Simplicial Maps for Progressive Transmission of Polygonal Surfaces

André Guéziec, Gabriel Taubin, Francis Lazarus and William Horn [1]

IBM T.J.Watson Research Center

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. The vertex representatives array encodes a succession of *simplicial maps* between the highest resolution LOD and other LODs. 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. We describe an implementation of our method in VRML.

Key-words : Simplicial Map, 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], Gueziec's [2], Hoppe's [3], and Xia and Varshney's [4] is to use a simplification algorithm that generates either 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 [2] (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, 3, 5, 4, 6, 7] 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 a Surface Partition; (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 a Surface Partition is introduced 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. In Section 4, we describe our VRML implementation.

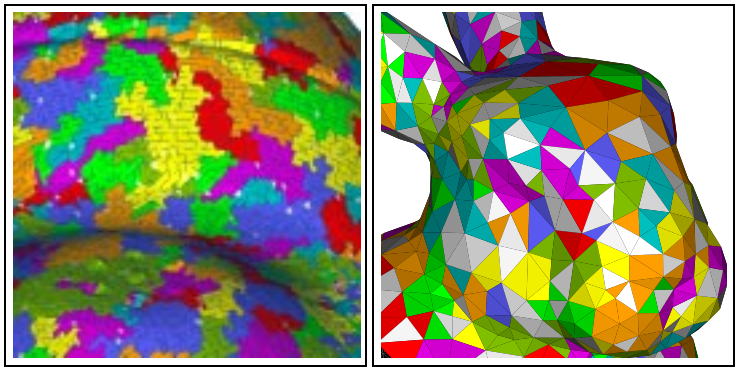

Figure 1: Forest of representative relationships obtained as the natural outcome of Guéziec's simplification algorithm (similar patterns could be visualized for any simplification method using edge collapses): vertices connected by marked edges simplify to the same location; triangles of the same color have the same *pc-rep* (see Section 2.1); grey triangles are pc-reps; see also Fig. 2B. and Fig. 9

1.1 Background

For the present paper, a *polygonal surface*, or simply *surface*, is composed of a set of vertices and a set of triangles, wherein 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 stars of the edge endpoints. To provide an intuitive surface model, 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 [8]): this assumption is also used in Section 2.4.

[1] IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, gueziec@watson.ibm.com

A *boundary edge* is such that only one triangle is incident to it. 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).

Background on surface simplification can be consulted in the bibliography of this paper. The principle is to produce a lower detailed surface approximation of a higher detailed surface. Several automatic methods have been proposed. In particular, a number of methods simplify a surface by applying a succession of edge collapses; they differ in the strategy used for collapsing edges. For instance Ronfard [1] and Guéziec [2] order the potential collapses according to different measures of the deviation from the original surface that results. Then, in that order, Guéziec performs only those collapses such that the resulting surface will not deviate more than a specified tolerance from the original. Optionally, he preserves the volume enclosed by a surface without boundary. Other criteria have been shown to be effective as well: Hoppe [3] defines a surface energy based upon pairwise vertex distances and attempts to minimize it.

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 [4], except that their parents and children do not form a partition of the vertices. Hoppe's data structures [9] 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 $\Delta(\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3})$ 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.

Subsequently each vertex, edge and triangle are replaced with their representative. For instance, the red triangles in Fig. 2 become larger as Vertex $\mathbf{v_1}$ is identified to its representative Vertex $\mathbf{v_2}$. Although the surface has changed (has been simplified), there is an onto mapping between the original surface and the simplified surface. There is no need to create 2 new triangles, we can as well use two existing triangles from the original surface, and before painting them, remember to replace Vertex $\mathbf{v_1}$ with Vertex $\mathbf{v_2}$: this is the information that is consigned in the representative array.

The original motivation for defining representatives was an efficient method for computing vertex stars during simplification by "pivoting" about a vertex representative. One 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 [10], 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.

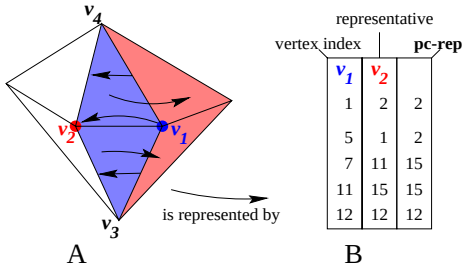

Figure 2: A: during an edge collapse blue Vertex $\mathbf{v_1}$ and blue triangles are removed; vertex, edge and triangle representatives are assigned as arrows indicate. B: vertex representative array, and path compressed representative (pc-rep) array for some vertices of Fig. 3.

