EUROGRAPHICS '96 / J. Rossignac and F. Sillion (Guest Editors), Blackwell Publishers

Volume 15, (1996), Number 3

Full-range approximation of triangulated polyhedra.

Rémi Ronfard [1] and Jarek Rossignac

IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598

Abstract

We propose a new algorithm for automatically computing approximations of a given polyhedral object at different levels of details. The application for this algorithm is the display of very complex scenes, where many objects are seen with a range of varying levels of detail. Our approach is similar to the region-merging method used for image segmentation. We iteratively collapse edges, based on a measure of the geometric deviation from the initial shape. When edges are merged in the right order, this strategy produces a continuum of valid approximations of the original object, which can be used for faster rendering at vastly different scales.

Keywords: Vizualisation, Polyhedral Surfaces, Levels of Detail, Region Merging.

1 Introduction

This paper describes a new method for automatically generating several levels of details of a polyhedral object, whose faces have been triangulated [9]. Our method is applicable to architectural walk-through, assembly mock-up, virtual reality, medical simulation, planning and monitoring. The difficulty in those applications is that a wide range of viewing conditions must be accommodated, so that the available levels-of-details should span several orders of magnitude [5, 12].

There are two sides to all simplification methods. On one side, it is desirable to be able to simply iterate the process until a desired level of detail is reached (in terms of the number of elements in the approximation), without having to guess for global parameters. Thus, incremental methods remove triangles from a 3D object, based on a measure of how much the shape changes locally in each move [15, 13, 16, 14]. On the other side, global control of the simplification error is also useful, and has been emphasized by recent authors [8, 2, 7, 6, 4].

Our method takes an intermediate view of the problem. It is based on local, incremental operations, but keeps track of the initial object, by tracing a *history* of all vertex moves. As a result, we can control approximation levels by prescribing geometric tolerances, and still get the benefits of working incrementally.

The previous work of Rossignac and Borrel [11] is based on space-partionning and clustering techniques in 3D space. By allowing topological changes, that method yield much higher compression ratios than most other previous work, at an extremely low computational cost.

[1] Current address: Institut National de l'Audiovisuel, 4, ave. de l'Europe, 94 366 Bry-sur-Marne, France.

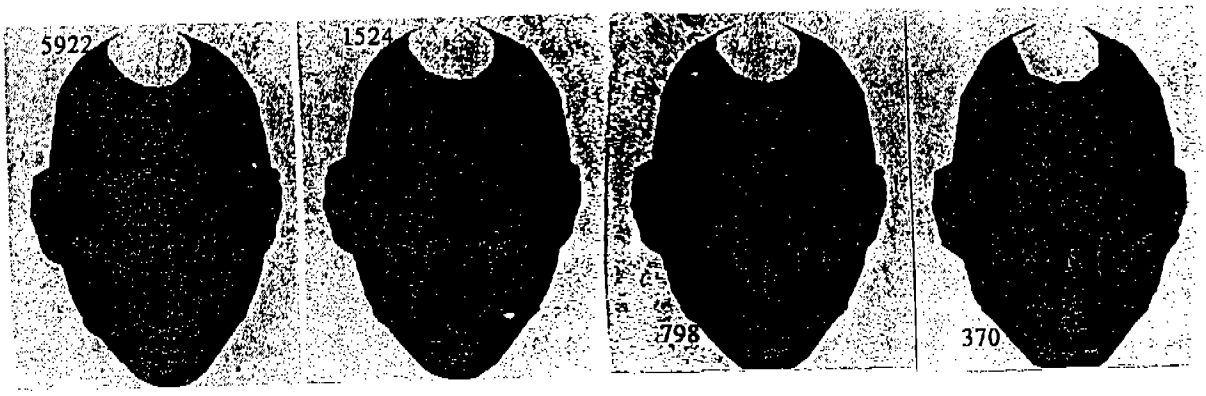

Figure 1: Surface of a human face, extracted from MR image data, with three levels of simplification, and the number of triangles for each level.

but with too little control over the quality of the approximation. In our new method, vertices are merged in a much more controlled way, based on a step by step evaluation of the approximation error.

The multiresolution approach advocated by Eck et al. [3] gives a suitable theoretical framework to surface simplification based on *wavelets*, but it is not yet clear that it is practical, because it cannot deal with surface discontinuities or topological simplifications.

