RC 20889 (92539) (6/16/97)
Computer Science/Mathematics 10 pages

Research Report

Surface Partitions for Progressive Transmission and Display, and Dynamic Simplification of Polygonal Surfaces

André Guéziec, Francis Lazarus and Gabriel Taubin

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, NY 10598

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich · Austin

Surface Partitions for Progressive Transmission and Display, and Dynamic Simplification of Polygonal Surfaces

André P. Guéziec, Francis Lazarus and Gabriel Taubin
IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
`gueziec,francis,taubin@watson.ibm.com`

ABSTRACT:

We present a new method for (1) automatically generating multiple Levels Of Detail (LODs) of a polygonal surface, (2) progressively loading, or transmitting, and displaying a surface, and for (3) changing interactively the LOD when displaying. We build the LODs using any algorithm that performs edge collapses and certain vertex removals to simplify surfaces, and provides an ordered list of ordered vertex pairs (edge collapse specifications). We propose a Surface Partition for encoding surface LODs: we define vertex and triangle *levels* during simplification; vertices and triangles are partitioned and sorted according to their level, and are passed to the display algorithm in decreasing level order, one level at a time, together with a vertex *representatives* array. Each level of vertices and triangles, together with higher levels and the vertex representatives, form a valid surface. We propose a data structure using a Directed Acyclic Graph (DAG) for recording a partial ordering among edge collapses, and varying the LODs across the surface.

Key-words : Edge Collapse, Vertex and Triangle Levels, Surface Levels of Detail, Surface Partition, Progressive Transmission and Display, Dynamic Simplification.

1. Introduction

Our starting point, as Ronfard and Rossignac's [1], Hoppe's [2], and Xia and Varshney's [3] is to use a simplification algorithm that generates edge collapses, or vertex removals corresponding to a particular edge collapse, to produce a series of intermediate levels of detail. The actual order in which the collapses occur is irrelevant. However, when a given Collapse i is validated by the algorithm, the collapsed edge neighborhood is in a particular configuration, resulting from a few identifiable edge collapses, say Collapses j and k: we record that Collapse i must occur after Collapses j and k, defining a *partial ordering* on the collapses that we use in our method.

We developed this method after observing the patterns of vertex and triangle *representatives* and *pc-reps* naturally resulting from the simplification method of Guéziec [4] (see Fig. 1 and explanations on representatives and pc-reps of Section 2.1 and Fig. 2B). The main differences with the related work on interactively defining LODs [1, 2, 5, 3, 6] are that: (1) by concentrating on global LODs and progressive surface display, we develop a LOD format of small overhead, i.e., an array of vertex representatives; (2) we introduce the principle of Surface Partitions; (3) we provide a new definition for a vertex and triangle *levels*; (4) we exploit the one-to-one correspondence between edge collapses and removed (*blue*) vertices: we store old attributes of the remaining (*red*) vertex using the blue, subsequently unreferenced, vertex during collapses, thus allowing vertices to move or normals and other attributes to change.

In Section 2 we introduce Surface Partitions and use them for changing interactively the LOD in a display using a single surface model and an array of vertex representatives. We also propose a format for progressively loading and displaying a surface. In Section 3, we define a data structure using a DAG to represent a partial ordering among edge collapses; we describe work in progress for varying the LODs across the surface.

1.1. Background

Background on surface simplification can be found for instance in Guéziec [4]. For the present paper, a *surface* is a set of triangles, where each triangle is a triple of vertex references. An *edge* is a pair of vertices, called *endpoints*, used in a triangle. The *star of a vertex* is the set of triangles that share that vertex. The *star of an edge* is the union of the edge endpoint stars. To provide an intuitive surface model, and because we start with the simplification method of [4], we suppose that the star of each vertex is homeomorphic to a disk or a half disk at the boundary, yielding a *2-manifold* (see Hoffmann [7]): this assumption is also used in Section 2.4. A *boundary edge* is shared by one triangle only. Edges shared by two triangles are *interior* edges. An *edge collapse* consists of bringing both endpoints of an edge to the same position, thereby eliminating two triangles (or one triangle when collapsing a boundary edge).

