Proximity-aware multiple meshes decimation using quadric error metric

Anahid Ghazanfarpour

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *22/11/2019* par :

Anahid GHAZANFARPOUR

Proximity-Aware Multiple Meshes Decimation using Quadric Error Metric

---

École doctorale et spécialité :

MITT : Image, Information, Hypermédia

Unité de Recherche :

Institut de Recherche en Informatique de Toulouse (UMR 5055)

Directeurs de Thèse :

Jean-Pierre JESSEL	*Professeur à l'Université Toulouse III Paul Sabatier*
Nicolas MELLADO	*Chargé de Recherche au CNRS*

JURY

Loïc BARTHE	*Professeur à l'Université Toulouse III Paul Sabatier*	Examinateur
David CAZIER	*Professeur à l'Université de Strasbourg*	Rapporteur
Marc DANIEL	*Professeur à l'Université d'Aix-Marseille*	Examinateur
Gilles GESQUIÈRE	*Professeur à l'Université Lyon II Lumière*	Rapporteur
Nancy RODRIGUEZ	*Maître de Conférences à l'Université de Montpellier*	Examinatrice

Proximity-Aware Multiple Meshes Decimation using Quadric Error Metric

Anahid GHAZANFARPOUR

Abstract

Progressive mesh decimation by successively applying topological operators is a standard tool in geometry processing. A key element of such algorithms is the error metric, which allows to prioritize operators minimizing the decimation error. Most previous work focus on preserving local properties of the mesh during the decimation process, with the most notable being the Quadric Error Metric which uses the edge collapse operator. However, meshes obtained from CAD scenes and describing complex systems often require significant decimation for visualization and interaction on low-end terminals. Hence preserving the arrangement of objects is required in such cases, in order to maintain the overall system readability for applications such as on-site repair, inspection, training, serious games, etc. In this context, this thesis focuses on preserving the readability of proximity relations between meshes during decimation, by introducing a novel approach for the joint decimation of multiple triangular meshes with proximities.

The works presented in this thesis consist in three contributions. First, we propose a mechanism for the simultaneous decimation of multiple meshes. Second, we introduce a proximity-aware error metric, combining the local edge error (i.e. Quadric Error Metric) with a proximity penalty function, which increases the error of edge collapses modifying the geometry where meshes are close to each other. Last, we devise an automatic detection of proximity areas. Finally, we demonstrate the performances of our approach on several models generated from CAD scenes.

Keywords Computer graphics, Mesh processing, Mesh decimation, Quadric error metric, Virtual disassembly

Résumé

La décimation progressive de maillage par l'application successive d'opérateurs topologiques est un outil standard de traitement de la géométrie. Un élément clé de tels algorithmes est la métrique d'erreur, qui donne la priorité aux opérateurs minimisant l'erreur de décimation. La plupart des travaux précédents se concentrent sur la préservation des propriétés locales du maillage lors du processus de décimation, le plus notable étant la métrique d'erreur quadrique qui utilise l'opérateur d'effondrement d'arête. Toutefois, les maillages obtenus à partir de scènes issues de CAO et décrivant des systèmes complexes requièrent souvent une décimation significative pour la visualisation et l'interaction sur des terminaux bas de gamme. Par conséquent, la préservation de la disposition des objets est nécessaire dans de tels cas, afin de préserver la lisibilité globale du système pour des applications telles que la réparation sur site, l'inspection, la formation, les jeux sérieux, etc. Dans ce contexte, cette thèse a trait à préserver la lisibilité des relations de proximité entre maillages lors de la décimation, en introduisant une nouvelle approche pour la décimation conjointe de multiples maillages triangulaires présentant des proximités.

Les travaux présentés dans cette thèse se décomposent en trois contributions. Tout d'abord, nous proposons un mécanisme pour la décimation simultanée de multiples maillages. Ensuite, nous introduisons une métrique d'erreur sensible à la proximité, combinant l'erreur locale de l'arête (i.e. la métrique d'erreur quadrique) avec une fonction pénalisant la proximité, ce qui augmente l'erreur des effondrements d'arête là où les maillages sont proches les uns des autres. Enfin, nous élaborons une détection automatique des zones de proximité. Pour finir, nous démontrons les performances de notre approche sur plusieurs modèles générés à partir de scènes issues de CAO.

Mots-clés Informatique graphique, Traitement de maillage, Décimation de maillage, Métrique d'erreur quadrique, Démontage virtuel

Remerciements

Je tiens à remercier en premier lieu mes encadrants grâce à qui cette thèse fut une expérience tout aussi enrichissante que plaisante, tant d'un point de vue scientifique que d'un point de vue humain. Je m'estime très chanceuse d'avoir passé ces années à travailler à leurs côtés.

Je remercie mon directeur de thèse Jean-Pierre Jessel pour la confiance qu'il m'a accordée tout au long de ma thèse, pour ses encouragements répétés, pour m'avoir aidée à naviguer dans les affres de l'administration et enfin pour sa disponibilité ainsi que son soutien fort précieux dans les moments difficiles.

Je remercie également mon co-directeur de thèse Nicolas Mellado, un jeune chercheur fort talentueux dont la rigueur et l'esprit critique ont pesé de façon significative sur ma thèse et furent très formateurs pour moi. Sa guidance, son aide face aux challenges techniques ainsi que pour l'écriture de nos papiers, ont été indispensables à mes travaux. Je le remercie aussi pour sa patience et sa bienveillance à mon égard, et j'en profite également pour remercier son adorable femme Charlène.

Enfin je remercie Loïc Barthe qui a été pour moi tout comme un directeur de thèse et dont l'expérience et la positivité ont eu un grand impact sur mes travaux. Son soutien et sa sympathie me furent également très précieux tout au long de ma thèse.

Je tiens ensuite à remercier David Cazier et Gilles Gesquière pour avoir accepté d'être les rapporteurs de ma thèse, ainsi que Marc Daniel et Nancy Rodriguez, pour leur participation à mon jury de thèse. Je les remercie pour le temps qu'ils m'ont consacré ainsi que pour leurs remarques constructives.

Je remercie également David Vanderhaegue et Mathias Paulin pour leur accueil chaleureux au sein de l'équipe STORM, Véronique Gaildrat pour avoir partagé son bureau avec moi pendant les derniers mois de ma thèse, ainsi que Géraldine Morin pour l'intérêt qu'elle a manifesté pour mes travaux.

J'ai une pensée pour les nombreux doctorants, post-doctorants, ingénieurs et stagiaires que j'ai eu le plaisir de côtoyer tout au long de ma thèse. Je remercie en particulier Caroline pour m'avoir prise sous son aile lors de mon arrivée au laboratoire, Céline et Florian pour m'avoir souvent aidée, pour leur soutien et surtout pour leur amitié, Nadine qui fut pour moi une source d'inspiration, Thomas, Charly, Baptiste, Even, Maurizio, mon voisin de table Thibault, Jie, François, Hugo, Olivier, Chems Eddine, Kevin, et Pierre. Je remercie enfin Valentin qui a partagé mon quotidien de doctorante dès le premier jour, je garde un excellent souvenir de nos discussions diverses et passionnantes. Je le remercie également pour avoir continué à me soutenir sur la fin malgré la distance, ainsi que pour la relecture minutieuse de ma thèse.

Enfin je remercie mes collègues avec qui j'ai eu le plaisir d'enseigner lors de

mon année d'ATER à Limoges, en particulier Damien, Guillaume et Vincent. Je remercie également tous mes étudiants, aussi bien à Toulouse qu'à Limoges, qui m'ont beaucoup apporté et dont les progrès m'ont emplie de joie.

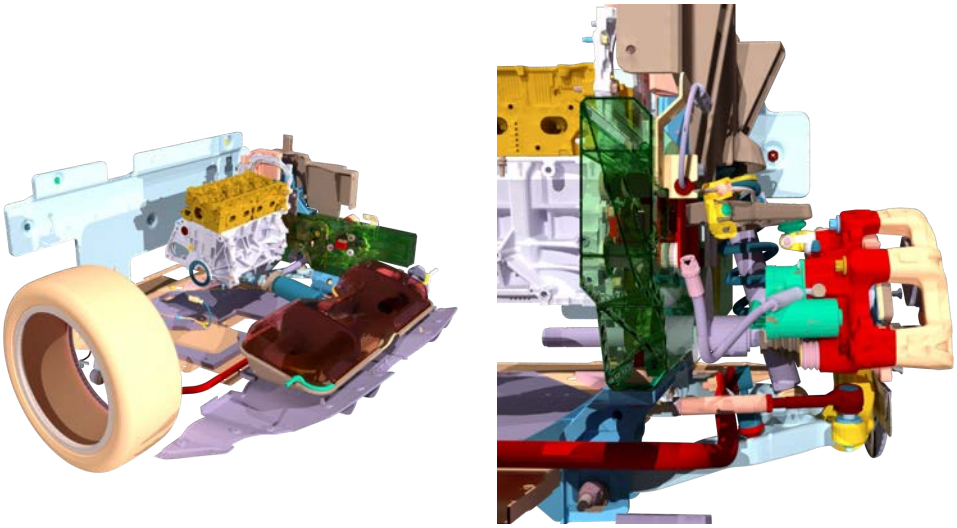
Je tiens enfin à remercier mes amis, aussi bien à Toulouse qu'aux quatre coins de la France et du monde, pour leur soutien tout au long de ma thèse. Je remercie en particulier Charlotte qui fut également une doctorante toulousaine, Christelle pour sa longue amitié, Arezou et Sara avec qui j'ai passé un bon nombre de mes meilleurs moments à Toulouse, ainsi qu'Agathe pour son amitié indéfectible et tous les bons moments qu'on a passé ensemble. Je remercie enfin Alban pour son soutien constant pendant les moments les plus importants de ma thèse et pour m'avoir inspirée à donner le meilleur de moi-même.

Pour finir, je remercie évidemment ma famille, et en particulier mes parents pour leur amour, leur soutien inconditionnel et pour m'avoir toujours poussée à aller au bout de moi-même dans mes études, et mon petit frère Arya, qui m'a toujours entourée d'une affection sans limites.

Contents

Introduction

Context and Motivations



(a) Scene overview (b) Close-up view

Figure 1: Complex 3D scene [Gha+19]. (a) Multiple models compose the scene. (b) Different models are related to each other and these relations encode the semantics of the scene.

Complex 3D scenes are often modeled by very large and detailed models representing the geometric information. Furthermore, the models representing man-made or Computer-Aided Design (CAD) scenes also encode the functional meaning of a scene through their relations with each other, as illustrated in Figure 1. Although CAD primitives contain much information about a scene, CAD scenes are usually exported from CAD software in the form of meshes as these informations are often kept confidential and costly to render, while meshes are easier to process.

Visualizing and interacting with complex 3D data is very challenging since the data scale and complexity 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 data transfer and rendering even on low-end terminals. Despite their limited performances, mobile platforms are very attractive as they bring access to rich on-site information, and help users to perform 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.

The works of this thesis fit in the broader context of an industrial research project aiming at training and bringing on-site information for machinery maintenance by means of a serious game on low-end mobile terminals. As complex CAD scenes are composed by numerous objects, some objects are usually partially or completely hidden by other objects. Thus, in order to have access to every object in a scene, the user should be able to interact with the scene through virtual disassembly. Furthermore, as the arrangement of the objects encodes the semantics of a scene, virtual disassembly enables the user to look into the relations between objects and hence to have a better understanding of the scene functionality. Therefore, virtual disassembly allows to consider an object both individually and as part of a scene.

Accordingly, in order to enable the visualization and interaction with a complex CAD scene in real-time on a low-end terminal, through virtual disassembly in particular, the complexity of the 3D data modeling the scene has to be reduced while preserving both the geometry and the functional meaning of the scene through the geometry of each object and its relations with other objects, with each object being modeled by a mesh.

Several approaches can mitigate the increasing complexity of 3D datasets for low performance hardware. The visibility [BW03] can be used to not load hidden parts of a scene. However, interacting with a scene requires access to all meshes and when working offline, no loading can be achieved. The instances of models composing a scene can be detected to avoid loading duplicates. However, meshes exported from CAD software are usually different for instances of a same model as they have different topology, i.e. different vertices, edges and faces, even though they have the same geometry, i.e. the same shape. The most common and convenient approach to reduce data complexity is mesh simplification, which consists in reducing the polygon count of a large mesh using decimation algorithms [Lue01]. Most decimation algorithms are incremental and consist in applying a topological operator on a mesh at each decimation step. An error metric estimates the local changes that would occur in the geometry following an operation on a mesh. Thus, an error value is associated to each possible operation, allowing to prioritize the operation with the lowest error value. The most popular decimation algorithm combines the edge collapse operator with the *Quadric Error Metric (QEM)* [GH97]. This algorithm collapses an edge into a vertex minimizing the point-to-plane distance with

Figure 2: Triangular mesh decimation [Cac19]. The complexity of the topology describing the geometry of the mesh is reduced.

regard to its local neighborhood at each decimation step. Mesh simplification can also use additional CAD data if provided, such as features [Gao+10] or appearance attributes [GH98; Hop99]. However, view-dependant mesh decimation [Hop97] is not an option for the same reasons as the visibility cannot be used to optimize data loading.

Meshes exported from CAD software often display some inconsistencies, such as cracks or intersections. Therefore, mesh repairing [BK05; ACK13] is performed to clean up meshes before decimation. Furthermore, as triangle meshes are the most commonly used polygon meshes in computer graphics, most topological operators are designed for triangle meshes. Hence, polygon triangulation algorithm [NM95] is applied to non-triangle meshes prior the decimation. A basic example of triangular mesh decimation can be seen in Figure 2.

CAD scenes are often very large and complex scenes modeled by multiple meshes with each mesh encoding the geometry of a model. Moreover, the relations between such meshes encode the functional meaning of a scene. Therefore, preserving the semantics of a scene in addition to its geometry is mandatory to still understand the modeled system, even though the meshes have been highly decimated to be manipulated on low-end remote terminals. As the semantic information is usually not provided as input, the decimation of scenes composed by multiple meshes relies only on the geometry, and preserving both the individual properties of each mesh and the relations between meshes is very challenging.

Contributions and Outline

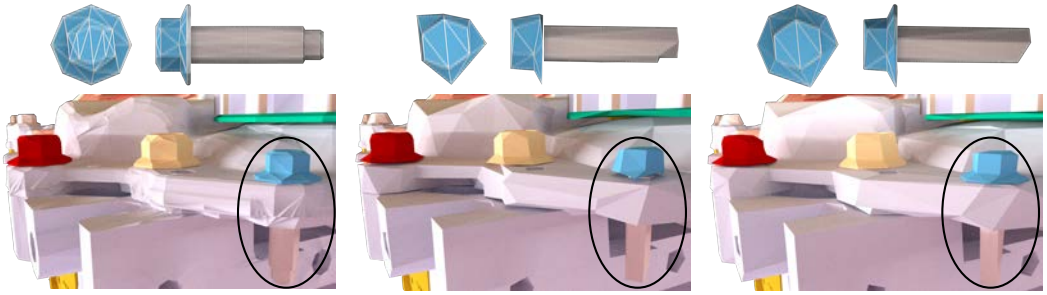
The problematic of this thesis is to decimate 3D scenes consisting of multiple meshes while preserving the geometry of each mesh along with its relations with other meshes in the scene. We observed that in mechanical systems, the shape of an object is designed according to its functionality and its interactions with the surroundings, e.g. contacts and arrangement. As such, we consider the neighboring meshes for the decimation of a given mesh in a scene. We define the concept of neighborhood in a scene by the proximity between parts of different meshes, as opposed to the local neighborhood which is defined within a mesh itself.

We propose a solution to this problematic by extending the incremental decimation algorithm for a mesh to multiple meshes, and using it with the edge collapse operator along with a novel error metric considering both the local geometry and the proximity relations with other meshes, which we refer to as *proximity-aware error metric*. We formulate the neighboring information as a proximity error which we use to mitigate the importance of the geometric structures according to the surrounding meshes. Thereby, we use the proximity error to penalize the local error introduced by decimation operations in proximity parts, thus reducing their priority as compared to operations in other parts of the scene, which yields to delay their decimation and better preserve the relations between meshes, i.e. the shape of proximity parts, as seen in Figure 3. Furthermore, we provide a definition of proximity to parameterize the proximity-aware error metric.

Hence, our solution consists in three independent yet complementary contributions as they form a fully automatic pipeline for the proximity-aware decimation of multiple meshes, and each contribution can be overviewed as follows:

Simultaneous decimation of multiple meshes We combine the decimation operations of all meshes in a scene by means of a global structure, thus ensuring that all meshes are decimated with a consistent error. This multiple meshes decimation scheme does not increase the complexity required to update the affected operations following an operation on a mesh as compared to the single mesh decimation scheme, and performs with a minimal memory overhead due to the global structure.

Proximity-aware error metric We evaluate a new error for edge collapses at the collapsing vertex minimizing the QEM by penalizing the QEM with the proximity error. The idea is to benefit from the robustness of the QEM collapsing vertex placement while increasing the error associated to edge col-



(a) Original scene (b) QEM (c) Proximity-Aware EM

Figure 3: Scene decimation [Gha+18]. (a) A close-up on the scene seen in Figure 1 (425 meshes and a total of about 3 million faces). (b) The decimation with the Quadric Error Metric. (c) Our Proximity-Aware Error Metric preserves the shape of mesh parts that are close to other meshes in the scene, as illustrated by the hex head bolt and the preservation of the contact surface with its support. Both decimated scenes have 150 000 faces each.

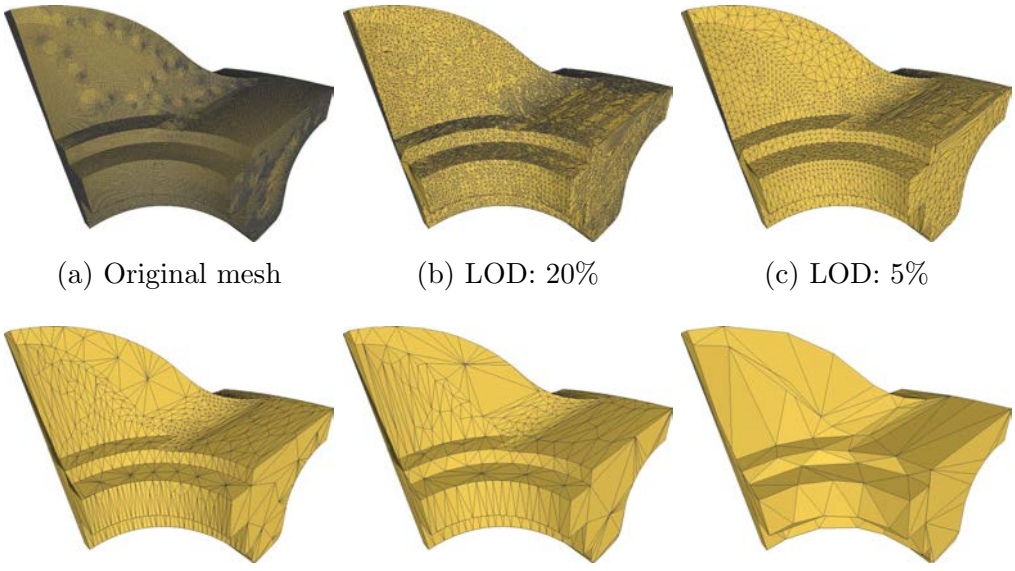
lapses in proximity parts of a scene. Thus, we do not change such collapses but only delay them, in order to better preserve the geometry where multiple meshes are close, at the expense of mesh parts outside proximity areas in the scene.

Proximity analysis in a scene We propose a generic approach to detect proximity areas in a scene. We analyze the spatial arrangement of the input meshes and their distances, from which we derive a distance, referred to as proximity threshold, for the automatic configuration of the proximity-aware error metric.

We first present a state of the art on *mesh decimation* in the aforementioned context (Chapter 1), which serves as basis for our contributions. Then we introduce *multiple meshes decimation* (Chapter 2), which we use for the *proximity-aware decimation* (Chapter 3) parameterized following a *proximity analysis* (Chapter 4). Finally, we review the *results* of this fully automatic pipeline (Chapter 5) and conclude this thesis by discussing perspectives of our works.

Chapter 1

Mesh Decimation



(a) Original mesh (b) LOD: 20% (c) LOD: 5%

(d) LOD: 1% (e) LOD: 0.5% (f) LOD: 0.1%

Figure 1.1: Levels of detail for a fan disk model. The original mesh has about 200 000 faces.

Mesh decimation, which is part of remeshing [All+08; Hu+16], is the process of modifying the tessellation of an input mesh in order to reduce its complexity. To do so, the mesh is turned into a mesh with fewer faces, edges and vertices, while preserving as much as possible the geometry and therefore the overall aspect of the mesh. A hierarchy of meshes with decreasing number of faces can be generated for a mesh by using several *levels of detail (LODs)* depending on the face budget, as illustrated by Figure 1.1.

Mesh decimation is usually performed on *manifold* meshes since certain mesh operations are not compatible with non-manifold meshes. Hence, the

manifoldness of a mesh is a very desirable property. A mesh is manifold if each edge is incident to no more than two faces and if each vertex is shared only by faces sharing the same fan, i.e. each of these faces shares at least one edge with another one of these faces. Figure 1.2 and Figure 1.3 show a non-manifold vertex and a non-manifold edge, respectively.

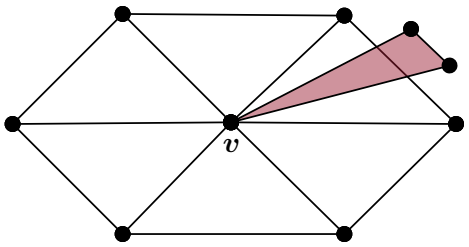


Figure 1.2: Non-manifold vertex. The vertex $\boldsymbol{v}$ is non-manifold as it is also shared by a face which is not part of its fan, making the mesh non-manifold.

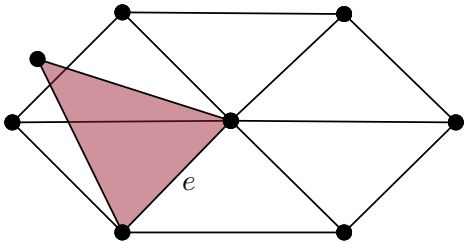


Figure 1.3: Non-manifold edge. The edge e is non-manifold as it is shared by more than two faces, making the mesh non-manifold. The endpoints of e are also non-manifold.

Mesh decimation has been extensively studied over the last few decades [HG97; CMS98; Lue01; Lue+03; BK04; MP06; Bot+10]. This chapter first reviews general mesh decimation by presenting the two main families of methods, on the one hand *vertex clustering* techniques which group sets of vertices and replace them by a single vertex (Section 1.1), and on the other hand *incremental decimation* techniques which sequentially apply a given decimation operation on the mesh (Section 1.2). Then it introduces some derived methods focusing on *feature preservation* while decimating the mesh (Section 1.3). Unlike mesh decimation, the focus on *scene decimation* with multiple meshes has been less notable (Section 1.4). Finally, the technical

choices with regard to the previously mentioned approaches, which constitute the basis for the contributions of this thesis, are discussed (Section 1.5).

1.1 Vertex Clustering

Vertex clustering methods group sets of vertices and replace them by a single vertex. Vertex clustering generates discrete LODs as the algorithm needs to be run n times to get n LODs.

Some approaches first set new vertices and then replace each original vertex by one these new vertices. The new vertices can be a subset of the original vertices. Turk [Tur92] evenly distributes a new set of vertices over a triangulated surface. This intermediate model is called mutual tessellation as it contains both the original and the new vertices. The decimation is performed by removing each original vertex and locally reconnecting affected edges in a way that matches the local connectivity of the initial surface. Boubekeur and Alexa [BA09] use stochastic vertex selection based on a local feature estimator to better preserve areas of high curvature. First, each initial vertex is assigned to the closest selected vertex and then, the triangles are re-indexed, with only triangles having all three vertices assigned to different selected vertices being kept.

Some other approaches first group vertices in clusters and then replace each cluster by a representative vertex. Similarly, the representative vertices can be a vertex of the corresponding cluster, i.e. a subset of the original vertices. Hence, the bounding space around the mesh is partitioned into cells and a representative vertex is computed for each cell. The representative vertex of a cell is then assigned to all vertices falling into this cell. This process is illustrated by Figure 1.4.