The paper is organized as follows. First, we present a general overview of the algorithm in section 2. Then, we discuss our definition of the approximation error, in section 3. Implementation issues are discussed in section 4, with a focus on data structures and topology issues. We end with a discussion of results and performance analysis.

2 Algorithmic foundations of the method.

Our method is based on a merging algorithm, which removes edges one at a time from a polyhedral object. Each edge removal is applied by moving the first vertex into the second vertex of the edge, as shown in fig. 3. We view this operation as a *region* merge, because all triangles around the merged vertices are modified. We associate a cost with each merge, which can be precomputed and stored for all edges. The cost function itself will be described in section 3. We also maintain the topological links between vertices, edges and triangles throughout the simplification process. As a general overview, we can outline our method as follows:

Pre-processing. Build the topology of the object, and compute the costs associated with all edges. Insert edges into a priority queue, so the the edge with the lowest cost is readily available at all times.

Iteration. Merge edges one at a time, by moving their first vertex at the location of the second vertex, and update the topological data structure accordingly.

Relaxation. After each merge, update the geometric location of the remaining vertex, based on its new neighborhood. Update the values of the cost function for all edges that have been affected, and update the priority queue accordingly.

Topology reconstruction. When the number of vertices, or the total approximation error, reach pre-specified levels, intermediate approximations of the object are extracted, and the whole process can be iterated until the full range of approximation levels have been obtained.

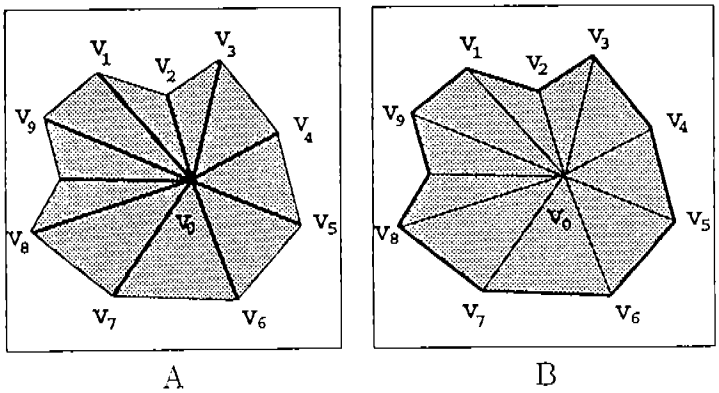

Figure 2: Star and crown of a vertex. A: Edges and faces touching a vertex are its *star*. B: Edges bounding the star of a vertex are its *crown*.

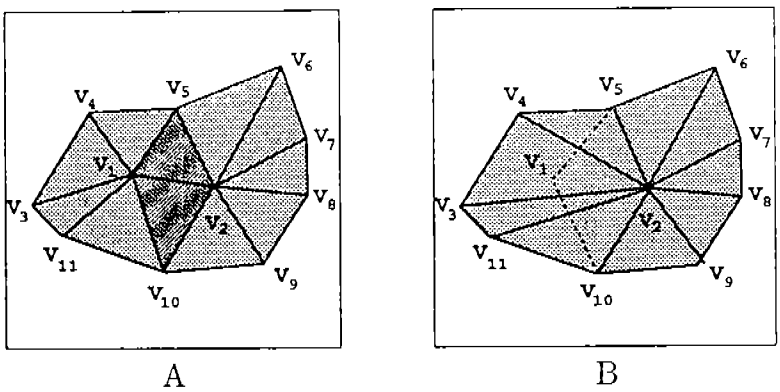

Figure 3: Merging vertices and their stars: A: Initial configuration around edge. B: Final configuration around collapsed vertex.

3 Geometric foundations of the method.

Local tesselation error.

We call the region composed of all edges and triangles around a given vertex its *star*. By definition, the neighborhood of a vertex is composed of edges and triangles in its star, as depicted in fig. 2. In the same figure, the *crown* of a vertex is introduced, consisting of the boundary of its star (i.e. the set of edges adjacent to, but not part of, its star).

Our single topological operation for simplifying polyhedral shapes consists in merging the stars of two adjacent vertices. Fig. 3 shows a typical example of the region-merging operator, where one vertex V_1 is merged into another vertex V_2.