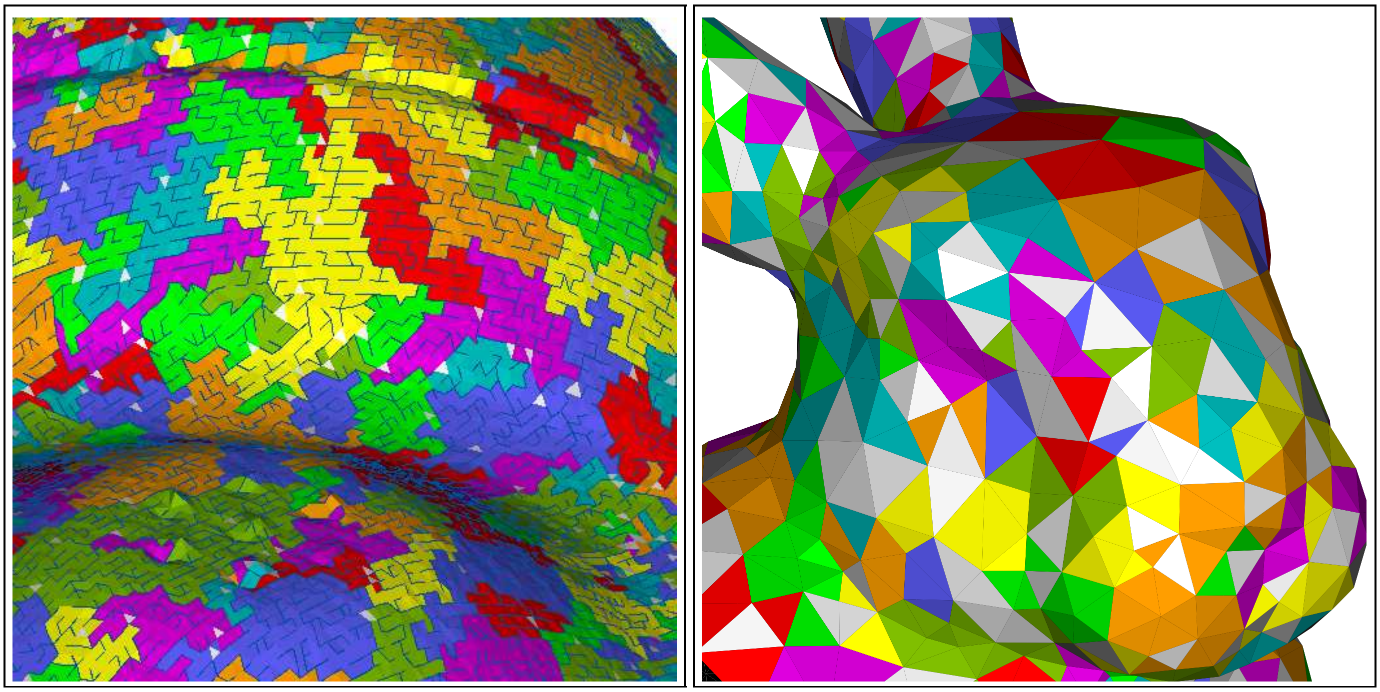


Figure 1: Forest of representative relationships obtained as the natural outcome of Guéziec's simplification algorithm: vertices connected by marked edges simplify to the same location; triangles of the same color have the same pc-rep; grey triangles are pc-reps; see also Figs. 2B and 6.

2. Global LODs using Surface Partitions

2.1. Red and Blue Vertices and Triangles

We give the edge collapse a direction, by calling a *blue* vertex and a *red* vertex both endpoints, such that: the blue vertex is removed and the red vertex remains (see Fig. 2A); the position of the red vertex can be modified; the red vertex is the *representative* of the blue vertex; Examples of edge collapses, with red and blue vertices, are shown in Fig. 3. More generally, each surface vertex has a representative, preferably stored using an array; red vertices are their own representative. At the start of the simplification process, each vertex is red; subsequently, some red vertices become blue; in the end, the simplified surface uses red vertices exclusively. There is a one to one correspondence between blue vertices and edge collapses. Red vertices are similar to the "parents" and blue vertices to the "children" in Xia and Varshney's method [3], except that their parents and children do not form a partition of the vertices. Hoppe's data structures [10] ignore the direction of the collapsed edge.

blue triangles are removed during an edge collapse, *red triangles* remain. In Fig. 2A we illustrate how vertex and triangle representatives are updated when performing an interior edge collapse $\mathbf{v_1} \rightarrow \mathbf{v_2}$ (a boundary edge collapse is treated similarly). We note $\mathbf{v_3}$ and $\mathbf{v_4}$ the vertices adjacent to both $\mathbf{v_1}$ and $\mathbf{v_2}$. Edge $(\mathbf{v_1}, \mathbf{v_2})$ is now degenerate; it will not be referenced subsequently. The representative of Edge $(\mathbf{v_1}, \mathbf{v_3})$ is Edge $(\mathbf{v_2}, \mathbf{v_3})$. The representative of Edge $(\mathbf{v_1}, \mathbf{v_4})$ is Edge $(\mathbf{v_2}, \mathbf{v_4})$. Triangle