Several partitioning techniques have been developed. The mesh can be clustered using a regular subdivision of its bounding box [RB93; Lin00]. Low and Tan [LT97] extended this idea to arbitrary shapes using voxel grids. Likewise, there are many ways to find the representative vertices. The representative vertex of a cell can be defined as the center of mass of its weighted vertices [RB93] or as its vertex of maximal weight [LT97]. Lindstrom [Lin00] used the Quadric Error Metric [GH97], described in Section 1.2.3, to compute the representative vertices. Each cell has a quadric which is initialized to zero. For each initial face having each vertex belonging to a different cell (i.e. the face is not discarded), the quadric is computed and added to the quadric of each cell it has a vertex in. Hence, once every face has been processed, the position of each vertex representative is computed using the quadric of its cell, which is more accurate than selecting the vertex

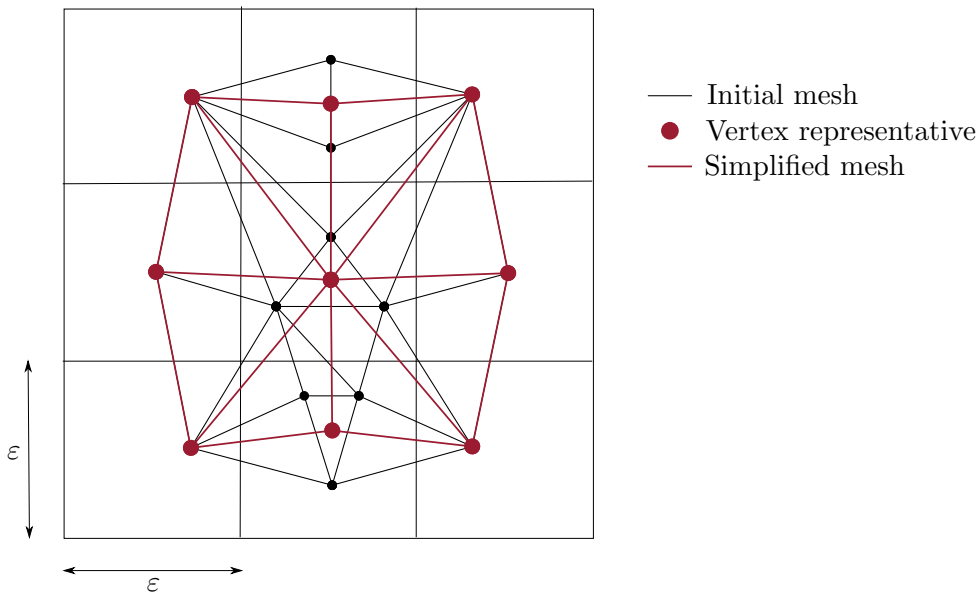


Figure 1.4: Vertex clustering. The bounding space around the mesh is partitioned into cells according to a given approximation tolerance ε, and a vertex representative is computed for each cell. The mesh is decimated by replacing each original vertex by its representative.

of maximal weight or a weighted average of vertices.

While vertex clustering is fast with a linear complexity with regard to the number of vertices of the mesh, it performs a global decimation, without granularity and hence not much control over the decimation process. Moreover, as it generates discrete LODs, there is no transition between the different levels, which leads to popping effects when switching from one LOD to another. Another drawback of vertex clustering is that it may lead to a non-manifold mesh even though the original mesh is manifold, as a portion of a surface could collapse to a vertex, as seen in Figure 1.5.

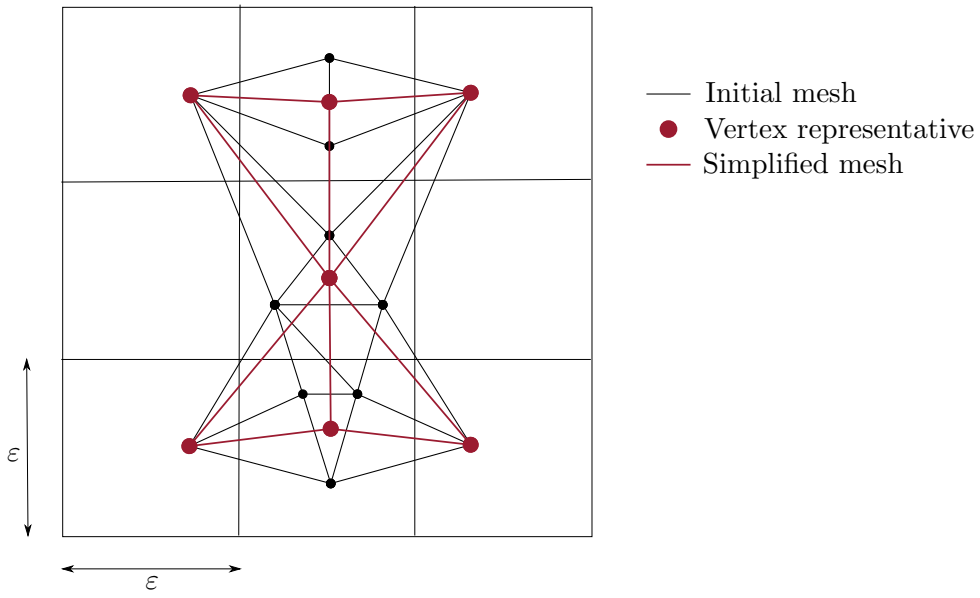


Figure 1.5: Non-manifold vertex clustering. The vertex representative of the central cell turns into a non-manifold vertex when used to generate the decimated mesh.

1.2 Incremental Decimation

Incremental decimation methods iteratively decimate the topology while minimizing a local error, until a stopping criterion is reached. Meshes processed this way are thus called *Progressive Meshes* [Hop96]. Incremental decimation performs a local decimation as the decimation is done step by step, with each step consisting of an approximation minimizing a local error,

and thus generates continuous LODs which results in smooth transitions between the different levels. Nonetheless, incremental decimation algorithms are greedy.

This section first introduces a generic *algorithm* for incremental decimation (Section 1.2.1). Then, it reviews different *topological operators* (Section 1.2.2) and *error metrics* (Section 1.2.3) that can be used as parameters of this algorithm.

1.2.1 Algorithm

Algorithm 1 Incremental decimation

Input: original mesh M, stopping criterion *stop_crit*
Output: decimated mesh M
$P \leftarrow$ empty priority queue
for each element e **do**
  if *op* is a possible operation on e **then**
    $err \leftarrow$ compute error associated to *op*
    Insert (*op*,*err*) in P
  end if
end for
while P not empty and *stop_crit* not reached **do**
  op $\leftarrow$ top operation of P
  Remove *op* from P
  Apply *op* on M
  for each neighbor operation op_i **do**
    Remove op_i from P
    $err_i \leftarrow$ compute error associated to op_i
    Insert (op_i,err_i) in P
  end for
end while

Incremental decimation algorithms combine a topological operator with an error metric, as well as a stopping criterion [KCS98]. Any topological operation is designed for a specific element (vertex, edge, or face) and associated to a cost. This cost quantifies the changes that would be caused to the geometry of the mesh by performing the operation, hence it is also referred to as the error associated to the operation.

For a given topological operator, the error of each possible operation on an input mesh is computed using a given error metric and sorted into a priority queue by ascending error value. Then, the decimation is performed by iteratively popping out and applying the operation with the smallest error value

on the mesh, i.e. the one located on top of the priority queue. After each operation, the operations associated to affected elements in the neighborhood are updated in the priority queue and their new error value is computed. This process is run as long as the priority queue is not empty or the stopping criterion is not reached. The stopping criterion can be either a target number of faces, edges, or vertices, a target number of operations, an error threshold, some custom quality criteria for the mesh, etc, or a combination of some criteria. Algorithm 1 is a generic algorithm for incremental decimation regardless the topological operator, error metric and stopping criterion used.

For each operation, the complexity of updating the priority queue is $O(2k \log(E))$, where k is the number of operations to update (find to remove, and reinsert) in a queue of E elements.

1.2.2 Topological Operators

The topological operator is the main parameter of any incremental decimation algorithm as most error metrics are designed for a specific operator. There are several topological operators, which consist in removing either a single vertex, edge, or face, along with its adjacent entities. Topological operators are local as they affect only the neighborhood of the entity they are associated to.

An incremental decimation algorithm consists of many small and successive steps. At each step, a topological operator is applied to the mesh. As most topological operators have an inverse operator, it is therefore possible to reinject detail to the mesh.

Vertex Removal

The *vertex removal* operator consists in first removing a single vertex along with its adjacent edges, which leaves a hole into the mesh, and then triangulating this hole [SZL92; KLS96], as illustrated by Figure 1.6.

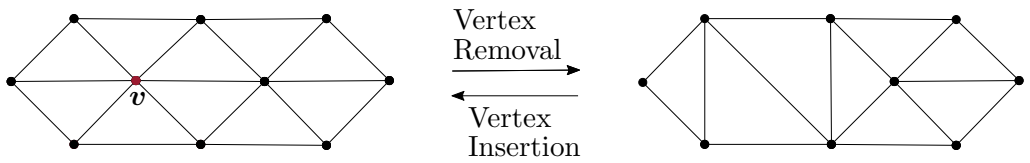


Figure 1.6: Vertex Removal. The vertex $\boldsymbol{v}$ is removed and the subsequent hole is triangulated.

The removal of a vertex of valence k results in a $k-1$-sided opening in the case of an open mesh in which the removed vertex is a border vertex, and a

k-sided hole if the removed vertex is not a border vertex or in the case of a closed mesh. This hole has to be triangulated by adding back $k-2$ faces. As a consequence, one vertex and one or two faces are removed overall by a vertex removal operator.

The inverse operator of vertex removal is *vertex insertion*, which consists in inserting a new vertex on the mesh and removing its neighbor faces, and then triangulating the resulting hole by inserting faces adjacent to this new vertex.

The main shortcoming of vertex removal is the fact that the triangulation of faces removed and faces inserted can differ significantly, which means that important changes can happen in the topology with a single operation.

Edge Collapse

The *edge collapse* operator consists in collapsing an edge into a single vertex [Hop+93]. As a consequence, an edge collapse removes an edge along with the faces sharing it, as illustrated by Figure 1.7.

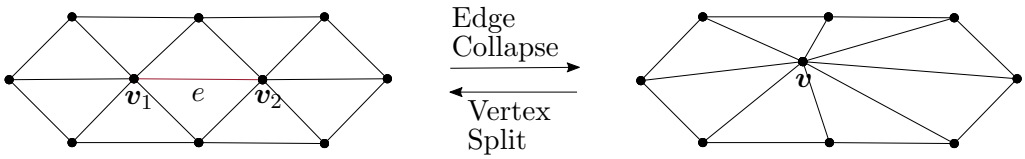


Figure 1.7: Edge Collapse. The edge e is collapsed into the vertex $\boldsymbol{v}$ and its neighborhood is updated with regard to $\boldsymbol{v}$.

In the case of an open mesh in which the collapsed edge is a border edge, one face is removed, and if the collapsed edge is not a border edge or in the case of a closed mesh, two faces are removed. Indeed, as mentioned earlier at the beginning of the chapter, an edge shared by more than two faces would not be manifold, turning the mesh into a non-manifold mesh.

The collapsing vertex can be either one endpoint of the edge, the middle of the edge, or an optimal collapsing position minimizing a local error metric [GH97; LT98] (see Section 1.2.3). Moreover, all edges and faces adjacent to one endpoint of the collapsed edge are updated as both endpoints of the collapsed edge are replaced by the vertex resulting from the collapse.

The inverse operator of edge collapse is *vertex split*, which consists in dividing a vertex into two new vertices to form an edge, as well as one or two resulting faces shared by these two new vertices and one or two other vertices, respectively.

Some edge collapses may introduce inconsistencies in the mesh. Firstly, an edge collapse might cause *mesh inversion*. Indeed, the choice of the collapsing vertex could change the orientation of faces in the area of the collapse, thus causing the mesh to fold over itself, as seen in Figure 1.8. To avoid such

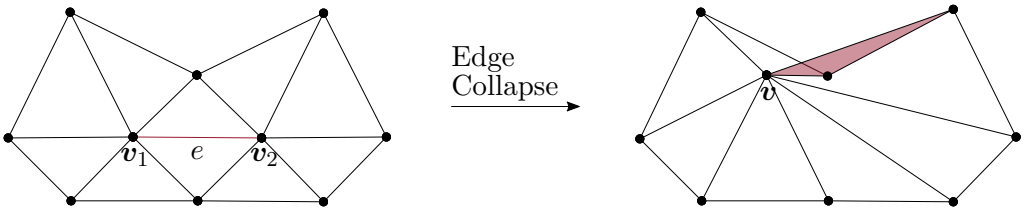


Figure 1.8: Mesh inversion. The collapse of edge e into vertex $\boldsymbol{v}$ causes a face to fold over the mesh.

a configuration, the normal vectors of the faces adjacent to an edge can be compared before and after the collapse. If the normal vector of a face changes by more than a certain threshold, the face is regarded as flipped. Garland [Gar99] proposes a more robust approach. For every face around an endpoint $\boldsymbol{v}_1$ of an edge e, excluding faces shared with the other endpoint $\boldsymbol{v}_2$, there is an edge opposite $\boldsymbol{v}_1$. The collapsing vertex of the edge must lie on the same side of a plane perpendicular to such a face through this opposite edge, as $\boldsymbol{v}_1$. The same condition applies to the faces around $\boldsymbol{v}_2$, excluding faces shared with $\boldsymbol{v}_1$. Secondly, an edge collapse might cause a *non-manifold edge*. Depending on the topology around an edge, its collapse could generate an edge shared by more than two faces, turning it into a non-manifold mesh. A non-manifold edge appears following the collapse of an edge for which the one-ring neighborhood of both endpoints intersect in more than two vertices, i.e. an edge whose endpoints form edges with more than two same vertices, as seen in Figure 1.9.

Figure 1.9: Non-manifold edge. The collapse of edge e into vertex $\boldsymbol{v}$ generates a non-manifold edge.

Such edge collapses causing a mesh inversion or a non-manifold edge are usually discarded and only reconsidered when topological changes occur in their local neighborhood, thus modifying the resulting vertex.

Halfedge Collapse

The *halfedge collapse* operator consists in collapsing an edge into one of its endpoints [RR96], as illustrated by Figure 1.10. Therefore, the halfedge collapse is a particular case of edge collapse in which the collapsing vertex is an endpoint of the edge. Thus, one endpoint of the edge is removed while the other one remains as it is.

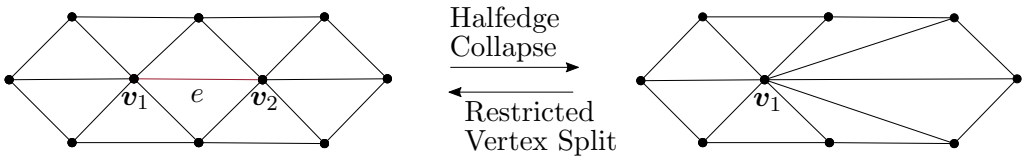


Figure 1.10: Halfedge Collapse. The edge e is collapsed into its endpoint $\boldsymbol{v}_1$ and the neighborhood of its other endpoint $\boldsymbol{v}_2$ is updated with regard to $\boldsymbol{v}_1$.

As for the edge collapse, a halfedge collapse removes an edge along with the faces sharing it, but unlike edge collapse, only edges and faces adjacent to the removed endpoint of the edge are updated.

The inverse operator of halfedge collapse is *restricted vertex split*, which consists in inserting a new vertex to form an edge with another vertex, as well as one or two resulting faces shared by these two vertices and one or two other vertices, respectively.

The main shortcoming of halfedge collapse is the fact that the endpoints of an edge are usually not optimal candidates for its collapse with regard to its local neighborhood.

Vertex Contraction

The *vertex contraction* operator consists in merging two unconnected vertices into a single vertex [GH97; Sch97], as illustrated by Figure 1.11.

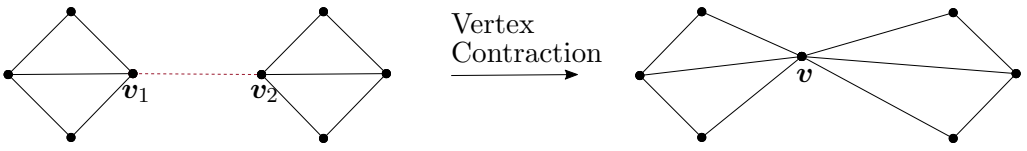


Figure 1.11: Vertex Contraction. The vertices $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are merged into vertex $\boldsymbol{v}$.

It is an extension of the edge collapse operator to unconnected yet close pairs of vertices. Such pairs of vertices are considered as virtual edges in order

to close small holes rather than extending them during the decimation process, for instance in the case of meshes with several connected components.

The vertex contraction operator removes only a single vertex since the contracted vertices do not share any edge nor face as they are unconnected. As a consequence, a limitation of this operator is the generation of non-manifold meshes.

Face Removal

The *face removal* operator consists in collapsing a face into a single vertex [Ham94], as illustrated by Figure 1.12.

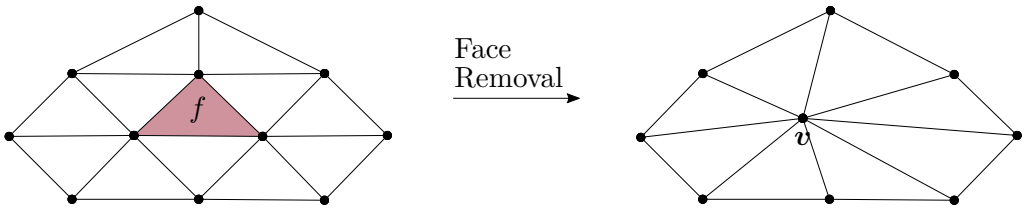


Figure 1.12: Face Removal. The face f is collapsed into vertex $\boldsymbol{v}$ and its neighborhood is updated with regard to $\boldsymbol{v}$.

The removal of a face results also in the removal of all faces it shares an edge with. Hence, $k+1$ faces are removed overall, with k the number of edges of the removed face shared with another face. As a face has three edges, a maximum of four faces can be removed at once. Moreover, all edges and faces adjacent to the removed face are updated.

The main shortcoming of face removal is the fact that it collapses at once three edges and the faces sharing them, which provides less control over the topology than with operators removing less entities.

1.2.3 Error Metrics

The decimation process of a mesh consists in applying successive topological operators. The cost of an operation, which defines its position in the priority queue as compared to the other operations, is characterized by an error metric. Hence, the decimation is driven by the error metric and the choice of such a metric will be highly influenced by the choice of the topological operator. Furthermore, as the topological operators are local, so are the error metrics.

Besides its position in the priority queue, a topological operator is also influenced by the error metric regarding its outcome as the error metric computes the cost of an operation by minimizing the error induced in the mesh.

For instance, the vertex resulting from an edge collapse will be the one minimizing the error metric, i.e. the one that has the less impact on the geometry of the mesh. It should be noted that when an operation takes place on a planar region of the mesh, its error is equal to zero, meaning that it impacts only the topology and not the geometry.

The following paragraphs provide an overview of the most commonly used error metrics.

Hausdorff distance

The *Hausdorff distance* measures the distance between two surfaces. Hence, it can be used to compute the maximum distance between the original and the decimated mesh, which represents the geometric deviation caused by a topological operator.

Let $\mathcal{A}$ and $\mathcal{B}$ be two meshes. The minimum distance to $\mathcal{B}$ of a vertex $\boldsymbol{a}$ belonging to $\mathcal{A}$ is:

$$d(\boldsymbol{a}, \mathcal{B}) = \min_{b \in \mathcal{B}} ||\boldsymbol{a} - b|| \tag{1.1}$$

The Hausdorff distance from $\mathcal{A}$ to $\mathcal{B}$ is the maximum minimum distance to $\mathcal{B}$ of a vertex belonging to $\mathcal{A}$:

$$\begin{aligned} h(\mathcal{A}, \mathcal{B}) &= \max_{\boldsymbol{a} \in \mathcal{A}} d(\boldsymbol{a}, \mathcal{B}) \\ &= \max_{\boldsymbol{a} \in \mathcal{A}} \min_{b \in \mathcal{B}} ||\boldsymbol{a} - b|| \end{aligned} \tag{1.2}$$

The Hausdorff distance is non-symmetric as shown in Figure 1.13:

$$h(\mathcal{A}, \mathcal{B}) \neq h(\mathcal{B}, \mathcal{A}) \tag{1.3}$$

Therefore, the Hausdorff distance from one surface to another surface is called the one-sided Hausdorff distance. A way to symmetrize the Hausdorff distance between two surfaces $\mathcal{A}$ and $\mathcal{B}$ is to compute the maximum of both one-sided Hausdorff distances:

$$H(\mathcal{A}, \mathcal{B}) = \max(h(\mathcal{A}, \mathcal{B}), h(\mathcal{B}, \mathcal{A})) \tag{1.4}$$

Klein, Liebich, and Straßer [KLS96] use the Hausdorff distance between the original and the decimated mesh as error metric while decimating a mesh using the vertex removal operator. As the vertex removal is a local topological operator, a change in the Hausdorff distance can be evaluated only locally

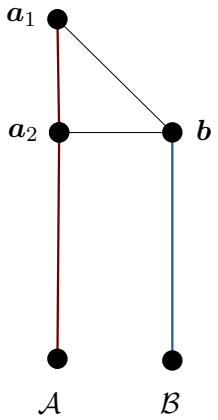


Figure 1.13: Non-symmetric Hausdorff distance. The Hausdorff distance between $\mathcal{A}$ and $\mathcal{B}$ is non-symmetric as $h(\mathcal{A}, \mathcal{B}) = d(\boldsymbol{a}_1, \boldsymbol{b})$ and $h(\mathcal{B}, \mathcal{A}) = d(\boldsymbol{b}, \boldsymbol{a}_2)$.

around the affected neighborhood. Thus, at each step of the decimation, the Hausdorff distance is updated thanks to a mapping between the original and the decimated mesh.

The Hausdorff distance is the most accurate error metric but also the most computationally expensive, while an estimated error is generally enough for most applications.

Distance to average plane

Schroeder, Zarge, and Lorensen [SZL92] use the distance of a vertex to an average plane as error metric while decimating a mesh using the vertex removal operator. This average plane is computed from the faces adjacent to the vertex. The normal vector of an average plane is the average of the normal vectors of the associated faces weighted by their area. A point on the average plane is computed using the average of the centroids of the associated faces, also weighted by their area.

Let $\tau(\boldsymbol{v})$ be the set of faces in the one-ring neighborhood of a vertex $\boldsymbol{v}$, and $\mathcal{A}_i$, n_i and c_i be the area, normal vector and centroid of a face f_i, respectively.

The average plane of $\boldsymbol{v}$ is defined by its normal vector n and a point c:

$$n = \frac{\sum\limits_{f_i \in \tau(\boldsymbol{v})} \mathcal{A}_i n_i}{\sum\limits_{f_i \in \tau(\boldsymbol{v})} \mathcal{A}_i} \tag{1.5}$$

$$c = \frac{\sum\limits_{f_i \in \tau(\boldsymbol{v})} \mathcal{A}_i c_i}{\sum\limits_{f_i \in \tau(\boldsymbol{v})} \mathcal{A}_i} \tag{1.6}$$

The equation of the average plane leads to:

$$\begin{aligned} & n^T c + d = 0 \\ \Leftrightarrow\ & d = -n^T c \end{aligned}$$

where d is the signed distance to the origin. The signed distance of $\boldsymbol{v}$ to its average plane is:

$$\begin{aligned} \Delta(\boldsymbol{v}) &= n^T \boldsymbol{v} + d \\ &= n^T \boldsymbol{v} - n^T c \end{aligned}$$

Therefore, the error value associated to the removal of $\boldsymbol{v}$ is:

$$\Delta(\boldsymbol{v}) = |n^T(\boldsymbol{v} - c)| \tag{1.7}$$

This error metric computes the deviation from the current mesh and not from the initial mesh. Thus, its main drawback is the fact that it does not keep track of the error accumulation over the course of the decimation process.

Maximum distance to supporting planes

Ronfard and Rossignac [RR96] use the maximum squared distance of a vertex to its *supporting planes* as error metric while decimating a mesh using the halfedge collapse operator. Each vertex of the initial mesh has a supporting plane per adjacent face. As a consequence, the set of supporting planes for the vertex resulting from the collapse of an edge is the union of the sets of supporting planes of both endpoints of the edge. This set of planes grows during the decimation process, as each collapsed edge passes on its set of supporting planes to its collapsing vertex which does the same when collapsed as part of an edge, etc.

Let $\boldsymbol{v}$ be the collapsing vertex of an edge e, $\mathcal{S}(e)$ the set of supporting planes of e, n_i and d_i the normal vector and signed distance to the origin of a plane $\mathcal{P}_i$, respectively. The error value associated to the collapse of e is the maximum squared distance from $\boldsymbol{v}$ to the set of supporting planes:

$$\Delta(\boldsymbol{v}) = \max_{\mathcal{P}_i \in \mathcal{S}(e)} (n_i{}^T \boldsymbol{v} + d_i)^2 \tag{1.8}$$

This error metric computes the deviation from the initial mesh as the sets of supporting planes are merged during each collapse operation. Thus, the error accumulation is taken into account throughout the decimation process. However, keeping track of the sets of supporting planes is expensive memory-wise.

Quadric Error Metric

Garland and Heckbert [GH97] introduce the *Quadric Error Metric (QEM)*. A set of supporting planes is associated to each vertex of the mesh as in the work of Ronfard and Rossignac [RR96]. The squared distance of a vertex to its set of planes is used as error metric while decimating a mesh using the edge collapse operator, and the vertex minimizing this distance is chosen as the result of the edge collapse. Figure 1.14 illustrates this quadric error.

(a) Original model (b) Decimated model (c) Quadric error

Figure 1.14: Decimation of the bunny model using QEM [GH97]. (a) The original bunny model has about 70 000 faces. (b) The bunny is approximated using 1000 faces. (c) The ellipsoids on the approximated bunny represent the quadric error and are centered at the isovalue of the quadric. They follow the local shape of the surface.

