Computers & Graphics 33 (2009) 542–553

Contents lists available at ScienceDirect

Computers & Graphics

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

Technical Section

CHuMI viewer: Compressive huge mesh interactive viewer

Clément Jamin [a,b,*], Pierre-Marie Gandoin [a,c], Samir Akkouche [a,b]

[a] Université de Lyon, CNRS
[b] Université Lyon 1, LIRIS, UMR5205, F-69622, France
[c] Université Lyon 2, LIRIS, UMR5205, F-69676, France

ARTICLE INFO

Article history:
Received 21 December 2008
Received in revised form
21 March 2009
Accepted 28 March 2009

Keywords:
Lossless compression
Interactive visualization
Large meshes
Out-of-core

ABSTRACT

The preprocessing of large meshes to provide and optimize interactive visualization implies a complete reorganization that often introduces significant data growth. This is detrimental to storage and network transmission, but in the near future could also affect the efficiency of the visualization process itself, because of the increasing gap between computing times and external access times. In this article, we attempt to reconcile lossless compression and visualization by proposing a data structure that radically reduces the size of the object while supporting a fast interactive navigation based on a viewing distance criterion. In addition to this double capability, this method works out-of-core and can handle meshes containing several hundred million vertices. Furthermore, it presents the advantage of dealing with any *n*-dimensional simplicial complex, including triangle soups or volumetric meshes, and provides a significant rate-distortion improvement. The performance attained is near state-of-the-art in terms of the compression ratio as well as the visualization frame rates, offering a unique combination that can be useful in numerous applications.

1. Introduction

Mesh compression and mesh visualization are two fields of computer graphics that are particularly active today, whose constraints and goals are usually incompatible, even contradictory. The reduction of redundancy often goes through signal complexification, because of prediction mechanisms whose efficiency is directly related to the depth of the analysis. This additional logical layer inevitably slows down the data access and creates conflicts with the speed requirements of real-time visualization. Conversely, for efficient navigation through a mesh integrating the user's interactions dynamically, the signal must be carefully prepared and hierarchically structured. This generally introduces a high level of redundancy and sometimes comes with data loss, if the original vertices and polyhedra are approximated by simpler geometrical primitives. In addition to the on-disk storage and network transmission issues, the data growth implied by this kind of preprocessing could become detrimental to the visualization itself since the gap between the processing times and the access times from external memory is increasing.

In this article, we attempt to reconcile compression and interactive visualization by proposing a method that combines good performance in terms of both the compression ratio and the visualization frame rates. The interaction requirements of a viewer and a virtual reality system are quite similar in terms of the computational challenge but the two applications use different criteria to decide how a model is viewed and updated. Our goal here is to take advantage of excellent view independent compression and introduce distance-dependent updates of the model during the visualization. As a starting point, the in-core progressive and lossless compression algorithm introduced by Gandoin and Devillers [1] has been chosen. On top of its competitive compression ratios, out-of-core and LOD capabilities have been added to handle meshes with no size limitations and allow local refinements on demand by loading the necessary and sufficient data for an accurate real-time rendering of any subset of the mesh. To meet these goals, the basic idea consists in subdividing the original object into a tree of independent meshes. This partitioning is undertaken by introducing a primary hierarchical structure (an *n*SP-tree) in which the original data structures (a kd-tree coupled to a simplicial complex) are embedded, in a way that optimizes the bit distribution between geometry and connectivity and removes the undesirable block effects of kd-tree approaches.

After a study of related works (Section 2) focusing on the method chosen as the starting point (Section 3), our contribution is introduced by an example (Section 4), then detailed in two complementary sections. First, the algorithms and data structures of the out-of-core construction and compression are introduced (Sections 5 and 6), then the visualization point of view is adopted

* Corresponding author Tel.: +33 472448064.
E-mail addresses: clement.jamin@liris.cnrs.fr (C. Jamin), pierre-marie.gandoin@liris.cnrs.fr (P.-M. Gandoin), samir.akkouche@liris.cnrs.fr (S. Akkouche).

0097-8493/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2009.03.029

to complete the description (Section 7).Finally, experimental results are presented and the method is compared to prior art (Section 8).

2. Previous works

2.1. Compression

Mesh compression is a domain situated between computational geometry and standard data compression. It consists in efficiently coupling geometry encoding (the vertex positions) and connectivity encoding (the relations between vertices), and often addresses manifold triangular surface models. We distinguish single-rate algorithms, which require full decoding to visualize the object, and multiresolution methods that allow one to see the model progressively refined while decoding. Although historically geometric compression began with single-rate algorithms, we have chosen not to detail these methods here. The reader can refer to surveys [2,3] for more information. For comparison purposes, since progressive and out-of-core compression methods are very rare, Section 8 will refer to Isenburg and Gumhold's well-known single-rate out-of-core algorithm [4]. Similarly, lossy compression, whose general principle consists in a frequential analysis of the mesh, is excluded from this study.

