View-Dependent Topology Simplification

Jihad El-Sana and Amitabh Varshney

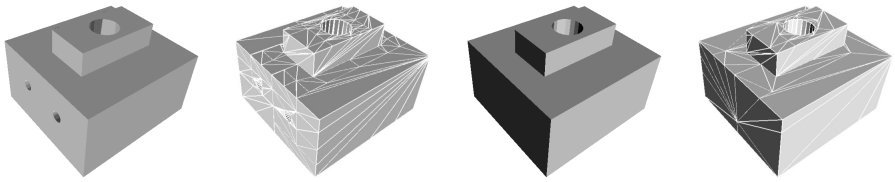
Department of Computer Science, State University of New York at Stony Brook,
Stony Brook, NY 11794-4400
{jihad,varshney}@cs.sunysb.edu

Abstract. We propose a technique for performing view-dependent simplifications for level-of-detail-based renderings of complex models. Our method is based on exploiting frame-to-frame coherence and is tolerant of various commonly found degeneracies in real-life polygonal models. The algorithm proceeds by preprocessing the input dataset into a binary tree of vertex collapses. This tree is used at run time to generate the triangles for display. Dependencies to avoid mesh foldovers in manifold regions of the input object are stored in the tree in an implicit fashion. This obviates the need for any extra storage for dependency pointers and suggests a potential for application to external memory prefetching algorithms. We also propose a distance metric that can be used to unify the geometry and genus simplifications with the view-dependent parameters such as viewpoint, view-frustum, and local illumination.

1 Introduction

Interactive visualization of large geometric datasets in computer graphics is a challenging task due to several reasons. One of the main reasons is that the sizes of several present geometric datasets are one or more orders of magnitude larger than what the current graphics hardware can display at interactive rates. Further, the rate of growth in the complexity of such geometric datasets has outpaced the advances in the graphics hardware rendering capabilities. As a result, several algorithmic solutions have been proposed to bridge this gap between the actual and desired rendering performances on such large datasets. These include visibility-based culling, geometric multiresolution hierarchies, levels of detail in illumination and shading, texture mapping, and image-based rendering. The focus of this paper is on defining geometric multiresolution hierarchies to enable a view-dependent simplification of the geometry as well as topology of the model for interactive walkthroughs of high complexity polygonal datasets.

For some graphics systems such as those used in mechanical tolerancing and medical volume visualization preservation of the topology of the input dataset is an important criterion. However, for a wide variety of real-time graphics applications where interactivity is essential, preservation of the topology of the input polygonal dataset is often not a requirement. For such applications geometry simplification has been shown to yield significantly lower complexity approximations if performed with genus simplification than without. In this paper we demon-



(a) (b) (c) (d)

Fig. 1. Genus-simplification based on light direction

strate a technique for performing topology simplifications in a view-dependent manner. In our approach a hierarchy of vertex-pair collapses is identified to form a *view dependence tree*. Appropriate levels of detail are selected from this tree at runtime to generate view-dependent simplifications. We also propose a distance metric that uses the coordinates and the normals of vertices, to define view- and light-dependent topology and geometry simplifications for polygonal environments in a unified manner.

2 Related Work

Related work on geometry simplification has been well surveyed in several recent papers [2, 3, 6, 9, 12]. In this paper we shall overview the related work in genus simplifications and view-dependent simplifications. These two categories have almost no overlap with the notable exception of [13].

2.1 Genus Simplifications

Rossignac and Borrel's algorithm [16] uses a global grid to subdivide a model. Then the vertices of one cell are collapsed to a single vertex and the polygonal mesh is appropriately updated. This approach can simplify the topology if the desired simplification regions fall within a grid cell. He *et al* [8] have used an low-pass filter to perform a controlled simplification of the genus of a volumetric objects. However, polygonal objects need to be voxelized. El-Sana and Varshney [5] perform genus simplification by extending the concept of α-hulls from points and spheres to triangles. Their approach is based on convolving individual triangles with a L_∞ cube of side α and then computing their union. The convolution operation effectively eliminates all holes that are less than size α.

