Eurographics Symposium on Parallel Graphics and Visualization (2015)
C. Dachsbacher, P. Navrátil (Editors)

Out-of-Core Framework for QEM-based Mesh Simplification

Hiromu Ozaki[1,2], Fumihito Kyota[1], Takashi Kanai[2]

[1]Silicon Studio Corporation, [2]The University of Tokyo

Abstract

In mesh simplification, in-core based methods using Quadric Error Metric (QEM), which apply a sequence of edge-collapse operations, can generate high-quality meshes while preserving shape features. However, these methods cannot be applied to huge meshes with more than 100 million faces, because they require considerable memory. On the other hand, the quality of simplified meshes by previous out-of-core algorithms tends to be insufficient. In this paper, we propose an out-of-core framework to establish high-quality QEM-based simplification for huge meshes. To simplify a huge mesh using limited memory, the mesh is first partitioned into a set of patches in the out-of core framework using linear classifiers which are trained by clustered points based on the machine learning approach. Also, a scheme to guarantee the exact matching of boundary vertices between neighbor patches is proposed even when each patch is simplified independently. Based on this scheme, out-of-core simplification is established while generating a simplified mesh with almost the same quality as that of the in-core QEM-based method. We apply the proposed method to multiple models including huge meshes and show the superiority of our method over previous state-of-the-art methods in terms of the quality of simplified meshes.

1. Introduction

Due to the advance of technology in the field of computer graphics, e.g. 3D-scanning devices such as range scanners, there has been a great increase in the number of applications dealing with huge meshes. Out-of-core mesh simplification algorithms have been developed to simplify the huge mesh while keeping shape features. Most of these algorithms simplify huge meshes by decomposing them into small submeshes. In this case, the constraint for neighbor boundaries tends to decrease the quality of the simplified meshes. In other approaches, spatial data structures and quantization can be utilized in an out-of-core manner. However, the quality of the simplified meshes is insufficient.

On the other hand, Garland and Heckbert proposed a fast high quality edge collapse-based simplification algorithm using Quadric Error Metric (QEM) [GH97]. The QEM evaluates the cost of the collapse operation for an edge by the squared length from a new vertex position to a plane. Then the edge with minimum cost is collapsed. For this reason, the resulting mesh has considerably higher quality with minimum shape changes owing to the use of QEM. One drawback of this method is that the QEM-based simplification of huge meshes requires a large amount of memory due to the need to construct their graph connectivity.

Our purpose here is to simplify huge meshes without losing their geometrical details and to archive at the same level of quality as in-core QEM-based methods. To achieve this, we consider partitioning an original mesh into a set of smaller sub-meshes so that each sub-mesh can be simplified individually. An output mesh is then constructed by merging simplified sub-meshes. With such a divide-and-conquer scheme, it is of utmost importance for the boundary vertices between simplified sub meshes to be exactly matched.

Our key idea is to add additional processes so as to guarantee the exact matching of boundaries based on the idea of instant Level Of Detail [GDG11]. By applying this idea, our method can, in principle, simplify huge meshes with more or less the same quality as in-core algorithms within reasonable computational time. One of the advantages of our method is that the simplification process can be done independently for each sub-mesh. Out-of-core algorithms are required for mesh simplification itself and also for mesh partition. Here, we propose an out-of-core mesh partition method based on utilizing sampled points as a guide for partitioning a huge mesh.

In later sections, we describe the details of our key idea in Sec. 3 and a detailed overview of our method in Sec. 4.1.

2. Related Work

Mesh simplification and segmentation are now fundamental techniques in geometry processing. Numerous algorithms have been proposed for these topics. Detailed surveys of these subjects in Botsch et al.'s book [BKP*10] and articles [HLS07, SPR06, Lue01] were referenced for further understanding. Here, we will review only the researches on mesh simplification and segmentation which handle huge meshes and are closely related to our work.

Mesh Simplification. Garland and Heckbert proposed a fast high-quality simplification algorithm by applying a sequence of edge collapse operations and by using Quadric Error Metric (QEM) to optimize the vertex positions as minimizing shape changes [GH97]. Although this QEM-based method provides high-quality simplification results, it tends to consume considerable memory and is not suitable for huge meshes. Later, memory-less simplifications were proposed where QEMs are not explicitly stored in the main memory [LT99, Hop99]. However, handling a huge mesh is still a serious problem in terms of memory usage.

More recently, algorithms for parallel and distributed processing have been proposed. Approaches based on total or partial atomic operations have been proposed [GDG11, SN13] for the parallel processing on GPUs efficiently. Other possibilities are based on using vertex clustering [RB93, Lin00, LS01, SG01, SW03, SG05] and RSimp [BP02]. In these approaches, spatial data structures are used and vertices in each cell of such a structure are put together, which seem to be suitable for processing huge meshes. However, the quality of simplified mesh tends to be lower than that of the edge collapse based approaches.

To limit memory usage during simplification, Isenburg et al. proposed mesh simplification via the file streaming process [ILGS03, IL05]. In addition, a method to reduce memory usage has been proposed by spatially subdividing a huge mesh into smaller sub-meshes [CMRS03]. While both methods are based on spatial subdivision, they are different in terms of how data is treated: The former accelerates simplification using a special data structure for the streaming process, and the latter temporarily stores subdivided sub-meshes on a disk.

When a mesh is partitioned into sub-meshes, their boundaries must be handled carefully to simplify them in parallel independently. The vertex clustering approaches described above serve as one of the possible solutions for the boundary issue. In addition, the use of a firewall [DC10] was proposed to prevent propagation effects to neighbor regions in edge collapse operations.

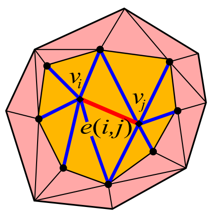
In contrast, our proposed method is a partition-based approach applicable for huge meshes and also provides the same level of quality as in-core QEM-based methods.

Mesh Segmentation. Spectral analysis is one of the basic tools for mesh segmentation. A Laplacian matrix based on neighbor vertices can be used in the spectral analysis. Zhou et al. [ZSGS04] proposed a method using such Laplacian matrices taking into consideration geodesic distances. However, the construction of a Laplacian matrix consumes considerable memory and is inappropriate for huge meshes.