Vertex representatives implement a particular map between vertices of the original surface and vertices of the simplified surface that is a *simplicial map* in the sense of Agoston [11]: each *simplex* (triangle, edge, vertex) in the original surface maps to a simplex in the simplified surface. Moreover, the vertex representatives array encodes a succession of simplicial maps between the original surface and each level of detail. Using these maps, we can specify each LOD using triangles of the original surface, and can partition the surface in batches of triangles corresponding to particular LODs.

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).

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, starting from the highest to the lowest level (red vertices and tri-

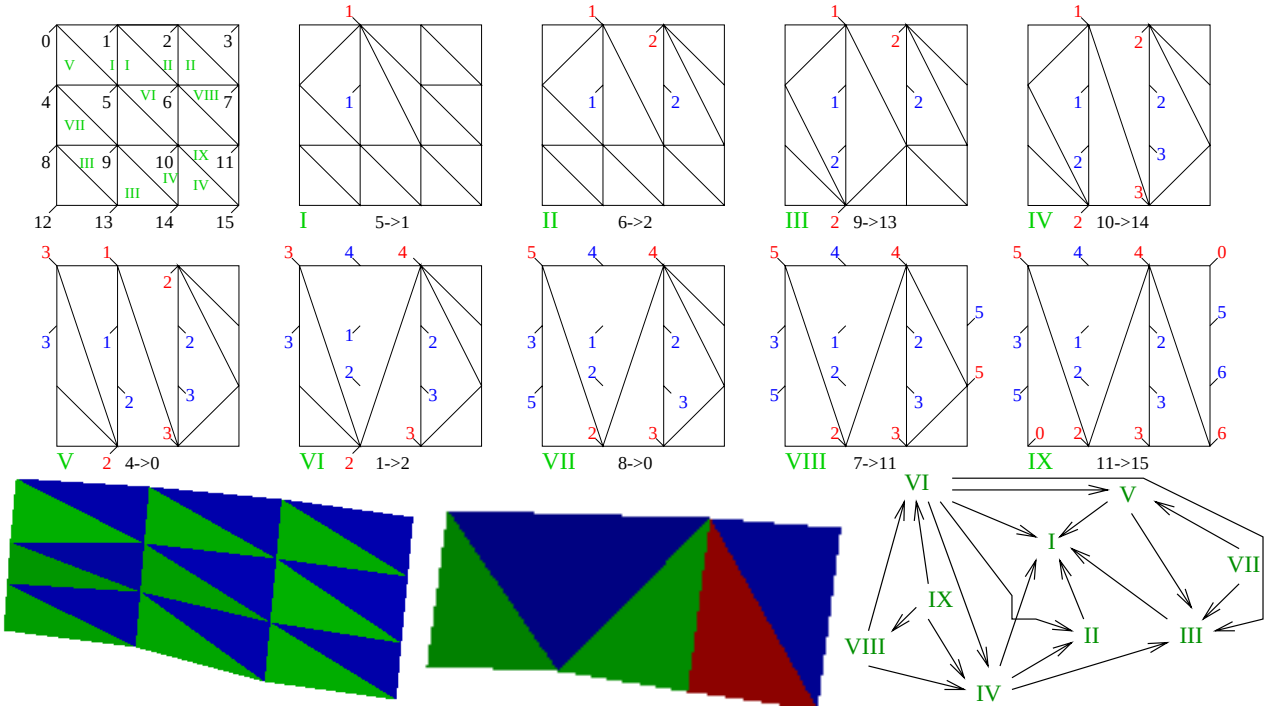

Figure 3: Simple surface: edge collapses, numbered I through IX, affect the levels of the red and blue vertices as shown. Triangles removed during the collapses are shown on the top left. The Level 1 and level 7 surfaces are shown below (colors are irrelevant), together with the DAG representing the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