The squared distance from a vertex $\boldsymbol{v}$ to a plane $\mathcal{P}$, defined by its normal vector n and its signed distance to the origin d, is:

$$\begin{aligned} D^2(\boldsymbol{v}) &= (n^T\boldsymbol{v} + d)^2 \\ &= (\boldsymbol{v}^T n + d)(n^T\boldsymbol{v} + d) \\ &= (\boldsymbol{v}^T nn^T\boldsymbol{v} + 2dn^T\boldsymbol{v} + d^2) \end{aligned} \tag{1.9}$$

This distance can be defined as a quadratic form represented by a quadric Q:

$$Q = (A, b, c) \tag{1.10}$$

with

$$\begin{cases} A = nn^T \\ b = dn \\ c = d^2 \end{cases}$$

As a consequence, the squared distance from v to $\mathcal{P}$ is:

$$Q(\boldsymbol{v}) = \boldsymbol{v}^T A\ \boldsymbol{v} + 2b^T\boldsymbol{v} + c \tag{1.11}$$

The quadric Q can also be represented as an homogeneous matrix:

$$Q = \begin{pmatrix} A & b \\ b^T & c \end{pmatrix} \tag{1.12}$$

Hence, the squared distance from a vertex v to the plane associated to Q can be represented as the quadratic form:

$$Q(\boldsymbol{v}) = \boldsymbol{v}^T Q\ \boldsymbol{v} \tag{1.13}$$

Therefore, the quadric of a face f associated to a plane $\mathcal{P}$, defined by $p = (n, d)^T$ with n its normal vector and d its signed distance to the origin, is:

$$Q_f = pp^T \tag{1.14}$$

The quadric of a vertex $\boldsymbol{v}$ is the weighted sum of the quadrics of its adjacent faces:

$$Q_{\boldsymbol{v}} = \frac{\sum\limits_{f_i \in \tau(\boldsymbol{v})} w(f_i)\ Q_{f_i}}{\sum\limits_{f_i \in \tau(\boldsymbol{v})} w(f_i)} \tag{1.15}$$

with $\tau(\boldsymbol{v})$ the set of faces in the one-ring neighborhood of $\boldsymbol{v}$ and w the weighting scheme used for the faces. The faces can all have the same weight, be weighted using their area or cotangent weights, etc.

The quadric of an edge e is the mean of the quadrics of its endpoints:

$$Q_e = \frac{1}{2}(Q_{\boldsymbol{v}_1} + Q_{\boldsymbol{v}_2}) \tag{1.16}$$

with $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ the endpoints of e.

The error value associated to the collapse of an edge e is the evaluation of its quadric at its resulting vertex $\boldsymbol{v}$:

$$\Delta(\boldsymbol{v}) = \boldsymbol{v}^T Q_e \, \boldsymbol{v} \tag{1.17}$$

There are multiple possible placements for the resulting vertex of an edge collapse. The most intuitive choice is either one of the endpoints of the edge, as for a halfedge collapse, i.e. the one having the smaller QEM value. Thus, the vertices of the decimated mesh would be a subset of the vertices of the original mesh. However, an optimal placement of the resulting vertex provides a better approximation. Therefore, the resulting vertex is the one minimizing the QEM, as seen in Figure 1.15.

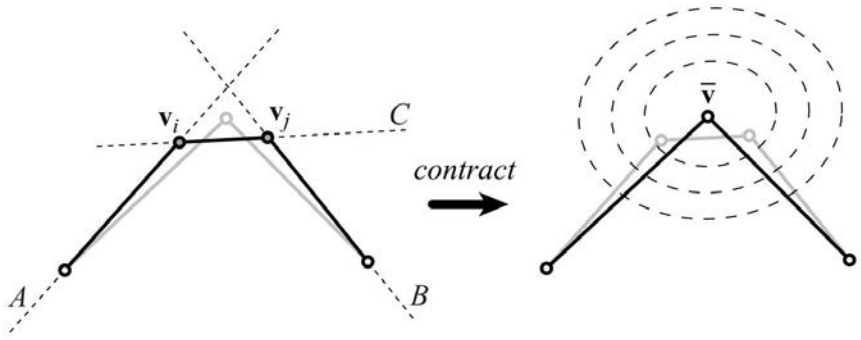


Figure 1.15: Edge collapse using QEM [GH97]. The resulting vertex minimizes the QEM. The ellipses around it represent isocontours representing various error values and correspond to the ellipsoids in 3D.

The gradient of Equation 1.11 is:

$$\nabla Q(\boldsymbol{v}) = 2A \, \boldsymbol{v} + 2b \tag{1.18}$$

Solving $\nabla Q(\boldsymbol{v}) = 0$ leads to the optimal placement:

$$\boldsymbol{v} = -A^{-1}b \tag{1.19}$$

Hence, the associated error is obtained by applying Equation 1.19 to Equation 1.11:

$$Q(\boldsymbol{v}) = -b^T A^{-1} b + c \tag{1.20}$$

However, the matrix A may not be invertible, i.e. a unique optimal position may not exist. In that case, an optimal position might be found along the edge. Otherwise, a halfedge collapse has to be performed by choosing the endpoint having the smallest QEM value.

Therefore, a single 4x4 symmetric matrix is enough to store all the data required for an edge collapse. Furthermore, since the QEM is quadratic, finding its minimum is a linear problem.

Volume preservation

Lindstrom and Turk [LT98] use volume preservation as error metric while decimating a mesh using the edge collapse operator. An edge is collapsed into a vertex minimizing the volume change of the model. For every face adjacent to one or both endpoints of an edge, a tetrahedron is formed with the vertex resulting from the collapse of the edge, as seen in Figure 1.16. The tetrahedral volume is positive if the resulting vertex is outside the model and negative if it is inside the model.

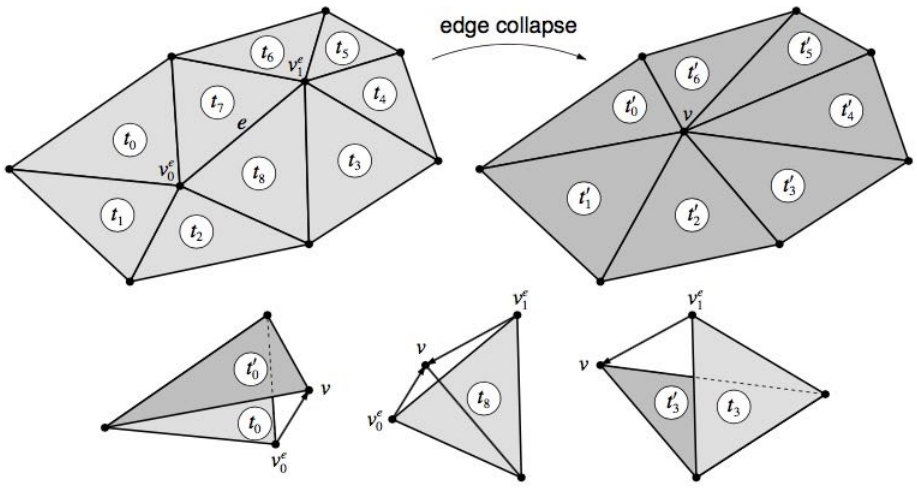


Figure 1.16: Edge collapse using tetrahedral volumes [LT98]. The resulting vertex minimizes the tetrahedral volumes associated to the faces adjacent to the collapsed edge.

To preserve the volume of the model, the vertex $\boldsymbol{v}$ resulting from the collapse of an edge e is found by solving the following equation:

$$\sum_{f \in \tau(e)} \mathcal{V}(\boldsymbol{v}, f) = 0 \tag{1.21}$$

with $\mathcal{V}$ the volume of a tetrahedron formed by $\boldsymbol{v}$ and a face f belonging to the set $\tau(e)$ of faces in the one-ring neighborhood of e.

As the aim is to minimize the volume of each tetrahedron, unsigned tetrahedral volumes are used to compute the error value while collapsing e into $\boldsymbol{v}$:

$$\Delta(\boldsymbol{v}) = \sum_{f \in \tau(e)} \mathcal{V}(\boldsymbol{v}, f)^2 \tag{1.22}$$

This error metric is memoryless as it does not retain any information about the original mesh and makes decisions based only on the current approximation.

1.3 Feature Preservation

The error metrics presented previously decimate a mesh by minimizing a local error. Thus, there is not much control over the global error, which may be problematic, especially in the case of extreme decimation, as successive approximations accumulate.

Furthermore, mesh decimation can aim at preserving certain properties of the mesh more so than others. These properties can be either detected on the mesh or specified directly by the user. Therefore, such an algorithm would trade off some global quality of the decimated mesh for preserving specific properties.

This section first introduces some approaches striving to control the global error by the mean of *tolerance volumes* (Section 1.3.1) or to preserve the structure of a mesh by using *planar proxies* (Section 1.3.2). Then, it reviews some methods focusing on preserving various *major features* (Section 1.3.3) and *user-guided* methods to do so (Section 1.3.4).

1.3.1 Tolerance Volumes

Several approaches strive to control the global error in order to avoid the error accumulation induced by local error metrics throughout the decimation process. Thus, the topological operations are restricted so that the decimated mesh stays within a tolerance volume regarding the original mesh. As a

consequence, the level of decimation of a mesh is directly influenced by the error bound. The less tight the bound is, the coarser the decimated mesh will be.

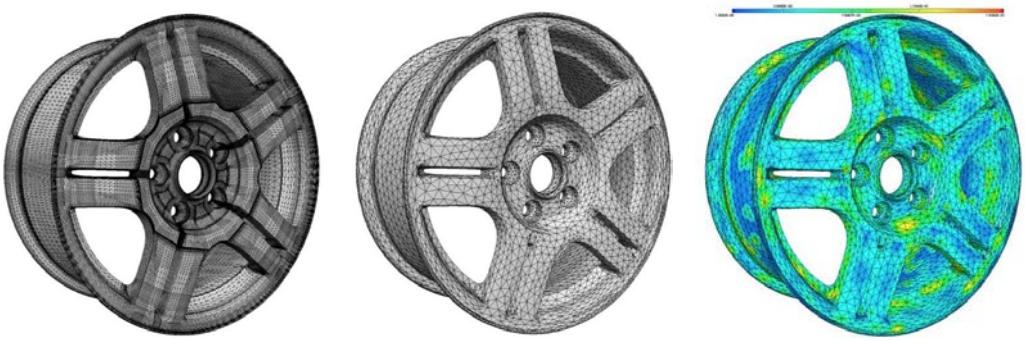
Cohen et al. [Coh+96] propose the idea of *decimation envelopes*. An inner and an outer envelope distanced from the original mesh by a certain threshold, and corresponding to an error bound, are created. These envelops are constructed by displacing the original vertices by this threshold along their normal vectors. As a consequence, a vertex removal operator is applied only if it does not cause the mesh to intersect with either envelope, i.e. if it has a re-triangulation scheme where none of the faces created to fill the resulting hole intersect an envelope.

Inspired by these envelopes surrounding a mesh, Borouchaki and Frey [BF05] also add to it cones centered at each vertex. The axis of a cone is defined by the normal vector of its vertex to the surface as well as a given aperture common to all cones. Hence, each face resulting from a topological operation must on the one hand belong to the global envelope of the mesh and on the other hand have its normal vector contained within the regularity cones associated with its vertices. Therefore, this approach is driven by two parameters, the tolerance volume and the aperture, referred to as the tolerance pair, as illustrated by Figure 1.17.

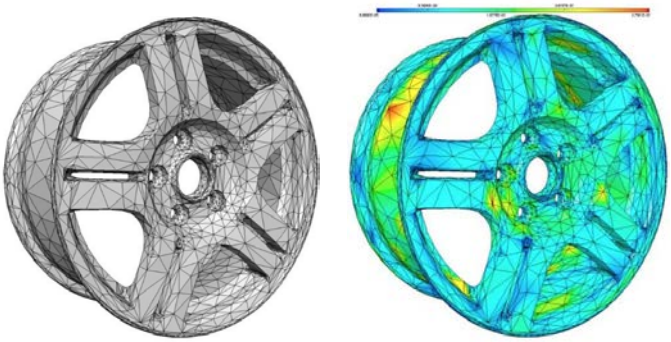
However there is no optimization of the error bound, as it remains the same for each operation throughout the decimation process.

Guéziec [Gué99] also uses a bounding volume, with the difference that the error bound is optimized for each operation. An error volume is measured locally per vertex, in the form of spheres centered at their corresponding vertex. Thus, linearly interpolating the spheres defines the current error volume of the mesh. The sphere of a vertex resulting from an edge collapse operation contains the spheres of both endpoints of the edge. As a consequence, the error volume of the mesh is grown iteratively throughout the decimation process.

Zelinka and Garland [ZG02] introduce the concept of *permission grid*. As in the work of Guéziec [Gué99], a sphere is defined per vertex of the original mesh, with the difference that the radius is the same for all spheres. A bounding volume is created per face by linearly interpolating the spheres of its vertices. These bounding volumes are then voxelized using a uniform grid. The voxels are divided into two categories, on the one hand those that are completely within the bounded volumes of the faces and can subsequently be intersected by new faces, and on the other hand those that are not entirely within these boundaries and thus cannot be intersected by new faces. Therefore, an edge collapse can be performed only if the faces it generates do not intersect unauthorized voxels.



(a) Original mesh (b) Decimated mesh (c) Error map



(d) Decimated mesh (e) Error map

Figure 1.17: Decimation of the wheel model [BF05]. (a) The original wheel model has about 160 000 faces. The wheel is approximated using the tolerance pairs $(0.2\%, 33°)$ (b) and $(0.5\%, 36°)$ (d), and the subsequent error is given as compared to the original model (c) and (e), respectively. The tolerance volume is given as a percentage of the diagonal of the original mesh bounding box.

However, global error bounds do not account for local properties, as a consequence, they might not preserve the structure of the mesh.

1.3.2 Planar Proxies

Salinas, Lafarge, and Alliez [SLA15] introduce a *structure-aware* approach for mesh decimation. The local Quadric Error Metric, described in Section 1.2.3, is extended to account for the global structure of the mesh characterized by planar proxies detected in a pre-processing step and structured by an adjacency graph. These proxies are taken as input of the algorithm alongside with the original mesh, as seen in Figure 1.18.

An hybrid error metric is devised by mitigating the local quadrics of QEM with the quadrics of proxies. Let $Prox(f)$ be the set of proxies containing a

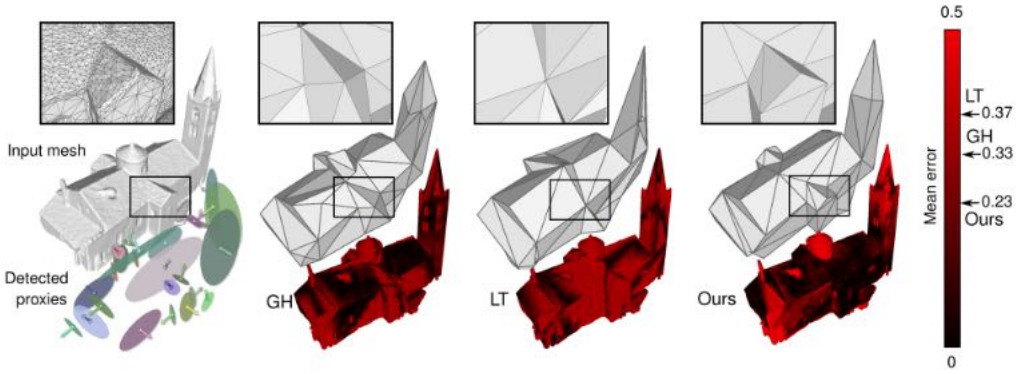


Figure 1.18: Structure-aware mesh decimation [SLA15]. The combination of a local error with a global error related to proxies better preserves the structure of the mesh than a simple local error.

face f and Q_φ the quadric of a proxy φ. The quadric of f becomes:

$$Q'_f = \begin{cases} Q_f & \text{if } Prox(f) = \varnothing, \\ (1-\lambda)Q_f + \lambda \sum\limits_{\varphi \in Prox(f)} Q_\varphi & \text{otherwise.} \end{cases} \tag{1.23}$$

where Q_f is the local quadric of f as defined in Equation 1.14 and λ is an abstraction parameter which sets the influence of the quadrics of proxies as compared to the local quadric. This quadric Q'_f, which also includes the quadrics of proxies, replaces Q_f in Equation 1.15 to compute the quadric of a vertex.

If the abstraction parameter λ is equal to zero, a classic quadric error is computed, without any consideration at the global scale of the mesh. In a similar fashion, if λ is equal to one, the error is solely based on the proxies, without any consideration of the local geometry. Hence, the error value of an edge collapse changes if at least one face in the one-ring of the edge has a proxy and the value of λ is not zero. The more the value of λ is important, the more influence the proxies have over the error and the subsequent position of the vertex resulting from an edge collapse.

Furthermore, the resulting vertex also inherits the union of proxies of the endpoints of the collapsed edge. Hence, some additional structure-preserving rules prevent collapses that would alter proxies or their adjacency graph.

This approach reduces the accumulation of local approximations by considering the structure of the mesh and thus tends to move the vertices towards the proxies during the decimation process, which contributes to keep the overall

aspect of the model and improve its resilience to noise.

1.3.3 Major Features

The distance error metrics commonly used in most decimation algorithms are very efficient to measure geometric error, but they often fail at detecting important shape features of a mesh such as regions with high curvature. Therefore, some approaches focus on preserving the major features of a mesh during the decimation process.

Kim, Kim, and Levin [KKL02] use a *discrete curvature norm* as error metric while decimating a mesh using the edge collapse operator. The sum of absolute principal curvatures of each endpoint of an edge is computed, by combining Gaussian curvature and mean curvature. The Gaussian curvature of a vertex is related to the angles between successive edges in its one-ring neighborhood, and the mean curvature is related to both the length of such an edge and its dihedral angle (i.e. the angle between the supporting planes of faces sharing the edge). The discrete curvature norm is checked before and after an edge collapse and the difference represents the error value associated to the collapse of this edge. The vertex resulting from such a collapse minimizes the change of discrete curvature norm. However, this error metric is restricted to curvature and does not account for other geometric properties.

Marinov and Kobbelt [MK05] use the face merge operator [KT96] along with an integral error metric [CAD04] to derive a subdivision control mesh whose structure is adjusted and aligned to the major geometric features. Faces are referred to as regions and the face merge operator merges two nearly coplanar regions by removing their common edges. Thus, the regions tend to grow during the decimation process until a maximum error bound or a target number of regions is reached, as illustrated by Figure 1.19. At the beginning of the algorithm, each face makes up a region on its own and the proxy of a region is characterized by a point on the face and the normal vector of the face. When two regions are merged, a new proxy is computed for the resulting region by using an area-weighted average of the points and normal vectors of the regions. Adjacent pairs of regions are iteratively merged in order to generate a coarser mesh. It should be noted that the algorithm must maintain an injectivity constraint throughout the decimation process in order for the mesh to stay consistent: the projection of a face onto a proxy must be injective, which guarantees on the one hand that there is no foldover as a face cannot be part of several regions, and on the other hand that there is no degeneration as a face is always part of a region. Nonetheless, as in the work of Kim, Kim, and Levin [KKL02], this approach is also restricted to curvature.

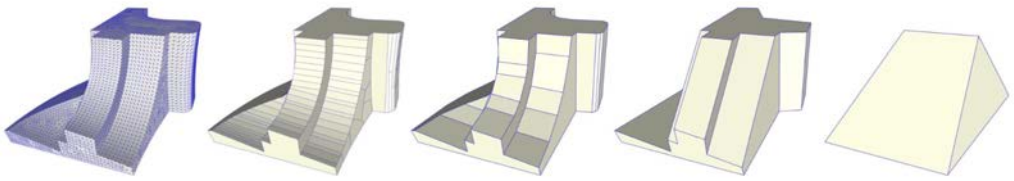


Figure 1.19: Decimation of the fan disk model [MK05]. From left to right: the original fan disk model having about 13 000 faces, and the fan disk approximated using 200, 50, 17 and 5 regions, respectively.

Vivodtzev, Bonneau, and Le Texier [VBL05] propose a decimation scheme for meshes with embedded polylines. These polylines are a subset of edges that characterize the features of the mesh. In the context of CAD models, if the materials of a mesh are available, the polylines may be extracted as interfaces in between materials defined on the surface. An edge collapse can be performed only if it preserves both the topology of the mesh and the topology of the embedded polylines. However, not all meshes have embedded polylines or attributes enabling smooth polylines extraction.

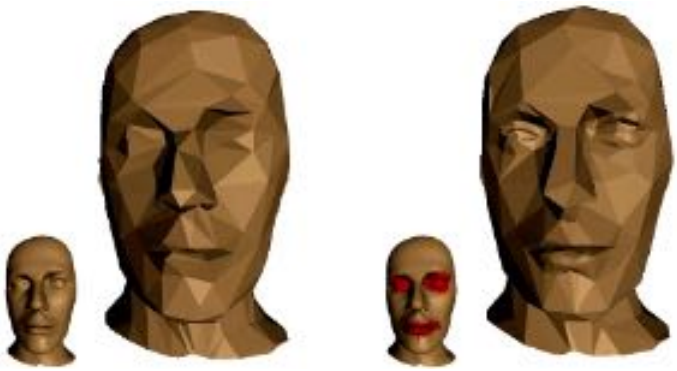
While efficient at preserving the high level structure of a mesh, these approaches often miss smaller details that might be semantically important, if they are not annotated.

1.3.4 User-Guided

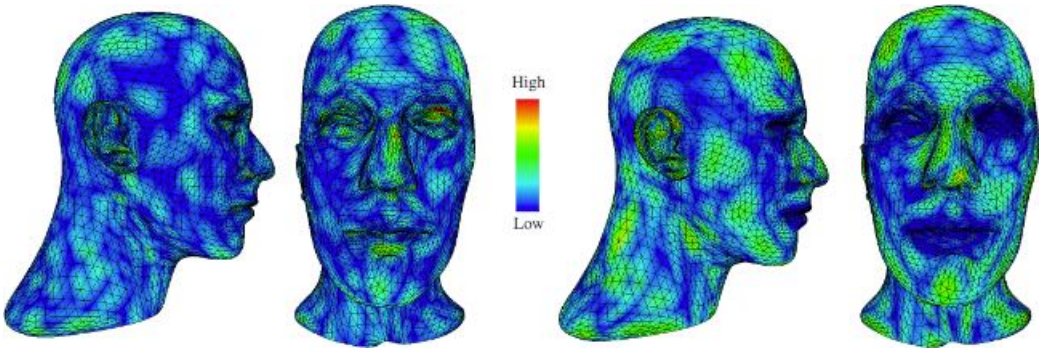
Automatic mesh decimation algorithms ignore the semantic meaning of models. For instance, some features with small size can be semantically important even if they have a low geometric error, but not be detected as major features. Thus, they might be discarded by algorithms relying on distance error metrics or feature sensitivity. Therefore, user-guided decimation enables to provide semantic information that will be used along with the geometry during the decimation process.

Kho and Garland [KG03] propose a user-guided decimation in order to preserve semantically important features of a mesh. The user drives the decimation by interactive control of an automatic decimation method consisting in iterative edge collapses sorted with the QEM. This interactive control is done using both an adaptive decimation and geometric constraints. The adaptive decimation consists in selecting some areas of importance which will be accordingly less decimated than other areas. For this purpose, the quadrics of vertices in such areas are assigned a high weight. Moreover, different types of

geometric constraints (contour, plane and point) can be imposed with additional constraint quadrics to preserve various features of a mesh. As a result, the error values of edges in important areas increase, delaying their collapse and changing the optimal position of their resulting vertices. Thus, heavily weighted vertices are more likely to preserve their position throughout the decimation process, as illustrated by Figure 1.20.



(a) Decimation



(b) Error distribution

Figure 1.20: Decimation and error distribution of the face model [KG03]. (a) Decimation using fully automatic QEM on the left and user-guided QEM on the right. The painted regions are significantly improved on the right. (b) Error distribution of approximation with QEM on the left and user-guided QEM on the right. The error distribution is less uniform on the right with less error for perceptually important areas as the eyes and the lips.

Likewise, Pojar and Schmalstieg [PS03] enable the user to interact with the same automatic decimation algorithm in order to specify areas of importance of a mesh. However, unlike the work of Kho and Garland [KG03] in which the user can weight quadrics of vertices and potentially add constraint quadrics to them, the user can directly change the cost of an edge collapse by weighting it. Another notable difference is the fact that the position of the vertex resulting

from an edge collapse remains unchanged, i.e. the optimal position is computed using only the QEM, regardless of the user weighting scheme. Hence, this approach does not change the edge collapse operations, but only reorders them.

Following user input, while Kho and Garland [KG03] make the most of the QEM as the user can only weight quadrics of vertices and not the cost of an edge collapse directly, Pojar and Schmalstieg [PS03] make use of the robust vertex placement scheme of QEM as the collapses are not modified.

Ho et al. [Ho+06] introduce a generic user-assisted approach which can be used with any error metric. Based on the observation that the weights applied to edge collapses, either directly or indirectly, in order to reorder them, have no relation to the error value and are empirical, the weights are rather applied to the rank of an edge collapse in the priority queue. Thus, this method overcomes the dependency over the error metric.

The main drawback of these non-automatic algorithms is their full dependency over user input. Another limitation, shared by all the algorithms mentioned so far in this state of the art, is that none of these methods are designed for multiple meshes decimation, as they process a single mesh and thus can not deal with relations between different meshes.