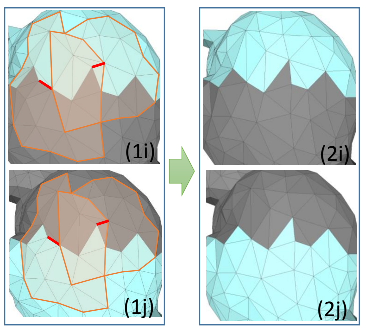
Clustering-based approaches are more suitable for segmenting huge meshes. Julious et al. proposed an iterative clustering method using quasi-developable patches that approximate each sub-mesh to a developable cone [JKS05]. An important aspect in these approaches is how to evaluate the distance between two elements. Liu and Zhang proposed a method by using the K-means clustering and spectral analysis [LZ04]. For CAD applications, Xiao et al. proposed an agglomerative hierarchical clustering algorithm [XLXG11]. Recently, the supervised learning approach using preliminarily labeled meshes is applied to the semantic segmentation of meshes [KHS10, BLVD11].

In contrast, our approach for the partition is based on machine learning and requires less memory than the approach based on Laplacian matrices. Our approach also considers geodesic distances by using clustered points on a mesh as training data.

3. Minimum Decimation Domain and Out-Of-Core Simplification



(a) MDD



(b) Using MDD

Figure 1: (a) MDD (red region) is defined as 2-ring neighbors of end vertices of an edge $e(i,j)$ (red line). The yellow region shows a 1-ring neighbor for $e(i,j)$. A LME $e(i,j)$ has minimum QEM cost over neighbor edges (in blue) locally. (b) The same edges are always selected as LMEs if MDDs are included in two neighbor patches. Also, the boundary vertices of two neighbor patches are matched after the decimation of LMEs. Red lines: LMEs, orange polygons: MDDs.

In this section, we describe the main idea of our out-of-core simplification method for huge meshes. First, we introduce *Local Minimum Edge* (LME) in Instant Level Of Detail (ILOD) proposed by Grund et al. [GDG11]. We then describe the idea of *Minimum Decimation Domain* (MDD) in Sec. 3.1. Finally, we explain how the MDD is applied to out-of-core simplification in Sec. 3.2.

3.1. Defining Minimum Decimation Domain

In the ILOD approach, special edges referred to as LME are introduced. An edge $e(i,j)$ is defined as LME satisfying the following equation, where i and j are the indices of vertices.

$$(Q_i+Q_j)(\bar{v_{ij}}) = \min_{k\subseteq\Omega_i} |(Q_i+Q_k)(\bar{v_{ik}})|, \quad (1)$$

$$= \min_{k\subseteq\Omega_j} |(Q_j+Q_k)(\bar{v_{jk}})|, \quad (2)$$

where Q denotes a Quadric Error Metric (QEM) [GH97], $\bar{v_{ij}}$ denotes an optimal position after $e(i,j)$ is collapsed, and Ω_i is a set of 1-ring neighbor vertices around a vertex v_i. The cost of $e(i,j)$ is defined as $(Q_i+Q_j)(\bar{v_{ij}})$ in ILOD. Eqs. (1) and (2) indicate that $e(i,j)$ has minimum cost among all edges which have v_i or v_j as end points. Such LMEs never have a shared vertex on a mesh, thus allowing collapsing operations in parallel for multiple LMEs.

Another important property of LMEs is that they are *uniquely* determined, that is, the same LMEs are selected independently in terms of search methods and order of searches. This property is the basis of independent simplification for each sub-mesh.

Now we consider a neighbor region needed for deciding a LME. To compute the *QEM cost* for $e(i,j)$, the logical sum of two 1-ring vertices of v_i and v_j is required (a yellow region in Fig. 1). By considering QEM costs for all edges around v_i and v_j, a region of 2-ring neighbor vertices (a red region in Fig. 1) is required to guarantee that $e(i,j)$ can be a LME and has the minimum QEM cost. We define such a red region as MDD.

3.2. Using MDDs for Out-Of-Core Simplification

MDD has a significant advantage in the simplification of huge meshes. By using MDD, partitioning a huge mesh into a set of smaller sub-meshes (called *patches*) is reasonable for simplification. As shown in Fig. 1b, let us suppose that two neighbor patches ((1i) and (1j)) share an overlapped boundary region (light blue regions), and that MDDs (orange regions) are included in such a region. When all LMEs (red lines) in MDDs of two patches are collapsed ((2i) and (2j)), those boundaries become exactly matched. Each patch can be then simplified independently while the boundaries of a neighbor patch are matched.

However, the problems listed below have to be considered when applying the idea of MDDs to the partition of a huge mesh.

1. Partitioning a huge mesh in the out-of-core manner itself is not a trivial task.
2. It is desirable that the faces of each patch are *connective*. If a patch has non-connective regions, the number of neighbor patches tends to increase. Also, a very small region may appear.
3. To collapse more LMEs all at once, each patch should have as much overlapped regions between neighbor patches as possible.
4. To execute simplification processes in parallel, each patch should have more or less the same size, namely, the number of faces in each patch should not differ considerably.

One possible solution is to use spatial data structures e.g. uniform grid. However, partitioning a huge mesh into a set of blocks by using such data structure does not seem to be a good choice for our purpose. If we use such data structure, we cannot predict what patches are constructed; a sub-mesh in a block may have separate pieces of faces or small number of faces. In this case, the problems 2. and 3. described above can arise.

4. Algorithm Details

In this section, we describe details of our method. We first explain the overview of our algorithm in the next subsection.

4.1. Overview

Fig. 2 illustrates our method. Our method consists of the following four phases; sampling, clustering, learning/partitioning, and simplification.

- In the sampling phase, points are evenly sampled on an input mesh (Fig. 2(b), Sec. 4.3) so as to fit the main memory. Such sampled points are used instead of an original mesh for the subsequent processes.
- In the clustering phase, sampled points are clustered so that points in each cluster are not unevenly distributed (Fig. 2(c), Sec. 4.3).
- In the learning/partitioning phase, an original huge mesh is partitioned into a set of patches based on the learned knowledge of the clustered points (Fig. 2(d), Sec. 4.4).
- In the simplification phase, two sub-phases; expansion and decimation, are repeatedly processed (Fig. 2(e) and 2(f), Sec. 4.5), and finally the merging process is executed only once (Fig. 2(g)).

