Computers & Graphics 51 (2015) 81–89

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

SMI 2015

Large mesh simplification for distributed environments

Daniela Cabiddu *, Marco Attene

CNR–IMATI Genova, Italy

ARTICLE INFO

Article history:
Received 12 March 2015
Received in revised form
12 May 2015
Accepted 13 May 2015
Available online 1 June 2015

Keywords:
Indexed mesh
Out-of-core
Big data
Parallel algorithm

ABSTRACT

An algorithm is described to simplify arbitrarily large triangle meshes while leveraging the computing power of modern distributed environments. Our method combines the flexibility of out-of-core (OOC) techniques with the quality of accurate in-core algorithms, while representing a particularly fast approach thanks to the concurrent use of several computers in a network. When compared with existing parallel algorithms, the simplifications produced by our method exhibit a significantly higher accuracy. Furthermore, when our algorithm is run on a single machine, its speed is comparable with state-of-the-art OOC techniques, whereas the use of more machines enables relevant speedups. Noticeably, we observe that the speedup increases as the size of the input mesh grows.

1. Introduction

In the last few decades, the evolution of 3D acquisition technologies called for methods to simplify meshes that have become large and larger. Earlier simplification algorithms [1,2] could focus on efficiency and accuracy only, and today we know that methods based on iterative edge collapses driven by quadric error metrics are both efficient and, under certain conditions, provably optimal [3]. Soon, however, too large meshes appeared that could not fit in main memory, and existing algorithms needed to be redesigned to account for an appropriate out-of-core elaboration. In most of these methods the mesh is partitioned in several sub-meshes, each small enough to be processed with traditional algorithms. In some cases the mesh can be partitioned using an in-core algorithm: this is appropriate when memory is enough to store the mesh, but no further space is available to host all the support data structures necessary for the simplification (e.g. quadric matrices and priority queues) which are often more memory-demanding than the mesh itself. Conversely, when even the plain mesh is too large, out-of-core partitioning is required to produce the sub-meshes.

After having simplified each of the sub-meshes separately, these partial objects can be merged back into a single mesh. If the resulting simplifications are comprehensively small enough to fit in memory, in-core methods can be exploited to merge and polish the final result. Some algorithms, for example, avoid to simplify the sub-mesh boundaries to guarantee an exact match after the simplification. These contact regions are then unified in an in-core merging phase and, if necessary, their neighborhood is simplified using traditional algorithms. When the resulting simplification is still too large for in-core post-processing, contact borders can either be kept unsimplified or treated using a local, though sub-optimal, approach. For example, vertex clustering can be used to simplify sub-meshes so that their eventual boundaries are guaranteed to match, but this has a cost in terms of quality, especially when simplifying meshes with fair morphological variations (i.e. large flat areas mixed with feature-rich areas).

Furthermore, it is important to consider that the most diffused formats employ an indexed mesh representation: a first block in the file represents the vertex coordinates, whereas a second block represents each triangle as a triplet of indexes referred to the first block. Thus, while partitioning the vertices is relatively easy, indexes must be dereferenced to partition triangles, and the latter is not a trivial operation when memory is not sufficient to host all the vertex coordinates. For this reason, most existing methods assume that the input mesh is represented as a list of triangles, each directly encoded by the coordinates of its three vertices. Note that the high redundancy of such “triangle soups” represents a severe limitation when even an indexed mesh requires giga or even terabytes of disk space [4,5].

A last important aspect that deserves attention is the efficiency. Even if multi-core architectures can be exploited to accelerate the process, the memory available on a single machine has a limit that imposes a sequentialization in any case. Conversely, distributed computer networks give access to virtually infinite resources while still enabling concurrent processing. However, typical multi-core methods communicate based on a fast-access shared memory [6] which is not available in a standard computer network. Hence, these methods are not suitable and innovative solutions are required.

* Corresponding author.
E-mail addresses: daniela.cabiddu@ge.imati.cnr.it (D. Cabiddu), marco.attene@ge.imati.cnr.it (M. Attene).

http://dx.doi.org/10.1016/j.cag.2015.05.015
0097-8493/

To summarize, the following aspects should be taken into account when designing a large mesh simplification algorithm: (1) Out-of-core initial partitioning; (2) Out-of-core final merging/post-processing; (3) High quality of the simplification/ adaptivity; (4) Treatment of indexed meshes; (5) Efficiency and distributable load.

Herewith, we propose an original algorithm that satisfies all the aforementioned requirements while being comparable with, or even outperforming, less flexible state-of-the-art approaches. To the best of our knowledge, no existing method encapsulates all these characteristics.

2. Related work

In parallel algorithms [7–11] a "master" processor partitions the input mesh and distributes the portions across different "slave" processors that perform the partial simplifications simultaneously. When all the portions are ready, the master merges the results together. The many slave processors available in modern GPU-based architectures are exploited in [12]. In these methods the main goal is to speedup the process and, with the exception of [7], the typical approach to partition the input exploits in-core algorithms.

