INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11
MPEG96/3057
February 1998

Source:
Status: proposal
Title: Multi-Triangulations for Managing the Level-of-Detail of Polygonal Surfaces
Author: Leila De Floriani, Paola Magillo, Enrico Puppo

Multi-Triangulations for Managing the Level-of-Detail of Polygonal Surfaces: Selective Refinement, Locality, Progressive Transmission, Dynamic Update, and Geometry Compression

Leila De Floriani, Paola Magillo
Dipartimento di Informatica e Scienze dell'Informazione
Universit^ di Genova
Via Dodecaneso, 35 -- 16146 Genova, ITALY
Email: {deflo,magillo}@disi.unige.it

Enrico Puppo
Istituto per la Matematica Applicata, C. N. R.
Via De Marini, 6 - 16149 Genova - Italy
tel.: x-39-10-6475 1; fax: x-39-10-6475 660
Email: puppo@ima.ge.cnr.it

1. Introduction

Mesh simplification is a popular approach to the lossy compression of polygonal surfaces, which is based on the reduction of the number of elements (vertices and faces) used to represent a mesh. Mesh simplification induces some reduction of accuracy in representing the surface. The amount of loss of accuracy is generally referred to as Level-of-Detail (LOD). Many algorithms for mesh simplification have been proposed in the literature. See [Hec97,Pup97] for surveys.

Mesh simplification is a slow operation that cannot be performed in real time in general. Hence, either simplified models are produced and stored off-line (e.g., in VRML LOD nodes), or data are pre-processed, and organized in a framework able to do simplification on-line. The latter solution is especially suitable to optimize performance when the LOD required by an application may be variable over different parts of the surface (e.g., flight simulation, virtual worlds).

The Multi-Triangulation (MT) is a general framework for representing and managing triangular surface meshes at variable level of detail. Concepts and theoretical properties about the MT have been introduced in [Pup96]; algorithms for building an MT from a generic surface mesh are described in [DeF97]; applications of the MT to the management of LOD has been demonstrated in [DeF97,DeF98a]. This document summarizes briefly the characteristics of this model, by emphasizing its application as a mesh compression tool.

A Multi-Triangulation is obtained by organizing surface patches produced by a mesh simplification algorithm in the context of a DAG, which provides a hierarchy of local Levels-of-Detail. A mesh can be extracted in real time from the MT, possibly restricted to a region of interest, and having an arbitrary LOD, which may be smoothly variable through space. Mesh extraction is performed by simple algorithms that essentially manipulate cuts in the DAG.

2. LOD management

We assume the architecture depicted in Figure 1. A server manages a database of objects (possibly a scene in a virtual world). The server provides methods for accessing surface meshes upon request. A client is connected to the server through a communication channel. The client sends requests, and receives meshes through the channel. In this scenario, a primary goal is to maximize performance by tansmitting, and loading into the client as less information as possible, while satisfying the requirements of the running application.

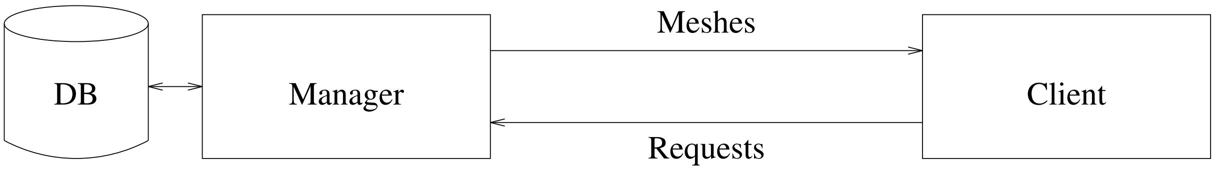

Figure 1: A client-server architecture

We face this problem by exploiting features of the MT framework, which allow us not only to reduce the size of data stored and transmitted, but also to maximize the quality of transmitted meshes. This latter goal is achieved by tuning the loss of accuracy introduced by compression (simplification) according to the needs of the running application.