1.4 Scene Decimation

A scene consists of several models that usually have relations with each others, especially in the case of CAD scenes where these relations encode the functional meaning of the scene. Therefore, decimating each model separately is not a suitable approach to decimate a scene as the relations between models might not be preserved. For instance, Figure 3(b) illustrates a significant loss of contact between a hex head bolt an its contact surface.

In most CAD representations, geometrical primitives and their arrangement define high level information that can be used to support the simplification process. Thereby, Kwon et al. [Kwo+15] propose a feature-based simplification of multiple models, based on geometric information imported from CAD data. By the same means, Erikson, Manocha, and Baxter III [EMB01] build hierarchical levels of detail by grouping nodes with regard to 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, which leads to seek for a geometric-only multiple mesh decimation approach.

The joint decimation of multiple meshes has also been studied, though it has received much less attention than single mesh decimation.

Gumhold, Borodin, and Klein [GBK03] introduce an intersection-free approach, which focuses on avoiding collisions between close-by meshes while iteratively performing edge collapses sorted with the QEM. Each time a collision arises from the collapse of an edge into the vertex minimizing its QEM, a new intersection-free position is computed for the resulting vertex. Hence, the cost of such an edge collapse also changes and the edge is re-inserted into the priority queue of collapses with its 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, for instance by generating holes, cracks or removing small geometrical features.

González et al. [Gon+09] propose a user-assisted method to decimate meshes while preserving their boundaries. Like many other approaches, the decimation process consists in iteratively performing edge collapses sorted with the QEM. To preserve the boundaries of each mesh, vertices nearby other meshes in the scene are tagged as boundary and preserved by performing halfedge collapses as opposed to edge collapses when no endpoint is a boundary vertex. The limitations of this approach are twofold. Firstly, the edges having both endpoints tagged as boundary are considered as regular edges and thus, standard edge collapse is performed, which prevents from penalizing shape variations inside boundary areas. Secondly, the approximation error in boundary areas is checked only along the edges, and not on the faces.

1.5 Conclusion

The incremental decimation algorithm combining the edge collapse operator with the Quadric Error Metric is the most widely used mesh decimation approach. Indeed, the edge collapse is the most convenient topological operator as it allows to put the resulting vertex at a position minimizing the changes in the mesh. It only removes one vertex per decimation step and does not require any re-triangulation. Likewise, the QEM is the most convenient error metric to sort the edge collapses into the priority queue as it computes both the error of the edge and the optimal position for its collapse, all in a computationally efficient way. Furthermore, filters can be added to the QEM. Therefore, numerous approaches extending standard mesh decimation are based on this approach.

Thereby, after introducing a priority queue gathering multiple meshes in Chapter 2, we extend the QEM in Chapter 3 to account for proximity relations between meshes, turning it into a proximity-aware error metric.

Chapter 2

Multiple Meshes Decimation

The joint decimation of multiple meshes raises several challenges that do not arise for the decimation of a single mesh. While all the available resources are allocated to a mesh when it is decimated alone, it is not obvious how to split them between several meshes during their joint decimation. Furthermore, the joint decimation of multiple meshes often means dealing with more complex data, as a scene can consist of many meshes as opposed to a single mesh.

This chapter first reviews possible *decimation schemes* for multiple meshes (Section 2.1). We then demonstrate how we combine them to devise a *global priority queue* (Section 2.2). Finally, we present an *algorithm* for the joint decimation of multiple meshes (Section 2.3), along with a *complexity analysis* of its global priority queue (Section 2.4).

2.1 Decimation Schemes

Mesh decimation consists in decreasing the data complexity by reducing the number of faces. A convenient stopping criterion for mesh decimation is a certain level of detail, i.e. a percentage of the initial number of faces, which is equivalent to a target number of faces, as it is easier to handle for a user than an error bound. In the case of a scene with multiple meshes, there are several possible decimation schemes depending both on the distribution of the face budget between the different meshes and the management of topological operations.

This section reviews two intuitive decimation schemes for multiple meshes, *decimating each mesh separately* (Section 2.1.1) and *decimating all meshes together* (Section 2.1.2), which we then compare to each other and discuss (Section 2.1.3).

2.1.1 Decimating Each Mesh Separately

Decimating each mesh separately implies splitting the face budget between meshes. The most intuitive and easiest solution is to take the desired level of detail for the scene and apply it to each mesh, thus performing a comparable number of topological operations on each mesh relatively to its initial number of faces. Thus, each mesh is decimated independently with its own target number of faces derived from the target level of detail, without any consideration for the other meshes. Therefore, a priority queue of operations is built for each mesh and standard incremental decimation is performed, until the target number of faces is reached for the mesh or it is no longer possible to decimate it, i.e. there are no more consistent operations left in the priority queue of the mesh.

While the complexity of such a decimation scheme is equivalent to that of standard incremental decimation as each mesh is decimated separately, the decimation is heavily dependant on the initial tessellation of each mesh. Indeed, the distribution of the face budget between meshes is not optimized as the error values of the different operations are not considered. As a consequence, some meshes might be overdecimated while others are barely decimated.

2.1.2 Decimating All Meshes Together

Decimating all meshes together means considering the scene as a single mesh and performing standard mesh decimation on this mesh. Thus, all topological operations of the scene are sorted into a unique priority queue and the operation with the smallest error value, regardless its mesh, is performed iteratively, until the target level of detail is reached or no more consistent operations are left in the priority queue of the scene.

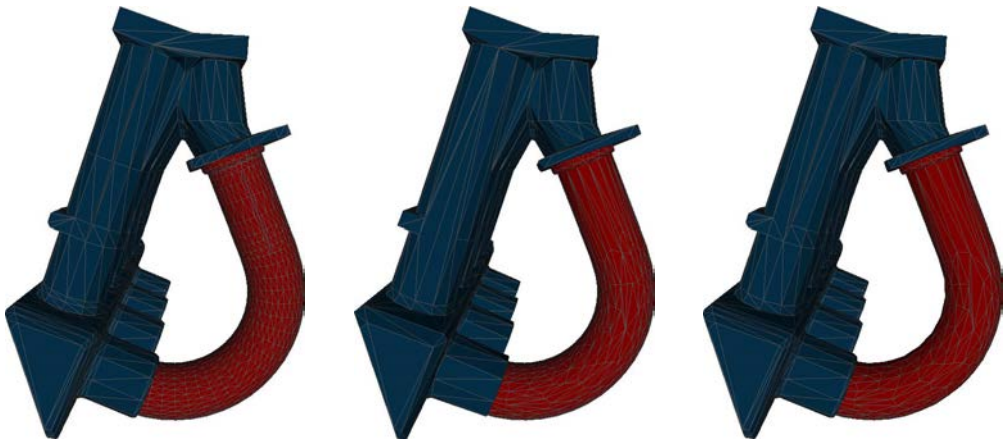
While the face budget is efficiently distributed between meshes as their operations are sorted all together based on the error value, the complexity of such a decimation scheme is influenced by both the number and complexity of meshes in the scene as a unique priority queue is used for the whole scene. Therefore, the size of this priority queue would be very large in case of a complex scene, which would slow down operation insertion and removal throughout the decimation process.

2.1.3 Combining Both Decimation Schemes

We seek for the joint decimation of multiple meshes to efficiently allocate faces between meshes, and to do so with the lowest possible complexity.

A decimation scheme decimating each mesh separately scales well to very large scenes with numerous meshes as each mesh has its own priority queue. Conversely, a decimation scheme decimating all meshes together overcomes the dependence over the tessellation of the scene, as the face budget is split between meshes in the same way as if they were merged into a single mesh.

As seen in Figure 2.1, the scene in Figure 2.1(a) is better preserved when its meshes are decimated together in Figure 2.1(c) than when they are decimated separately in Figure 2.1(b), as the rounded corners of the base are retained in the former and turned into sharp corners in the latter, while the tube is equally preserved in both. Indeed, the base has more faces than the tube, but its size is much larger and it is significantly less tessellated, though it has more shape variations. Hence, the tube should be more decimated than the base, which is the case when decimating them together as the distribution of faces is efficient.



(a) (b) (c)

Figure 2.1: Joint decimation of two meshes. (a) The original scene has 5508 faces, with 2252 faces for the tube (red mesh) and 2956 faces for the base (blue mesh). (b) The meshes are decimated separately: the tube has 1126 faces and the base 1478 faces. (c) The meshes are decimated together: the tube has 754 faces and the base 2000 faces. Both scenes are decimated down to 50%.

Therefore, we propose the joint decimation of multiple meshes by combining the advantages of both decimation schemes, namely decimating all meshes together in order to efficiently distribute the face budget between meshes, while using a global priority queue derived from the priority queues of each mesh in the scene to overcome a possibly very large unique priority queue, in order to limit the complexity of the decimation.

2.2 Global Priority Queue

We build a global priority queue driving the simultaneous decimation of multiple meshes from the priority queues of each mesh. The top operation of each priority queue is inserted with regard to its error value into the global priority queue. Therefore this global priority queue interleaves the different priority queues, which is strictly equivalent to a unique priority queue sorting all operations of all meshes. As there is no more than a single operation per mesh in the global priority queue at each step of the incremental decimation, the maximum size of this global priority queue is the number of meshes in the scene.

In order to process N meshes at once, a priority queue P_i is computed for each mesh. As illustrated by Figure 2.2, the top operation of each mesh is sorted into a global priority queue P_{GLOB} of size N. Let M_k be the mesh associated to an operation originally belonging to a priority queue P_k. At each decimation step, the top operation of P_{GLOB} is popped out and performed on its associated mesh M_k. It is also popped out of its own priority queue P_k which is updated following the operation. If P_k is not empty, its new top operation is sorted into P_{GLOB}.

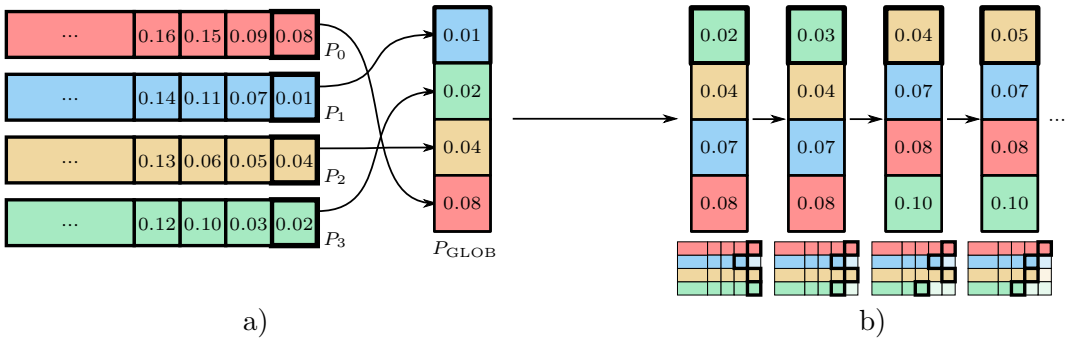


Figure 2.2: Simultaneous decimation of multiple meshes using interleaved priority queues [Gha+19]. a) State of the priority queues before the start of the decimation. The top operation of each priority queue P_i is copied and sorted by increasing error into the global priority queue P_{GLOB}. b) When the operation on top of P_{GLOB} is performed, its next operation in the priority queue of the mesh it belongs to is added and sorted into P_{GLOB}.

The benefits of the global priority queue are twofold. First, interleaving the operations for all meshes yields to an adaptive decimation of the meshes in the scene, as it allows to balance the decimation rate between meshes according to the error values of their operations. Second, the use of an interleaved priority

queue avoids to maintain and update a unique and possibly very large priority queue, while having a very small memory overhead. The cost to update the priority queue of a mesh when decimating a scene is the same as when decimating the mesh on its own, plus the cost of the global priority queue update.

2.3 Algorithm

Algorithm 2 Incremental decimation of N meshes

Input: original meshes M_i, stopping criterion *stop_crit*
Output: decimated meshes M_i

$P_{i,i\in[0,N[} \leftarrow$ empty priority queue of M_i
$P_{\text{GLOB}} \leftarrow$ empty global priority queue
for each mesh M_i **do**
  for each element e_i **do**
    if op_i is a possible operation on e_i **then**
      $err_i \leftarrow$ compute error associated to op_i
      Insert (op_i,err_i) in P_i
    end if
  end for
  $op_i \leftarrow$ top operation of P_i
  Insert op_i in P_{GLOB}
end for
while P_{GLOB} not empty and *stop_crit* not reached **do**
  $op_k \leftarrow$ top operation of P_{GLOB}
  $M_k \leftarrow$ mesh associated to op_k
  Remove op_k from P_{GLOB} and P_k
  Apply op_k on M_k
  for each neighbor operation $op_{k,j}$ **do**
    Remove $op_{k,j}$ from P_k
    $err_{k,j} \leftarrow$ compute error associated to $op_{k,j}$
    Insert ($op_{k,j}$,$err_{k,j}$) in P_k
  end for
  if P_k not empty **then**
    $op_k \leftarrow$ top operation of P_k
    Insert op_k in P_{GLOB}
  end if
end while

Algorithm 2 extends Algorithm 1, which describes incremental mesh decimation for a single mesh, to include the global priority queue managing the

topological operations on different meshes.

This algorithm interleaves the execution of Algorithm 1 for each mesh. At first, it computes the priority queue for each mesh, and the top operation of each priority queue is inserted into the global priority queue. Then, the top operation of the global priority queue is performed on the associated mesh by reactivating its algorithm during an iteration step, as only operations on the same mesh can be affected and thus need to be updated. Following such an iteration step, the new operation on top of the priority queue of the associated mesh, if existing, is inserted into the global priority queue. Again, the operation on top of the global priority queue is performed and the decimation process continues this way until no more operations are left or the stopping criterion is reached.

As with single-mesh decimation, the stopping criterion can be defined as an error bound, a number of topological operations or a target number of faces, defined as desired, globally or per mesh.

2.4 Complexity Analysis

When considering the joint decimation of N meshes having about E elements each, in terms of complexity, the cost of updating the operation having the minimal error in the global priority queue is $\mathcal{O}(\log(N))$ and the cost of re-sorting (deleting and reinserting) the k affected operations in the corresponding mesh priority queue is $\mathcal{O}(2k\log(E))$. Hence the complexity of an operation managed by the global priority queue is $\mathcal{O}(\log(N) + 2k\log(E))$. The complexity of the same operation managed by a unique priority queue gathering all elements of all meshes would be $\mathcal{O}(2k\log(NE))$.

As stated in Section 1.5, as the QEM is extended in this thesis, its associated topological operator, the edge collapse, is used. In order to attest the advantage of our algorithm with a global priority queue over an algorithm using a unique priority queue, we demonstrate that the number k of operations to update following an edge collapse is substantial.

Figure 2.3 displays the entities of a mesh affected by an edge collapse. While the collapse of an edge directly impacts the error of its adjacent edges as one or both of their endpoints will become the new vertex resulting from the collapse, it also impacts more distant edges through quadrics. The error of an edge is computed using the quadrics of its endpoints, and the quadric of a vertex is computed by combining the quadrics of its adjacent faces. Hence, when one vertex of a face is displaced following a collapse, its quadric changes as its plane changes. Therefore, edges having one or both endpoints whose quadric has changed as an aftermath of an edge collapse, are also impacted. As a result, all edges influenced by a collapse are removed from their priority

queue, and if not degenerated, their error is recomputed, and they are inserted again into their priority queue.

As seen in Figure 2.3, for a valence-6 mesh, an edge collapse affects 10 edges directly, 8 edges through both endpoints, and 22 edges through one endpoint, for a total of 40 edges. Hence the number k of operations to update at each decimation step is usually large. As a consequence, the approach using a global priority queue scales better when the number of meshes N is large than the approach using a unique priority queue.

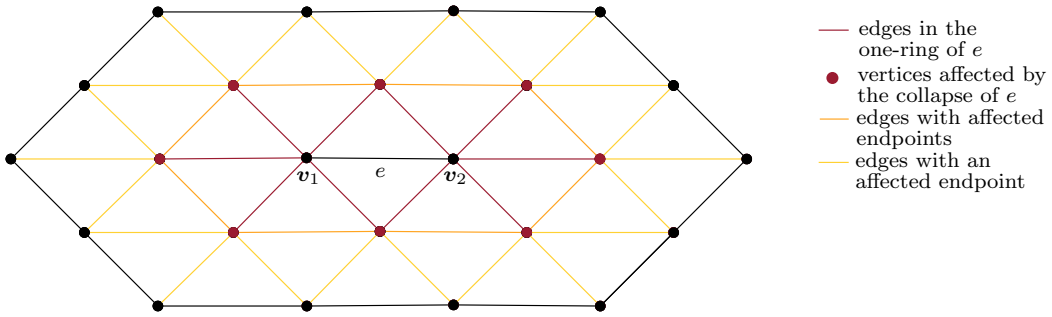


Figure 2.3: Impact of an edge collapse on a mesh. Edges in the neighborhood of edge e will be affected to varying degrees by its collapse. By order of influence: red edges are directly adjacent to e so one of their endpoints will be displaced, orange edges have both endpoints whose quadrics will be changed as a consequence of the plane of some of their adjacent faces changing, and yellow edges have one such endpoint.

2.5 Conclusion

The decimation of a scene is carried out by the joint decimation of its meshes. For this purpose, we devise a global priority queue of operations common to all meshes in order to interleave the decimation of the meshes in a scene. This global priority queue is derived and updated from the priority queue of each mesh for complexity purposes. Therefore, such a decimation scheme displays a suitable decimation rate for each mesh in the scene, while having just a slight memory overhead as compared to the standard decimation of a single mesh.

Nonetheless, the joint decimation of multiple meshes does not to account for their proximity relations. Hence, we extend the QEM into a proximity-aware error metric in Chapter 3.

Chapter 3

Proximity-Aware Decimation

Figure 3.1: `Tube` scene. This scene is composed of two interlocking models. The tube (red model) fits into the base (blue model).

The QEM computes the error associated to an edge collapse by computing a quadric for the edge and minimizing a point-to-plane distance using this quadric. The quadric of an edge aggregates information about the planes of its adjacent faces. In other words, it stores the local information restricted to the neighborhood of the edge in the topological sense. Hence, in the case of multiple meshes decimation, the information stored in the quadrics is incomplete, as faces close to or in contact with an edge, but belonging to another mesh, are not considered. Meshes in a scene usually have proximity relations representing semantic information. In that case, such an information is as important as the local information for a mesh in the scene context. For instance, the tube seen in Figure 3.1 is defined by its relation with the base and would loose its functionality in the scene would it no longer fit into the base following decimation. Therefore, the decimation of a scene consisting

of multiple meshes should also account for the non-local properties of each mesh, i.e. its close faces on other meshes.

This chapter first introduces a *proximity definition* (Section 3.1) and reviews different possible *decimation strategies* accordingly (Section 3.2). We then devise a *proximity quadric* to store non-local information (Section 3.3) and use it to compute a *proximity error* (Section 3.4). Finally, we propose a *proximity-aware error metric* by combining the QEM with the proximity error (Section 3.5).

3.1 Proximity Definition

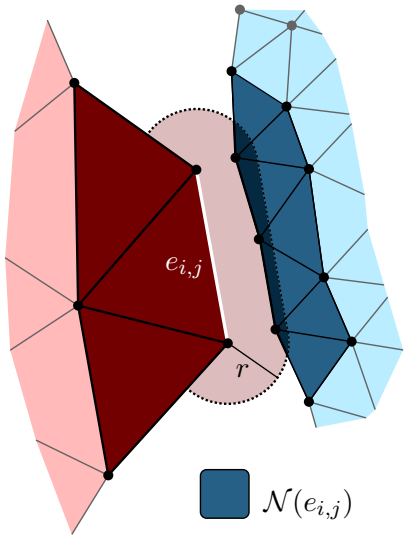


Figure 3.2: Neighborhood of an edge [Gha+18]. The local neighborhood of the edge $e_{i,j}$ is its set of adjacent faces belonging to the same mesh and its non-local neighborhood is the set $\mathcal{N}(e_{i,j})$ of faces within the proximity threshold r and belonging to other meshes.

We denote a scene as a set of N triangular meshes $M_i, i \in [0\,..\,N[$, with $\boldsymbol{v}_{i,j}$, $e_{i,j}$ and $f_{i,j}$ the j^{th} vertex, edge and face of a mesh M_i, respectively.

We define an edge to be in proximity with a face of a neighbor mesh if the edge-to-face distance is lower than a scene-specific proximity threshold r, which is assumed to be given in this chapter. This threshold can be determined analytically from the scene, which is the focus of Chapter 4.

Figure 3.2 illustrates the neighborhood of an edge $e_{i,j}$. Its local neighborhood consists of its set of adjacent faces on its own mesh M_i and its non-local

neighborhood consists of the set $\mathcal{N}(e_{i,j})$ of faces from close-by meshes of M_i that fall in its Euclidean neighborhood of radius the proximity threshold r.

We propose to also account for the non-local neighborhood of an edge belonging to a given mesh when computing the error associated to its collapse, i.e. faces from close-by meshes fitting the proximity definition. Edges having such a non-local neighborhood within the proximity threshold are said to have proximities and when considered together, they form proximity areas in the scene.

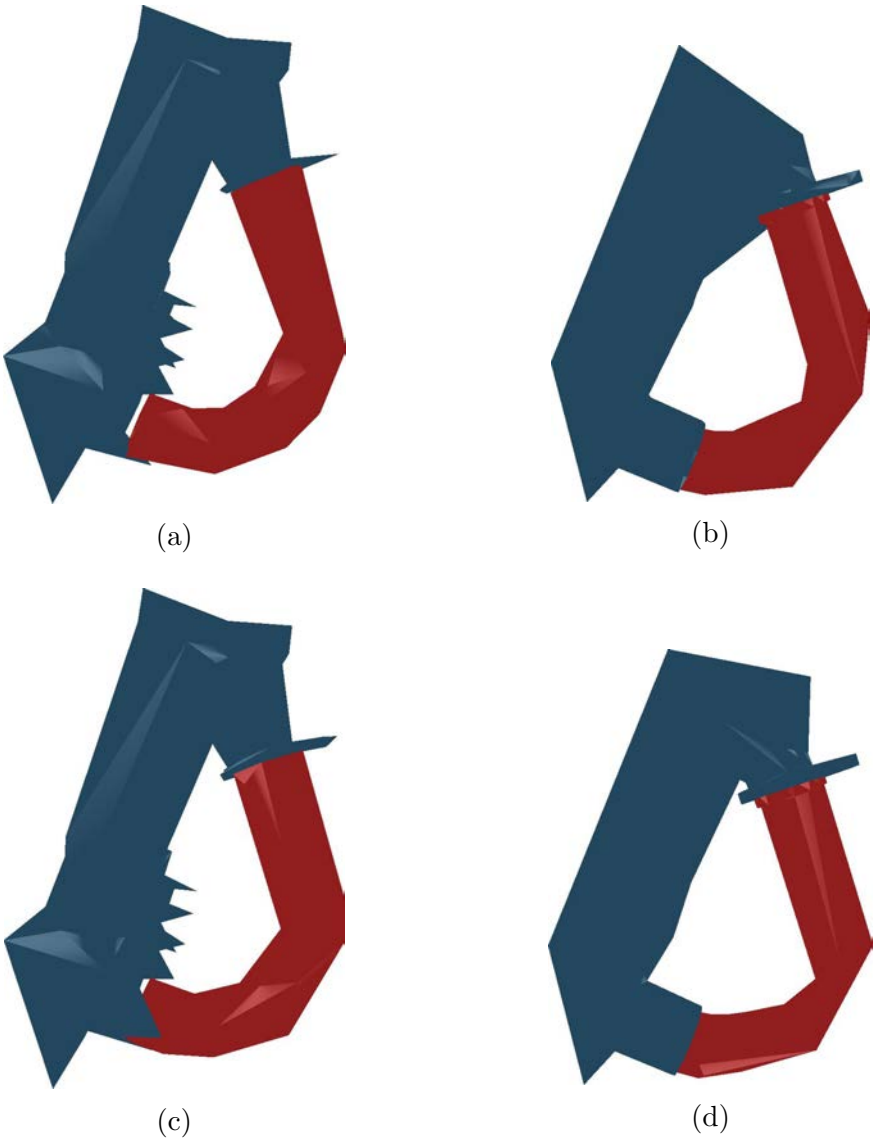
3.2 Decimation Strategies

This section reviews several approaches that may be considered to account for non-local properties of meshes during the decimation of a scene through successive edge collapses, namely *blocking collapses* (Section 3.2.1), *penalizing errors* (Section 3.2.2), *modifying quadrics* (Section 3.2.3), or *reordering collapses* (Section 3.2.4).

3.2.1 Blocking Collapses

A first approach to preserve proximity relations in a scene during its decimation is to block the collapse of edges having proximity with faces of other meshes. However, while this approach would perfectly keep the proximity parts of the mesh, in the case of a target number of faces as stopping criterion, it would lead to an undesired over-decimation in other parts of the mesh, outside proximity areas, especially in the case of massive decimation. This problem is illustrated by Figure 3.3(b) with both models having an altered shape even though their relation is perfectly kept. Similarly, in the case of an error threshold as stopping criterion, as the non-proximity areas would not be over-decimated, the proximity areas not being decimated at all means no massive decimation could be achieved.