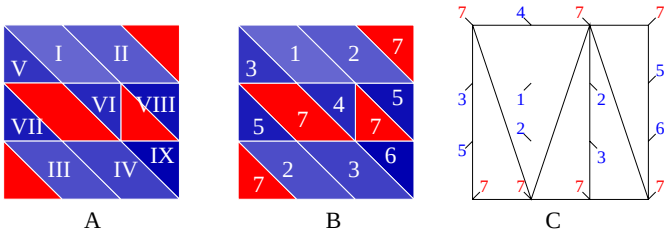

Figure 4: Simple surface: A: labeling blue triangles according to the collapse number. B: partitioning the surface triangles in 7 regions, using blue vertex levels; triangles with label i are used in surface LODs $i, i+1. \ldots, 7$; together with the vertex partition and vertex representatives, we define a Surface Partition. C: after the last edge collapse, all red vertices are assigned Level 7.

angles have the highest level), 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 [10].) 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 ignore the representatives of the vertices of level 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 ignore the representatives of the vertices marked 5; and so on. In Fig. 5 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 ignoring 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 [3], 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 Transmission or 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 and 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 of level $i, i = k-1, \cdots, i = 1$. We call this representation 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

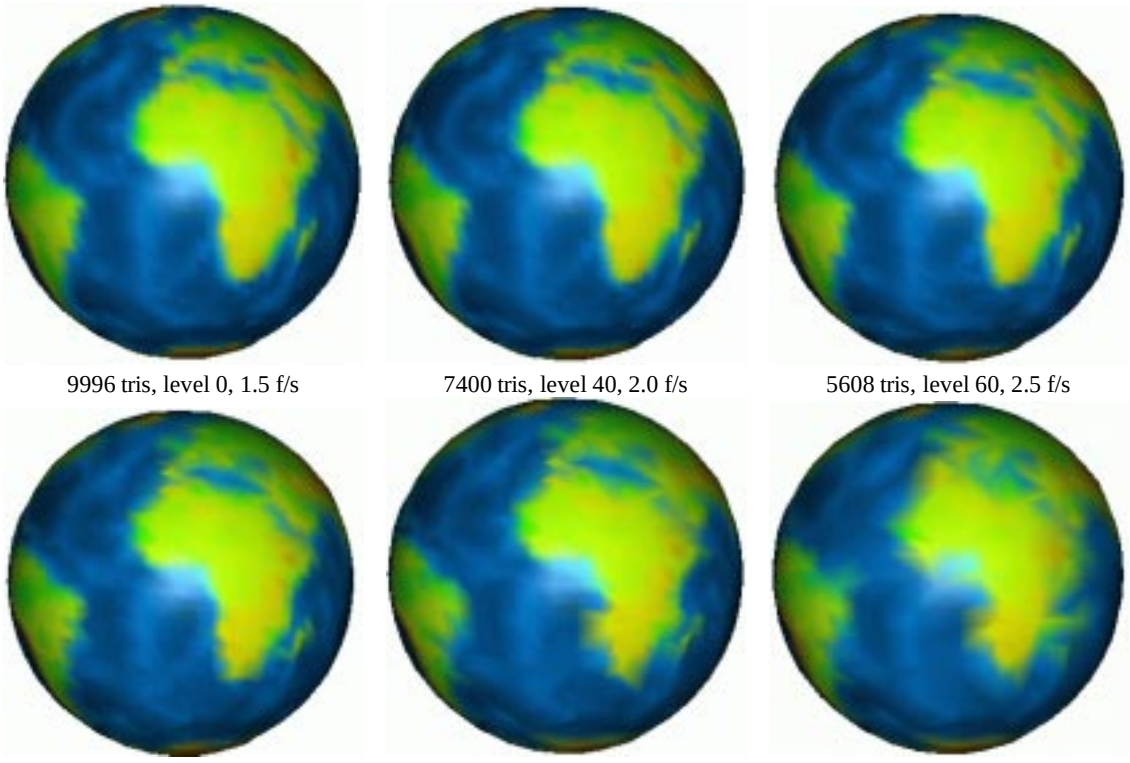

Figure 5: Successive LODs of the Earth model; for each LOD, we report the number of frames per second (f/s) during animation using an IBM POWER Gt4xi Graphics Adapter. The LODs were generated in 10.5 seconds CPU on an IBM 58F workstation.

levels are implicit, as each surface portion has a level corresponding to the order in which it is loaded (or 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 $40/36 \simeq 1.1$.