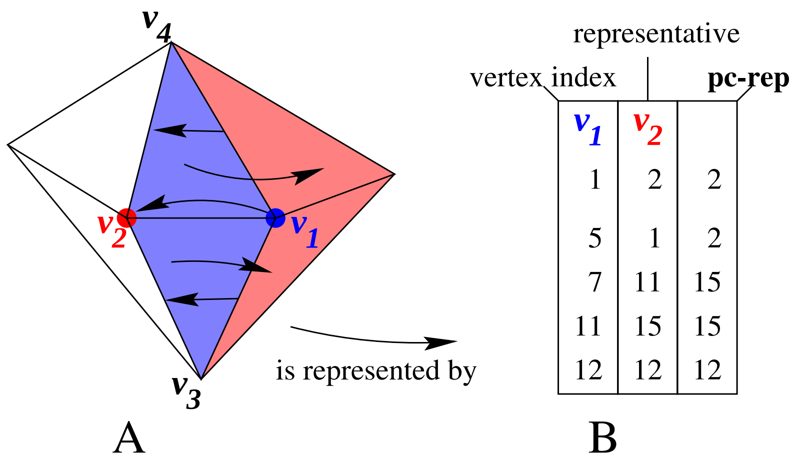


Figure 2: A: during an edge collapse blue Vertex $\mathbf{v_1}$ and blue triangles are removed; vertex, edge and triangle representatives are assigned as arrows indicate. B: vertex representative array, and path compressed representative (pc-rep) array for some vertices of Fig. 3.

$\Delta(\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3})$ is degenerate, its representative is the non-degenerate (red) triangle incident to $(\mathbf{v_1}, \mathbf{v_3})$. The representative of Triangle $\Delta(\mathbf{v_1}, \mathbf{v_4}, \mathbf{v_2})$ is chosen similarly.

The original motivation for defining representatives was an efficient method for computing vertex stars during simplification by "pivoting" around a vertex representative (see [4]). A benefit is that we can identify the blue and red triangles in the *original* list of triangles. To build a simplified surface, we path compress the vertex representative array as shown in Fig. 2B, resulting in the *pc-rep* array: path compression is discussed in Tarjan [11], and consists of following the representative hierarchy until a root is found, and of making each element point directly to the root. We store the triangles with original vertex indices; when using a particular triangle, we temporarily replace vertex indices with their pc-rep.

2.2. LOD Generation

Simple Example. We consider the model of Fig. 3, called the Simple surface, with 16 vertices and 18 triangles. Nine edge collapses that were computed automatically by Guéziec's algorithm are used to simplify the surface. For now, we suppose that red vertices do not move. We assign *levels* to vertices as follows: at the start all vertices are red with level 0. When an edge is collapsed, we compute the maximum level l in vertices of the edge star, and we assign level $l + 1$ to both red and blue edge endpoints. This is different from Xia and Varshney, who increment the level of all red vertices independent of their neighborhood. We generate more levels than they do. $l + 1$ is also the level assigned to the triangles that become blue during the collapse. We use levels of blue vertices and triangles to generate LODs. Levels of red vertices are only temporarily used for computing levels of blue vertices. To become familiar with this process, it is best to examine carefully Fig. 3, providing complete details of the edge collapses performed on the Simple surface. At the end of the simplification process, we increment the highest level and assign it to all red vertices and triangles (this is level 7 in Fig. 3).