Progressive compression is based on the notion of refinement. At any time of the decoding process, it is possible to obtain a global approximation of the original model, which can be useful for large meshes or for network transmission. This research field has been very productive for approximately 10 years, and rather than being exhaustive, here the choice has been to adopt a historical point of view. Early techniques of progressive visualization, based on mesh simplification, were not compression-oriented and often induced a significant increase in the file size, due to the additional storing cost of a hierarchical structure [5,6]. Afterward, several single-resolution methods were extended to progressive compression. For example, Taubin et al. [7] proposed a progressive encoding based on Taubin and Rossignac's algorithm [8]. Cohen-Or et al. [9] used techniques of sequential simplification by vertex suppression for connectivity, combined with position prediction for geometry. Alliez and Desbrun [10] proposed an algorithm based on progressive removal of independent vertices, with a retriangulation step under the constraint of maintaining the vertex degrees around 6. Contrary to the majority of compression methods, Gandoin and Devillers [1] gave the priority to geometry encoding. Their algorithm, detailed in Section 3, gives competitive compression rates and can handle simplicial complexes in any dimension, from manifold regular meshes to triangle soups. Peng and Kuo [11] took this paper as a basis to improve the compression ratios using efficient prediction schemes (sizes decrease by roughly 15%), still limiting the scope to triangular models.

2.2. Large meshes processing

Working with huge data sets implies to develop out-of-core solutions for managing, compressing or editing meshes. Lindstrom and Silva [12] proposed a simplication algorithm based on [13] whose memory complexity is neither dependent on the size of the input nor on the size of the output. Cignoni et al. [14] introduced a data structure called Octree-based External Memory Mesh (OEMM), which enable external memory management and simplification of very large triangle meshes, by loading only selected sections in main memory. Isenburg et al. [15] developed a streaming file format for polygon meshes designed to work with large data sets. Isenburg et al. [16] proposed a single-rate streaming compression scheme that encodes and decodes arbitrary large meshes using only minimal memory resources. Cai et al. [17] introduced the first progressive compression method adapted to very large meshes, offering a way to add out-of-core capability to most of the existing progressive algorithms based on octrees. The next section pays particular attention to out-of-core visualization techniques.

2.3. Visualization

Fast and interactive visualization of large meshes is a very active research field. Generally, a tree or a graph is built to handle a hierarchical structure which makes it possible to design out-of-core algorithms: at any moment, only necessary and sufficient data are loaded into memory to render the mesh. Level of detail, view frustum and occlusion culling are widely used to adapt displayed data to the viewport. Rusinkiewicz and Levoy [18] introduced QSplat, the first out-of-core point-based rendering system, where the points are spread in a hierarchical structure of bounding spheres. This structure can easily handle levels of detail and is well-suited to visibility and occlusion tests. Several million points per second can thus be displayed using adaptive rendering. El-Sana and Chiang [19] proposed a technique to segment triangle meshes into view-dependence trees, allowing external memory simplification of models containing a few millions polygons, while preserving an optimal edge collapse order to enhance the image quality. Later, Lindstrom [20] developed a method for interactive visualization of huge meshes. An octree is used to dispatch the triangles into clusters and to build a multiresolution hierarchy. A quadric error metric is used to choose the representative point positions for each level of detail, and the refinement is guided by visibility and screen space error. Yoon et al. [21] proposed a similar algorithm with a bounded memory footprint: a cluster hierarchy is built, each cluster containing a progressive submesh to smooth the transition between the levels of detail. Cignoni et al. [22] used a hierarchy based on the recursive subdivision of tetrahedra in order to partition space and guarantee varying borders between clusters during refinement. The initial construction phase is parallelizable, and GPU is efficiently used to improve frame rates. Gobbetti and Marton [23] introduced the *far voxels*, capable of rendering regular meshes as well as triangles soups. The principle is to transform volumetric subparts of the model into compact direction-dependent approximations of their appearance when viewed from a distance. A BSP tree is built, and nodes are discretized into cubic voxels containing these approximations. Again, the GPU is widely used to lighten the CPU load and improve performance. Cignoni et al. [24] proposed a general formalized framework that encompasses all these visualization methods based on batched rendering. Recently, Hu et al. [25] introduced the first highly parallel scheme for view-dependent visualization that is implemented entirely on programmable GPU. Although it uses a static data structure requiring 57% more memory than an index triangle list, it allows real-time exploration of huge meshes.

2.4. Combined compression and visualization

Progressive compression methods are now mature (the rates obtained are close to theoretical bounds) and interactive visualization of huge meshes has been possible for several years. However, even if the combination of compression and visualization is often mentioned as a perspective, very few papers deal with this problem, and the files created by visualization algorithms are often much larger than the original ones. In fact, compression favors a small file size to the detriment of fast data