Therefore, blocking all collapses in proximity areas is not a suitable solution, as some collapses are not problematic. For instance, the collapse of an edge located in a planar area does not introduce any changes in the geometry of the mesh. Moreover, the collapse of some edges in proximity areas may only introduce minor changes in the scene while the collapse of some edges outside proximity areas might introduce major changes in the associated mesh, making the corresponding model unrecognizable, e.g. breaking its structure.



(a) (b)

(c) (d)

Figure 3.3: Decimation of the `Tube` scene. (a) QEM. (b) QEM with blocking of collapses in proximity areas. (c) QEM with proximity weighted by a too low penalty separates the tube and the base like standard QEM. (d) QEM with proximity weighted by a too high penalty prevents the decimation in proximity areas but over-decimates non-proximity areas, as QEM with blocking of collapses in proximity areas.

3.2.2 Penalizing Errors

A second approach to preserve proximity relations in a scene is to delay the collapse of edges in proximity areas by increasing the associated error, using a fixed value. This approach is similar to the user-guided decimation method of Pojar and Schmalstieg [PS03] seen in Section 1.3.4, but without the user input to select and weight collapses, as only collapses in proximity areas are weighted and this weight is a constant.

As the QEM is not aware of proximity relations, we have implemented a simple weighting scheme. The new error Δ at the collapsing vertex $\boldsymbol{v}_{e_{i,j}}$ is obtained by multiplying the QEM error Δ_{qem} by a fixed value w for an edge $e_{i,j}$ in a proximity area:

$$\Delta(\boldsymbol{v}_{e_{i,j}}) = w\ \Delta_{qem}(\boldsymbol{v}_{e_{i,j}}) \tag{3.1}$$

However, there is no such unique optimal weight for a scene, and finding a good weight is a non-trivial problem. A too low weight would barely delay collapses in proximity areas, and as a result lead to a similar decimation as with the QEM. Likewise, a too high weight would drop collapses in proximity areas down to the bottom of their respective priority queue, and thus proceed practically as blocking the collapses.

Thus, we have chosen two extreme values: $w = \{2; 500\}$. The former preserves the order of magnitude of the collapse errors as seen in Figure 3.3(c), while the latter is very likely to block collapses in proximity areas as seen in Figure 3.3(d).

Therefore, weighting the errors associated to edge collapses in proximity areas is not a suitable solution either as, given the weight, it might not preserve the proximity between meshes as with the QEM in such a context or have the same drawbacks as with blocking collapses.

3.2.3 Modifying Quadrics

In its initial formulation, the QEM represents the point-to-plane distance for all the faces surrounding an edge, as the quadric of an edge is the normalized sum of the quadrics of its adjacent faces. The quadric of an edge in proximity areas can be modified to also incorporate the quadrics of faces in proximity from close-by meshes. This approach is in the same spirit as the virtual edge of Garland and Heckbert [GH97] seen in Section 1.2.2, which brings together quadrics of non-topologically but geometrically close faces through two unconnected vertices considered as an edge. It also displays similarities with the structure-aware decimation of Salinas, Lafarge, and Alliez [SLA15] seen in Section 1.3.2, which balances the quadric of a face with the quadrics of its proxies, as faces close to an edge but from other meshes

can be assimilated to proxies. However, the minimization of such a modified quadric would tend to move the optimal collapsing vertex of the associated edge towards the surrounding meshes.

Therefore, incorporating quadrics from faces in proximity with an edge but from close-by meshes into the quadric of this edge is not a suitable solution as close meshes would move towards each other and thus alter the proximity relations between meshes in the scene.

3.2.4 Reordering Collapses

Ideally, the faces from surrounding meshes and in proximity with an edge should contribute to delay its collapse, i.e. modifying its error value, but not to the collapse itself, i.e. modifying the resulting vertex. For this purpose, the collapses can be reordered as compared to their sorting with the QEM, by modifying their error with regard to their proximity relations, while still minimizing the QEM.

Therefore, we propose a new weighting strategy to delay the edge collapse operations in proximity areas. We emphasize that we keep the collapsing vertex given by the QEM since the aim is not to change the collapses but rather to reorder them considering proximity. To do so, we compute a proximity quadric from the surrounding meshes and evaluate it at the location of the collapsing vertex, i.e. a proximity error. We incorporate the proximity error along with the QEM into a proximity-aware error metric, in order to penalize collapses in proximity areas. This proximity-aware error metric is used to sort collapses in their respective priority queue and subsequently in the global priority queue of the scene throughout the decimation process.

3.3 Proximity Quadric

The local quadric of an edge represents the influence of the faces surrounding it on its associated mesh, through the normalized sum of their quadrics. Similarly, we compute a proximity quadric to represent the influence of faces in proximity with the edge, but belonging to other meshes in the scene.

While the local quadric is computed from the current mesh, the proximity quadric is computed from the original surrounding meshes, in order to avoid the accumulation of decimation errors into it. Furthermore, the quadrics of the faces from close-by meshes within the proximity threshold from the edge are weighted by a function of their distance to the edge, so that faces closer to the edge have more influence in the proximity quadric as compared to faces

further, but still close to the edge.

As seen in Figure 3.2, $\mathcal{N}(e_{i,j})$ is the set of faces from the non-local and original neighborhood of an edge $e_{i,j}$, i.e. faces on other and non-decimated meshes. The proximity quadric $\hat{Q}_{e_{i,j}}$ of $e_{i,j}$ combines the quadrics of such faces as follows:

$$\hat{Q}_{e_{i,j}} = \frac{1}{\mathbf{card}(\mathcal{N}(e_{i,j}))} \sum_{f_{k,l} \in \mathcal{N}(e_{i,j})} w(e_{i,j}, f_{k,l})\, Q_{f_{k,l}} \tag{3.2}$$

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

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

with $d(e_{i,j}, f_{k,l})$ the distance of the edge $e_{i,j}$ to the face $f_{k,l}$. In order to give more importance to the closest faces to the edge, a smooth polynomial kernel ϕ, drawn in Figure 3.4, is used:

$$\phi(x) = \begin{cases} (1 - (\frac{x}{r})^3)^2 & \text{if } x \leq r, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

It should be noted that in Equation 3.2, the proximity quadric $\hat{Q}_{e_{i,j}}$ is normalized using the number of faces in $\mathcal{N}(e_{i,j})$ rather than the sum of the respective weight of these faces. Therefore, the influence of a face surrounding an edge over the proximity quadric is solely determined by its distance to the edge, regardless of how many faces closer or further to the edge being also part of this non-local neighborhood.

Figure 3.5 analyzes the influence of proximity relations in a scene. The edges having proximities with the other mesh according to the definition of proximity in Section 3.1 are enhanced in Figure 3.5(a). Furthermore, the faces in the neighborhood of such edges on the other mesh are displayed in Figure 3.5(b) along with their corresponding weight. Obviously, most of these faces are in the neighborhood of more than one edge, hence the color of a face corresponds to the magnitude of its maximum weight. Moreover, there is a good symmetry between edges having proximities and faces in the neighborhood of such edges, as the faces formed by edges having proximities in a mesh are themselves in the neighborhood of the edges forming their surrounding faces on other meshes.

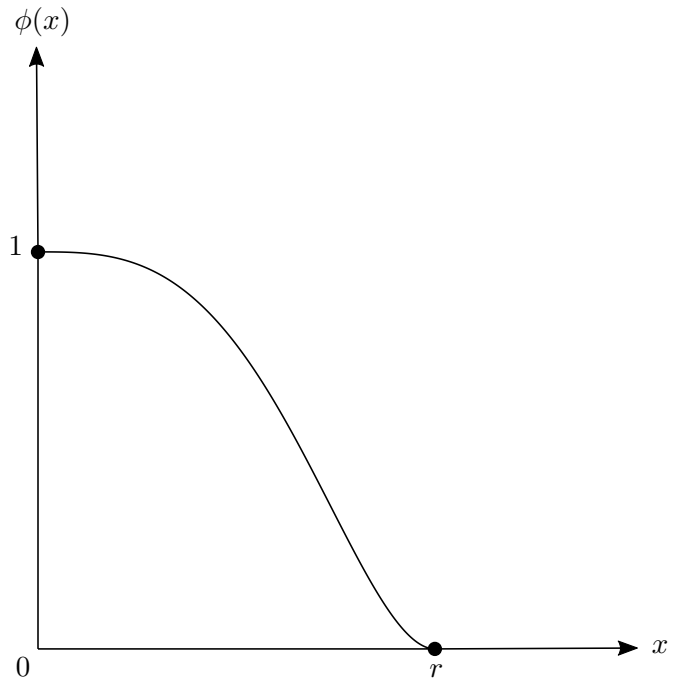
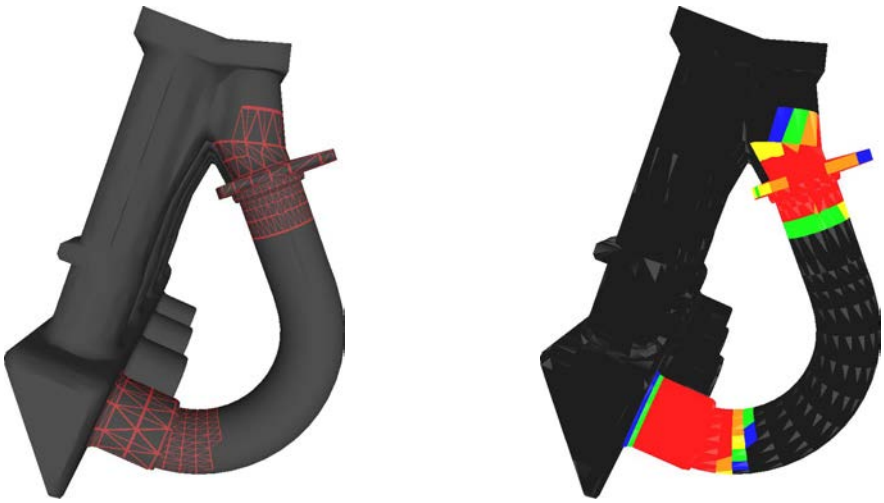


Figure 3.4: Smooth polynomial kernel. The weight of a face in the non-local neighborhood of an edge in the proximity quadric is function of the edge-to-face distance. Thus, the maximum weight is 1 in the case of a contact and decreases down to 0 when the proximity threshold r is reached.



(a) (b)

Figure 3.5: Proximity influence in the `Tube` scene. (a) The edges highlighted in red belong to proximity areas. (b) The maximum weight of each face with regard to the other mesh is displayed in proximity areas, by decreasing influence: the weight of red, orange, yellow, green and blue faces is equal to or below 1, 0.8, 0.6, 0.4 and 0.2, respectively.

3.4 Proximity Error

Just as the quadric of an edge is evaluated at the collapsing vertex minimizing it, defining the local error on the mesh in Equation 1.17, we likewise evaluate the proximity quadric at this collapsing vertex, defining the proximity error with regard to the surrounding meshes.

Let $\boldsymbol{v}_{e_{i,j}}$ be the collapsing vertex of an edge $e_{i,j}$, obtained by minimizing the quadric error. The proximity error is proportional to the deviation of the collapsing vertex from the non-local neighborhood of the edge:

$$\Delta_{prox}(\boldsymbol{v}_{e_{i,j}}) = \boldsymbol{v}_{e_{i,j}}^T \, \hat{Q}_{e_{i,j}} \, \boldsymbol{v}_{e_{i,j}} \tag{3.4}$$

This proximity error represents the impact of an edge collapse on its non-local neighborhood as for the quadric error with the local neighborhood. Hence, the proximity error can be viewed as a global error and used as a penalty function to increase the local error, in order to delay accordingly the collapse of the associated edge.

3.5 Proximity-Aware Error Metric

We define a proximity-aware error metric by combining the proximity error with the quadric error.

An intuitive solution is to add both errors, but this strategy would prevent the decimation of flat proximity areas, i.e. edges with a very low geometric error and a high proximity error. As a consequence, such an error metric is not a suitable solution as the decimation of flat areas does not degrade the geometry of a mesh and should thus occur early in the decimation process instead of being delayed at the expense of possibly structure-changing but proximity-free collapses, i.e. flat areas should still be associated with a low error, regardless any proximity relation.

Therefore, as a simple addition is not suitable, we propose to use the proximity error as a penalty factor modulating the quadric error. We recall that the collapsing vertex of an edge is computed exactly as in Section 1.2.3, by minimizing the standard QEM, i.e. regardless of proximity. We compute the proximity-aware error at the collapsing vertex $\boldsymbol{v}_{e_{i,j}}$ of an edge $e_{i,j}$ by combining its proximity error $\Delta_{prox}(\boldsymbol{v}_{e_{i,j}})$ with its quadric error $\Delta_{qem}(\boldsymbol{v}_{e_{i,j}})$ as follows:

$$\Delta(\boldsymbol{v}_{e_{i,j}}) = \Delta_{qem}(\boldsymbol{v}_{e_{i,j}}) \, (1 + \alpha \, \Delta_{prox}(\boldsymbol{v}_{e_{i,j}})) \tag{3.5}$$

where α scales the proximity error so that penalized edges are adequately sorted in their respective priority queue. Without this scale factor, the proximity error might be negligible or too important in the addition in which it takes parts. This addition ensures that the proximity-aware error is higher than the quadric error, as the aim is to increase the quadric error to delay edge collapses in proximity areas.

From Equation 3.5, the scale factor α is defined as follows:

$$\alpha = \frac{1}{\Delta_{prox}} \left(\frac{\Delta}{\Delta_{qem}} - 1 \right) \tag{3.6}$$

We estimated α so that the proximity-aware error Δ is in the same order of magnitude as a large error when the quadric error Δ_{qem} is relatively low and the proximity error Δ_{prox} is maximal.

As the aim of the proximity-aware error is to increase the quadric error of edges in proximity areas in order to delay their collapse, its value should be higher than the quadric error of most edges so that it drops down low in its corresponding priority queue. Thus Δ is approximated by a large error err_{high} set as the error at 90% of the QEM error histogram.

The proximity-aware error aims at increasing relatively low quadric errors, as a relatively high quadric error means that the associated edge already has a low priority in its queue of collapses. Thus, Δ_{qem} is approximated by a relatively low error $err_{\frac{1}{4}}$ set as the error at the first quartile of the QEM error histogram.

A high value for the proximity-aware error means a high value for the proximity error, since the proximity error modulates the quadric error. Accordingly, as the proximity error is the square value of a distance bounded by the proximity threshold r, Δ_{prox} is set at its maximum value r^2.

By applying these approximations to Equation 3.6, the scale factor α is estimated as follows:

$$\alpha = \frac{1}{r^2} \left(\frac{err_{high}}{err_{\frac{1}{4}}} - 1 \right) \tag{3.7}$$

We have first chosen the values of err_{high} and $err_{\frac{1}{4}}$ intuitively to respectively represent a high error and a relatively low error. It is meant to produce a penalty that is neither too high nor too low. A high penalty would prevent the decimation of edges even with a moderate proximity error, and a low penalty would have an insignificant impact on the simplified meshes, as compared to the standard QEM, even with a significant proximity error. We have then validated these values experimentally.

The proximity-aware error metric is used to compute the error associated to the collapse of each edge in the scene. It increases the quadric error in proximity areas and keeps the quadric error elsewhere. Therefore, this error metric is used for the joint decimation of multiple meshes modeling a scene. Algorithm 3 details the proximity-aware error computation for an edge. It takes as input the edge, its current mesh and the other meshes in their original state to avoid introducing the error accumulation related to decimation into the proximity computations. It also takes as input the specific parameters of the metric, namely the proximity threshold r and the scale factor α. This algorithm is used in Algorithm 2 to compute the error associated to an edge collapse while considering both its local and non-local neighborhood, thus preserving both the overall shape of meshes and the proximity relations between them.

Algorithm 3 Proximity-aware error computation for an edge

Input: edge $e_{i,j}$, current mesh M_i, original meshes $M_{k,k\neq i}$, proximity threshold r, scale factor α
Output: proximity-aware error err of $e_{i,j}$

$Q_{e_{i,j}} \leftarrow$ local quadric of $e_{i,j}$
$err_{qem} \leftarrow$ compute quadric error of $e_{i,j}$ using $Q_{e_{i,j}}$
$err \leftarrow err_{qem}$
$Q_{prox} \leftarrow$ null proximity quadric of $e_{i,j}$
$nb_{prox} \leftarrow 0$
for each mesh $M_{k,k\neq i}$ **do**
  for each face $f_{k,l}$ **do**
    $x \leftarrow$ compute distance from $e_{i,j}$ to $f_{k,l}$
    if $x \leq r$ **then**
      $Q_{f_{k,l}} \leftarrow$ local quadric of $f_{k,l}$
      $Q_{prox} \leftarrow Q_{prox} + (1 - (\frac{x}{r})^3)^2 \, Q_{f_{k,l}}$
      $nb_{prox} \leftarrow nb_{prox} + 1$
    end if
  end for
end for
if $nb_{prox} \neq 0$ **then**
  $Q_{prox} \leftarrow \frac{Q_{prox}}{nb_{prox}}$
  $err_{prox} \leftarrow$ compute proximity error of $e_{i,j}$ using Q_{prox}
  $err \leftarrow err \, (1 + \alpha \, err_{prox})$
end if

3.6 Conclusion

The proximity-aware error metric combines the local QEM with a penalty function representing the proximity relations. Therefore, its main parameter is the proximity threshold as it defines the proximity relations. We use this error metric to sort the edge collapses into the priority queue of their respective mesh and subsequently into the global priority queue of the scene throughout the decimation process.

So far, we assumed the proximity threshold as given. We propose to determine this threshold in Chapter 4 by analyzing the proximity between meshes in a scene.

Chapter 4

Proximity Analysis

All meshes in a scene possibly have proximity relations with each other. However, in most scenes, especially the complex ones with numerous meshes, these proximity relations differ significantly overall, ranging from perfect contact to remote proximity. Therefore, we define the proximity in a scene by analyzing all together the most relevant proximity relations between meshes.

This chapter first defines the *proximity between two meshes* (Section 4.1) which we use to build a *proximity distribution* (Section 4.2). We then introduce a *proximity distribution filtering* (Section 4.3) to single out candidates for the computation of a *proximity threshold* (Section 4.4). Finally, we propose a *generalization to N meshes* (Section 4.5).

4.1 Proximity 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 analysis is to find a proximity threshold r such that faces considered to have proximities with other meshes can be tagged, as illustrated in Figure 4.1. For this purpose, we analyze at first the face-to-mesh relations.

This section defines the relation between each face of a mesh and the other mesh by computing the *face-to-mesh distance* (Section 4.1.1) and the corresponding *asymmetry* (Section 4.1.2).

4.1.1 Face-To-Mesh Distance

Let $f_{i,k}$ be a face of a mesh M_i and $c_j(f_{i,k})$ its closest face on a mesh M_j. The face-to-mesh distance between $f_{i,k}$ and M_j is the distance between $f_{i,k}$

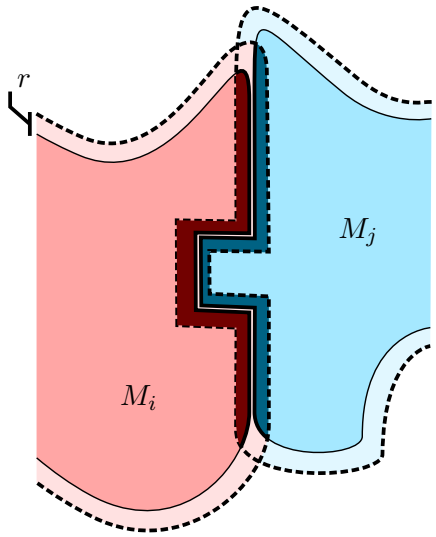


Figure 4.1: Proximity between two meshes [Gha+18]. The faces of M_i (respectively M_j) that fall within a distance r from M_j (respectively M_i) are highlighted.

and $c_j(f_{i,k})$:

$$d(f_{i,k}, M_j) = d(f_{i,k}, c_j(f_{i,k})) \tag{4.1}$$

with d the Euclidean distance.

We consider $f_{i,k}$ to have a proximity relation with M_j if the face-to-mesh distance is small enough, and also symmetric. Therefore, this distance should be the same as the corresponding face-to-mesh distance between $c_j(f_{i,k})$ and M_i:

$$\begin{aligned} d(f_{i,k}, M_j) &= d(c_j(f_{i,k}), M_i) \\ \Leftrightarrow d(f_{i,k}, c_j(f_{i,k})) &= d(c_j(f_{i,k}), c_i(c_j(f_{i,k}))) \end{aligned} \tag{4.2}$$

4.1.2 Asymmetry

In practice, a strict symmetry is not always desirable as the face-to-mesh distances might be affected by variations of tessellation level across meshes. Thus, rather than checking the symmetry, we measure the asymmetry between corresponding face-to-mesh distances.

The asymmetry of the face $f_{i,k}$ regarding the mesh M_j is computed as follows:

$$a(f_{i,k}, M_j) = \mid d(f_{i,k}, c_j(f_{i,k})) - d(c_j(f_{i,k}), c_i(c_j(f_{i,k}))) \mid \tag{4.3}$$

As the analysis is aimed at mesh parts that are the most likely to have proximities, only faces with a small asymmetry value regarding a given mesh are considered. Furthermore, faces with the smallest asymmetry values are given more importance in the course of the proximity analysis.

Therefore, we compute $\hat{\mathcal{A}}(f_{i,k}, M_j)$ as a function of the area $\mathcal{A}(f_{i,k})$ of the face $f_{i,k}$ weighted with regard to its asymmetry value regarding the mesh M_j, so that $\hat{\mathcal{A}}$ decreases when the asymmetry increases:

$$\hat{\mathcal{A}}(f_{i,k}, M_j) = \begin{cases} \mathcal{A}(f_{i,k}) \left(1 - \left(\frac{a(f_{i,k}, M_j)}{a_{\frac{1}{4}}}\right)^2\right)^2 & \text{if } a(f_{i,k}, M_j) \leq a_{\frac{1}{4}}, \\ 0 & \text{otherwise,} \end{cases} \tag{4.4}$$

with $a_{\frac{1}{4}}$ the first quartile of asymmetry measured on the face-to-mesh distances between the two meshes.

Intuitively, such a function allows to discard face-to-mesh relations that will not contribute to the proximity analysis. We have validated this function experimentally, and it should be noted that small changes in its value have a negligible impact over the proximity analysis.

4.2 Proximity Distribution

We populate a proximity distribution $\mathcal{D}$ with the face-to-mesh distances, where each distance sample is weighted with regard to its asymmetry by $\hat{\mathcal{A}}$. As the distance d is asymmetric by construction, $\mathcal{D}$ is built with distances from both M_i to M_j and M_j to M_i.

Figure 4.2 illustrates different possible distributions for two meshes. In the ideal case of Figure 4.2(a), all the faces in proximity areas are at the same distance d_{ideal} from the other mesh. This generates a clear peak in the proximity distribution $\mathcal{D}$ and the proximity threshold r can be obviously set as d_{ideal}. In another straightforward case shown by Figure 4.2(b), there are two different values possible for the face-to-mesh distances in proximity areas. Thus, there are two peaks in $\mathcal{D}$, which leaves two options for r, r_0 in the case of a restricted definition of proximity and r_1 in the case of a more extended notion of proximity. In practice, as seen in Figure 4.2(c), the geometric configuration of meshes are more ambiguous, generating a complex proximity distribution $\mathcal{D}$, with multiple peaks and thus numerous possibilities for the proximity threshold r, which is therefore more complicated to set.

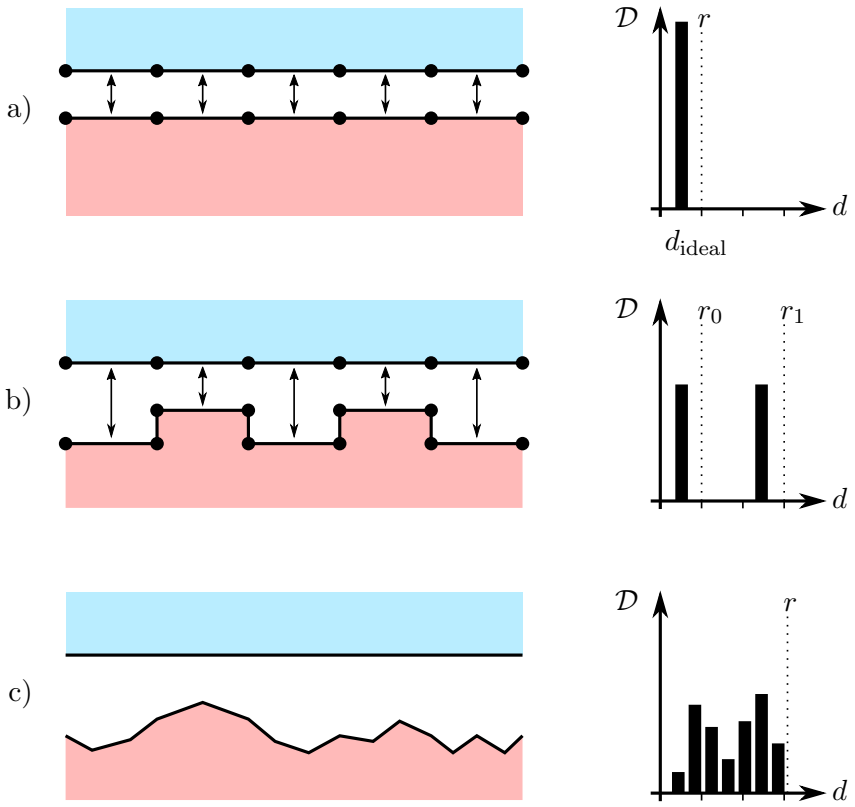


Figure 4.2: Distribution schemes [Gha+19]. Examples of proximity distributions between two surfaces, ranging from simple (top) to more realistic (bottom) cases. r_i are examples of proximity threshold candidates.