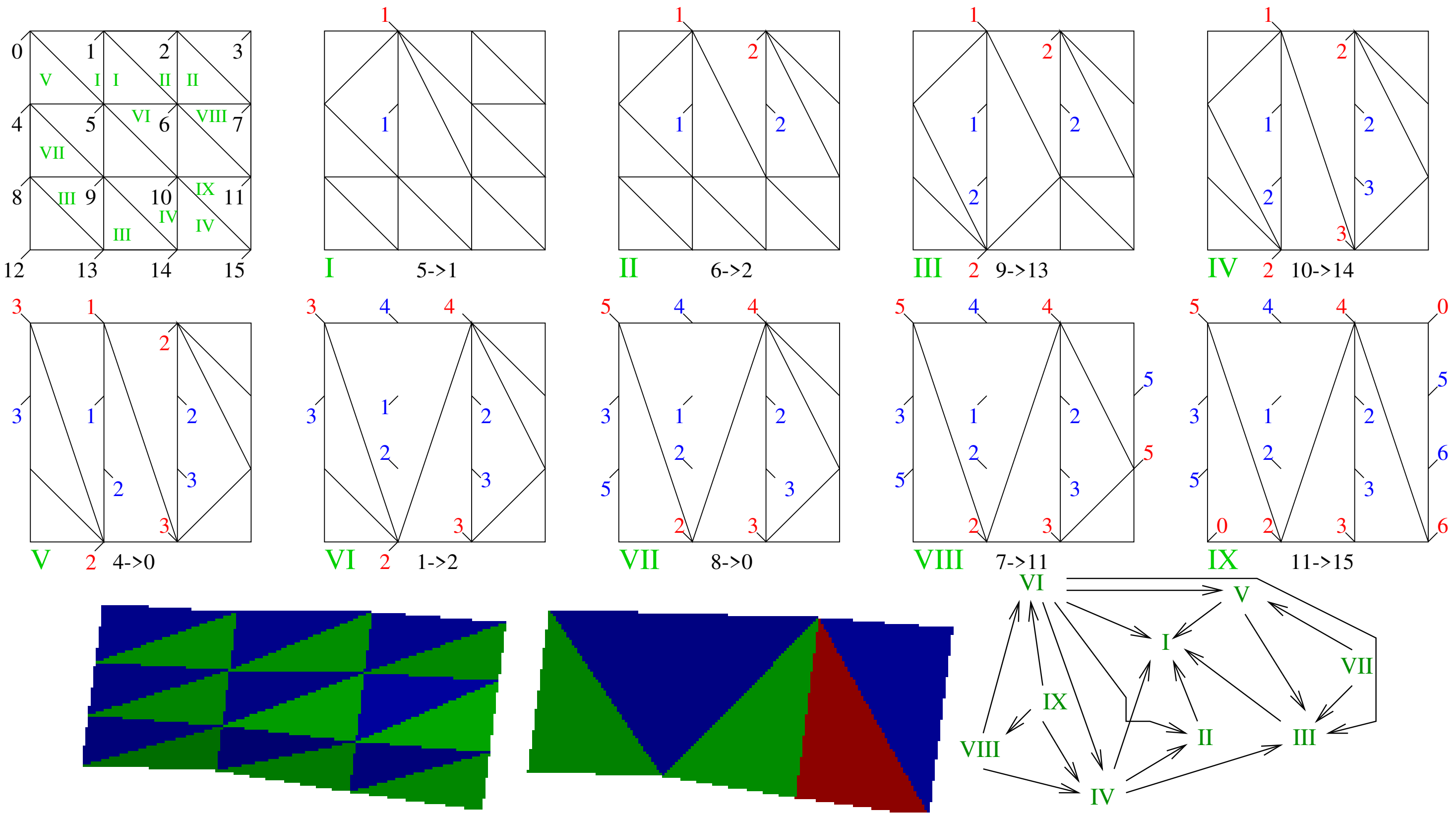


Figure 3: Simple surface: edge collapses, numbered I through IX, affect the levels of the red and blue vertices as shown. Triangles removed during the collapses are shown on the top left. The Level 1 and level 7 surfaces are shown below (colors are irrelevant), together with the DAG representing the partial ordering of edge collapses.

Partitioning a Surface in LODs. We have produced a *partition* of the vertices and the triangles using levels. For the Simple surface, the vertex partition is shown in Fig. 3, while the triangle partition is shown in Fig. 4. Surface LODs can be defined as follows: the ith surface level consists of all vertices and triangles of level greater or equal to i. In Fig. 3 the coarsest surface level is 7 and the finest is 1. To evolve from surface LOD i to $j < i$, we simply provide vertex and triangles of levels j to $i - 1$. We can create fewer levels by merging any number of consecutive levels in a single level.