Note that the process of clustering points are executed in an in-core manner, while other processes are executed in an out-of-core manner, as shown in the figure. To improve the computational performance, each process is made to run in parallel on a single, multicore CPU as much as possible. Details of the above phases will be discussed in the subsequent sections.

4.2. Input Meshes

An input mesh must have neighbor connectivity explicitly such as an indexed face set for edge-collapse operations in

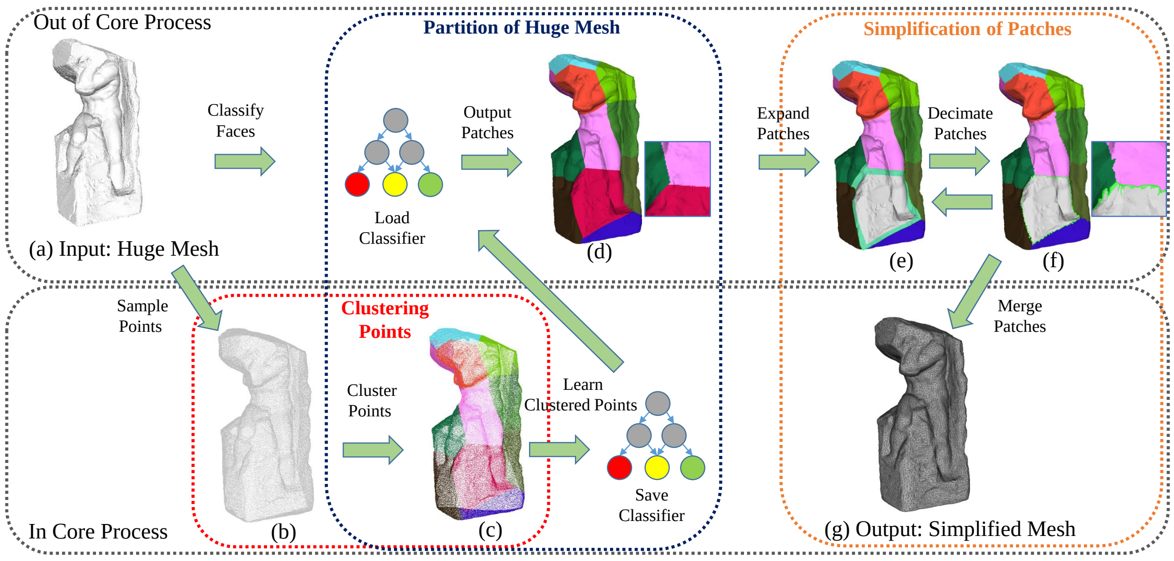


Figure 2: Outline of our out-of-core simplification framework. Gray dotted rectangles show the type of processes. Upper row: Out-of-core processes, Lower row: In-core processes. Green dotted rectangle: Partitioning huge mesh (Sec. 4.3-4.4), Red dotted rectangle: Clustering points (Sec. 4.3), Yellow dotted rectangle: Simplification of huge mesh (Sec. 4.5).

the simplification phase. For a mesh which does not explicitly have the neighbor connectivity such as a polygon soup, it has to be constructed after loading a mesh (or each sub-mesh) from a disk into the main memory. However in other phases, such neighbor connectivity is not required.

Most importantly, it is quite difficult to read a single file of a huge mesh from a disk at one time due to the limited memory. Thus, a three-dimensional bounding box which covers a mesh is spatially decomposed into blocks by using uniform grids, and vertices and faces of a sub-mesh in each block are separately stored on a disk. In the out-of-core phases such as sampling, partitioning and simplification, a part of the mesh in each block is then loaded into the main memory.

However, this decomposition does not consider any geometric information of huge meshes as described in Sec. 3.2. A simple decomposition scheme described above is therefore used only for loading a part of the mesh. Instead, another partitioning method has to be considered for the simplification.

A streaming mesh [ILGS03] is also available to load faces of a huge mesh in all out-of-core phases, although we did not use such a data structure because of implementation issues.

4.3. Clustering Points

Fig. 3 shows our framework of clustering sampled points lying on a huge mesh.

In the first phase in Fig. 3(a), points are sampled so that they are evenly distributed on a mesh by Parallel Poisson Disk Sampling (PPDS) [BWWM10]. PPDS checks that no two points are too close in terms of approximate geodesic distance. Specifically, the geodesic distance between any two neighbor points is not too close, is never less than a certain distance, and is more or less the same all the time. This sampling can be done by loading all parts of a mesh in parallel as described in Sec. 4.2.

The number of sampling points has to be carefully determined so as to fit the main memory. Based on the results of our experiments, the number of sampling points was set to less than 10% of the number of vertices in the original mesh. This is done by changing the sampling radius of PPDS.

Next, sampled points are clustered and the resulting clustered points are used as training data for mesh partitioning. Hence, this clustering directly affects the partition of a huge mesh. It is desirable that the number of points in each cluster is more or less equal, that is, the points in each cluster are more or less evenly distributed. Also, the points in each cluster should be adjacent on the mesh. It is not desirable for the points in distant regions of the mesh are on the same cluster as shown in Fig. 4a.

To address these issues, a Proximity Graph (PG) for sampled points, in which two neighbor points are connected by an edge, is first constructed as shown in Fig. 3b. In this construction, the resources of PPDS are maximally reused; the cell data structure of PPDS is used to find neighbor points, and the approximate geodesic distance is also used to evaluate the length of an edge. By using this distance, the edge between two neighbor points which are spatially close but

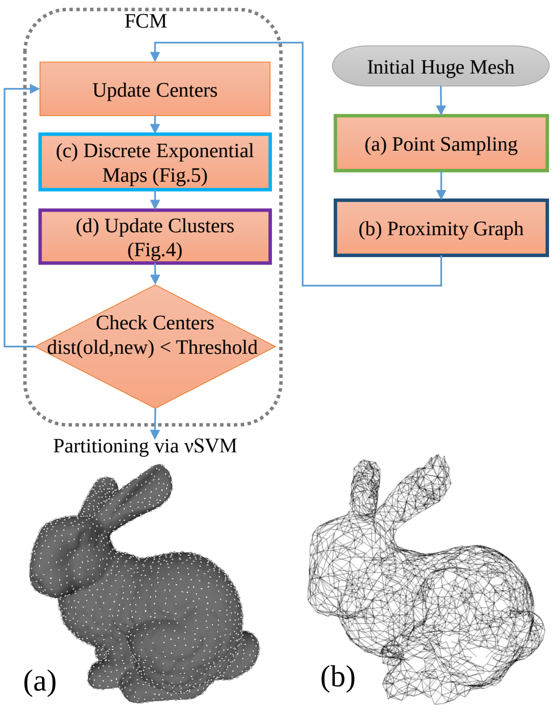

FCM
Update Centers
(c) Discrete Exponential Maps (Fig.5)
(d) Update Clusters (Fig.4)
Check Centers dist(old,new) < Threshold
Initial Huge Mesh
(a) Point Sampling
(b) Proximity Graph
Partitioning via νSVM
(a)
(b)

Figure 3: Our clustering framework. An initial huge mesh described in Sec. 4.2 is used in the sampling phase. Each process is explained in Sec. 4.3. Clustered points are used in the partitioning phase as the training data of νSVM. (a) Sampled points by PPDS and (b) Proximity Graph for Bunny model.

belong to different surface regions will not be included in a PG.