Therefore, a server stores a surface encoded by an MT, and provides on-line a mesh representing such a surface according to the requests of the client. In particular, the server can offer the following features:

- *Selective refinement.* A mesh can be obtained in real time, whose LOD is variable through space. The client must provide a threshold function, i.e., a mathematical law specifying how the loss of accuracy/resolution of the mesh should be distributed through space. For instance, if the running application is a flight simulator, the LOD of the observed objects will be decreasing with distance from the viewpoint; other applications might need to enhance accuracy in the proximity of regions of interest. The mesh obtained in this way is minimal, i.e., it is formed by the smallest number of triangles in the MT that can satisfy the threshold.
- *Locality*. The mesh can be bounded to a local Region of Interest (ROI), specified by the client. For instance, in a flight simulator, the client just needs the portion of landscape inside the view frustum. This not only permits to reduce the amount of information transmitted through the channel, and loaded by the client, but it also speeds up operations performed by the server to retrieve the mesh.
- *Progressive transmission*. The simplified mesh, possibly clipped to a ROI, can be transmitted by increasingly finer levels of detail. This allows the client to interrupt transfer before completion - e.g. because of space/time constraints - while getting a meaningful result, though at a lower LOD than desired. This also allows the server to put extraction and transmission of the mesh in a pipeline (i.e., some data are transmitted through the channel while further data are retrieved from the MT).
- *Dynamic update*. The mesh can be changed dynamically as the client changes its threshold and ROI. To this aim, only information necessary to update the mesh currently loaded by the client must be transmitted. In practice, only a very small amount of information is

needed for slight changes in the client threshold and ROI (e.g., frame-to-frame transition in the navigation of virtual worlds).

- *Geometry compression*. Depending on the simplification algorithm used to build the MT, different techniques can be adopted to compress meshes further, both in storing the MT on the server, and in trasmitting a simplified mesh to the client. It is possible to trade off storage and performance to different degrees, in order to obtain an optimal balance between the computing power of the server, and the rest of the architecture.

3. Related work

Selective refinement of meshes has been addressed by several authors using hierarchical structures based on either Progressive Meshes [Tau98,Hop97,Xia97], or Delaunay triangulations [Bro96,Cig97,deB95]. The MT is indeed a unifying framework for all such models, which is independent on the construction technique, and offers simple and efficient accessing methods based on the manipulation of cuts in a DAG. For a thorough discussion of MTs, and their relations with other models see [Pup97].

Static compression, and progressive transmission of a whole mesh at a constant LOD are addressed by simpler and more compact structures, like Progressive Meshes with linear encoding [Hop96]. Such structures, however, do not support directly selective refinement, locality, and dynamic update, hence they do not permit to drive progressive transmission from the client/application.

4. The Multi-Triangulation

The Multi-Triangulation (MT) is based on the intuitive idea to arrange a collection of local updates, aimed at progressively refining an initial coarse mesh, into a DAG that describes their interdependency.

A local update is an operation that replaces a sub-mesh Ti* of a mesh T with another mesh Ti having the same boundary of Ti*, such that the result T' = T - Ti* + Ti is still a consistent triangle mesh. We always consider refinement updates, i.e., updates where Ti has more triangles, and gives a more accurate representation of the surface than Ti*. In general, we implicitly assume that both Ti*, and Ti are formed by a small number of triangles. Local updates are the basis of many popular algorithms for mesh simplification. Vertex split (obtained by reversing edge collapse), and vertex insertion (obtained by reversing vertex decimation) are common examples. See Figure 2.

A Multi-Triangulation is a DAG whose nodes represent local updates. Its arcs are labeled by triangle sets, containing exactly one node with no incoming arcs - called the root, and one node with no outgoing arcs - called the drain. The root represents the whole surface, at a low level of detail, with a highly simplified mesh. The drain represents the surface, at the highest level of detail, with the most refined mesh available. Each internal node of the DAG represents a local modification of a mesh that replaces triangles labeling its incoming arcs with triangles labeling its outgoing arcs, or vice-versa. Figure 3a depicts a sequence of local updates that generates the MT shown in Figure 3b.