We next sort the vertices and triangles according to their level, with the red vertices and triangles coming first, and we update the triangle vertex indices and the vertex representatives to reflect the permutation (sorting) on the vertices. To use the various levels, we need a reference copy of the vertex representatives, a working copy that can be modified, and a pc-rep array that is re-computed each time the working copy is modified (this is done on the fly as triangles vertex indices are visited, and is inexpensive [11].) To use Level 7, we build the vertex pc-rep array from the vertex representative array, and exchange the vertices of red triangles with their pc-rep. To use Level 6, we *undo* the representatives of the vertices marked 6: in the working copy of the representative array, we overwrite the corresponding representative with the index of the vertex itself, thereby "splitting" the vertex; we then recompute the pc-rep. To use Level 5, we undo the representatives of the vertices marked 5; and

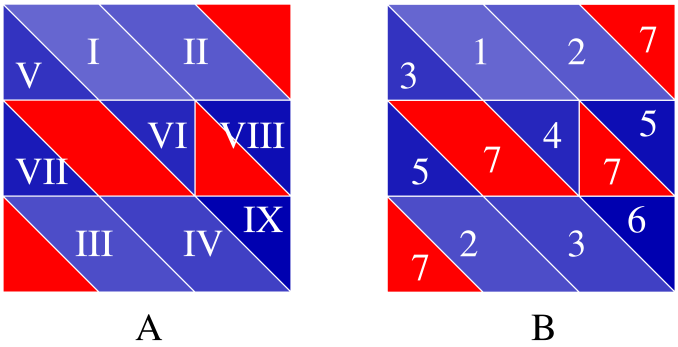


Figure 4: Simple surface: A: labeling blue triangles according the the collapse number. B: partitioning the surface triangles in 7 regions, using blue vertex levels; triangles with label i are used in surface LODs $i, i+1.\ldots, 7$; together with the vertex partition and vertex representatives, we define the Surface Partition.

so on. In Fig. **??** we illustrate the results of using this method to interactively change the LOD in a surface model representing the Earth's topography.

Consistent LODs. We prove that the above method produces *consistent*, or "good looking" surfaces. By *consistent*, we mean that the surface can be obtained from the original surface with a series of edge collapses that would be validated by the simplification algorithm; some collapses can be omitted or performed in a different order, provided that for each collapse the edge star is exactly the same as it was during simplification.

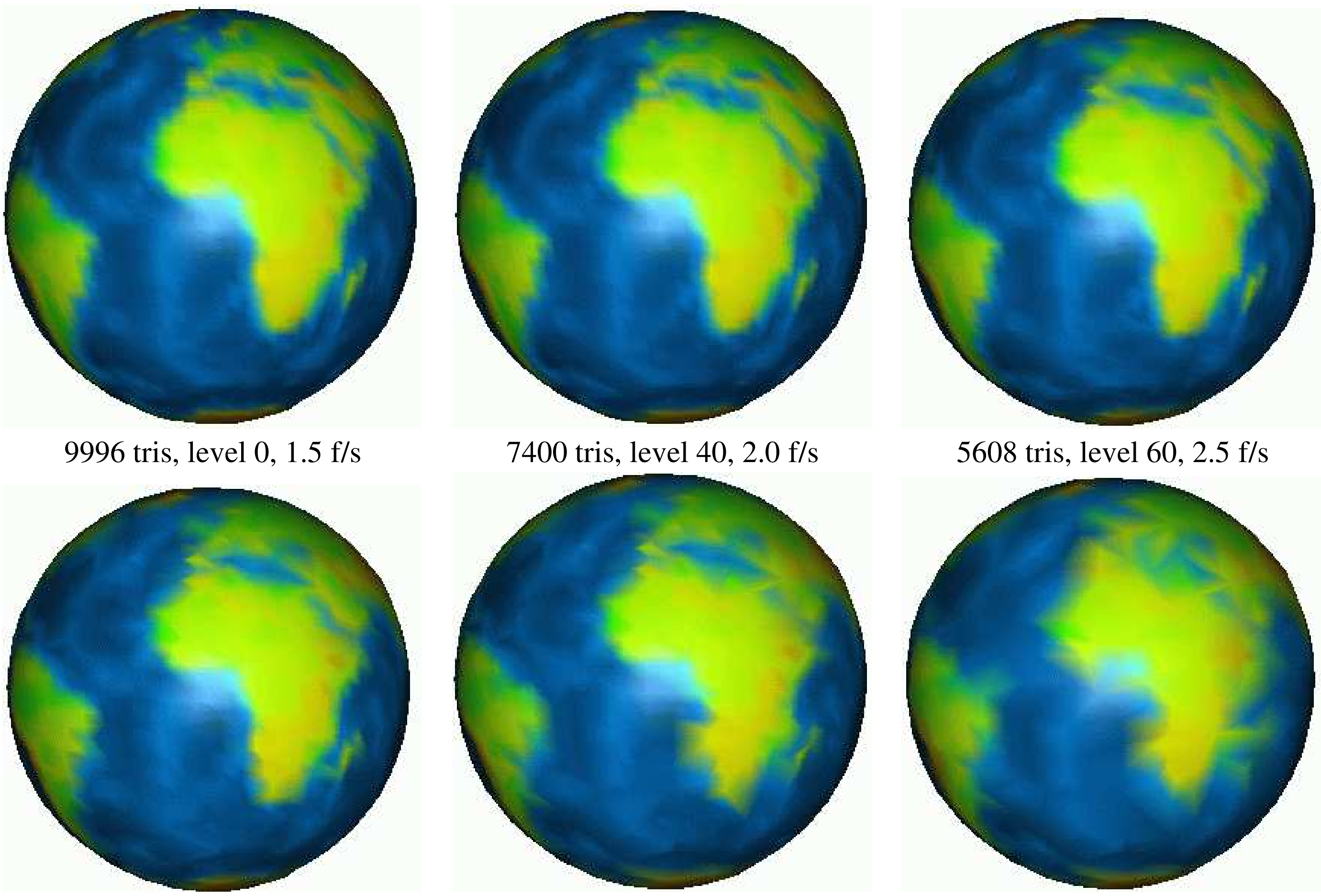
A global precedence amongst collapses is recorded using the vertex level. Suppose that we produce an inconsistent surface when undoing the representative $\mathbf{r}$ of vertex $\mathbf{b}$ with level l. Then there exists a vertex $\mathbf{w}$ in the star of $\mathbf{r}$ or $\mathbf{b}$ that relies on $\mathbf{b} \rightarrow \mathbf{r}$ to be collapsed, which implies that $\mathbf{w}$ has level at least $l+1$. But we operate level by level, and at this time, no level $l+1$ vertex remains.