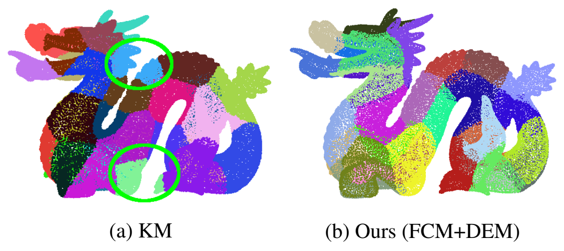


Figure 4: Comparison of clustering results. (a) Clustering by K-Means. In green circles, points on non-connective regions are clustered in a group. (b) Clustering by Fuzzy C-Means and by using approximate geodesic distances. Each cluster is composed of points on a connective region.

To cluster sampled points which are nodes of a PG, the geodesic distance between two arbitrary points must be calculated. In this case, the edge length of a PG is not sufficient for the approximation of the geodesic distance. Alternatively, the Discrete Exponential Map (DEM) [SGW06] is used here for this computation.

In each cluster, a barycentric center of all points is computed and its closest node point in $\mathbb{R}^3$ is found on a PG. A DEM on a tangent plane centered at the closest node point is then computed as shown in Fig. 5. Note that the geodesic distances between the same node points on two different DEMs are not equal. By nature of the DEM computation, geometric errors of geodesic distances are larger the further away are the node points on a DEM from its center. Due to these errors, it is difficult to cluster sampled points using hard clustering methods such as K-Means with DEM.

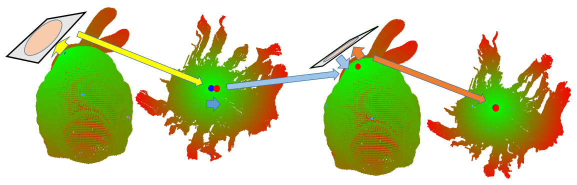


Figure 5: Computing DEM for cluster. From left to right: In the l-th iterative loop, a DEM is computed with a center point (in blue) of each cluster. Next, a center point is moved to a new point (in red), and then it is proceeded to the $(l+1)$-th loop.

To address the issue described above, Fuzzy C-Means (FCM) method is adopted for the clustering, since geometric errors described above do not affect the clustering results significantly. In FCM, points are clustered stochastically and it is more suitable to our clustering method compared to standard K-Means clustering. FCM implies that points are scattered in a circular shape for recognition. FCM also tries to equalize the size of these circles. Therefore, clustered points tend to be as evenly distributed as possible. Fig. 4b shows our clustering result.

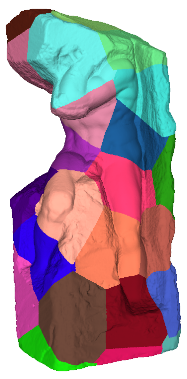
4.4. Partitioning Mesh via Nu-Support Vector Machine

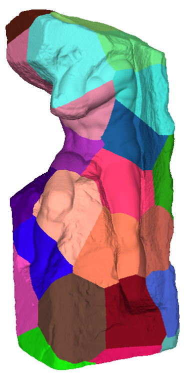
Now we describe how a huge mesh is partitioned into small patches. Our intention here is to partition a huge mesh using little memory and also at minimum computational costs by using clustered points. To resolve these demands, a *nu-Support Vector Machine* (νSVM) [SSWB00] based scheme is considered.

Basically, SVM is a method to classify unknown data into two classes. If the one-vs.-one (OvO) reduction strategy is adopted for the multi-class classification, $C(C-1)/2$ binary classifiers of SVM are required to decide a class containing a point. At the same time, classification is done by measuring the margin distance from hyperplanes by Eq. (3). Therefore, the total computation time for the partition strongly depends on the number of SVs and classes.

$$y = \sum_{k \in SV_{ij}} \alpha_k y_k K(\vec{x}_k, \vec{x}) - \rho,\; 0 \le i, j \le C, \tag{3}$$

where C is the number of classes, $\vec{x}$ is a point to be classified, $K(,)$ is a kernel function of SVM, SV_{ij} is a set of SVs for planes that separate into two classes i and j, $\{\vec{x}_k, y_k\}$ is a pair of a SV and a class id (= -1 or 1), and α, ρ are parameters computed by SVM.


(a) Linear


(b) Polynominal

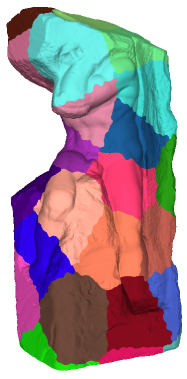

(c) RBF

Figure 6: Partitioning results by using different νSVM kernels. (a) Linear kernel. (b) Polynomial kernel. (c) RBF kernel. There is little distinct differences among the three kernels except around the intersection regions between linear and polynomial kernels. However, there are waving boundary shapes in the result for the RBF kernel.

Linear, polynomial, and RBF kernel are well-known kernel functions of SVM. A kernel function has to be selected depending on the data type and desired classification precision. As shown in Fig. 6, however, the linear kernel function is enough to partition a mesh well with our method.