We evaluate the cost of merging vertices, with the *maximum* of two functions of those vertices, LGE and LTE. The local geometric error LGE is the variation of a geometric error

G during the merge (a measure of the distance between the initial and approximated shapes). The local tesselation error LTE is a penalty function, used to keep the triangular mesh valid and smooth. With reference to fig. 3, the merge involves deleting vertex V_1, edges V_1V_2, V_1V_5 and V_1V_{10}, as well triangles $V_1V_2V_5$ and $V_1V_2V_{10}$. Four more triangles are modified, and a cost LTE is associated with the amount of rotation undergone by their normal vectors. In addition, vertex V_1 moves at a distance $\|V_1V_2\|$ from its previous location, thus increasing the global geometric error in the approximation by an amount LGE.

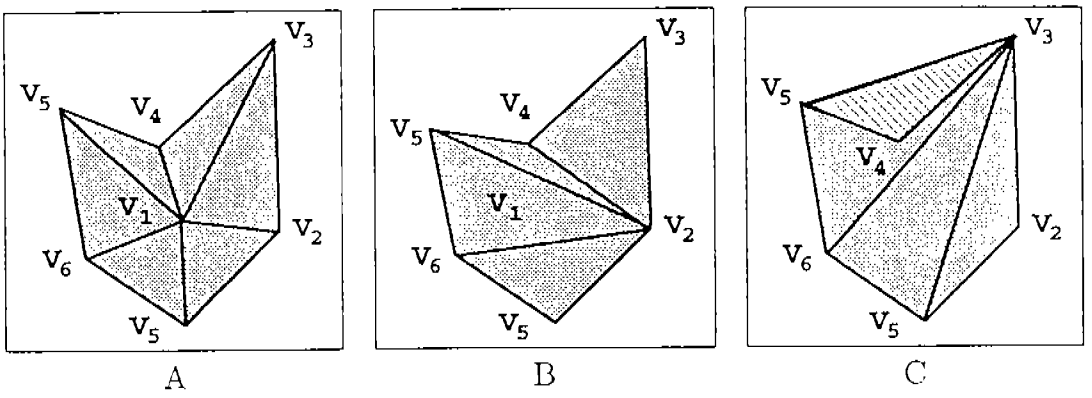

Figure 4: Enforcing validity constraints on a triangular mesh. A: Initial configuration. B: $V_1 \rightarrow V_2$ is an authorized move: all triangle normals keep their orientation. C: $V_1 \rightarrow V_3$ is a forbidden move: one triangle ($V_1V_4V_5$)has reversed its orientation.

A triangular mesh defined on a plane or a surface is a valid tesselation when no edges intersect, and no triangles overlap on that plane or surface. In our case, the surface is not explicitly given, but the validity of the mesh can still be approximately evaluated, if we assume that the initial shape is a valid mesh. When we merge vertices, we deform triangles in their star. For each triangle $V_i.V_j.V_k$, the deformation can be represented by the rotation of the normal vector to the plane of the triangle, and measured with just one angle A_{ijk} between 0 and π.

The validity and the quality of the mesh are violated when a triangle reverses its orientation, i.e. when it turns around the surface of the object with an angle equal to π. In the planar case, an example of this situtation is given in fig. 4. To prevent such cases, and to keep the mesh as balanced and smooth as possible, we impose the penalty function:

$$LTE(V_1.V_2) = K \max_{i.j.k \in star(V_1) i.j.k \notin star(V_2)} A_{ijk}$$

With a large coefficient K, this function can be used to prevent *flipping* triangles tangent to the surface of the object, if we use a data structure allowing us to pre-compute LTE (without merging). Color plates 7 and 8*show the mesh at different stages of the simplification process, on two simple examples.

Local geometric error.

We have devised a novel geometric error function, based on the distance of vertices as they move perpendicular to the surface of the initial object. We use this geometric error as a cost function in combination with the penalty function, which puts constraints on how vertices move tangent to the surface. As a result, all vertices that have been merged with costs inferior to a given tolerance remain within this tolerance of the original shape (see [10]).