4.3 Proximity Distribution Filtering

The proximity threshold r aims at classifying faces that are in proximity areas, such that their distance to the other mesh is less or equal to r, as illustrated in Figure 4.2. When the proximity distribution $\mathcal{D}$ is noisy, several proximity threshold candidates r_i might be considered. The candidates r_i split $\mathcal{D}$ into groups of faces with consistent distances. The aim is to group faces sharing the same distance, e.g. local maxima in $\mathcal{D}$. A natural solution consists in extracting proximity threshold candidates as local minima of the input distribution $\mathcal{D}$, i.e. simplifying the proximity distribution by filtering it using its local extrema.

As visible in Figure 4.3(a), the input distribution $\mathcal{D}$ is in practice too noisy for a direct extraction of its local minima, and thus 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 suitable candidate thresholds r_i. Among all the existing approaches for signal analysis,

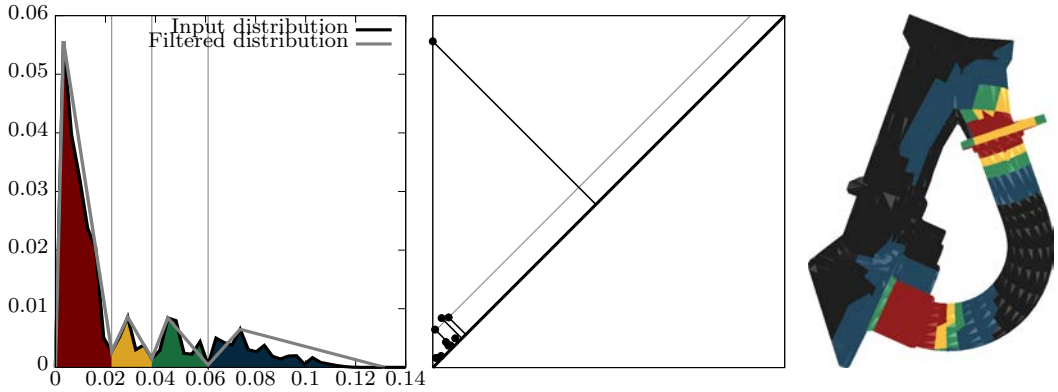


Figure 4.3: Proximity distribution filtering. (a-black curve) The input distribution $\mathcal{D}$ of a scene and (b) its persistence diagram. (a-gray curve) The filtered distribution is constructed from the first four most persistent intervals in the persistence diagram. (c) The `Tube` scene, seen in Figure 3.1. The faces populating the input distribution are colored with regard to the group they belong to.

a relevant approach would be one that considers explicitly the shape of the distribution, and allows to select the number of groups required to explain the data.

Topological Data Analysis (TDA) provides approaches for data analysis by using techniques from topology. Specifically, *persistent homology* [ELZ00] computes topological features of the data, filtering out the noise. The proximity distribution is thus processed as a one-dimensional curve by persistent homology.

Persistent homology views a one-dimensional curve as a height field. For a given height, a virtual horizontal line can be drawn on the curve domain. The interesting parts of such a line are its segments that are above the curve. Hence, when increasing the height of the line, some segments might appear at local minima and some existing segments might be merged at local maxima, depending on the shape of the curve. Such events at local extrema are referred to as topological events. The lines at various heights corresponding to topological events are displayed in Figure 4.4 and used to represent the topological events in a persistence diagram. The persistence diagram is a two-dimensional graph containing a set of points, each point representing an aforementioned segment, with its x coordinate corresponding to the height where the segment appears, and its y coordinate to the height where it is merged with another segment, as seen in Figure 4.4. The persistence of a

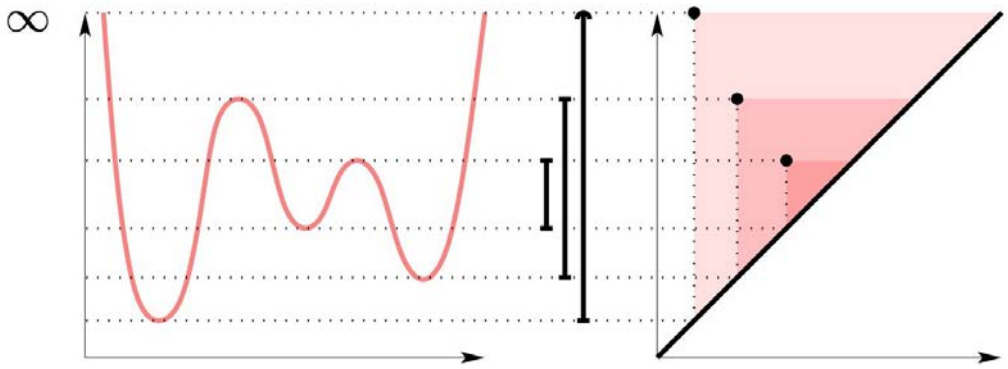


Figure 4.4: Persistent homology [Coh05]. The virtual lines at local extrema of the curve (on the left) are used to represent the associated topological events in the persistence diagram (on the right).

segment is measured as the distance between its associated point and the line $y = x$ in the persistence diagram. Thus, each segment is associated to an interval of the curve. A high persistence value of a segment means that it represents a significant interval of the curve while a smaller value corresponds to a less substantial interval. Therefore, persistent homology classifies intervals of a curve by topological prominence.

To simplify the curve representing the proximity distribution $\mathcal{D}$, we select the m most prominent intervals, i.e. those with the highest persistence values, as seen in Figure 4.3(b). As each interval is represented in the persistence diagram by its local minimum and maximum, corresponding to its appearance and merge events, respectively, its local extrema are known. Hence, the filtered proximity distribution is constructed by linearly connecting the successive local extrema corresponding to the m most prominent intervals of the input proximity distribution, as seen in Figure 4.3(a). The proximity threshold candidates r_i are subsequently extracted as the local minima of the simplified curve. Therefore, each such interval forms a group of faces, as illustrated in Figure 4.3(c). By construction, this approach is guaranteed to interpolate the local extrema of $\mathcal{D}$, and generate m proximity threshold candidates r_i.

4.4 Proximity Threshold

The proximity threshold r is used to define the proximity area of an edge, along with the weight of faces in such an area with regard to the edge-to-face distance, by using a smooth polynomial kernel, as seen in Section 3.3. From

Equation 3.3, the proximity threshold r is defined as follows:

$$r = \frac{x}{\sqrt[3]{1 - \sqrt[2]{\phi(x)}}}. \tag{4.5}$$

with x an edge-to-face distance and $\phi(x)$ its corresponding weight.

In the case of a narrow proximity interval, e.g. the red interval in Figure 4.3(a), the smoothing using its proximity threshold r_i might over smooth the influence of the 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, so as to preserve the face influence. Thus, the proximity threshold r used in Equation 3.3 is computed so that:

$$\phi(r_i) = w_{ref} \tag{4.6}$$

with w_{ref} the reference weight for a face at distance r_i from an edge.

Therefore, for a chosen candidate r_i, we compute the proximity threshold r by applying Equation 4.6 to Equation 4.5:

$$r = \frac{r_i}{\sqrt[3]{1 - \sqrt[2]{w_{ref}}}}. \tag{4.7}$$

The smooth polynomial kernel in Figure 4.5 illustrates the relation between the candidate threshold r_i and the final threshold r, i.e. the influence of faces within a distance r_i of an edge as compared to faces within a distance r of an edge but further than a distance r_i.

For all the experiments produced in this thesis in Chapter 5, we used the first most prominent local minimum, out of four extracted as proximity threshold candidates. Likewise, we experimentally set the reference weight w_{ref} at 0.9 in order to compute the proximity threshold from the aforementioned candidate.

The value m corresponding to the number of most prominent intervals depends on the shape of the models and their organization in the scene. While the choice of $m = 4$ is effective for all the experiments presented, it may not be optimal for all scenes. Therefore, the system can output multiple suggestions regarding the proximity threshold, for instance by coloring the associated faces as seen in Figure 4.3(c). Thus, the user can choose the appropriate number of prominent segments m to single out and a subsequent proximity threshold candidate r_i to compute the proximity threshold r.

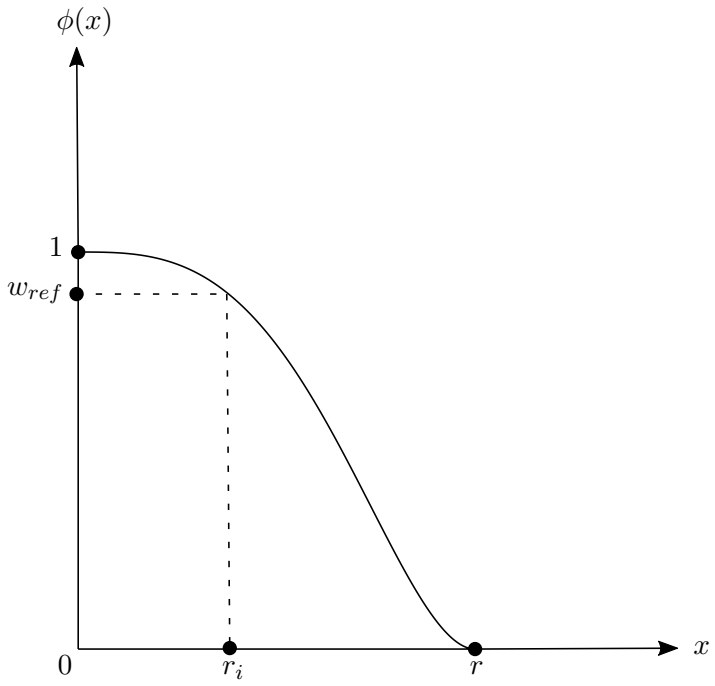


Figure 4.5: Smooth polynomial kernel. The weight of a face within a distance r_i from an edge is comprised between w_{ref} and 1 while the weight of further faces is bounded by w_{ref}.

4.5 Generalization to N Meshes

So far, the face-to-mesh relation computations and the subsequent proximity distribution construction and filtering, from which a candidate threshold is extracted to produce the proximity threshold, were done for scenes consisting of two meshes. We naturally extend this approach to N meshes by aggregating the pairwise proximity distance distribution for all meshes in the scene.

However, considering all pairs of meshes in the scene, especially those that are far away from each other, may lead to unwanted intervals and a relatively high distance might be computed as proximity threshold for the scene. To prevent this, we only considered pairs of meshes that have intersecting or close-by axis-aligned bounding boxes. Therefore, a fixed epsilon value is used for all the experiments presented, set as 0.1% of the axis-aligned bounding box of the scene. Hence, only pairs of meshes possibly having proximity relations are processed during the proximity analysis.

4.6 Conclusion

The proximity analysis consists in aggregating face-to-mesh distances in the scene, weighted by a function of their asymmetry, in a proximity distribution. We filter this distribution to extract proximity threshold candidates using persistent homology. Finally, we compute the proximity threshold of a scene out of a candidate threshold.

We propose an automatic computation of the proximity threshold r used in Chapter 3. We also compute the proximity threshold used to parameterize the proximity-aware decimation of all scenes in Chapter 5 with this approach. Furthermore, this computation can be easily edited if required by the user.

Chapter 5

Results

We decimate a scene using multiple meshes decimation as explained in Chapter 2. Therefore, we run Algorithm 2 with the edge collapse operator along with a given error metric, and we set the stopping criterion as a target number of faces for the scene. Each error metric that can be used in this algorithm defines a specific method, and we compare the performances of the proximity-aware error metric devised in this thesis with previous methods on various scenes.

This chapter first presents the *implementation* for multiple meshes decimation (Section 5.1). We then review the results on different *scenes* (Section 5.2). Finally, we perform a *quantitative analysis* of the results (Section 5.3).

5.1 Implementation

Each method consists in running the multiple meshes decimation algorithm with a specific error metric. We implemented this algorithm and its possible error metrics, i.e. the compared methods, in C++ and we run it on an Intel Xeon E5-2609 1.90GHz, 16GB of RAM. The incremental decimation is performed on a single thread. It should be noted that all compared methods have homogeneously non-optimal implementations to provide fairer comparisons.

This section details how the *quadric error metric* and the *boundary-aware decimation* are compared to the *proximity-aware error metric* developed in this thesis.

Quadric Error Metric

We compare the standard Quadric Error Metric (QEM) of Garland and Heckbert [GH97], detailed in Section 1.2.3, to the proximity-aware error metric by deactivating the proximity error computation both during the computation of the priority queues of the meshes forming a scene, prior to any edge collapse operation, and during the update routine of the priority queue of the mesh where a collapse operation has been performed. As the QEM has been designed to decimate a single mesh, inter-mesh proximity relations do not fall under its scope.

Boundary-Aware Decimation

We also compare the boundary-aware decimation of González et al. [Gon+09], seen in Section 1.4, to the proximity-aware error metric. Contrary to the proximity-aware error metric that changes the error of an edge with regard to proximities, i.e. delays the collapses in proximity areas, and keeps the collapsing vertex computed with the QEM, the boundary-aware decimation uses the QEM to compute the error of all edges, including those having proximities, i.e. does not delay collapses in proximity areas, and might just change the collapsing vertex of an edge having proximities. The boundary-aware decimation tags vertices in proximity areas as boundary, i.e. vertices having a vertex-to-face distance within the proximity threshold with a face of another mesh, and if only one endpoint of an edge is a boundary vertex, this boundary vertex is kept by performing a halfedge collapse instead of an edge collapse. As a consequence, edges with both endpoints tagged as boundary vertices, i.e. edges inside proximity areas, are not preserved.

Proximity-Aware Error Metric

We introduce the proximity-aware error metric in Chapter 3 and the proximity threshold parameterizing it in Chapter 4. The error associated to an edge is computed using Algorithm 3. To speed up the edge-to-face distance queries to detect the possible faces in proximity with an edge, a k-d tree storing the faces of a mesh is computed for each original mesh in the scene, as the potential proximities of an edge are computed with regard to the original surrounding meshes. In order to avoid going through the k-d tree of each other mesh in the scene to compute the proximity relations of an edge, a pre-filtering is done. Thus, two meshes are referred to as neighbor meshes if the distance between their axis-aligned bounding boxes is below the proximity threshold of the scene. Furthermore, to speed up the error computation for an edge, the distance queries are parallelized, using one thread per neighboring mesh.

Scene	# objects	# input faces	# output faces	Prox. threshold (%)	Dec. time (s)	Method
`Tube` (Figure 5.1)	2	5508	500	5.29	8	Proximity-Aware EM
					2	Boundary-Aware
					2	QEM
`Tube*` (Figure 5.2)	2	11526	500	4.68	19	Proximity-Aware EM
					4	Boundary-Aware
					3	QEM
`Connector` (Figure 5.3)	2	5820	580	1.98	5	Proximity-Aware EM
					2	Boundary-Aware
					2	QEM
`Engine` (Figures 5.4 to 5.6)	17	42286	3800	8.34	811	Proximity-Aware EM
					9	Boundary-Aware
					8	QEM
`Car` (Figures 5.7 to 5.9)	425	3075108	150000	0.00829	67680	Proximity-Aware EM
					25980	Boundary-Aware
					23460	QEM

Table 5.1: Scenes details [Gha+19]. The proximity threshold is computed automatically and given as a percentage of the diagonal of the scene's axis-aligned bounding box.

5.2 Scenes

We run the aforementioned methods on different scenes. The characteristics of each scene, as well as the timings for each method, are described in Table 5.1. The stopping criterion of the incremental decimation is set as a target number of faces, regardless the method. As this number of faces is set for the whole scene, each method is expected to balance the face budget differently on each individual mesh.

This section first analyzes the results of each method on a basic scene consisting of *interlocking models* (Section 5.2.1). We then use these methods to decimate objects with *cylindrical shapes* (Section 5.2.2). Finally, we discuss the results on a *medium complexity scene* (Section 5.2.3) and a *realistic complex scene* (Section 5.2.4).

5.2.1 Interlocking Models

The `Tube` scene is composed of two objects that mechanically fit into one another, as seen in Figure 5.1(a). Preserving the geometric features associated to this relation, e.g. the mechanical shoulder, is necessary to preserve the understanding of the scene.

As seen in Figure 5.1(b), the QEM fails at preserving the tube (red model) outer walls on the base (blue model), which retracts during decimation. As expected, the boundary-aware approach in Figure 5.1(c) displays a similar decimation as it mostly performs like the QEM in proximity areas. In comparison, as seen in Figure 5.1(d), the proximity-aware error metric nicely preserves the top shoulder on the tube and the outer tube slot on the base,

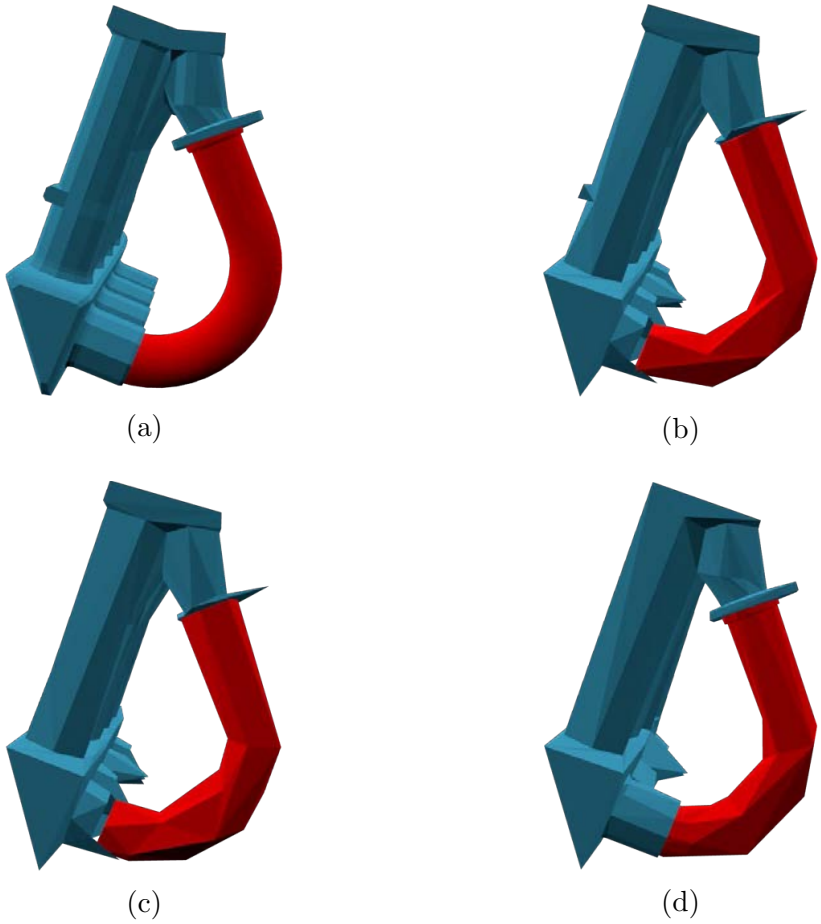
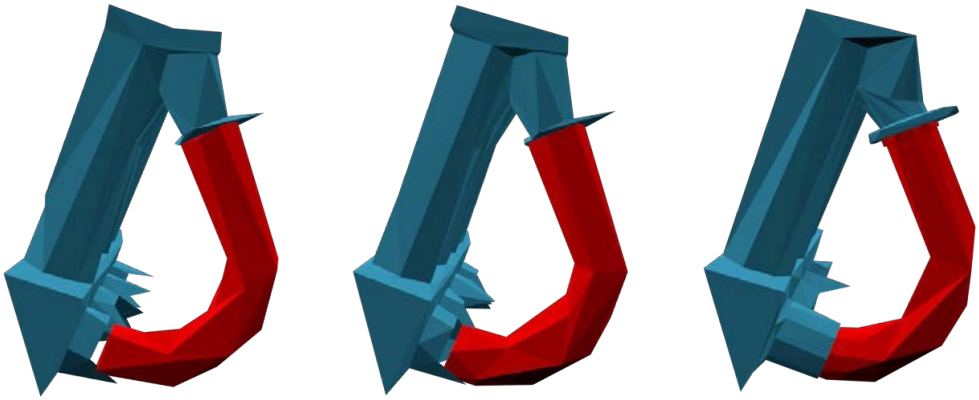


Figure 5.1: `Tube` scene [Gha+19]. (a) Original scene. (b) QEM and (c) Boundary-Aware separate the tube and the base. (d) Proximity-Aware EM keeps the two objects as a whole.

while outside of proximity areas, the results look similar to the QEM. The only noticeable difference is regarding the small structure at the back of the base, which is discarded by the proximity-aware error metric to provide a larger face budget in favour of proximity areas.

We also evaluated the stability of the proximity-aware error metric with regard to variations of tessellation on this first scene as compared to the QEM and the boundary-aware decimation. We generated the `Tube*` scene by subdividing the base of the `Tube` scene using midpoint subdivision. The midpoint subdivision of a mesh is detailed in Section 5.3. As seen in Figure 5.2, the proximity-aware error metric displays similar stability as the previous methods, while still preserving the proximity areas.



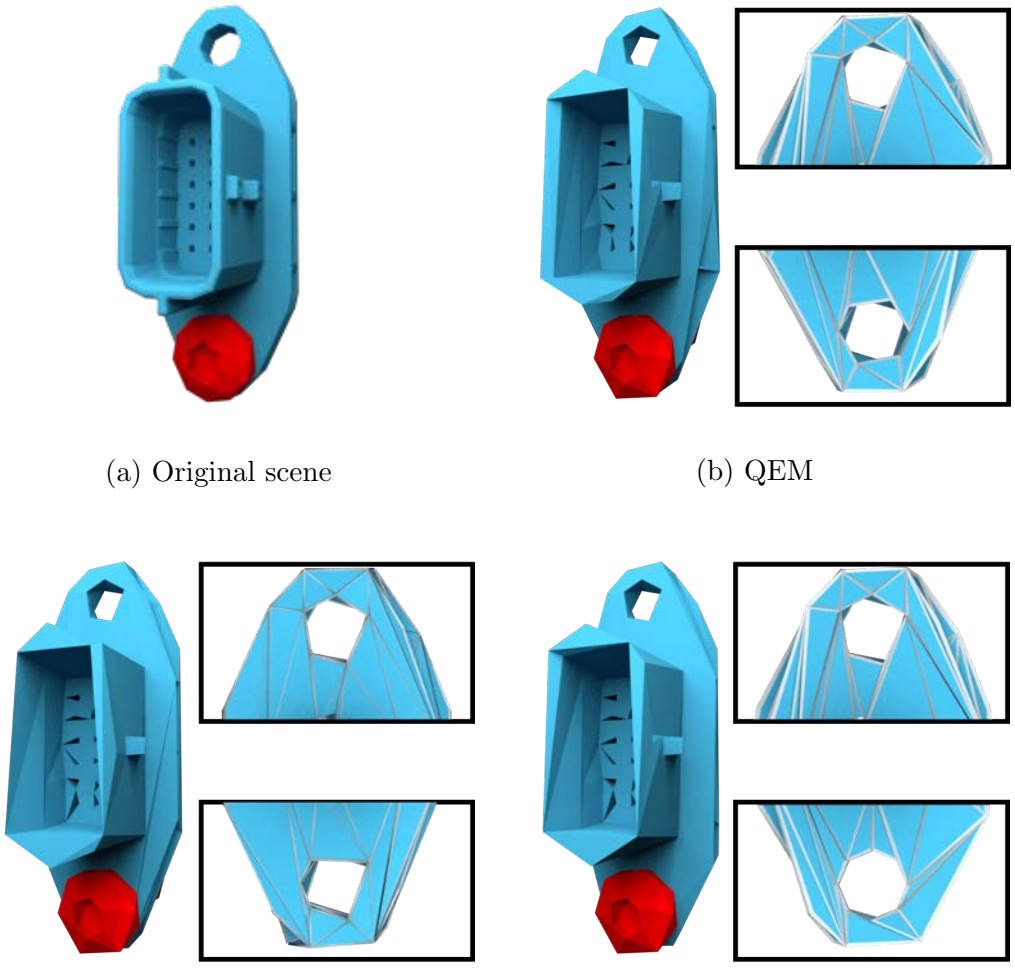
(a) (b) (c)

Figure 5.2: Decimation of the `Tube*` scene [Gha+19]. `Tube*` is obtained by tessellating `Tube` before decimation with (a) QEM, (b) Boundary-Aware and (c) Proximity-Aware EM. All three methods produce similar results to those obtained by simplifying the original `Tube` scene.

5.2.2 Cylindrical Shapes

The `Connector` scene is composed of two objects, a connector and a screw passing through one of the connector holes, as seen in Figure 5.3(a). To preserve the functionality of this scene, the cylindrical shape of the connector hole where the screw belongs has to be preserved so that the screw still fits into it, without any intersection between the two meshes.