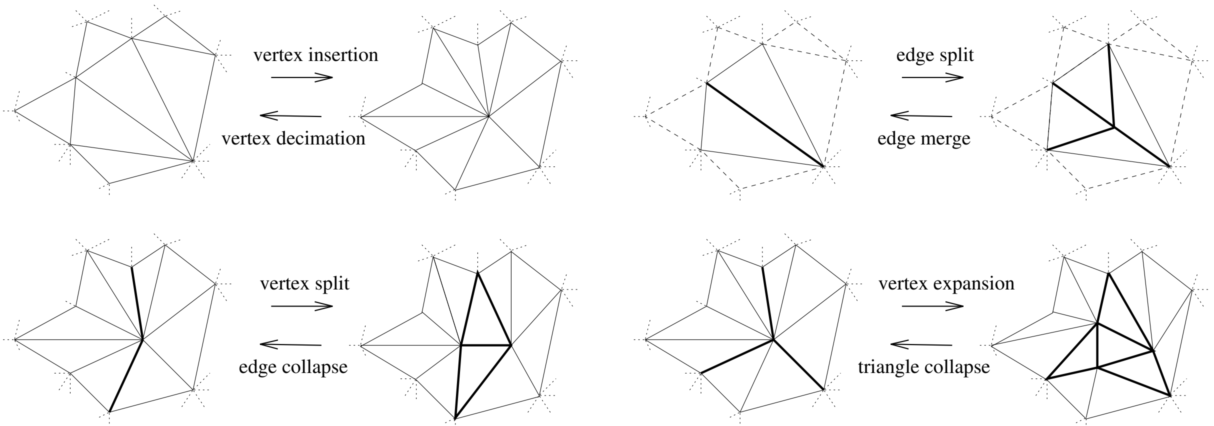

Figure 2: Some typical examples of local modifications: refinements from left to right, simplifications from right to left.

A cut of an MT is a set of arcs containing exactly one arc on each root-to-drain path. The collection of triangles labeling a cut gives a mesh representing the surface at some level of detail, possibly variable through space. High LOD is achieved where the cut lies closer to the drain, while low LOD (and low complexity) is obtained where the cut lies closer to the root. Since any cut gives a meaningful mesh, the LOD can be adjusted upon request to meet different requirements in different parts of the surface. Moving a cut beyond a node corresponds to a local refinement of the mesh (i.e. more vertices and triangles, better LOD); moving a cut before a node corresponds to a local simplification (i.e., less vertices and triangles, worse LOD). Therefore, the accuracy and size of a mesh increases/decreases while sweeping a cut forward/backward through the DAG. The mesh corresponding to the cut of Figure 3b is depicted in Figure 3c.

It has been shown in [DeF96,Pup97] that the MT framework encompasses all multiresolution models based on polygonal decompositions that have appeared in the literature so far. An MT may have additional properties, which depend on the mesh simplification technique used to build it:

- *Normal form*: every refinement operation is minimal (it cannot be split in two refinement operations that can be performed independently), and non-redundant (it does not recreate triangles and edges eliminated by updates preceding it*).*
- *Bounded degree*: the total cost of arcs incident at a node is bounded by a constant.
- *Logarithmic depth*: the depth of the DAG (i.e., the length of the longest path from root to drain), is logarithmic in the size of the MT (i.e., the total number of its triangles).

If an MT is in normal form, then every possible mesh formed by triangles of an MT can be obtained from a cut, and traversal algorithms are able to detect in optimal time the smallest possible mesh whose triangles satisfy a given LOD [Pup96]. Bounded degree and logarithmic depth are also highly desirable because they affect the efficiency of traversal algorithms (see next two sections).