* See page C-462 for figures 7 and 8

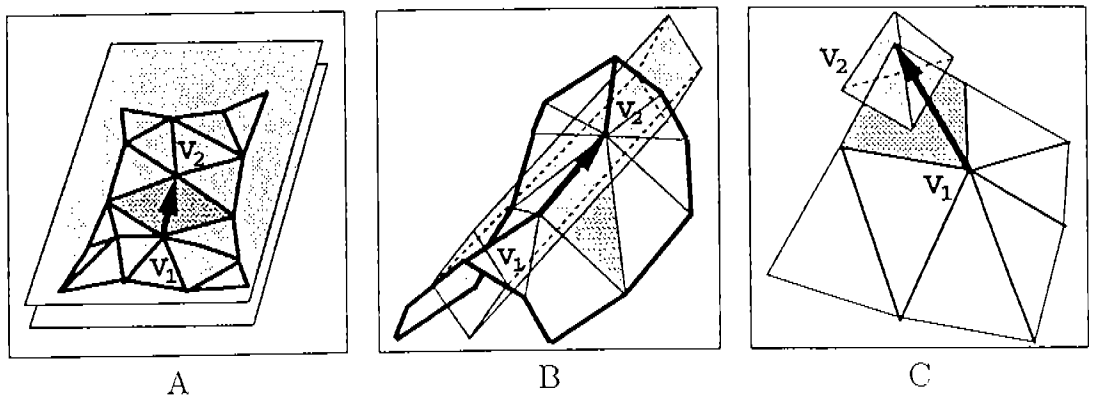

Figure 5: Special cases for simplification. The shaded regions materialize zones with zero, one or two degrees of freedom around vertex V_2. A: Merge almost coplanar vertices, two degrees of freedom. B: Merge vertices aligned on a sharp edge, one degree of freedom. C: Merge vertices into a sharp corner, zero degree of freedom.

Intuitively, the different ways of merging vertices without changing the shape of the object are (a) to merge vertices whose regions are entirely co-planar; (b) to merge vertices along prominent edges, where the dihedral angle is greatest; (c) to merge vertices into prominent corners (see fig. 5). In order to deal with those three cases simultaneously, we measure the deviation of the merged vertices from their *tangent planes*, i.e. a finite set of planes around each vertex. We start with a simple observation: in the original object, every vertex is a solution of a linear system of equations, obtained with the plane equations for all the triangles in its star. Let us write this system, with π_k denoting any such plane:

$$\pi_k(V_i) = a_k x_i + b_k y_i + c_k z_i + d_k = 0$$

If we move a vertex V_i to an arbitrary new position (x, y, z), we can associate costs C_k proportional to the distances from those planes to the new position, i.e. $C_k = \pi_k(x, y, z)$.

As a generalization, we associate an arbitrary set of planes, called its *zone*, to each vertex in the object. Initially, those planes are picked from the star of the vertex, with a total cost of zero. If we now move one vertex, there will be a corresponding deviation $\pi_k(V_i)$ for each plane in its zone. If we *merge* vertices, we will also update their zones.

In order to maintain a constraint such as $\pi_k^2(V_i) \leq \epsilon$, we need to confine the vertex V_i in a *slab* around plane π_k. In order to maintain *all* the constraints in its zone, the vertex must be confined in a *region* which is the intersection of all the slabs in the zone. The exact shape of this region depends on the neighborhood of the vertex to be merged. There are three possible cases. In the case of a planar neighborhood, the region is just a single slab with an infinite extent because there is only one equation (see fig. 5-A). In the case of fig. 5-B, the region is an infinite tube around a sharp edge, because there are only two different equations in the zone. In most other cases, however, the vertex is really confined in a finite region, because three or more equations are in its zone. This is the case of the corner in fig. 5-C.

We define the geometric error function G around a vertex V_i as the maximum distance from V_i to planes in the zone of V_i. When vertex V_1 is merged into vertex V_2, the new region inherits plane equations from the zones of the two merged vertices. The local geometric error LGE is the variation of the error, therefore $G^{after} = \max\{G^{before}, LGE(V_1, V_2)\}$. Because we merge vertices by rank of increasing geometric errors, the maximum error after the merge is necessarily one of the new constraints in $zone(V_1)$. Therefore, an equivalent definition for LGE is simply:

$$LGE(V_1, V_2) = \max_{j \in zone(V_1)} \pi_j^2(V_2)$$

In other words, LGE is the largest new constraint associated with the vertex being merged. Note that we always choose the final vertex position to be V_2 itself, i.e. we only use existing vertices at this point. Using a combination of LGE and LTE as a cost function, we can merge edges with increasing costs, first in the smoother regions of the object, then along sharp edges, and finally into corners in the object (see examples showing the triangular mesh in fig. 7 and fig. 8 in the color plates).

4 Implementation details.

Vertex-based data structure.