While the topology of the top and the bottom of the connector might differ as it is often the case for meshes exported from CAD software, the geometry is very similar. Therefore, the QEM, which is not sensitive to proximity, and by extension the boundary-aware decimation, display a substantially similar decimation for both holes in the connector, with none of them looking cylindrical anymore, as seen in Figure 5.3(b) and Figure 5.3(c). Conversely, as seen in Figure 5.3(d), the proximity-aware error metric displays a significant difference between the two holes of the connector, as the shape of the one where the screw belongs is preserved, contrary to the other one where the decimation is similar to previous approaches. Thus, the reordering of collapses carried out by the proximity-aware error metric is relevant as the face budget is appropriately used. Furthermore, there are no notable overall differences as compared with previous approaches regarding non-proximity areas.



(a) Original scene

(b) QEM

(c) Boundary-Aware

(d) Proximity-Aware EM

Figure 5.3: `Connector` scene [Gha+19]. Contrary to previous approaches, Proximity-Aware EM preserves the shape of the hole where the screw belongs in the connector.

5.2.3 Medium Complexity Scene

The `Engine` scene is composed by 17 meshes with very different shapes, e.g. thin belts, cylinders, box-like shapes, etc, and types of contacts, e.g. interlocked cylinders, partially colled belts, coplanar surfaces, etc, as illustrated by Figure 5.4(a). To preserve the semantics of the scene, these contacts and therefore the mesh parts composing them have to be preserved throughout the decimation.

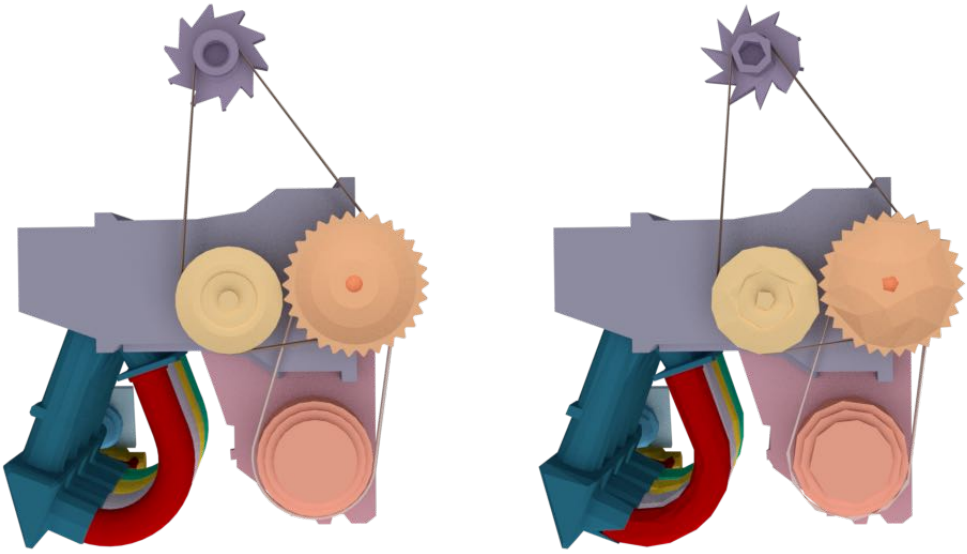
Although the overall decimation looks similar with all three methods in Figure 5.4, the proximity-aware error metric better preserves the geometry of meshes in proximity areas and improves the readability of the system functionality when the view is closer to the models. For instance both the QEM and the boundary-aware decimation generate intersections between the wheel and the belt as seen in Figure 5.5(b) and Figure 5.5(c), which are avoided by the proximity-aware error metric as seen in Figure 5.5(d). Likewise, in Figure 5.6, as for the `Tube` scene, the proximity-aware error metric better preserves the connection between the tubes and the base as compared to previous approaches.

In this scene, the proximity-aware error metric preserves details in proximity areas by discarding some details in other parts of the scene in order to balance pertinently the face budget. For instance, small geometrical components on top and at the bottom of the oil pan (purple model) are removed by the proximity-aware error metric as they are not part of any proximity relation nor are they necessary for the identification of the model they are part of.

5.2.4 Realistic Complex Scene

The `Car` scene consists of 425 meshes modeling the front part of a real car, including systems and mechanical pieces. A global view of this scene can be seen in Figure 1. As for the previous scenes, by taking into account the proximity relations in the decimation process, the associated functional information of the scene are better preserved.

As illustrated by Figure 5.7, the proximity-aware error metric, seen in Figure 5.7(d), better preserves the circular shape of the screw head as compared to the boundary-aware decimation in Figure 5.7(c) and especially the QEM in Figure 5.7(b) which not only destroys the shape but also generates an intersection with the underneath connector. Figure 5.8 illustrates how the proximity-aware error metric better preserves both the shape and the alignment of two connected axis, seen in Figure 5.8(d), as compared to the boundary-aware decimation in Figure 5.8(c) and especially the QEM in Figure 5.8(b) which



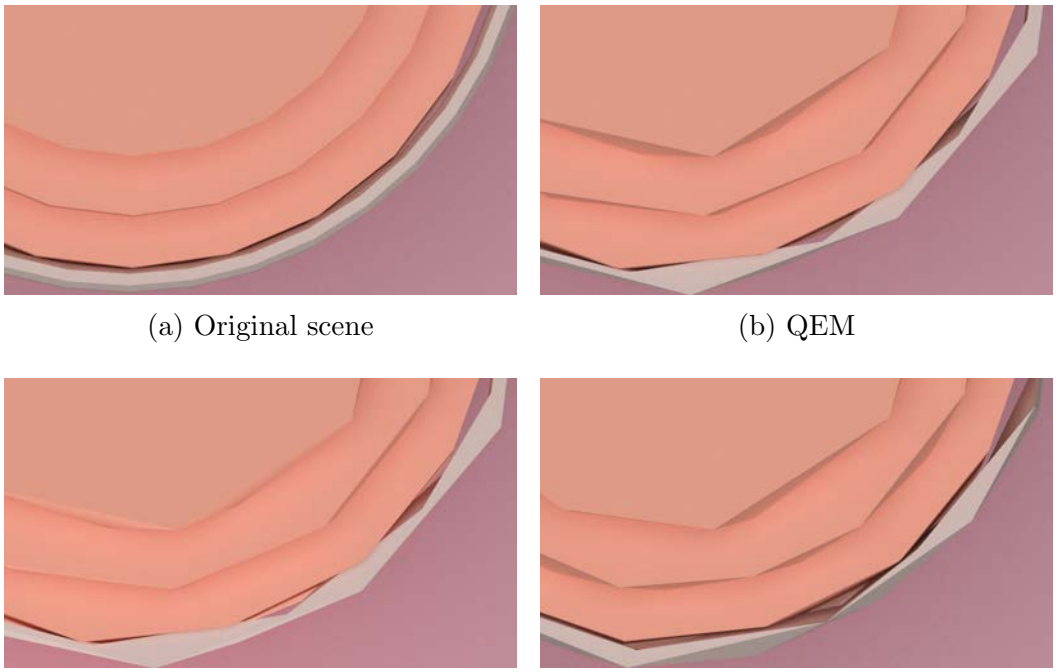
(a) Original scene

(b) QEM

(c) Boundary-Aware

(d) Proximity-Aware EM

Figure 5.4: `Engine` scene [Gha+19]. The decimation with previous approaches and Proximity-Aware EM produce globally similar results for the overview of the scene.



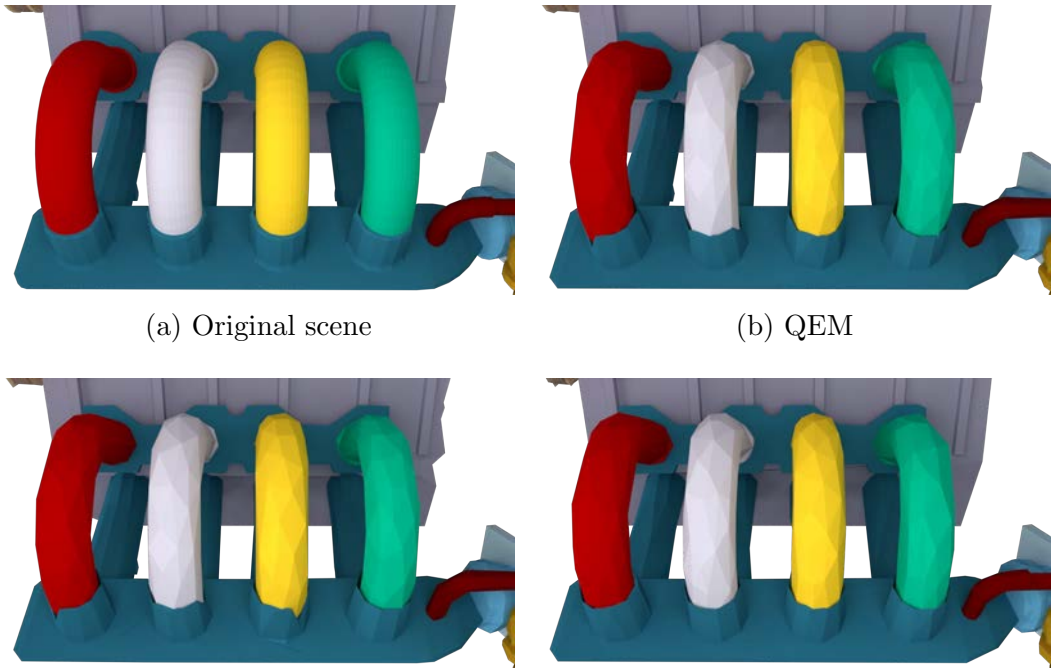
(a) Original scene

(b) QEM

(c) Boundary-Aware

(d) Proximity-Aware EM

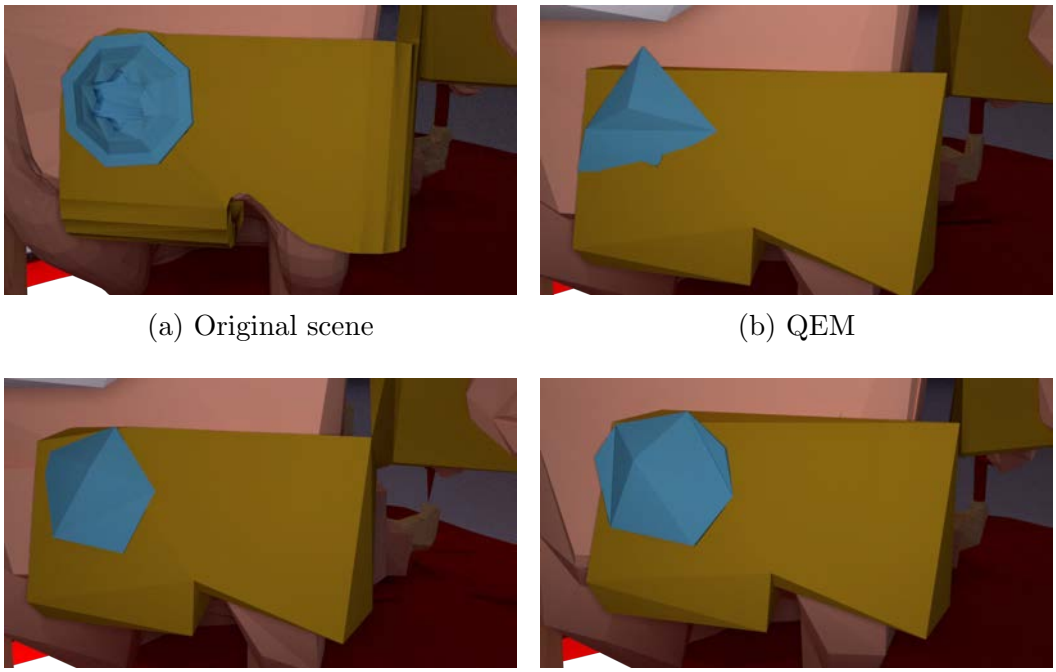
Figure 5.5: `Engine` scene, first close-up view [Gha+19]. Contrary to previous approaches, Proximity-Aware EM does not generate intersections between the wheel and the belt.



(a) Original scene (b) QEM

(c) Boundary-Aware (d) Proximity-Aware EM

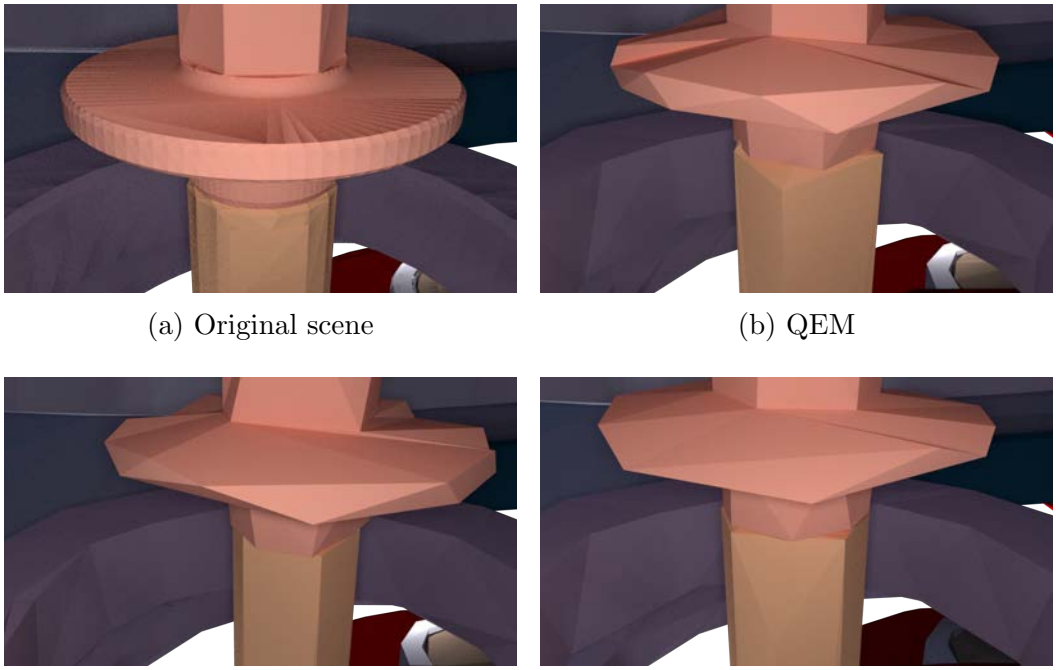
Figure 5.6: `Engine` scene, second close-up view [Gha+19]. Unlike previous approaches, Proximity-Aware EM better preserves the connection between the tubes and the base.



(a) Original scene (b) QEM

(c) Boundary-Aware (d) Proximity-Aware EM

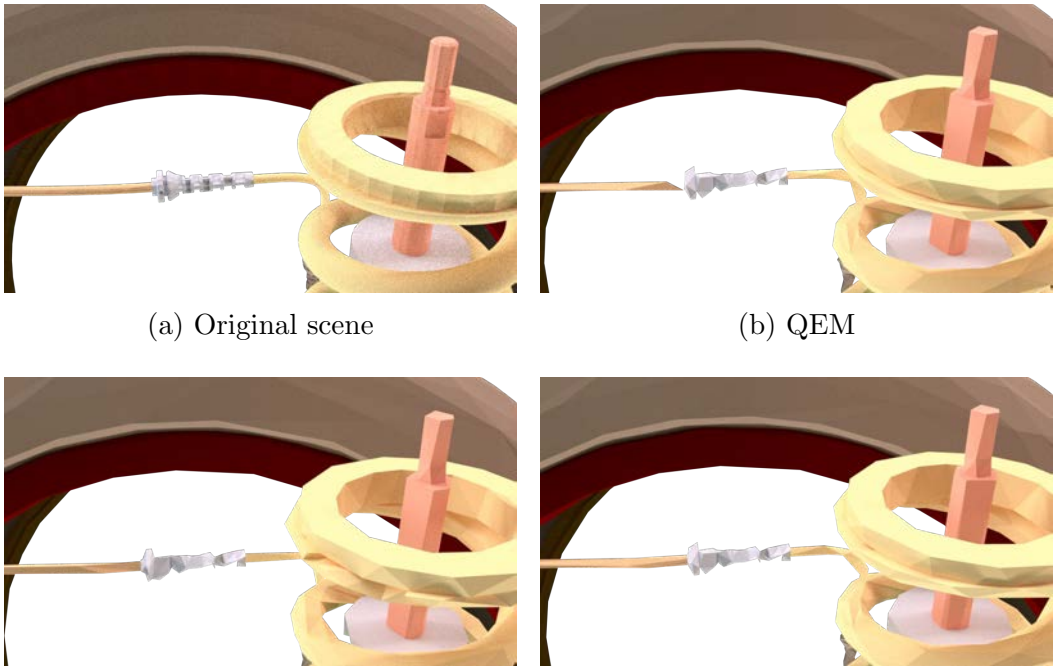
Figure 5.7: `Car` scene, first close-up view [Gha+19]. Compared to previous approaches, Proximate-Aware EM better preserves the shape of the screw head and does not generate an intersection with the connector unlike QEM.



(a) Original scene (b) QEM

(c) Boundary-Aware (d) Proximity-Aware EM

Figure 5.8: `Car` scene, second close-up view [Gha+19]. Compared to previous approaches, Proximity-Aware EM better preserves both the shape and the alignment of the two connected axis, and does not create an intersection between them in contrast to QEM.



(a) Original scene (b) QEM

(c) Boundary-Aware (d) Proximity-Aware EM

Figure 5.9: `Car` scene, third close-up view [Gha+19]. Proximity-Aware EM preserves the shape of the pipe shortened by QEM, and therefore its connection with the neighbor object.

additionally creates intersections. The functional readability of the scene may be severely deteriorated in Figure 5.9 by very thin pipes shortened by the QEM and consequently disconnected from their neighborhood as illustrated by Figure 5.9(b), while they are well preserved by both the proximity-aware error metric as seen in Figure 5.9(d) and the boundary-aware decimation in Figure 5.9(c) as the proximity area is very small and basically consists of a boundary.

5.3 Quantitative Analysis

We corroborate the visual results of Section 5.2 by a quantitative analysis. The most accurate approach to compare two surfaces is the Hausdorff distance, which is detailed in Section 1.2.3. While we assume that the original meshes are sampled enough to enable robust Hausdorff distance computations, as they otherwise would not need to be simplified, we resample the decimated meshes.

This section introduces *mesh subdivision* to resample the decimated meshes, in order to compute the *Hausdorff distances* for both the proximity and non-proximity parts of meshes distinctly.

Mesh Subdivision

The midpoint subdivision is both the most intuitive and widely used approach to subdivide a mesh. As illustrated by Figure 5.10, each face of a mesh undergoes a subdivision by the introduction of midpoints into its edges, which results in four faces instead of one previously. This process can be performed repeatedly until a satisfactory sampling is obtained.

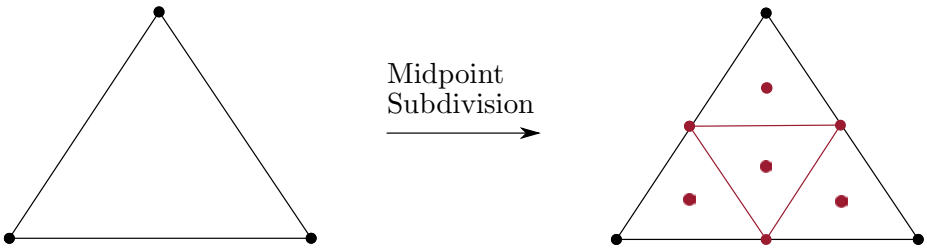


Figure 5.10: Midpoint subdivision of a face. The face is divided into four faces by introducing a vertex in the middle of each edge. Furthermore, a vertex is additionally created at the centroid of each such face.

We perform a single pass of such a mesh subdivision on the meshes of the decimated scenes of Section 5.2. Moreover, the centroids of all faces are added to the list of vertices following the subdivision process, in order to have more points for the Hausdorff distance computations, as seen in Figure 5.10, since it consists in computing point-to-plane distances.

Hausdorff Distances

The Hausdorff distance can be computed efficiently between two surfaces, i.e. an original and a simplified mesh. In the case of a scene consisting of multiple meshes, the Hausdorff distance of the scene is the maximum Hausdorff distance of all pairs of original and simplified meshes. We thus compare numerically the results of the proximity-aware error metric with those of the QEM and the boundary-aware decimation. However, as the proximity-aware error metric aims at preserving specifically the geometry in

Scene	Proximity mean	Non-proximity mean	Proximity max	Non-proximity max	Method
`Tube`	**0.0897**	0.446	**0.594**	5.23	Proximity-Aware EM
(Figure 5.1)	0.336	0.395	4.82	**5.01**	Boundary-Aware
	0.384	**0.348**	4.81	5.02	QEM
`Tube*`	**0.0774**	0.388	**2.45**	5.63	Proximity-Aware EM
(Figure 5.2)	0.455	0.273	5.00	5.01	Boundary-Aware
	0.462	**0.248**	4.98	**4.98**	QEM
`Connector`	**0.0540**	0.305	**0.514**	**2.33**	Proximity-Aware EM
(Figure 5.3)	0.254	0.313	1.15	2.44	Boundary-Aware
	0.240	**0.301**	0.787	3.83	QEM
`Engine`	**0.0475**	0.115	**1.08**	2.39	Proximity-Aware EM
(Figures 5.4 to 5.6)	0.0527	0.205	1.47	2.91	Boundary-Aware
	0.0487	**0.0488**	1.47	**0.389**	QEM
`Car`	**0.00591**	0.0124	**0.659**	**1.10**	Proximity-Aware EM
(Figures 5.7 to 5.9)	0.0143	0.0116	1.20	1.88	Boundary-Aware
	0.0138	**0.0105**	1.75	2.50	QEM

Table 5.2: Hausdorff distances [Gha+19]. The distances between meshes are given as a percentage of the diagonal of the scene's axis-aligned bounding box (the less the better), and computed separately for proximity and non-proximity areas.

proximity areas, while preserving the overall geometry in non-proximity areas, we measure separately the errors introduced inside and outside the proximity areas.

Therefore we divide an original scene into a proximity part and a non-proximity part, by using the proximity definition of Section 3.1 with the proximity threshold computed in Chapter 4. Hence, the edges having proximities with other meshes are detected. As the Hausdorff distance is a point-to-plane distance, it translates into a vertex-to-face distance. Moreover, as mentioned in Section 1.2.3, the Hausdorff distance is two-sided. As a result, for each mesh, the vertex-to-face distances have to be computed for both its original and simplified form, and the maximum of such distances among all meshes is the Hausdorff distance of the corresponding part of the scene, i.e. either the proximity or the non-proximity part. To this mean, the vertices of the original meshes are divided into two categories, those in the proximity part of the scene, i.e. endpoints of edges in proximity areas, and those that are not. Likewise, the vertices of the simplified meshes, resampled for the purpose of Hausdorff distance computations, are also divided into these two categories. Such a vertex is considered to be in the proximity part if its closest face on the corresponding original mesh, i.e. the face defining its vertex-to-face distance, belongs to the proximity part of the scene, i.e. has all its three edges in proximity areas. Thus, the vertex-to-face distances are assigned to either the proximity or the non-proximity part of a scene.

We evaluate the performances of the proximity-aware error metric, to ensure on the one hand that the proximity part of the scene is better preserved

than with previous approaches, and on the other hand that it does not significantly underperform in the non-proximity part as compared to previous approaches. Table 5.2 reports the mean and maximum Hausdorff distance of both the proximity and non-proximity part of each scene seen in Section 5.2, for the proximity-aware error metric and the previous approaches. As expected, the proximity-aware error metric yields to much lower errors in proximity areas, at the cost of higher errors in other areas of the scene. In most cases however, the proximity-aware error metric outputs a comparable error in non-proximity areas, and in two cases, for the `Connector` and `Car` scenes, it introduces less error than the previous approaches, due to side effects of the proximity areas.

5.4 Conclusion

In the context of the joint decimation of multiple meshes with proximity relations contributing to the semantics of a scene, the proximity-aware error metric displays better results than previous approaches on various scenes. It performs better in proximity areas as it preserves the inter-model relations and consequently the scene as a whole, with negligible aftermath on other areas.

We validate these visual findings by a quantitative analysis on both the proximity and the non-proximity part of a scene, using Hausdorff distances.

Conclusion

Summary

This thesis introduces the decimation of multiples meshes using a proximity-aware error metric which is parameterized following a proximity analysis of the scene, in order to preserve both the geometry of each mesh and its relations with other meshes when decimating a complex 3D scene consisting of multiple meshes. Our solution is based on the following contributions.

First, we propose a mechanism for the joint decimation of multiple meshes by extending the incremental decimation algorithm for a single mesh. We compute a priority queue of topological operations for each mesh by using an error metric to estimate the error introduced by an operation on its associated mesh, and interleave the operations of all meshes by constructing a global priority queue with the top operation of each priority queue. The top operation of the global priority queue is iteratively popped out and performed on its associated mesh, and following the update of the corresponding priority queue, the new top operation is inserted in the global priority queue. The advantage of this global priority queue is twofold, as it allows to perform operations with consistent errors on all meshes and thus efficiently distributes the face budget between meshes, while doing so with the cost to update the priority queue of a mesh being the same as in the case of single mesh decimation.