Besides [7], other effective out-of-core partitioning techniques are described in [13,14]. These methods typically require their input to come as a triangle soup. When the input is represented using an indexed format, it must be dereferenced using out-of-core techniques [15], but this additional step is time-consuming and requires significant storage resources. As an exception, the method proposed in [16] is able to work with indexed representations by relying on memory-mapped I/O managed by the operating system; however, if the face set is described without locality in the file, the same information is repeatedly read from disk and thrashing is likely to occur. Instead of partitioning the input into fixed portions, processing sequences are used in [17] and approaches based on streaming simplification are proposed in [18,19]. These approaches are very elegant and do not need to deal with boundary coherence. Even in these cases, however, conversion to appropriate processing sequences and mesh pre-sorting operations are non-trivial processes that require a significant time [20].

In [13] the vertex clustering approach by Rossignac and Borrel [21] is modified to use a quadric error metric to compute the representative vertex. With respect to [21], this choice improves the quality of the resulting mesh, and the use of a clustering-based simplification guarantees that adjacent mesh portions have coherent common boundaries. The adaptive clustering employed in [16] leads to an even higher quality result, whereas in [22] boundaries are kept consistent at each iteration thanks to a smart octree-based external memory data structure. Both [13,16] solve the problem of boundary coherence, but the quality of their simplifications is still not comparable with traditional methods based on global priority queues [1]. On the other hand, Cignoni et al. [22] provide high quality simplifications, but the approach is not suitable for a distributed setting.

The possibility to exploit distributed environments is scarcely treated in the literature. In [7] this possibility is considered but, due to the use of a distributed shared memory, the approach proposed is appropriate only on high-end clusters where local nodes are interconnected with particularly fast protocols. To the best of our knowledge, the only existing technique that can operate without any shared memory is described in [11], but out-of-core partitioning is not supported.

3. Distributed simplification

In our reference scenario a number of consumer PCs are connected within a standard lab network, and a very large mesh M represented as an OFF file is stored on the local disk of one of these machines. In analogy with previous work on parallel processing, we call this latter machine the "master", whereas all the other PCs are "slaves". To simplify M, our method considers a target accuracy, the master's available memory, the number N_s of available slaves, and the available memory on each of the slaves. For the sake of simplicity, our exposition assumes that all the slaves have an equally-sized memory and a comparable speed.

Fig. 1 shows how the distributed simplification algorithm works. In the first step, the master partitions the mesh into a collection of submeshes using an out-of-core algorithm. Submeshes are then grouped into independent sets (ISs), so that submeshes in each IS are fully disjoint (i.e. they do not share any element) (Fig. 1b). Each IS is guaranteed to contain at most N_s submeshes to be simultaneously sent to the slaves for simplification. In the first iteration, each submesh is simplified in all its parts according to the target accuracy (Fig. 1c). Besides the simplified mesh, each slave also produces an additional file identifying which vertices on the submesh boundary were removed during simplification. This information is appended to adjacent submeshes and used as a constraint during their own simplification (Fig. 1d–e). When all the ISs have been processed, the master employs an out-of-core algorithm to join the simplified submeshes along their boundaries, which are guaranteed to match exactly (Fig. 1f).

3.1. Input partitioning

The input to our partitioner is an indexed mesh $M = \langle V, T \rangle$, where V is a list of vertices and T is a set of triangles. Each vertex v_i in V is encoded by its three coordinates, whereas each triangle t_i in T is encoded by three integer indexes: an index k identifies the kth vertex in the list V. An analogous encoding is used to describe each output submesh $M_i = \langle V_i, T_i \rangle$. In the remainder, we distinguish between local indexes and global indexes: a local index k in a submesh $M_i = \langle V_i, T_i \rangle$ identifies the kth vertex in the list V_i, whereas a global index j identifies the jth vertex in the overall V.

Our solution requires two integer parameters: the number of vertices N_v that we wish to assign to each submesh (based on the

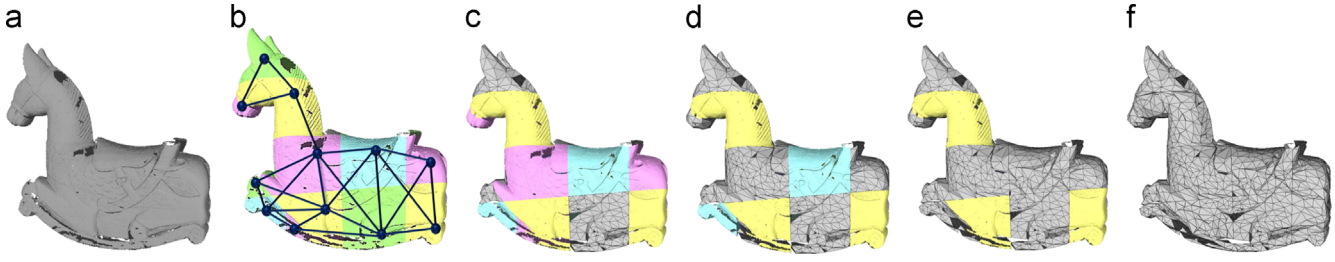

Fig. 1. Distributed simplification. (a) Input mesh. (b) Independent sets of submeshes. (c)–(e) Simplification steps. (f) The final output.