We have devised a vertex-based data structure with easy access to edges around a local center, enabling us to compute the local cost functions as efficiently as possible. The local tesselation error LTE involves edges in the crown of a vertex. The local geometric error LGE involves edges in its star, as well as equations in its zone.

We therefore represent each vertex as a triplet: its star, its crown and its zone. The star contains a list of edges incident to the vertex, which means that we maintain a complete representation of the graph of vertices and edges. In addition, the crown specifies an imbedding of that graph on a particular surface: each edge in the crown corresponds to one triangle on the surface. The zone contains the history of the approximation, and relates its vertices to their ancestors in the original object. The construction of this dynamic data structure is quite straight-forward, and is described in much more detail in our research report [10].

Edge ordering based on local errors.

In order to merge edges with increasing costs in the right order, we maintain an auxillary data structure, in the form of a heap of directed edges. We associate a priority to each directed edge, which is just the opposite of its cost function. We use the heap as a priority queue, such that (a) the first element in the queue always has the highest priority, and (b) we can efficiently remove and update edges when their priority changes. The heap is a particularly efficient implementation of the priority queue, as a balanced, partially ordered tree [1]. The heap is easily updated when an edge priority changes, by *bubbling* up (if the priority increases) or down (if the priority decreases). The heap is initialized by reading the triangulation data sequentially. Its elements are updated as we merge edges, as we will now explain in more details.

In each move, we update the local of the dynamic graph structure around the merged vertex, as well as the heap, and the position (coordinates) of the merged vertex. The heap updating operations consist in removing some edges (two opposite directed edges at a time), and reordering the heap based on new priorities, for those edges that have been modified but not deleted. Edges pointing from V_2 must be recomputed, because of the additional constraints inherited from V_1. Edges pointing toward V_2 need only be recomputed if we also update the position of V_2 at this point.

Relaxation.

After each merge, it is possible to optimize the coordinates of the central vertex, based on the (fixed) coordinates of its new neighbors. The general idea is to minimize an energy function

that will make the triangular mesh as smooth and balanced as possible, while keeping the geometric errors as small as possible.

An interesting choice for the relaxation energy is taken directly from the geometric error function, except that we now use the sum of errors, rather than the maximum error, i.e.

$$E_R(V_2) = \sum_{j \in zone(V_1)} \pi_j^2(V_2)$$

Deforming the object locally by moving V_2 into its *minimum* local error, enhances the method at all scales. We must acknowledge that a more elegant solution would incorporate that optimization into the estimate of the merging costs, i.e. we should use E_R as our local geometric error, and the optimized value of V_2 in our cost function. But this would be terribly inefficient, and we prefer to apply those two steps separately. As a heuristic, we have found that the two cost functions worked together quite nicely, because the optimized value for V_2 was never very far from its initial position, and the prediction used for LTE and LGE remained correct.

Local topology changes.

Collapsing an edge (V_1, V_2) can change the topology of the object. For instance, a through-hole may become filled-up (in topological terms, *handles* may be removed, e.g. a torus may be transformed into a sphere). Vertices sharing an edge with both V_1 and V_2 play a very important role in this step of our algorithm. Their number characterizes the local topological structure of the object (zero at a wireframe edge, one at the border of an open surface, two inside a closed surface, three or more on a surface with self-intersections). Our vertex-based data structure allows all those cases to be represented, which is a key to changing the topological genus of the shape, i.e. removing holes, handles, and separating components [2, 7, 6]. Our dynamic data structure can tolerate such changes, because it is based on vertices, and vertices can be incident to arbitrary numbers of edges and triangles. In many cases, topological simplification is in fact acceptable, and we have used it to achieve the higher compression ratios in most of the examples shown here.

While we are able to correctly reconstruct approximations in presence of topological changes, we do not know how to re-organize constraints. For instance, if several components are separated, which constraints should be associated wth each component ? In those cases, an appropriate enhancement of our method would consist in resolving the new structure, so that the constraints for each of several new components could be separated, and the constraints which came from filled-up holes could be eliminated. This appears to be a difficult issue in general, and is not a part of our current implementation. As a result, we keep too many constraints in those cases, and the approximation process artificially slows down in those regions.

5 Results and comments.

As an input, the algorithm is given a number L of levels and a geometric tolerance for each level (ϵ_1 , ϵ_2 ..... ϵ_L). Each tolerance can be expressed as a percentage of the object size, or as a multiple of the initial costs found during the pre-computing step. As an alternative, the numbers of triangles in each level may be pre-specified (N_1 , N_2 , ... , N_L).