Exemplary File Format Recall that consecutive levels can be combined to form a single level. This property is useful to form levels comprising batches of triangles of similar sizes. Figure 7 shows an exemplary file containing all the information necessary to distribute the Simple surface progressively in three different LODs. Recall that after assigning levels to vertices and triangles, the vertices and triangles are re-enumerated according to their levels. For the Simple surface, new vertex indices resulting from this process are shown in Fig. 6. The vertex representative array must be recomputed to reflect the new indices. There are no representatives for Vertices 0 through 6, which are at the highest levels. Hence the representative array lists representatives of Vertices 7 through 15: {6, 0, 7, 1, 0, 5, 1, 4, 10}. The representative array, which encodes a collection of simplicial maps as mentioned above, can be specified between the first batch of vertices and the first batch of triangles. The representative array can also be send progressively as

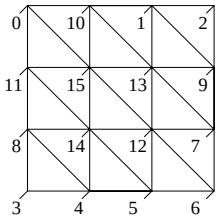

Figure 6: Simple surface: vertices are re-enumerated according to their level, starting with higher levels.

representatives are required. This is shown in Fig. 7. For instance, as there are seven vertices in the first vertex batch, when writing the first triangle batch, each vertex index larger than six must be followed with a representative, which will be followed with its representative if larger than six, etc. Clearly, representatives need only be specified once. Note that in Fig. 7, the entire representative array was specified in the first batch; this is not the general case.

Red Vertices are Allowed to Move We now drop the requirement of fixing red vertices. This offers the possibility of moving the vertex resulting from the edge collapse. Returning to Fig. 2, when Vertex $\mathbf{v_1}$ collapses to Vertex $\mathbf{v_2}$, instead of keeping the original position of Vertex $\mathbf{v_2}$, we may as well choose a position for which the deviation is minimized, or the face normals are less perturbed, or the overall volume is preserved. In practice, it is frequent for methods using edge collapses to displace the vertex resulting from the collapse.

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

```
#BATCH 1
{#coordinates of vertices at level 7
 x y z,  #v0
 ...
 x y z,  #v6
}
{#triangles at level 7, followed with map
  1, 9, 2,-1,           7, 6,
  7,13,12,-1,           1, 5,
 11,14,15,-1,           0, 4, 10, 1
 12,15,14,-1,
  3, 4, 8,-1,           0,
}
#BATCH 2
{#coordinates of vertices at levels 6 through 3
 x y z, #v7
 ...
 x y z, #v12
}
{#triangles at levels 6 through 3; no map is necessary
  6, 7,12,-1,
  8,14,11,-1,
  7, 9,13,-1,
 12,13,15,-1,
  0,11,15,-1,
  5,12,14,-1,
  5, 6,12,-1,
}
#BATCH 3
{#coordinates of vertices at levels 2 and 1
 x y z, #v13
 x y z, #v14
 x y z, #v15
}
{#triangles at levels 2 and 1; no map is necessary
 10,13, 1,-1,
  1,13, 9,-1,
  8, 6, 9,-1,
 14, 4, 5,-1,
  0,15,10,-1,
 10,15,13,-1,
}
```

Figure 7: Exemplary ascii file for distributing the Simple surface in three progressive LODs; batches of triangles and vertices correspond to LODs; the symbol # precedes a comment.

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. As the blue vertex cannot be collapsed further (it is not referenced in any edge), this position is "safely" stored, and will not be overwritten. When increasing the levels of detail (ignoring the vertex representatives), we restore the red position of the representative. Because this is performed in order of decreasing levels from highest to lowest, 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 (There are several possibilities here: essentially, for each vertex of the DAG, we want to have a list of all the directed edges that enter the vertex and all the directed edges that exit the vertex). 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 [9] 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. Our approach is also related to De Floriani *et al.*'s [7] who also use a DAG to represent local surface updates. However, our representation is simpler, as each vertex represents exactly one edge collapse, and the directed edges represent dependencies between collapses.

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 influences 1. 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 [2] 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 a display list, represented for instance using a doubly linked list of triangles.

```
#VRML V2.0 utf8
PROTO ProgIfs
[
 field    SFString urlData
 eventIn  SFFloat  setLevel
]
{
  DEF ifs IndexedFaceSet {
    coord Coordinate { }
  }
  DEF script Script {
    url      "ProgIfs.class"
    field    SFString urlData  IS  urlData
    field    SFNode   ifs      USE ifs
    eventIn  SFFloat  setLevel IS  setLevel
  }
}
Shape {
 geometry ProgIfs {
   urlData ''sample.data''
 }
}
```

Figure 8: Exemplary VRML file describing a progressive IndexedFaceSet.