When a huge mesh is partitioned, the computations described above are the primary factor which increases the total computation time. Now two approaches are employed to speed up the classification. One approach is that the classification computation by SVs with the linear kernel function is replaced with a simple inside/outside judgment of a position over a three-dimensional plane $\vec{w}^T\vec{x} + h = 0$ whose parameters $\vec{w}$ and h are obtained by,

$$\vec{w} = \sum_{k \in SV_{ij}} \alpha_k y_k \vec{x}_k, \tag{4}$$

$$h = y_l - \vec{w}\vec{x}_l, \quad l \in SV_{ij}. \tag{5}$$

Note that this speed-up scheme is available only with a linear kernel. Another approach is that a DAG-SVM [PSTC00] is adopted to reduce the computational complexity for the classification from $O(C^2)$ to $O(C)$.

Fig. 7 shows our framework of partitioning a huge mesh via νSVM. Faces of an original mesh are first loaded from a disk in an out-of-core manner as described in Sec. 4.2. Such faces are next classified in parallel by νSVM. For the classification of a face by νSVM, only its vertex positions are required. Here a barycenter of these vertices in a face is used. After the classification, faces in each class are separately stored on a disk.

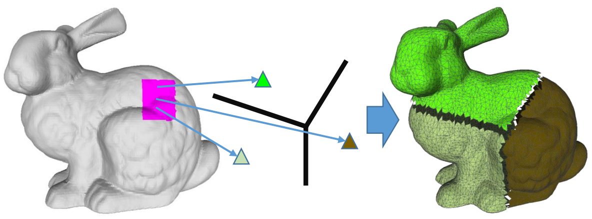


Figure 7: Our framework for partitioning huge mesh. Left: Loading a part of a huge mesh from a disk (Sec. 4.2). Middle: Classifying faces in a list by DAG-SVM classifier. Right: Classified faces are saved to a disk as separate files for patches.

4.5. Simplification Algorithm

The simplification phase is composed of three sub-phases, *expansion*, *decimation*, and *merging*. Fig. 8 shows the flowchart of the simplification phase for a huge mesh.

In the first sub-phase, the boundary of each patch is expanded towards the outside of a patch so that the 2-ring neighbors are included in the extended regions. To keep MDDs for boundary vertices (Fig. 8a), DAG-SVM classifiers constructed in Sec. 4.4 are used again to compute a margin distance from the boundary. After the expansion, we check that MDD is included in an expanded region. For a classified point $\vec{x}$, a margin distance $\Delta_{ij}(\vec{x})$ from a three-dimensional plane to such a point is also computed at the same time.

We consider here a parameter δ to determine whether $\vec{x}$ is on the expanded region or not; if $\Delta_{ij}(\vec{x}) < \delta$, $\vec{x}$ is on the expanded region. A more largely expanded region is computed for a larger δ. One issue is that the value of δ has to be determined empirically so that the region is expanded to more than 2-ring neighbor vertices. However, it is helpful to determine δ so that the number of neighborhoods from $\vec{x}$ to a border vertex can be computed using the graph connectivity of a patch. In practice, to check whether MDD is guaranteed for border vertices or not, edges connecting border vertices of the expanded region are traversed and 1 is added as the cost of each 1-ring neighbor vertex. If all costs of vertices in a region are less than 1, MDD is guaranteed. Otherwise, δ is slightly increased and the expansion sub-phase is repeated again.

The decimation sub-phase is the same process as that of ILOD; for each expanded patch, a set of LMEs is computed and edge collapse operations are executed for all LMEs (Fig. 8(b)). Both processes can be run in parallel on a CPU. After edge collapse operations are finished, mesh connectivity information is updated and the total number of faces is

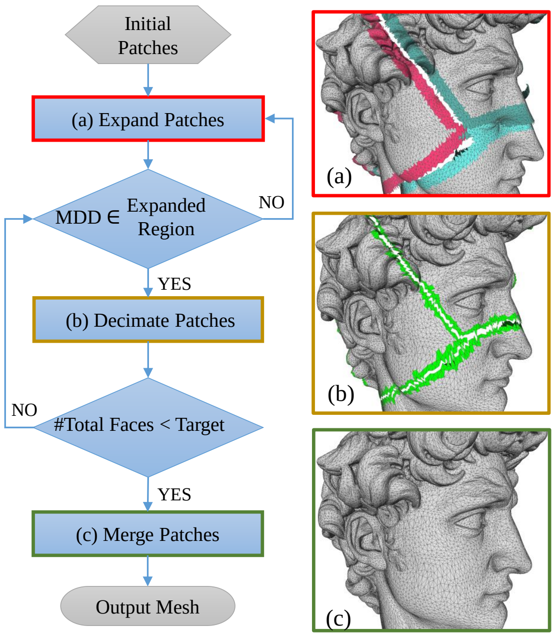


Figure 8: Workflow of simplification process. Patches described in Sec. 4.4 are used as input. (a) Expansion of patches. (b) Decimation of patches. (c) Merging to a mesh. (a) and (b) are executed for each patch individually.

checked. If the total number of faces reaches its target number, all patches are ready to be merged in the next sub-phase. Otherwise, border vertices of a simplified patch is checked whether MDD is still included. If all border vertices satisfy MDDs, the decimation sub-phase can be executed again. Otherwise, there is a need to go back to the expansion sub-phase. These processes are repeated until the target number of faces is reached. Note that our method cannot generate exactly the same number of faces as the target number, since the number of deleted edges (or faces) is determined from the number of LMEs counted by the traverse of all patches at once.

In the last sub-phase, all simplified patches are merged into an output mesh (Fig. 8(c)). In this sub-phase, the faces of expanded regions are deleted and several overlapped vertices are integrated to a vertex at the end of the algorithm.