A typical example is shown fig. 6 (top), with a maximum compression ratio of 1:80, leading to just under a hundred triangles. Subtle topological changes occur betwen the second and third approximation levels; holes disappear, antennas retract, eventually leading to the almost minimal representation of the fourth level of the figure.

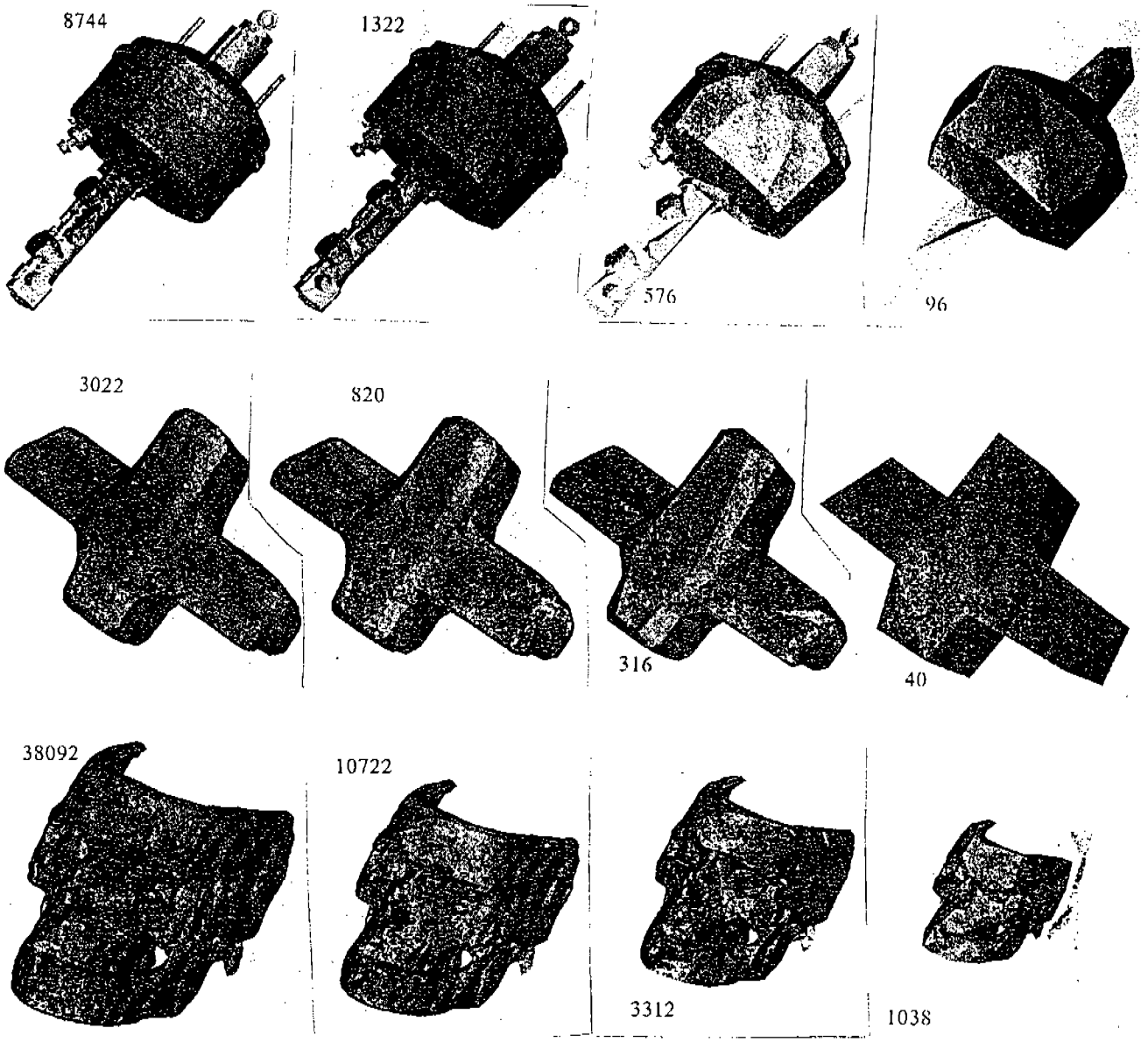

Figure 6: Top: Power brake assembly. Middle: Radiation iso-dose surface. Bottom: Skull surface extracted from X-ray scanner image data. Three levels of approximation for each example, and corresponding numbers of triangles.