4 Vrml Implementation

In this section we briefly describe our VRML2.0 implementation, which is based on defining a new node using the `PROTO` mechanism and Java in the script node for the logic. Figure 8 shows a typical VRML file. The new Node behaves as an IndexedFaceSet, has the `URL` of the file containing the data as the only `field` (instance variable) to be set up when the node is instantiated, and one `eventIn` used by the browser to request a new level of detail.

The Java program in the script node performs two functions. Upon instantiation, a thread is started to download the data from the `URL` provided in the `urlData` field, immediately returning control to the browser. After the browser regains control, it can request a change in level of detail by sending a `setLevel` event to the node. The changes in level of detail of the IndexedFaceSet node are handled by the main thread of the script code.

The download thread progressively downloads the total number of levels of detail, vertex, triangle, and attribute data, and periodically updates the corresponding arrays (triangle array, vertex pc-rep array, vertex representative array, and optionally attribute arrays). These arrays, which are private to the script code but persist after the download thread finishes, are used later on by the main thread of the script code to update the IndexedFaceSet fields responding to browser requests. The main thread does so by setting and changing values of the `coord` and `coordIndex` fields (and optionally of the other attribute fields), as a function of the data downloaded so far by the download thread, and the requested level of detail.

In our implementation the `setLevel` value is first clipped to the currently valid range as a function of the data downloaded so far by the download thread. If after the clipping `setLevel` has not an integer value, the vertex coordinates are interpolated between the two consecutive integer values immediately preceding and succeeding the given value. This step increases the complexity of the script code, but provides support for smooth transitions between consecutive levels of detail. Alternatively, the `setLevel` event should be defined as `SFInt32`. Note that we have decided not to show in figure 8 the VRML logic necessary to trigger the change in level of detail events. This can be done in many different ways, depending on the application. For a simple demo, a TimeSensor can be set up to periodically send *setLevel* events to the node with cycling values.

To show the progress of the download thread, the script code can be modified to automatically update the IndexedFaceSet fields with the highest resolution level of detail available as soon as all the data associated with it finishes downloading. Alternatively, the node could behave as a `LOD` node instead, with the script adding new levels the the `LOD` node as soon as they are available. However, the smooth transition between levels would be more difficult to implement.

5 Conclusion

We have described a new scheme for partitioning a surface in Levels of Detail (LODs), using the output of any known algorithm that performs edge collapses [1, 2, 3, 4]on a surface.

For global LODs, our representation simply consists of a batch of surface portions, each surface portion being represented as usual with point positions and an indexed face set. Each surface portion corresponds to an increment from one level to the next. In addition, one array of vertex representatives is provided. Vertex indices are replaced with their representatives before display. We have described a preliminary implementation of this representation in VRML. We thus provide a simple and efficient framework for progressive transmission and display of polygonal surfaces.

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] A. Guéziec. Surface simplification with variable tolerance. In *Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery*, pages 132–139, Baltimore, MD, November 1995.

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

[4] 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.

[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 and C. Erikson. View dependent simplification of arbitrary polygonal environments. In *Siggraph*, pages 199–208, Los Angeles, August 1997. ACM.

[7] L. De Floriani, P. Magillo, and E. Puppo. Building and traversing a surface at variable resolution. In *Visualization 97*, pages 103–110. IEEE, 1997.

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

[9] H. Hoppe. View dependent refinement of progressive meshes. In *Siggraph*, pages 189–198, Los Angeles, August 1997.

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

[11] M. K. Agoston. *Algebraic Topology. A First Course*. Pure and Applied Mathematics. Marcel Dekker, Inc., New York, 1976.

[12] L. De Floriani, B. Falcidieno, and C. Pienovi. Delaunay-based representation of surfaces defined over arbitrarily shaped domains. *CVGIP*, 32:127–140, 1985.