This method is different from Hoppe's Progressive Meshes [2], because we do and undo the edge collapses in batches according to their level, which is not directly related to the order in which they were performed.

2.3. Progressive Surface Loading and Display

By analogy with what is commonly performed for progressively loading and displaying image files in Web Browsers, we propose a surface format suitable for progressive loading or transmission as follows: we first send the coarsest surface level k (vertices and triangles) together with the vertex representative array; then, in $k-1$ successive batches, we send the vertices and triangles only of level $k-1$ to 1. We call this format a Surface Partition.

Supposing that no data compression technology is used on the surface, the same information is provided as usual as we have the same number of vertices and triangles. The vertex and triangle levels are implicit, as each surface portion has a level corresponding to the order in which it is loaded (or



9996 tris, level 0, 1.5 f/s | 7400 tris, level 40, 2.0 f/s | 5608 tris, level 60, 2.5 f/s

3894 tris, level 75, 3.5 f/s | 2722 tris, level 90, 4.5 f/s | 1476 tris, level 106, 6.5 f/s

Figure 5: Successive LODs of the Earth model; for each LOD, we report the number of frames per second (f/s) during animation using an IBM POWER Gt4xi Graphics Adapter. The LODs were generated in 10.5 seconds CPU on an IBM 58F workstation.

transmitted through a network). The vertex representative array must be provided separately. The representatives of the red vertices are implicit and thus not provided.

We next perform a simple byte count for specifying a generic surface, ignoring vertex or triangle properties. We suppose, as commonly done, that there are n vertices and approximately $2n$ triangles (this depends upon the surface genus and number of boundaries; it is exact for a torus) and that 4 bytes are used to store each vertex coordinate (typically a 4 byte `float`) and vertex index (typically an 4 byte `int`). The generic surface would be stored using $36n$ bytes. The Surface Partition would be stored using *less* than $40n$ bytes, since the vertex representatives array, which is the sole addition, ignores red vertices. The Surface Partition incurs an additional cost factor of at most $40/36 \simeq 1.1$.

Red Vertices are Allowed to Move. We now drop the requirement of fixing red vertices. We create a copy of the array of vertex positions, called the *blue positions* array. We call the original array *red* *positions*. We can also create a copy of vertex normals, the *blue normals*, or of other vertex attributes.

Whenever we perform a collapse, before updating the red vertex position, we store its current position in the blue position array entry corresponding to the blue vertex. When using the levels of detail and undoing a vertex representative, we give back the red position to the red vertex. Because we undo the representatives in decreasing levels, the positions and attributes that we restore are always correct. The additional cost on the Surface Partition representation is ns bytes per attribute of size s (s = 12 for vertex positions or normals).