We also show results obtained with iso-surface data (courtesy of G. Turk [15]) in fig. 6 (middle). Our results are comparable to previously published work, only with a much wider range of approximation levels. Examples in fig. 1 and 6 (bottom) are applications of our method to medical data. Iso-surface extraction methods of that sort typically generate results with hundreds of thousands of polygons, and efficient data reduction methods are therefore particularly important for dynamic simulations, surgery planning and other related applications. Although our approach is heuristically based on only a partial measure of the approximation error, we are able to control the tolerances at each level, so that no vertices move further away from their initial configuration than 0.1, 0.5 and 1.0 per cent of the image size. As can be seen in fig. 6 (bottom), very good approximations can be obtained at intermediate compression ratios (1:10 to 1:20), while much coarser approximation result at further stages. The initial shapes were extracted from MR and CAT-scan data as iso-surfaces

(courtesy of A. Guéziec).

We estimate that the complexity of the method is $N_0 \log^2 \frac{N_0}{N_L}$ for bringing the number of vertices down from N_0 to N_L triangles. In our analysis, we consider the heap operations, which are approximately logarithmic, and the tests on all the equations in the vertex zones. Because equations are inherited, their number is constant thoughout the simplification process, hence the result. In practice, the computing times for all examples run from a few seconds to several minutes on a workstation.

6 Conclusion.

We have presented a new method for simplifying three-dimensional shapes, based on an particular measure of the approximation error, that we derive from simple plane equations. Our approach allows us to reach very high compression ratios in a variety of cases, while keeping the general appearance of the original shapes. It has been noted by other researchers that an important part of obtaining minimal approximations of shapes was the elimination of small features. No previous method has addressed that issue successfully. Our method shows that feature elimination can be achieved, partly at least, with purely geometric reasoning (i.e., without any detection or even understanding of the eliminated features), if we allow the topology of the shape to be modified.

References

[1] A.V. Aho and J.D. Ullman. *Foundations of Computer Science*. Computer Science Press, 1992.

[2] T. DeRose, H. Hoppe, T. Duchamp, W. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics (Proc. SIGGRAPH)*, pages 71–78, 1992.

[3] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsberry, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. *Computer Graphics (Proc. SIGGRAPH)*, pages 173–182, 1995.

[4] A. Guéziec. Surface simplification with variable tolerance. In *Second Annual Symposium on Medical Robotics and Computer Assisted Surgery*, 1995.

[5] P.S. Heckbert and M. Garland. Multiresolution modeling for fast rendering. *proc. Graphic Interface*, pages 43–50, 1994.

[6] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, W. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. *Computer Graphics (Proc. SIGGRAPH)*, pages 295–302, 1994.

[7] H. Hoppe, T. DeRose, T. Duchamp, W. McDonald, and W. Stuetzle. Mesh optimization. *Computer Graphics (Proc. SIGGRAPH)*, pages 19–26, 1993.

[8] T. Phillips, R. Cannon, and A. Rosenfeld. Decomposition and approximation of three-dimensional solids. *Computer Vision, Graphics, and Image Processing*, 33:307–317, 1986.

[9] R. Ronfard and J. R. Rossignac. Triangulating multiply-connected polygons: A simple yet efficient algorithm. In *Computer Graphics Forum, proceedings of Eurographics'94*, Oslo, Norway, September 1994.

[10] R. Ronfard and J. R. Rossignac. Full-range approximation of triangulated polyhedra. Technical Report 20423, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1996.

[11] J. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering complex scenes. *Modeling in Computer Graphics*, pages 455–465, 1993.

[12] J.R. Rossignac and M. Novak. Research issues in model-based visualization of complex data-sets. *IEEE Computer Graphics and Applications*, pages 83–85, March 1994.

[13] L. Schroeder, J.A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–69, 1992.

[14] W. Schroeder, K. Martin, and B. Lorensen. *The visualisation toolkit, an object-oriented approach to 3D graphics*. Prentice Hall, 1985.

[15] G. Turk. Re-tiling polygonal surfaces. *Computer Graphics*, 26(2):55–64, 1992.

[16] A. Varshney. *Hierarchical geometric approximation*. PhD thesis, University of North Carolina, Chapel Hill, 1994.