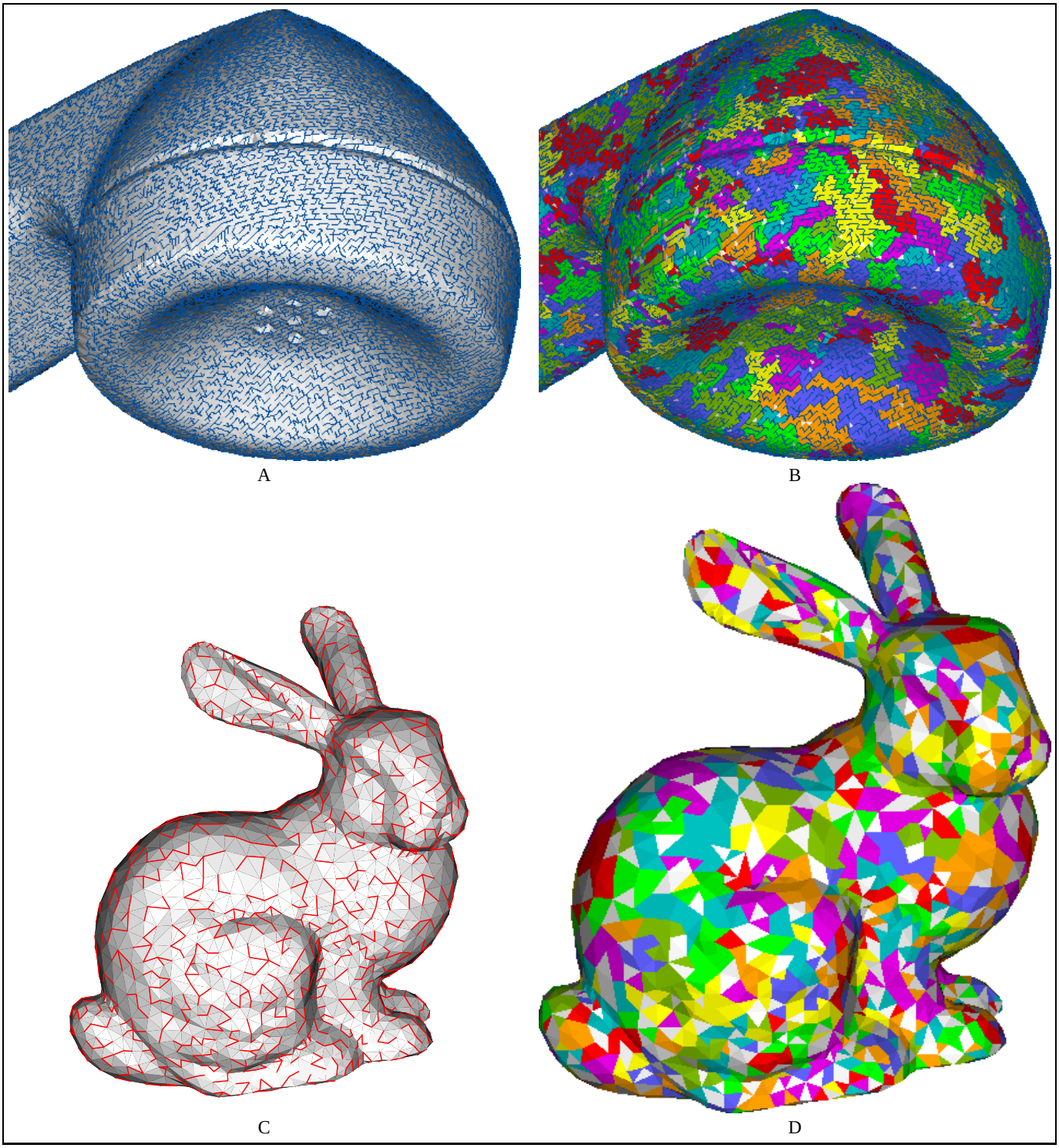

Figure 9: This figure complements Fig.1. A: Phone model: we show a forest of edges (in blue) such that each tree collapses to exactly one vertex with Guéziec's method. B: we use colors to indicate which are the triangles (in grey) that are their own representative after simplification: the simplified surface is obtained by taking these triangles and replacing their vertex indices with their representatives. Triangles of a given color have the same representative. The colored triangles are the ones that are removed during simplification. C: same as A shown on a simplified version of the Bunny model. D: same as B on the model of C.

[13] C. Bajaj and D. Schikore. Error-bounded reduction of triangle meshes with multivariate data. volume 2656, pages 34–45. SPIE, 1996.

[14] J. Cohen, D. Manocha, and M. Olano. Simplifying polygonal models using successive mappings. In *IEEE Visualization*, pages 395–402, 1997.