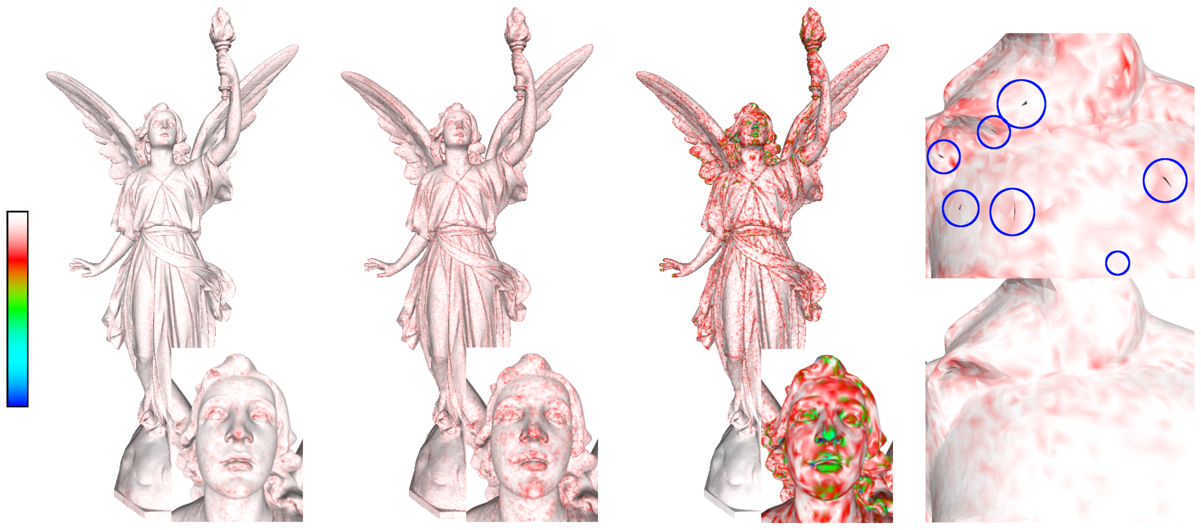
5. Results and Discussion

In this section, we discuss our experimental results to demonstrate the advantages of the proposed simplification algorithm for huge meshes. We implemented our algorithm in C++. For the νSVM in the partitioning phase, we used LIBSVM, a C++ SVM library by Chang et al. [CL11]. We conducted all of our experiments on a PC with Intel Core i7-3770 CPU, 16GB RAM and a 500GB SSD with 500MB/s R/W. In our implementation, several processes are parallelized by using a multi-core CPU. In the multi-core CPU computation, we use one of four cores for loading data from a disk to optimize the data-loading process. However, the writing process to a disk cannot be parallelized because synchronization is required.

The huge meshes used in our experiments were reconstructed from points in the Digital Michelangelo Project [LPC*00] by using the out-of-core surface reconstruction algorithm [BKBH07]. These meshes are; David (161,482,359 faces), St. Matthew (166,725,000 faces) and Atlas (197,182,000 faces). In addition, we also used smaller sized models from the Stanford 3D Scanning Repository [Sta] to compare the quality of simplified meshes with other methods. To measure the quality of simplified meshes, we used a Hausdorff distance between two models (typically the original model and its simplified model) divided by the diagonal length of the bounding box of a model. We used the Metro software [CRS98] for those measurements.

Quality Comparison to Other Methods. Fig. 9 shows the visual comparison of the simplification results for the Lucy model and shows the computation time and the Root Mean Square error (RMS) for each result. As for other methods, we compared two methods in [nex] (NXS) and in [ILGS03] (SM) using their own executables. Those executables, however, have upper limits on the number of faces. We then compared the Lucy model (28,055,742 faces) with our method. As seen from the figures, our method achieves higher accuracy than both NXS and SM obviously. There are a number of flipped faces (black dots in Fig. 9(d)) on the simplified mesh by SM, which dramatically worsen the visual quality. In addition, our method achieved less computation time than NXS. In contrast, the computation time of SM is faster than our method for partitioning a mesh into four patches.

Computation Time and Memory Usage According the Number of Clusters. In the second experiment, we discuss the changes in computation time according to the number of clusters. Fig. 10 illustrates a graph of computation times of our method for the Atlas model, a reconstructed model which has 197 million faces, at different number of clusters. This graph shows the computation time for partition and expansion, simplification, and total time. It also shows the time taken for the disk I/O process and computation on CPU. It can be seen that the most time-consuming part is the simplification phase. This is because the total number of faces in the expansion phase increases according to the number of patches. On the other hand, the costs for the partition and expansion do not radically increase. This graph shows the effect of the reduced computational complexity of partition and expansion to $O(C)$ as described in Sec. 4.4. In addition, the cost of the disk I/O process does not increase with the number of clusters, although the total number of loading files does. This is because the disk I/O process and other computations are carried out in parallel with a multi-core CPU.



(a) Ours (1.4M)
Time: 262 sec.
RMS: 0.000797

(b) SM (1.4M)
Time: 83 sec.
RMS: 0.00104

(c) NXS (1.4M)
Time: 793 sec.
RMS: 0.00193

(d) Close up view of face

Figure 9: Visual comparison of Lucy model with other methods. Color bar means the error distance from low (white) to high (blue). (a) With our method, a model was partitioned into 4 patches. (d) shows the close-up view of faces in the simplified meshes (upper: SM, bottom: our method). A simplified mesh by SM has many flipped triangles (shown in blue circles).

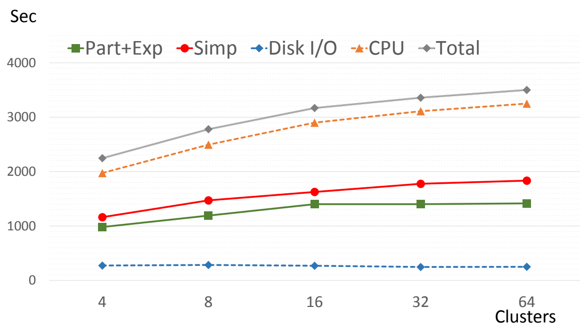


Figure 10: Computation time for Atlas model partitioned into 4 to 64 clusters. A linear kernel is used for SVM classifiers. Note that the total computation time is equal to the sum of Part. + Exp. (green) and Simp. (red), and also to the sum of Disk I/O (blue) and CPU (orange).

We next discuss the memory needed with our method. The simplification phase uses considerable memory, since the construction of graph connectivity using a half-edge data structure is required and this consumes considerable memory. In our experiments, the peak memory was approximately 10.7GB and 1.7GB when an Atlas model was simplified with the number of clusters being 4 and 64, respectively. The peak memory in 64 clusters is significantly increased than expected because more faces for each patch have to be added in the expansion phase.

On the whole, the decision of the number of clusters is a trade-off between the computation time and memory usage. Fortunately, we can predict the peak memory used by the number of clusters. To accelerate our method, the number of clusters which consumes the most amount of memory in the PC environment can be determined in advance. If we want to make the whole process faster by making full use of the PC memory, it is better to partition a mesh into as little patches as possible.

Simplification of Huge Meshes. Fig. 11 shows the simplification results of a David model, and Tab. 1 shows the computational time for three huge meshes. As shown in this table, the computation time of each model is not proportional to the number of faces. This is because the number of LMEs differs according to the mesh. Nevertheless, each mesh, which has over 100M faces, can be simplified within 2,500 seconds.