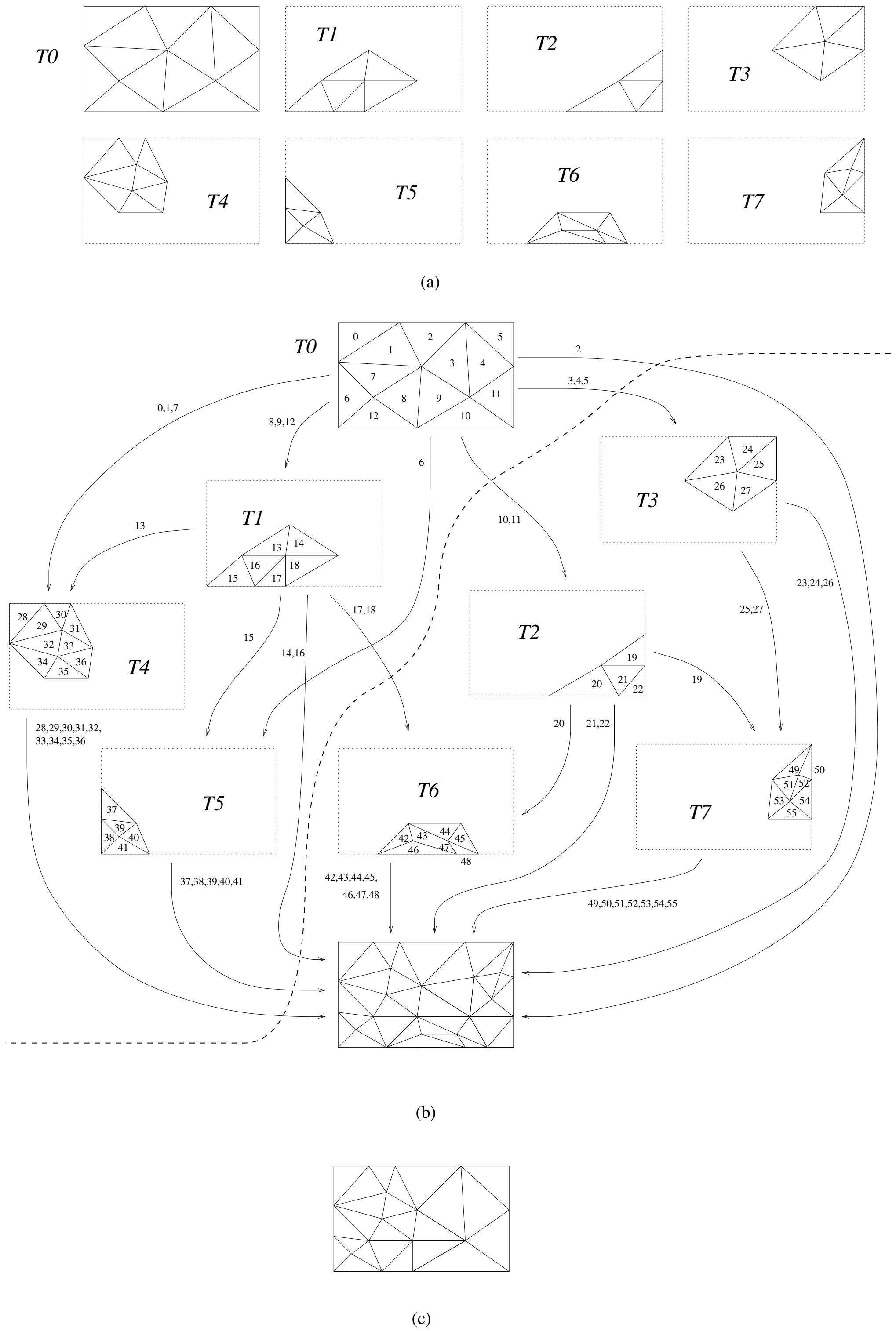

Figure 3: (a) A triangle mesh, and a sequence of local refinements; (b) the DAG describing the corresponding MT, with a cut denoted by a dashed curve; (c) the mesh corresponding to the cut.

5. Building an MT

An MT can be built through any mesh simplification technique based on the concept of local modification. Methods of interest must warrant normal form, bounded width, and logarithmic height. Moreover, they must be based on fixed rules for local modifications. Such a characteristic permits to encode local modifications through a minimal amount of information.

In [DeF97], construction algorithms based on vertex decimation, and Delaunay triangulations have been discussed, and compared. Best results are obtained with an algorithm that iteratively eliminates independent sets of vertices with bounded degree, while trying to delete less relevant details first. The Delaunay rule permits to obtain a compact data structure that encodes each local modification only through its central vertex.

Similar results may be achieved by methods based on either edge collapse [Hop96,Xia97], or triangle collapse [Ham94]. Such methods should be modified to achieve bounded degree and logarithmic height, by performing collapses of independent sets of entities with bounded degree, while trying to collapse edges corresponding to less relevant details first. Also in this case, the rules that define a local modification (vertex split/expansion) permit to obtain a compact data structure, as for Progressive Meshes.

6. Traversing an MT

A mesh is extracted from an MT by sweeping a cut through the DAG, based on conditions imposed by the client application, namely a threshold for the LOD, and a Region Of Interest, plus possibly space and time constraints. A triangle in the ROI is called active, while a triangle satisfying the LOD threshold is called valid. The server must provide the smallest sub-mesh representing the portion of surface in the ROI, and such that all its triangles are valid. If space/time requirements are violated before such a task is completed, the server must return a mesh representing the portion of surface in the ROI, and having the best possible LOD that can be achieved within such constraints.

Algorithms for selective refinement are described in detail in [DeF98a]. In the following, we briefly outline the main concepts about such algorithms. A mesh is extracted from an MT by traversing the DAG top down, i.e., starting at the cut just below its root. A local modification is performed (i.e., the cut is advanced beyond a node) each time an active triangle is not valid. Local modifications are iteratively performed until all active triangles in the cut are valid. A dynamic variant of the algorithm starts instead from an existing cut, and updates the mesh by adjusting such a cut according to new client conditions. In this case, parts of the cut might be moved forward, while other parts are moved backward through the DAG.