2.4. Using Other Simplification Algorithms

We can use the specification of edge collapses as an ordered series of ordered vertex pairs blue $\rightarrow$ red, provided by any simplification algorithm; we then recreate the representative relationships and the levels. The Surface Partition uses triangle representatives, whose maintenance currently relies on having a manifold surface. However, we can abandon this requirement if the blue triangles are provided along the edge collapse specification by the other algorithms.

3. Local Levels of Detail

Each edge collapse has a status: S stands for "split", meaning that the collapse is not performed currently. C stands for "collapsed", meaning that the collapse is currently performed.

Building a DAG for Storing a Partial Ordering of Edge Collapses. If performing a certain collapse, for instance in Fig. 3 Collapse V with blue vertex 4 and red vertex 0 ($4 \rightarrow 0$), requires that other collapses be performed beforehand, e.g., Collapse I ($5 \rightarrow 1$) and Collapse III ($9 \rightarrow 13$), we add two edges ($V \rightarrow I$) and($V \rightarrow III$) to the DAG. This means that the situations in which V has status C and I has status S or III has status S are impossible. This DAG is stored as two separate hash tables, one that for V, stores both I and III, and another one that for I and III stores V. We also note that V has two *collapse constraints* and that I and III each have one *split constraint*. When we split V, then we can decrement the number of split constraints of I and III. Similarly, we can increment or decrement collapse constraints. The complete DAG for the Simple surface is represented in Fig. 3. In comparison, Hoppe's vertex hierarchies [10] simply encode the blue $\rightarrow$ red representative relationship, and "valid" collapses and split must be defined separately; our DAG completely defines valid collapses and splits. Vertex representatives are analogous to Luebke's triangle proxies [6]; he uses an octree to represent vertex hierarchies which is different from the present method.

Once the final ordered list of edge collapses is known, we redo the collapses in their original order, ignoring the geometry, which was examined during simplification. The vertex levels can be computed at this stage. For instance, when redoing Collapse I, we note that Vertex 5 *influences* Vertex 1. In a vertex *influence hash table*, we record for Vertex 1 that 5 is one of its *influences*. We also examine the current level of all vertices in the star of the collapsed edge. Each level greater than zero indicates that the corresponding vertex was the outcome of a collapse. We retrieve the edge collapses that influenced that particular vertex using the influence hash table. For instance, we remark that Collapses I and III must have occurred before, and we consign the information $V \geq I, III$ in the DAG.

Dynamic Local LOD Specification. We refer to an edge collapse using the corresponding blue vertex, which has either C status or S status (if it became red). Initially, the surface is fully simplified and all blue vertices except the ones with split constraints are entered in a *split priority queue*. The split priority queue is keyed with an *error* before split, with larger values coming first: the error depends upon the location of the vertex, simplification error measurements [4] and the projection parameters. A *collapse priority queue* is initially empty. The collapse queue is keyed with the *error* after collapse, with lower values coming first.

The following step applies for a series of views: while the error of the top vertex in the split priority queue is larger than a maximum error, we split the blue vertex, update the corresponding collapse constraints, and the collapse queue (such vertices can now be collapsed). Then, while the error of the top vertex in the collapse priority queue is smaller than the maximum error, we collapse blue vertices on that queue. Corresponding triangles are added or removed to the display list, represented for instance using a doubly linked list of triangles. After processing a few views, we flush the two priority queues, and refill them using up-to-date (viewpoint dependent) error information.

References.

[1] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3):C67–C76, 1996. Proc. Eurographics'96.

[2] H. Hoppe. Progressive meshes. In *Siggraph*, pages 99–108, New Orleans, August 1996. ACM.

[3] J.C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In Yagel and Nielson, editors, *Visualization 96*, pages 327–334. IEEE, October 1996.

[4] A. Guéziec. Surface simplification inside a tolerance volume. Technical report, IBM T.J. Watson Research Center, Yorktown Heights, New York, March 1997. Revised version of RC 20440, March 1996; available from the author.

[5] P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. Real-time, continuous level of detail rendering of height fields. In *Siggraph*, pages 109–118, New Orleans, August 1996. ACM.

[6] D. Luebke. View dependent simplification of arbitrary polygonal environments. In *Siggraph*. ACM, 1997. To appear; pre-print provided by the Author.

[7] C.M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann, San Mateo, California, 1989.

[8] L. De Floriani, B. Falcidieno, and C. Pienovi. Delaunay-based representation of surfaces defined over arbitrarily shaped domains. *Computer Vision, Graphics, and Image Processing*, 32:127–140, 1985.