Model	Orig. (#faces)	Simp. (#faces)	Time (sec.)
David	161,482,359	1,652,283	2,060
St. Matthew	166,725,000	1,737,906	2,151
Atlas	197,182,000	1,820,499	2,233

Table 1: Computation time and number of faces (reduced to 1%, Level 1) in simplification of huge meshes.

6. Conclusions and Future Work

In this paper, we have proposed a novel out-of-core framework that can simplify huge meshes by using QEM-based

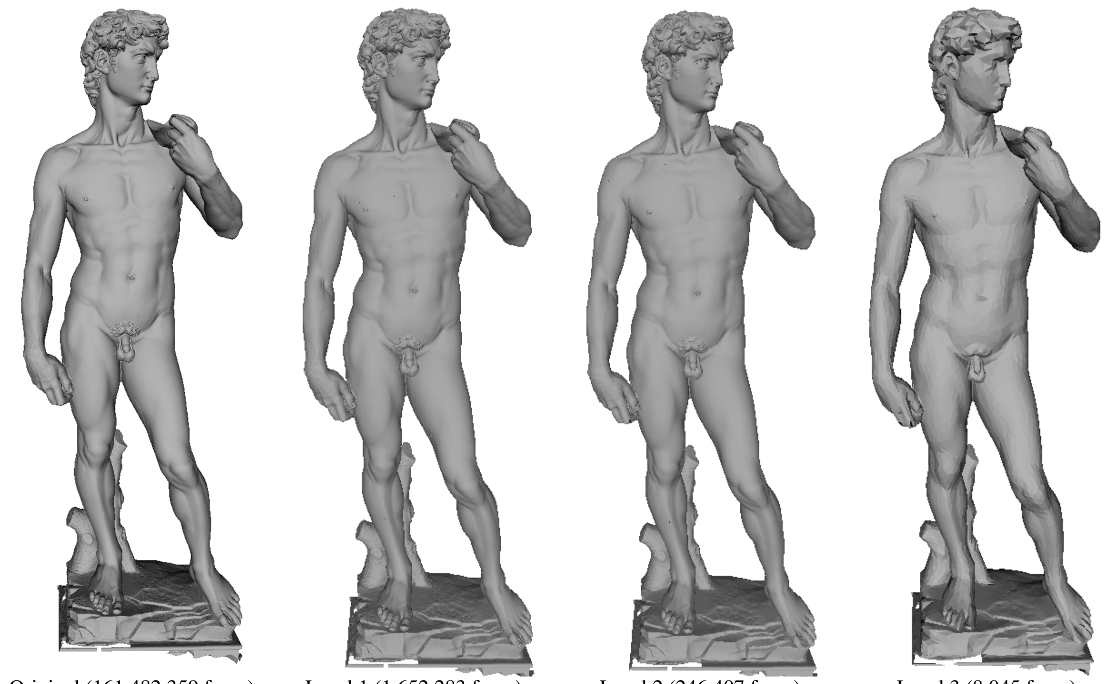


Figure 11: Simplification results for David model. The number of faces is reduced to approximately 1% (Level 1), 0.15% (Level 2) and 0.005% (Level 3) from that of the original model, respectively. The computational time for the simplification to Level 1 is shown in Tab.1.

mesh simplification. For simplifying the small sub-meshes, the notion of MDD was introduced to enable independent per-patch simplification while considering the boundary issue. In the partitioning phase, a Fuzzy C-Means method was used to cluster the sampled points and νSVM as well as DAG-SVM were used in the partitioning phase. A huge mesh can be then partitioned into a set of patches in the out-of-core framework. In summary, it can be said that our method can achieve the partition and simplification of huge meshes with high quality simplified models as those generated by in-core QEM-based methods.

Our method is capable of parallel computation of per-patch simplification process in principle. Faster computation may be possible by using multiple CPUs. In this case, there would be an additional task to effectively distribute each patch to multiple processing units.

Our method has several limitations to be resolved as future work. First, our partitioning method does not ensure that each cluster (or patch) has the same number of points (or faces). To partition a graph into a set of sub-graphs with each graph having almost the same number of nodes, graph partitioning methods such as METIS [KK95] are applicable. In this case, a graph has to be constructed from sampled points while keeping the topology of the original mesh.

Secondly, the different number of LMEs between patches has a significant effect on the computation time of the simplification sub phase. Hence, the computational performance could be further improved by parallel computation of per-patch simplification if a mesh is partitioned into patches with the same number of LMEs. To address this presumption, we will investigate the relationship between mesh geometry and the number of LMEs.

Acknowledgements

We would like to thank Tatsuya Yatagawa and Hideki Todo for providing many valuable comments for this paper. We would also like to thank Peter Lindstrom, Martin Isenburg, and Federico Ponchio for providing their executables for the evaluation of our research. Data used in this paper are courtesy of the Digital Michelangelo Project and the Stanford 3D Scanning Repository.

References

[BKBH07] BOLITHO M., KAZHDAN M., BURNS R., HOPPE H.: Multilevel streaming for out-of-core surface reconstruction. In *Proc. Fifth Eurographics Symposium on Geometry Processing (SGP '07)* (2007), Eurographics Association, Aire-la-Ville, Switzerland, pp. 69–78. 7

[BKP*10] BOTSCH M., KOBBELT L., PAULY M., ALLIEZ P., LEVY B.: *Polygon Mesh Processing*. AK Peters, 2010. 2

[BLVD11] BENHABILES H., LAVOUÉ G., VANDEBORRE J.-P., DAOUDI M.: Learning boundary edges for 3D-mesh segmentation. *Computer Graphics Forum 30*, 8 (2011), 2170–2182. 2

[BP02] BRODSKY D., PEDERSEN J. B.: Parallel model simplification of very large polygonal meshes. In *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '02)* (2002), vol. 3, CSREA Press, Athens, GA, pp. 1207–1215. 2

[BWWM10] BOWERS J., WANG R., WEI L.-Y., MALETZ D.: Parallel poisson disk sampling with spectrum analysis on surfaces. *ACM Transaction on Graphics 29*, 6 (Dec. 2010), 166:1–166:10. 4

[CL11] CHANG C.-C., LIN C.-J.: LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology 2*, 3 (May 2011), 27:1–27:27. 7