Active triangles in the output mesh may be piped through the channel, and received by the client for further processing. In this way, the client only receives the portion of surface lying in its ROI, represented with a mesh that optimizes the quality/space ratio. The key idea here is that only the cut corresponding to the current mesh, or, better, only the portion of such a cut lying inside the ROI needs to be transmitted to the client. Since the size of such a cut can be much smaller than the whole mesh at full resolution, the compression factor obtained by combining selective refinement and locality can be very high, while often the loss of quality of the mesh received by the client can be hardly perceived.

7. Progressive transmission

Since the algorithm for selective refinement obtains the output mesh by performing local modifications, it is possible to transmit the mesh progressively. Instead of sending a cut once it has been detected, the server sends initially the mesh corresponding to the root, possibly clipped to the ROI. Then, each time a cut is moved either forward beyond a node, or backward before a node, the server sends the corresponding local modification, which depends on the information stored at that node.

If the MT has been built on the basis of either vertex decimation, or edge collapse, each modification can be encoded in a compact way, thus achieving further compression of the extracted mesh. In other words, the implicit rules to obtain a local modification may help compressing connectivity information, hence the bitstream in the progressive transmission might result even smaller than a bitstream encoding just the final cut. For instance, in the case of an MT based on Delaunay triangulations, it is sufficient to send either a new vertex to insert (for refinement), or a reference to an existing vertex to remove (for decimation, only in the dynamic case). Therefore, no connectivity information is needed. In the case of an MT based on vertex split, information at each node is slightly higher (this case is analogous to that of Progressive Meshes [Hop96]).

8. Dynamic update

In a dynamic scenario, the client stores a portion of a current cut, namely that contained in its ROI. The server keeps track of the whole cut, and also knows which portion of it has been loaded by the client. If the server adopts the dynamic version of the traversal algorithm, then the current cut is updated on the basis of conditions given by the client. Each time a local update affects a part of the cut already loaded by the client, such an update can be transmitted directly. Otherwise, the portion of the current cut that is affected must be provided to the client before performing the update. The portion of the old cut that move outside the ROI can be eliminated directly by the client, in order to save space.

In this way, only information strictly necessary to update the mesh are sent through the channel each time client conditions change. This is especially convenient for small changes, like frame-to-frame transition in visualizing a scene while either the viewpoint moves, or zoom-in/zoom-out occur. Note that the size of the current cut may remain nearly constant, in order to fit space/time constraints of the application, while the accuracy of the mesh can be highly variable depending on requests of the client (this situation occurs, e.g., in all zooming operations).

9. Geometry Compression

Mesh simplification obtained through selective refinement and clipping operate directly by reducing the size of the mesh. This may results in a drastic compression of information transmitted through the channel, and loaded by the client. However, it is possible to obtain further compression at the server, through the channel, and at the client, by adopting suitable encoding techniques.

A mesh must be encoded by providing vertex information (i.e., position, surface normal, photometry, etc., at each vertex), and connectivity information (i.e., which triplets of vertices form triangles). Vertex information usually takes the largest space, because a single vertex may need many bits. Such information can be compressed through lossy techniques based on quantization, or lossless techniques like Huffman codes, or both [Dee95]. Connectivity information is also important, not only because it intrinsically requires some space, but also because, depending on the technique adopted to encode it, it may be necessary to store/transmit each vertex more than once (this is called vertex redundancy). In this section, we address compression of connectivity information, and redundancy, for the MT framework.

We wish to remark here that mesh simplification is a form of compression that does not require any decompression. On the contrary, the further compression of connectivity information does require decompression that may affect time performance. For this reason, we have explored different possibilities to trade off space and time requirements. In the following, we briefly describe compression methods that we are developing for an MT based on Delaunay triangulation, and built through vertex decimation. Similar methods can be also developed for an MT built through edge collapse (which can be regarded equivalently as the MT encoding of Progressive Meshes).