Second, we devise a proximity-aware error metric by extending the Quadric Error Metric (QEM) [GH97] to account for proximity relations between meshes in a scene. The QEM uses the local quadric of an edge, characterizing the influence of its surrounding faces, to compute the local error associated to the edge collapse at the vertex minimizing it. Similarly, we also construct a proximity quadric for the edge from the surrounding meshes, by adding the quadrics of its close faces on other meshes weighted by a function of their distance to the edge, and normalize it using the total number of faces in proximity so that the influence of each such face does not change regarding other faces. We use this proximity quadric to compute a proximity error at the collapsing vertex defined with the QEM, as we do not aim at changing the collapses but only at reordering them with regard to proximity. Hence, we derive the proximity-aware error metric by combining the local error and the proximity error so that

the proximity error penalizes the local error and delays the collapse of edges in proximity areas.

Last, we analyze the spacial relationships between meshes in a scene to derive a proximity threshold defining the proximity relations between meshes, i.e. the proximity areas of the scene. We populate a proximity distribution with face-to-mesh distances weighted by a function of their asymmetry for each face of a pair of close meshes in a scene. We aggregate the distributions for all pairs of close meshes in a scene and filter the resulting proximity distribution to group faces with consistent distances using persistent homology [ELZ00]. Hence, we extract the local minima of this filtered distribution as proximity threshold candidates, which each candidate demarcating a group of faces. We chose to divide the faces of the proximity distribution into four groups and selected the candidate of the first group as the proximity threshold for all our experiments presented in this thesis. Therefore, this analysis can be used to suggest multiple proximity configurations, thus enabling semi-automatic configuration.

We combine our contributions to define a fully automatic pipeline for proximity-aware multiple meshes decimation. In pre-process, we run the proximity analysis to compute the proximity threshold, in order to parameterize the proximity-aware error metric. Then, we run the algorithm for the joint decimation of multiple meshes using the edge collapse operator with the proximity-aware error metric. As shown on various CAD scenes, the proximity-aware error metric produces comparable overall results to previous approaches, while better preserving the proximity relations between meshes.

Perspectives

While we use our contributions all together to decimate CAD scenes consisting of multiple meshes, we also point out that each one of these contributions can be used either independently or for other applications. The multiple meshes decimation scheme is generic can thus be used with any topological operator and error metric. The proximity-aware error metric can both be used as error metric with any other generic decimation scheme involving multiple meshes and have its proximity threshold defined with any other proximity analysis. Finally, the proximity analysis can be used to define proximity for any purpose.

Our approach is based on the assumption that proximity relations imply semantic or functional relationships between models, which is often accurate for CAD objects, with shapes designed to achieve a specific task. An interesting future investigation would be to extend this approach to other relations, such

as alignment, symmetry, instances, or user-defined relations. The proximity analysis might also be less accurate when the input meshes contain large faces, as the distance anisotropy function is sampled for each face of a mesh. This is not a problem in most cases as detailed meshes are to be decimated, however an interesting research direction would be to integrate the spatial relationship over the faces, in order to achieve a truly tessellation-independent method. Our current implementation introduces a time overhead as compared to the QEM, which could be significantly reduced by optimizing distance queries and neighborhood relations, e.g. by caching faces in proximity areas.

Bibliography

[ACK13] Marco Attene, Marcel Campen, and Leif Kobbelt. "Polygon mesh repairing: An application perspective". In: *ACM Computing Surveys (CSUR)* 45.2 (2013), p. 15.

[All+08] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. "Recent advances in remeshing of surfaces". In: *Shape analysis and structuring*. Springer, 2008, pp. 53–82.

[BA09] Tamy Boubekeur and Marc Alexa. "Mesh simplification by stochastic sampling and topological clustering". In: *Computers & Graphics* 33.3 (2009), pp. 241–249.

[BF05] H Borouchaki and PJ Frey. "Simplification of surface mesh using Hausdorff envelope". In: *Computer methods in applied mechanics and engineering* 194.48-49 (2005), pp. 4864–4884.

[BK04] Stephan Bischoff and Leif Kobbelt. "Teaching meshes, subdivision and multiresolution techniques". In: *Computer-Aided Design* 36.14 (2004), pp. 1483–1500.

[BK05] Stephan Bischoff and Leif Kobbelt. "Structure preserving CAD model repair". In: *Computer Graphics Forum*. Vol. 24. 3. 2005, pp. 527–536.

[Bot+10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. AK Peters/CRC Press, 2010.

[BW03] Jiří Bittner and Peter Wonka. "Visibility in computer graphics". In: *Environment and Planning B: Planning and Design* 30.5 (2003), pp. 729–755.

[Cac19] Fernando Cacciola. "Triangulated Surface Mesh Simplification". In: *CGAL User and Reference Manual*. 4.14. CGAL Editorial Board, 2019.

[CAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. "Variational shape approximation". In: *ACM Transactions on Graphics (ToG)*. Vol. 23. 3. ACM. 2004, pp. 905–914.

[CMS98] Paolo Cignoni, Claudio Montani, and Roberto Scopigno. "A comparison of mesh simplification algorithms". In: *Computers & Graphics* 22.1 (1998), pp. 37–54.

[Coh+96] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick Brooks, and William Wright. "Simplification Envelopes". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. ACM, 1996, pp. 119–128.

[Coh05] David Cohen-Steiner. "Persistance topologique". In: *Journées de Géométrie Algorithmique*. 2005.

[ELZ00] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. "Topological persistence and simplification". In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE. 2000, pp. 454–463.

[EMB01] Carl Erikson, Dinesh Manocha, and William V Baxter III. "HLODs for faster display of large static and dynamic environments". In: *Proceedings of the 2001 symposium on Interactive 3D graphics*. ACM. 2001, pp. 111–120.

[Gao+10] Shuming Gao, Wei Zhao, Hongwei Lin, Fanqin Yang, and Xiang Chen. "Feature suppression based CAD mesh model simplification". In: *Computer-Aided Design* 42.12 (2010), pp. 1178–1188.

[Gar99] Michael Garland. *Quadric-based polygonal surface simplification*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1999.

[GBK03] Stefan Gumhold, Pavel Borodin, and Reinhard Klein. "Intersection free simplification". In: *International Journal of Shape Modeling* 9.02 (2003), pp. 155–176.

[GH97] Michael Garland and Paul S. Heckbert. "Surface Simplification Using Quadric Error Metrics". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. ACM, 1997, pp. 209–216.

[GH98] Michael Garland and Paul S. Heckbert. "Simplifying Surfaces with Color and Texture Using Quadric Error Metrics". In: *Proceedings of the Conference on Visualization '98*. IEEE, 1998, pp. 263–269.

[Gha+18] Anahid Ghazanfarpour, Nicolas Mellado, Loïc Barthe, and Jean-Pierre Jessel. "Mesh Simplification for Virtual Objects Disassembly". In: *Journées Françaises d'Informatique Graphique (j.FIG)*. (Best Paper Award). 2018.

[Gha+19] Anahid Ghazanfarpour, Nicolas Mellado, Loïc Barthe, and Jean-Pierre Jessel. "Proximity-Aware Multiple Meshes Decimation using Quadric Error Metric". In: *under revision (Graphical Models)* (2019).

[Gon+09] Carlos González, Jesús Gumbau, Miguel Chover, Francisco Ramos, and Ricardo Quirós. "User-assisted simplification method for triangle meshes preserving boundaries". In: *Computer-Aided Design* 41.12 (2009), pp. 1095–1106.

[Gué99] André Guéziec. "Locally toleranced surface simplification". In: *IEEE Transactions on Visualization and Computer Graphics* 5.2 (1999), pp. 168–189.

[Ham94] Bernd Hamann. "A data reduction scheme for triangulated surfaces". In: *Computer aided geometric design* 11.2 (1994), pp. 197–214.

[HG97] Paul S Heckbert and Michael Garland. *Survey of polygonal surface simplification algorithms*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1997.

[Ho+06] Tan-Chi Ho, Yi-Chun Lin, Jung-Hong Chuang, Chi-Han Peng, and Yu-Jung Cheng. "User-assisted mesh simplification". In: *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*. ACM. 2006, pp. 59–66.

[Hop+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. "Mesh Optimization". In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. ACM, 1993, pp. 19–26.

[Hop96] Hugues Hoppe. "Progressive Meshes". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. ACM, 1996, pp. 99–108.

[Hop97] Hugues Hoppe. "View-dependent Refinement of Progressive Meshes". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. ACM, 1997, pp. 189–198.

[Hop99] Hugues Hoppe. "New Quadric Metric for Simplifying Meshes with Appearance Attributes". In: *Proceedings of the 10th IEEE Visualization 1999 Conference (VIS '99)*. IEEE, 1999.

[Hu+16] Kaimo Hu, Dong-Ming Yan, David Bommes, Pierre Alliez, and Bedrich Benes. "Error-bounded and feature preserving surface remeshing with minimal angle improvement". In: *IEEE transactions on visualization and computer graphics* 23.12 (2016), pp. 2560–2573.

[KCS98] Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. "A general framework for mesh decimation". In: *Graphics interface*. Vol. 98. 1998, pp. 43–50.

[KG03] Youngihn Kho and Michael Garland. "User-guided simplification". In: *Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM. 2003, pp. 123–126.

[KKL02] Sun-Jeong Kim, Chang-Hun Kim, and David Levin. "Surface simplification using a discrete curvature norm". In: *Computers & Graphics* 26.5 (2002), pp. 657–663.

[KLS96] Reinhard Klein, Gunther Liebich, and Wolfgang Straßer. "Mesh reduction with error control". In: *Proceedings of Seventh Annual IEEE Visualization'96*. IEEE. 1996, pp. 311–318.

[KT96] Alan D Kalvin and Russell H Taylor. "Superfaces: Polygonal mesh simplification with bounded error". In: *IEEE Computer Graphics and Applications* 16.3 (1996), pp. 64–77.

[Kwo+15] Soonjo Kwon, Byung Chul Kim, Duhwan Mun, and Soonhung Han. "Simplification of feature-based 3D CAD assembly data of ship and offshore equipment using quantitative evaluation metrics". In: *Computer-Aided Design* 59 (2015), pp. 140–154.

[Lin00] Peter Lindstrom. "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, pp. 259–262.

[LT97] Kok-Lim Low and Tiow-Seng Tan. "Model simplification using vertex-clustering". In: *Proceedings of the 1997 symposium on Interactive 3D graphics*. ACM. 1997, 75–ff.

[LT98] Peter Lindstrom and Greg Turk. "Fast and Memory Efficient Polygonal Simplification". In: *Proceedings of the Conference on Visualization '98*. IEEE, 1998, pp. 279–286.

[Lue+03] David Luebke, Martin Reddy, Jonathan D Cohen, Amitabh Varshney, Benjamin Watson, and Robert Huebner. *Level of detail for 3D graphics*. Morgan Kaufmann, 2003.

[Lue01] David P Luebke. “A developer’s survey of polygonal simplification algorithms”. In: *IEEE Computer Graphics and Applications* 21.3 (2001), pp. 24–35.

[MK05] Martin Marinov and Leif Kobbelt. “Automatic generation of structure preserving multiresolution models”. In: *Computer Graphics Forum*. Vol. 24. 3. 2005, pp. 479–486.

[MP06] Oliver Matias van Kaick and Hélio Pedrini. “A comparative evaluation of metrics for fast mesh simplification”. In: *Computer Graphics Forum*. Vol. 25. 2. Wiley Online Library. 2006, pp. 197–210.

[NM95] Atul Narkhede and Dinesh Manocha. “Fast polygon triangulation based on seidel’s algorithm”. In: *Graphics Gems V*. Elsevier, 1995, pp. 394–397.

[PS03] Erik Pojar and Dieter Schmalstieg. “User-controlled creation of multiresolution meshes”. In: *Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM. 2003, pp. 127–130.

[RB93] Jarek Rossignac and Paul Borrel. “Multi-resolution 3D approximations for rendering complex scenes”. In: *Modeling in computer graphics*. Springer, 1993, pp. 455–465.

[RR96] Rémi Ronfard and Jarek Rossignac. “Full-range approximation of triangulated polyhedra.” In: *Computer Graphics Forum*. Vol. 15. 3. Wiley Online Library. 1996, pp. 67–76.

[Sch97] William J. Schroeder. “A Topology Modifying Progressive Decimation Algorithm”. In: *Proceedings of the 8th Conference on Visualization ’97*. IEEE, 1997, 205–ff.

[SLA15] David Salinas, Florent Lafarge, and Pierre Alliez. “Structure-Aware Mesh Decimation”. In: *Computer Graphics Forum*. Vol. 34. 6. Wiley Online Library. 2015, pp. 211–227.

[SZL92] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. “Decimation of Triangle Meshes”. In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’92. ACM, 1992, pp. 65–70.

[Tur92] Greg Turk. “Re-tiling Polygonal Surfaces”. In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’92. ACM, 1992, pp. 55–64.

[VBL05] Fabien Vivodtzev, Georges-Pierre Bonneau, and Paul Le Texier. “Topology-preserving simplification of 2D nonmanifold meshes with embedded structures”. In: *The Visual Computer* 21.8-10 (2005), pp. 679–688.

[ZG02] Steve Zelinka and Michael Garland. “Permission grids: Practical, error-bounded simplification”. In: *ACM Transactions on Graphics (TOG)* 21.2 (2002), pp. 207–229.

Résumé en français

Contexte

Les scènes 3D complexes sont souvent modélisées par des modèles très larges et détaillés représentant l'information géométrique. Par ailleurs, les modèles représentant des scènes issues de la Conception Assistée par Ordinateur (CAO) encodent également le sens fonctionnel d'une scène au moyen de leurs relations les uns avec les autres. Bien que les primitives de CAO contiennent beaucoup d'informations à propos d'une scène, les scènes issues de CAO sont généralement exportées des logiciels de CAO sous forme de maillages, étant donné que ces informations sont souvent confidentielles et leur rendu coûteux, alors que les maillages sont plus simples à traiter.

La visualisation et l'interaction avec des données 3D complexes est très difficile car l'échelle des données ainsi que leur complexité nécessite de puissants composants matériels et logiciels pour des applications réelles. Ceci devient encore plus critique avec les plateformes mobiles et la visualisation 3D en ligne, ce qui requiert un transfert des données et un rendu rapide même sur des terminaux bas de gamme. Malgré leurs performances limitées, les plateformes mobiles sont très attrayantes car elles donnent accès à beaucoup d'information sur site, et aident ainsi les utilisateurs à effectuer des tâches dans des environnements complexes, par exemple pour la maintenance de machines. Le coût relativement bas et l'ubiquité des appareils mobiles les rendent très attrayants en tant que support pour l'enseignement et la formation.

Les travaux de cette thèse s'inscrivent dans le contexte plus large d'un projet de recherche industriel visant à la formation et à l'apport d'information sur site pour la maintenance de machines au moyen d'un jeu sérieux sur des terminaux mobiles ayant des performances limitées. Etant donné que les scènes complexes issues de CAO sont composées par de nombreux objets, certains objets sont partiellement ou complètement dissimulés par d'autres objets. Ainsi, pour avoir accès à tous les objets dans une scène, l'utilisateur doit être en mesure d'interagir avec la scène en faisant du démontage virtuel. Par ailleurs, étant donné que l'arrangement des objets encode la sémantique d'une scène, le démontage virtuel permet à l'utilisateur d'analyser les relations entre objets et ainsi de mieux comprendre la fonctionnalité de la scène. Par conséquent, le démontage virtuel permet de considérer un objet à la fois individuellement et dans le cadre d'une scène.

En conséquence, afin de permettre la visualisation et l'interaction en temps réel avec une scène complexe issue de CAO sur un terminal ayant des performances limitées, au moyen du démontage virtuel en particulier, la complexité des données 3D modélisant la scène doit être réduite tout en

préservant à la fois la géométrie et le sens fonctionnel de la scène à travers la géométrie de chaque objet et ses relations avec les autres objets, chaque objet étant modélisé par un maillage.

Plusieurs approches peuvent atténuer la complexité croissante des données 3D pour les terminaux bas de gamme. La visibilité peut être utilisée pour ne pas charger les parties cachées d'une scène. Cependant, l'interaction avec une scène nécessite un accès à tous les maillages et lors d'une utilisation hors-ligne, aucun chargement ne peut être effectué. Les instances de modèles composant une scène peuvent être détectées pour éviter le chargement de duplicatas. Néanmoins, les maillages exportés de logiciels de CAO sont souvent différents pour des instances d'un même modèle car ils ont des topologies différentes, c'est-à-dire des sommets, arêtes et faces différents, bien qu'ils aient la même géométrie, c'est-à-dire la même forme. L'approche la plus répandue et la plus pratique pour réduire la complexité des données est la simplification de maillage, consistant à réduire le nombre de polygones d'un maillage complexe en utilisant des algorithmes de décimation. La plupart des algorithmes de décimation sont incrémentaux et consistent à appliquer un opérateur topologique à un maillage à chaque étape de décimation. Une métrique d'erreur estime les changements locaux qui impacteraient la géométrie suite à une opération sur un maillage. Ainsi, une valeur d'erreur est associée à chaque opération, ce qui permet de prioriser l'opération ayant l'erreur la plus faible. L'algorithme de décimation le plus populaire combine l'opérateur d'effondrement d'arête avec la métrique d'erreur quadrique (QEM). Cet algorithme effondre une arête en un sommet minimisant la distance point-plan par rapport à son voisinage local à chaque étape de décimation. La simplification de maillage peut aussi utiliser des données de CAO supplémentaires si elles sont fournies, comme des caractéristiques ou des attributs d'apparence. Toutefois, la décimation de maillage dépendante du point de vue n'est pas une option pour les mêmes raisons que la visibilité ne peut pas être utilisée pour optimiser le chargement des données.

Les maillages exportés de logiciels de CAO comportent souvent des irrégularités, comme des fissures ou des intersections. Par conséquent, les maillages sont nettoyés avant leur décimation grâce à la réparation de maillage. Par ailleurs, étant donné que les maillages triangulaires sont les maillages polygonaux les plus utilisés en informatique graphique, la plupart des opérateurs topologiques sont conçus pour les maillages triangulaires. Ainsi, un algorithme de triangulation de polygones est appliqué aux maillages non triangulaires avant la décimation.

Les scènes issues de CAO sont très souvent des scènes très larges et complexes modélisées par de multiples maillages, chaque maillage encodant la

géométrie d'un modèle. De plus, les relations entre de tels maillages encodent le sens fonctionnel d'une scène. Par conséquent, il est indispensable de préserver la sémantique d'une scène en plus de sa géométrie afin de toujours comprendre le système modélisé, même si les maillages ont été grandement décimés afin d'être manipulés sur des terminaux distants ayant des performances limitées. Etant donné que les informations sémantiques ne sont en général pas fournies comme entrée, la décimation de scènes composées de multiples maillages s'appuie uniquement sur la géométrie, et la préservation conjointe des propriétés individuelles de chaque maillage et des relations entre maillages est très difficile.

Contributions

La problématique de cette thèse est de décimer des scènes 3D modélisées par de multiples maillages tout en préservant la géométrie de chaque maillage ainsi que ses relations avec les autres maillages de la scène. Nous avons observé que pour les systèmes mécaniques, la forme d'un objet est conçue selon sa fonctionnalité et ses interactions avec ses environs, par exemple sa position et ses contacts. Ainsi, nous considérons les maillages voisins pour la décimation d'un maillage donné dans une scène. Nous définissons le concept de voisinage dans une scène comme étant la proximité entre des parties de différents maillages, par opposition au voisinage local qui est défini sur le maillage lui-même.

Nous proposons une solution à cette problématique en étendant l'algorithme de décimation incrémentale d'un maillage à de multiples maillages, et en l'utilisant avec l'opérateur d'effondrement d'arête et une nouvelle métrique d'erreur prenant en compte à la fois la géométrie locale et les relations de proximité avec les autres maillages, que nous appelons métrique d'erreur sensible à la proximité. Nous formulons l'information de voisinage comme une erreur de proximité que nous utilisons pour moduler l'importance des structures géométriques selon les maillages environnants. Ainsi, nous utilisons l'erreur de proximité pour pénaliser l'erreur locale introduite par les opérations de décimation dans les zones de proximité, réduisant ainsi leur priorité comparé aux opérations dans les autres zones de la scène, ce qui retarde leur décimation et fait donc que les relations entre maillages soient mieux préservées, c'est-à-dire la forme des zones de proximité. Par ailleurs, nous fournissons une définition de la proximité dans une scène afin de paramétrer la métrique d'erreur sensible à la proximité.

Ainsi, notre solution est composée de trois contributions indépendantes mais complémentaires, formant un système automatique pour la décimation de maillages multiples prenant en compte la proximité :

Décimation simultanée de multiples maillages Nous rassemblons les opérations de décimation de tous les maillages d'une scène au moyen d'une structure globale, ce qui permet de décimer les maillages avec une erreur consistante. Ce schéma de décimation multi-maillage n'augmente pas la complexité de mise à jour des opérations affectées suite à une opération sur un maillage comparé au schéma de décimation d'un simple maillage, et s'exécute avec une surcharge minimale de mémoire du fait de la structure globale.

Métrique d'erreur sensible à la proximité Nous évaluons une nouvelle erreur pour les effondrements d'arête au sommet minimisant la QEM en pénalisant la QEM avec l'erreur de proximité. L'idée est de bénéficier de la robustesse de QEM pour le placement du sommet effondrant l'arête tout en augmentant l'erreur associée aux effondrements d'arête dans les zones de proximité d'une scène. Ainsi, nous ne changeons pas de tels effondrements d'arête mais les réordonnons seulement, afin de mieux préserver la géométrie là où plusieurs maillages sont proches, au détriment des parties de maillage situées hors des zones de proximité de la scène.

Analyse de proximité au sein d'une scène Nous proposons une approche générique pour la détection des zones de proximité d'une scène. Nous analysons l'arrangement spatial des maillages et leurs distances, dont on dérive une distance, que nous appelons seuil de proximité, pour la configuration automatique de la métrique d'erreur sensible à la proximité.

Nous montrons les performances de notre système sur diverses scène issues de CAO. Notre approche produit des résultats globaux comparables à l'état de l'art, tout en préservant mieux les relations de proximité entre maillages.

Notre approche se base sur l'hypothèse que les relations de proximité impliquent des relations sémantiques ou fonctionnelles entre modèles, ce qui est souvent le cas pour les objets issus de CAO, avec des formes conçues pour remplir une tâche spécifique. Une perspective intéressante serait d'étendre cette approche à d'autres relations, comme l'alignement, la symétrie, les instances, ou bien des relations définies par un utilisateur.

Abstract Progressive mesh decimation by successively applying topological operators is a standard tool in geometry processing. A key element of such algorithms is the error metric, which allows to prioritize operators minimizing the decimation error. Most previous work focus on preserving local properties of the mesh during the decimation process, with the most notable being the Quadric Error Metric which uses the edge collapse operator. However, meshes obtained from CAD scenes and describing complex systems often require significant decimation for visualization and interaction on low-end terminals. Hence preserving the arrangement of objects is required in such cases, in order to maintain the overall system readability for applications such as on-site repair, inspection, training, serious games, etc. In this context, this thesis focuses on preserving the readability of proximity relations between meshes during decimation, by introducing a novel approach for the joint decimation of multiple triangular meshes with proximities.
The works presented in this thesis consist in three contributions. First, we propose a mechanism for the simultaneous decimation of multiple meshes. Second, we introduce a proximity-aware error metric, combining the local edge error (i.e. Quadric Error Metric) with a proximity penalty function, which increases the error of edge collapses modifying the geometry where meshes are close to each other. Last, we devise an automatic detection of proximity areas. Finally, we demonstrate the performances of our approach on several models generated from CAD scenes.

Résumé La décimation progressive de maillage par l'application successive d'opérateurs topologiques est un outil standard de traitement de la géométrie. Un élément clé de tels algorithmes est la métrique d'erreur, qui donne la priorité aux opérateurs minimisant l'erreur de décimation. La plupart des travaux précédents se concentrent sur la préservation des propriétés locales du maillage lors du processus de décimation, le plus notable étant la métrique d'erreur quadrique qui utilise l'opérateur d'effondrement d'arête. Toutefois, les maillages obtenus à partir de scènes issues de CAO et décrivant des systèmes complexes requièrent souvent une décimation significative pour la visualisation et l'interaction sur des terminaux bas de gamme. Par conséquent, la préservation de la disposition des objets est nécessaire dans de tels cas, afin de préserver la lisibilité globale du système pour des applications telles que la réparation sur site, l'inspection, la formation, les jeux sérieux, etc. Dans ce contexte, cette thèse a trait à préserver la lisibilité des relations de proximité entre maillages lors de la décimation, en introduisant une nouvelle approche pour la décimation conjointe de multiples maillages triangulaires présentant des proximités. Les travaux présentés dans cette thèse se décomposent en trois contributions. Tout d'abord, nous proposons un mécanisme pour la décimation simultanée de multiples maillages. Ensuite, nous introduisons une métrique d'erreur sensible à la proximité, combinant l'erreur locale de l'arête (i.e. la métrique d'erreur quadrique) avec une fonction pénalisant la proximité, ce qui augmente l'erreur des effondrements d'arête là où les maillages sont proches les uns des autres. Enfin, nous élaborons une détection automatique des zones de proximité. Pour finir, nous démontrons les performances de notre approche sur plusieurs modèles générés à partir de scènes issues de CAO.