[CMRS03] CIGNONI P., MONTANI C., ROCCHINI C., SCOPIGNO R.: External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics 9*, 4 (Oct 2003), 525–537. 2

[CRS98] CIGNONI P., ROCCHINI C., SCOPIGNO R.: Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (1998), 167–174. 7

[DC10] DU Z., CHIANG Y.-J.: Out-of-core simplification and crack-free LOD volume rendering for irregular grids. *Computer Graphics Forum 29*, 3 (2010), 873–882. 2

[GDG11] GRUND N., DERZAPF E., GUTHE M.: Instant level-of-detail. In *Proc. 16th International Fall Workshop Vision, Modeling and Visualisation (VMV 2011)* (2011), Eurographics Association, Aire-la-Ville, Switzerland, pp. 293–299. 1, 2, 3

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proc. SIGGRAPH '97* (1997), ACM Press, New York, NY, pp. 209–216. 1, 2, 3

[HLS07] HORMANN K., LÉVY B., SHEFFER A.: Mesh parameterization: Theory and practice. In *ACM SIGGRAPH 2007 Courses* (2007), ACM Press, New York, NY. 2

[Hop99] HOPPE H.: New quadric metric for simplifiying meshes with appearance attributes. In *Proc. 10th IEEE Visualization (VIS '99)* (1999), IEEE CS Press, Los Alamitos, CA, pp. 59–66. 2

[IL05] ISENBURG M., LINDSTROM P.: Streaming meshes. In *Proc. 16th IEEE Visualization (VIS 2005)* (2005), IEEE CS Press, Los Alamitos, CA, pp. 231–238. 2

[ILGS03] ISENBURG M., LINDSTROM P., GUMHOLD S., SNOEYINK J.: Large mesh simplification using processing sequences. In *Proc. 14th IEEE Visualization (VIS 2003)* (2003), IEEE CS Press, Los Alamitos, CA, pp. 465–472. 2, 4, 7

[JKS05] JULIUS D., KRAEVOY V., SHEFFER A.: D-charts: Quasi-developable mesh segmentation. *Computer Graphics Forum 24*, 3 (2005), 581–590. 2

[KHS10] KALOGERAKIS E., HERTZMANN A., SINGH K.: Learning 3D mesh segmentation and labeling. *ACM Transaction on Graphics 29*, 4 (July 2010), 102:1–102:12. 2

[KK95] KARYPIS G., KUMAR V.: METIS – unstructured graph partitioning and sparse matrix ordering system, version 2.0, 1995. 9

[Lin00] LINDSTROM P.: Out-of-core simplification of large polygonal models. In *Proc. SIGGRAPH '00* (2000), ACM Press, New York, NY, pp. 259–262. 2

[LPC*00] LEVOY M., PULLI K., CURLESS B., RUSINKIEWICZ S., KOLLER D., PEREIRA L., GINZTON M., ANDERSON S., DAVIS J., GINSBERG J., SHADE J., FULK D.: The digital michelangelo project: 3D scanning of large statues. In *Proc. SIGGRAPH '00* (2000), ACM Press, New York, NY, pp. 131–144. 7

[LS01] LINDSTROM P., SILVA C. T.: A memory insensitive technique for large model simplification. In *Proc. 12th IEEE Visualization (VIS 2001)* (2001), IEEE CS Press, Los Alamitos, CA, pp. 121–126. 2

[LT99] LINDSTROM P., TURK G.: Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (Apr. 1999), 98–115. 2

[Lue01] LUEBKE D. P.: A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications 21*, 3 (2001), 24–35. 2

[LZ04] LIU R., ZHANG H.: Segmentation of 3D meshes through spectral clustering. In *Proc. 12th Pacific Conference on Computer Graphics and Applications (PG 2004)* (2004), IEEE CS Press, Los Alamitos, CA, pp. 298–305. 2

[nex] NEXUS. `http://vcg.isti.cnr.it/nexus/contacts.php`. 7

[PSTC00] PLATT J. C., SHAWE-TAYLOR J., CRISTIANINI N.: Large margin DAGs for multiclass classification. In *Advances in Neural Information Processing Systems 12* (2000), pp. 547–553. 6

[RB93] ROSSIGNAC J., BORREL P.: Multi-resolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics*, Falcidieno B., Kunii T., (Eds.), IFIP Series on Computer Graphics. Springer Berlin Heidelberg, 1993, pp. 455–465. 2

[SG01] SHAFFER E., GARLAND M.: Efficient adaptive simplification of massive meshes. In *Proc. 12th IEEE Visualization (VIS 2001)* (2001), IEEE CS Press, Los Alamitos, CA, pp. 127–134. 2

[SG05] SHAFFER E., GARLAND M.: A multiresolution representation for massive meshes. *IEEE Transactions on Visualization and Computer Graphics 11*, 2 (March 2005), 139–148. 2

[SGW06] SCHMIDT R., GRIMM C., WYVILL B.: Interactive decal compositing with discrete exponential maps. *ACM Transaction on Graphics 25*, 3 (July 2006), 605–613. 5

[SN13] SHONTZ S. M., NISTOR D. M.: Cpu-gpu algorithms for triangular surface mesh simplification. In *Proc. 21st International Meshing Roundtable* (2013), Springer, Berlin, Germany, pp. 475–492. 2

[SPR06] SHEFFER A., PRAUN E., ROSE K.: Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision 2*, 2 (Jan. 2006), 105–171. 2

[SSWB00] SCHÖLKOPF B., SMOLA A. J., WILLIAMSON R. C., BARTLETT P. L.: New support vector algorithms. *Neural Comput. 12*, 5 (May 2000), 1207–1245. 5

[Sta] The stanford 3D scanning repository. `http://graphics.stanford.edu/data/3Dscanrep/`. 7

[SW03] SCHAEFER S., WARREN J.: Adaptive vertex clustering using octrees. In *Proc. 8th SIAM Geometric Design and Computing* (2003), Nashboro Press, Brentwood, TN, pp. 491–500. 2

[XLXG11] XIAO D., LIN H., XIAN C., GAO S.: CAD mesh model segmentation by clustering. *Computers & Graphics 35*, 3 (2011), 685–691. 2

[ZSGS04] ZHOU K., SYNDER J., GUO B., SHUM H.-Y.: Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In *Proc. Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP '04)* (2004), ACM Press, New York, NY, pp. 45–54. 2