9.1 Compression at the server

The server works with an extremely compressed disk structure, and a larger run-time structure. In the case of a functional surface, such as a terrain, the disk structure only requires the sequence of vertices of the MT, plus a single integer number r that specifies the number of vertices in the root mesh. The Delaunay triangulation of the first r vertices is the root mesh, while the remaining vertices are listed in inverse order of decimation. Each such vertex defines a local modification. The MT DAG is reconstructed from such a structure by an incremental algorithm for Delaunay triangulation that inserts one vertex at a time. Therefore, the disk structure in this case does not need to encode either connectivity, or the structure of the DAG.

The case of a sculptured surface is more complicated, since the Delaunay rules do not hold in space, and need to be used on a local basis. The root mesh must be encoded explicitly. To this purpose, we have developed a compression method that avoids vertex redundancy, and needs at most two extra bits per edge of the mesh to encode connectivity [DeF98b]. Each local update can be encoded by its central vertex, plus a pointer to a triangle of the existing mesh, which is used to localize the Delaunay criterion bt projecting the surface on the plane containing such a triangle. Also in this case, the MT DAG is obtained by an incremental algorithm for Delaunay triangulation, and the disk structure needs small information for connectivity.

The run-time structure needs to encode the DAG explicitly, in order to achieve efficiency of traversal algorithms. Therefore, some overhead due to storage of arcs in the DAG is necessary. We have developed two versions of this data structure, with different complexities and performances. In the implicit version, the root node contains the initial mesh, encoded in the compressed format; each internal node contains compressed information on the corresponding update, as in the disk structure (i.e., either just a vertex, or a vertex plus a reference to an existing triangle), plus a number to indicate the LOD of the corresponding portion of mesh after the update; no additional information is stored on arcs. The traversal algorithm needs also to encode explicitly the mesh corresponding to the current cut during traversal, with an uncompressed data structure (e.g., by storing its triangles, and adjacencies explicitly). However, the current mesh usually needs much less space than the MT DAG.

In the explicit data structure, the initial mesh, as well as all updates are encoded by listing their triangles (a triple of pointers to vertices, plus a LOD number for each triangle). Triangles are clustered to label arcs, while no additional information must be stored at nodes. In this case, the current mesh is encoded simply by listing the arcs of the current cut. The explicit data structure is about three times larger than the implicit one, but traversal algorithms run about ten times faster. See [DeF98a] for a comparison of the two structures.

9.2 Compression through the channel

Depending on the application, we can adopt either a static transmission (a mesh is transmitted after it has been retrieved by the traversal algorithm), or a progressive transmission (mesh updates are transmitted while the mesh is retrieved).

In the static case, it is easy to obtain and transmit a compressed bitstream, as described in [DeF98b], from the current mesh encoded by the implicit structure on the server. In the progressive case, the initial mesh can be transmitted through the same compressed bitstream, while each update is transmitted by sending its corresponding vertex, plus possibly a pointer to a triangle, and a few bits of control codes to drive the update. Depending on the amount of extra information transmitted, we could need to perform more or less computation at the client. Since our construction algorithm [DeF97] guarantees that each update can be obtained with only a small number of edge swaps, we are currently developing a compressed code to specify such swaps in order to avoid any numerical computation at the client. In both the static, and the progressive case vertex information is transmitted without redundancy.

In a dynamic scenario, we must take into account that the mesh changes through time. In this case, some information about a mesh may become useless for the client at a given time, because of an update, and be reused later because of another update. In principle, each vertex can be transmitted only once through the channel. Once it has been loaded by the client, it can be referred to by using a hash code, whenever necessary. In practice, if the memory size of the client is low, it may be necessary to discard vertices once they move outside the current cut, and to send them again if they become active again later.

9.3 Compression at the client

The client only needs to store the current mesh. In the static case, this can be compressed in the same structure used for transmission [DeF98b], which can be decompressed quickly for rendering purposes. In the progressive case, it is necessary to maintain an explicit data structure by encoding a list of vertices, plus a list of triangles, where for each triangle indexes to its vertices and its neighbors are maintained. Such a structure can be updated efficiently by dynamic Delaunay procedures to insert/delete vertices. The client either may or may not need performing numerical computation during the update, depending on the amount of information transmitted with any update (see above). More information through the channel may help the client saving time in computation. Thus, this is again a matter of balancing speed and memory load.

The client data structure used in the progressive case does not implement any mechanism for compressing connectivity, in order to achieve efficiency in mesh updates. On the other hand, the current mesh should be usually small enough to fit the client's memory, because of simplification performed by the selective refinement algorithm.