[9] M. Garland and P. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Carnegie Mellon University, September 1995.

[10] H. Hoppe. View dependent refinement of progressive meshes. In *Siggraph*. ACM, 1997. To appear; pre-print provided by the Author.

[11] R.E. Tarjan. *Data Structures and Network Algorithms*. Number 44 in CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, 1983.

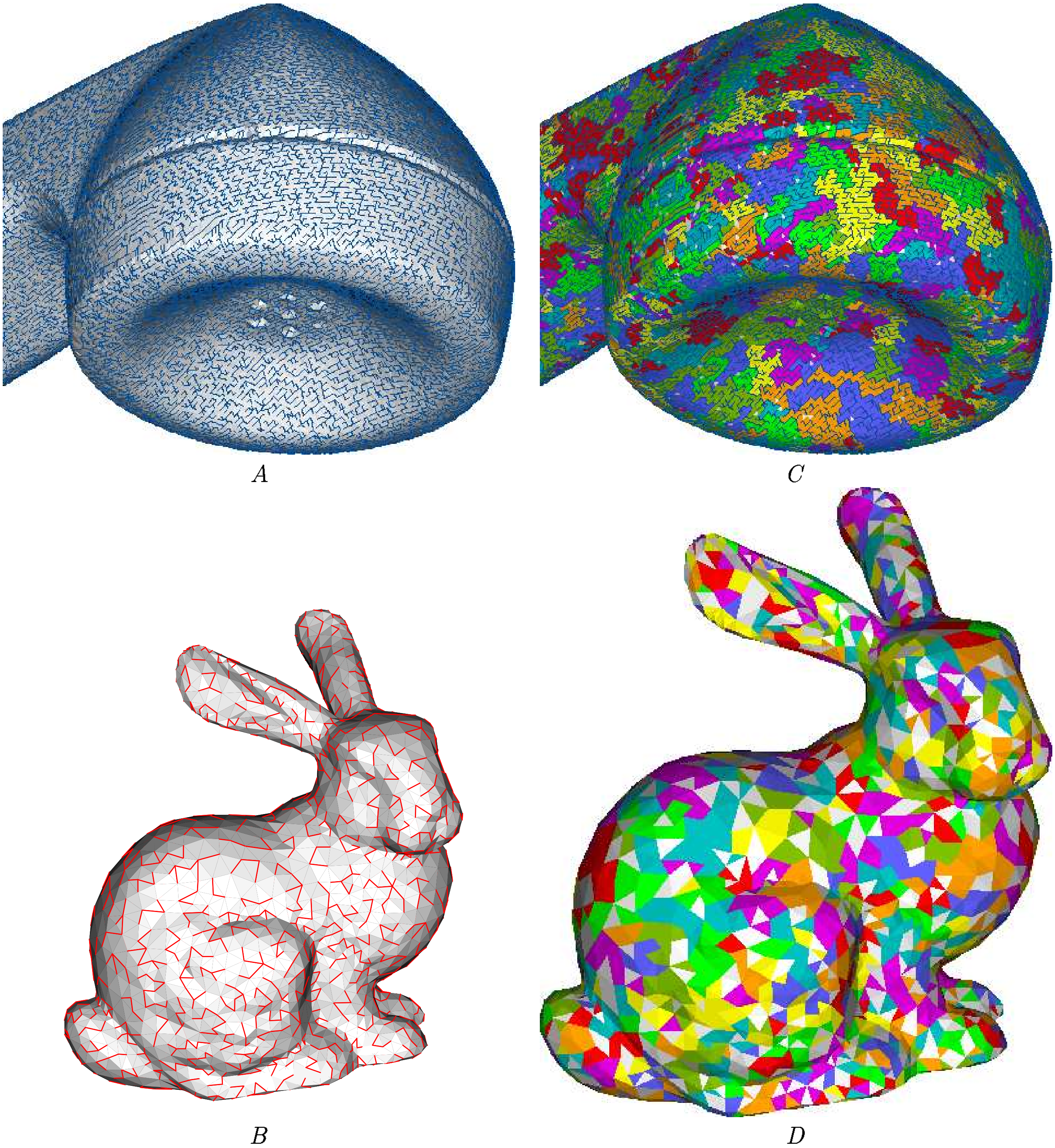


Figure 6: (complement to Fig.1) A, B: Vertex forests: vertices connected by marked edges simplify to the same location. C, D: Triangles of the same color have the same pc-rep, or root; grey triangles are roots.