10. Implementation and experiments

A prototype version of the MT server has been implemented by using the explicit encoding for the run-time structure, which provides primitives for selective refinement, and clipping to a ROI. Such a server includes both the static, and the dynamic version of the traversal algorithm. A simple viewer has been implemented as a client, which provides interactive functionalities like flythrough, and draggers to define focus areas (ROIs). In the current implementation, both the client and the server are running on the same computer, and communication occurs through a segment of shared memory. This implementation is made at an academic level, without any optimization in data conversion and rendering. Therefore, its performances are expectedly far below the possibilities of a properly engineered system. Nevertheless, such a system already supports real-time flythrough, and interaction with draggers in handling MTs built on meshes of reasonably large size (64K triangles).

Figures 4 and 5 show two pictures taken from interactive sessions. For more details on these experiments see [DeF98a]. In Figure 4, a dragger box is swept interactively through space containing a bunny model. The selective refinement algorithm extracts the mesh at the highest possible LOD inside the box, while LOD can be arbitrarily low outside it. In this case, the whole mesh at high resolution contains about 70,000 triangles, while the mesh visualized in the figure contains only about 2,500 triangles, with only about 500 triangles inside the box. On average, such a mesh is produced by the server in about 40 milliseconds, and the time decreases to about 20 milliseconds if the mesh is clipped to the box (ROI). A frame-to-frame transition needs making on average a few hundred local modifications to the current mesh (this is indeed highly dependent on the speed in moving the dragger).

Figure 5 shows the top view of a terrain, with a wedge region representing the projection of a view frustum. The viewpoint, at the vertex of the region, is moved interactively to simulate flythrough. In this case, the required LOD is decreasing linearly with distance from the viewpoint inside the frustum, and it is arbitrarily low outside it. The mesh at the highest LOD contains 32K triangles, while the mesh in the figure contains about 2,700 triangles, with only about 1,700 triangles in the view frustum. Extraction times are about 30 milliseconds both with and without clipping. Also in this case, a few hundred local modifications are sufficient to update the mesh at each frame.

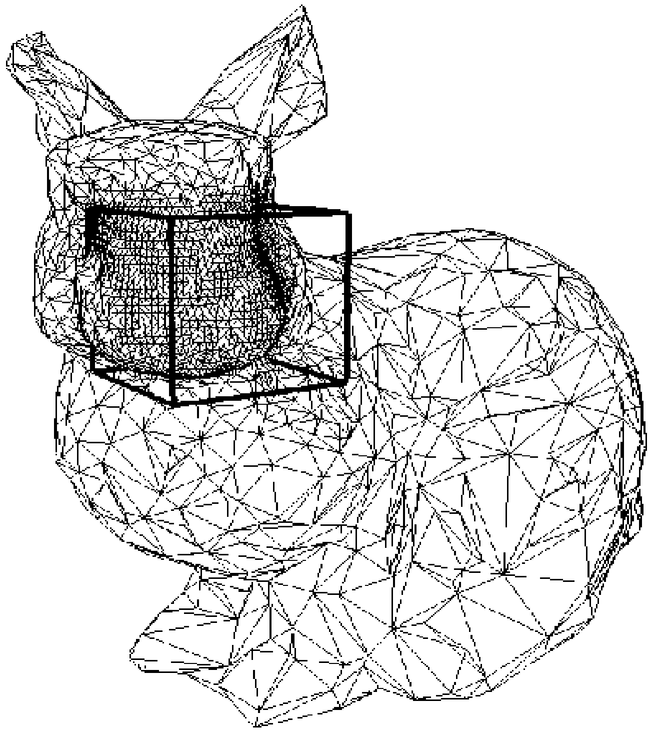

Figure 4: Full resolution is used inside the box, while low resolution is used outside it.

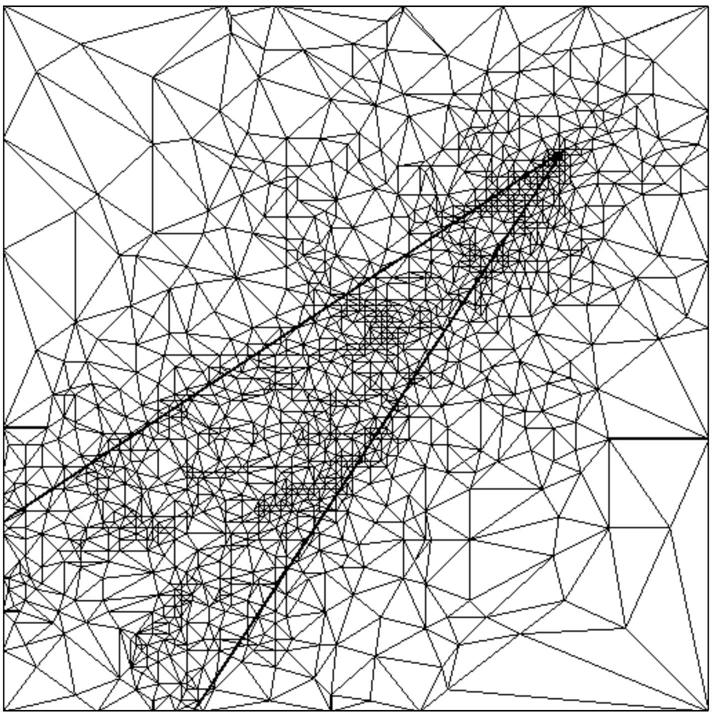

Figure 5: Top view of a terrain: resolution of the mesh inside the wedge is decreasing linearly with distance from the corner (viewpoint); resolution outside the wedge can be arbitrarily low.

References

[Bro96] Brown, P.J.C., 1996, A fast algorithm for selective refinement of terrain meshes, *Proceedings COMPUGRAPHICS 96,* pp.70-82.

[Cig97] P. Cignoni, E. Puppo, R. Scopigno, 1997, Representation and visualization of terrain surfaces at variable resolution, *The Visual Computer,* 13, 5, pp.199-217.

[deB95] de Berg, M., Dobrindt, K.T.G., 1995, On the levels of detail in terrains*, Proceedings 11th ACM Symposium on Computational Geometry,* Vancouver, BC (Canada), pp.c26--c27.

[Dee95] M. Deering, 1995, Geometry Compression*, Computer Graphics Proceedings (SIGGRAPH'95),* Los Angeles, August 1995, pp.13-20.

[DeF96] L. De Floriani, E. Puppo, P. Magillo, 1996, A formal approach to multiresolution modeling, in *Geometric Modeling: Theory and Practice,* W. Strasser, R. Klein, R. Rau (Editors), Springer-Verlag.

[DeF97] L. De Floriani, P. Magillo, E. Puppo, 1997, Building and traversing a surface at variable resolution, *Proceedings IEEE Visualization'97,* October 19-24, 1997, Phoenix, AZ.

[DeF98a] L. De Floriani, P. Magillo, E. Puppo, 1998, Selective refinement of surface meshes: data structures and algorithms*, Technical Report* DISI-TR-98-02, Department of Computer and Information Sciences, University of Genova.

[DeF98b] L. De Floriani, P. Magillo, E. Puppo, Compression of triangular and tetrahedral meshes through shelling, in preparation.

[Ham94] B. Hamann, 1994, A data reduction scheme for triangulated surfaces, *Computer Aided Geometric Design,* 11(2):197-214.

[Hec97] P. Heckbert, M. Garland, 1997, Survey of polygonal surface simplification algorithms, *Siggraph'97 Course Notes.*

[Hop96] H. Hoppe, 1996, Progressive meshes*, Computer Graphics Proceedings (SIGGRAPH 96),* New Orleans, LA, USA, August 4-9, pp.99-108.

[Hop97] H. Hoppe, 1997, View-dependent refinement of progressive meshes, *Computer Graphics Proceedings (SIGGRAPH 97),* August 1997.

[Pup96] Puppo, E., 1996, Variable resolution terrain surfaces*, Proceedings Eight Canadian Conference on Computational Geometry,* Ottawa (Canada), 12-15 August, pp.202-210. Also appeared in extended version as: Variable resolution triangulations*, Technical Report* 16/96, Institute for Applied Mathematics, National Research Council, (submitted for publication).

[Pup97] Puppo, E., Scopigno, R., 1997, Simplification, LOD, and multiresolution - Principles and applications*, Eurographics'97 Tutorial Notes.*

[Tau98] G. Taubin, A. Guéziec, W. Horn, F. Lazarus, 1998, Progressive forest split compression, *Research Report* RC-21082, IBM Research Division, Yorktown Heights, NY.

[Xia97] J.C. Xia, J. El-Sana, A. Varshney, 1997, Adaptive real-time level-of-detail-based rendering for polygonal models, *IEEE Transactions on Visualization and Computer Graphics,* 3, 2, pp.171-183.