Several algorithms for topology simplification are based on vertex-pair collapse method, though not in the context of view-dependent renderings. Schroeder [18] has introduced vertex-split and vertex-merge operations on polygonal meshes for modifying the topology of polygonal models. The simplification is based on the Euclidean distance and the vertex splits are performed along feature lines and at corners. Garland and Heckbert [6] present a quadric error metric that can be used to perform genus as well as geometric simplifications. The error at a

vertex v is stored in the form a 4×4 symmetric matrix. The algorithm proceeds by performing vertex-pair collapses and the error is accumulated from one vertex to the other by summing these matrices. Popović and Hoppe [15] introduce the operator of a generalized vertex split to represent progressive changes to the geometry as well as topology for triangulated geometric models. Progressive use of this operator results in representation of a geometric model as a *progressive simplicial complex*.

2.2 View-Dependent Simplifications

Most of the previous work on generating multiresolution hierarchies for LOD-based rendering has concentrated on computing a fixed set of view-independent levels of detail. At runtime an appropriate level of detail is selected based on viewing parameters and displayed. Such methods are overly restrictive and do not take into account finer image-space feedback such as light position, visual acuity, silhouettes, and view direction. Recent advances to address some of these issues in a view-dependent manner take advantage of the temporal coherence to adaptively refine or simplify the polygonal environment from one frame to the next. Since most of the work in view-dependent simplifications is closely related to the concept of progressive meshes we briefly overview them next.

Progressive meshes have been introduced by Hoppe [9] as an elegant solution for a continuous resolution representation of polygonal meshes. Progressive meshes are based upon two fundamental operators – edge collapse and its dual, the vertex split, as shown in Figure 2. In this example the vertices $n_0 \ldots n_6$ comprise the *neighborhood* of the edge pc. A polygonal mesh $\hat{M} = M^k$ is simplified

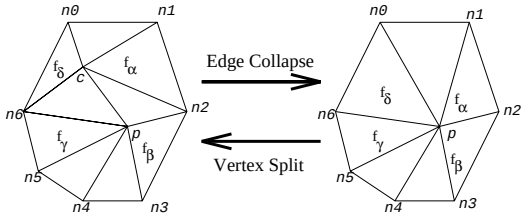


Fig. 2. Edge collapse and vertex split

into successively coarser meshes: $M^k \overset{ecol_{k-1}}{\rightarrow} M^{k-1} \overset{ecol_{k-2}}{\rightarrow} \ldots M^1 \overset{ecol_0}{\rightarrow} M^0$ by applying a sequence of edge collapses. One can retrieve the successively higher detail meshes from the simplest mesh M^0 by using a sequence of the dual transformation, vertex-split: $M^0 \overset{vsplit_0}{\rightarrow} M^1 \overset{vsplit_1}{\rightarrow} \ldots M^{k-1} \overset{vsplit_{k-1}}{\rightarrow} (\hat{M} = M^k)$

Merge trees have been introduced by Xia *et al* [19] as a data-structure built upon progressive meshes to enable real-time view-dependent rendering of an object. Let the vertex p in Figure 2 be considered the parent of the vertex c (as c is created from p through a vertex split). The merge tree is constructed in

a bottom-up fashion from the high-detail mesh $\hat{M}$ to a low-detail mesh M^0 by storing these parent-child relationships in a hierarchical manner over the surface of an object.

View-dependent simplification is achieved by performing edge-collapses and vertex-splits on the triangulation used for display depending upon view-dependent parameters such as lighting (detail is directly proportional to intensity gradient), polygon orientation, (high detail for silhouettes and low detail for backfacing regions) and screen-space projection. Since there is a high temporal coherence the selected levels in the merge tree change only gradually from frame to frame. Unconstrained edge-collapses and vertex-splits during runtime can be shown to result in mesh foldovers resulting in visual artifacts such as shading discontinuities. To avoid these artifacts Xia *et al* propose the concept of dependencies or constraints that necessitate the presence of the entire neighborhood of an edge before it is collapsed (or its parent vertex is split). Thus, for the example shown in Figure 2, the neighborhood of edge pc should consist exactly of vertices $n_0 \ldots n_6$ for c to collapse to p. Similarly, for the vertex p to split to c, the vertices adjacent to p should be exactly the set $n_0 \ldots n_6$.

View-Dependent Progressive Meshes Hoppe [10] has independently developed an algorithm that is similar to Xia *et al* [19]. Whereas Xia *et al* use the Euclidean distance metric and collapse the shortest edge first to construct the merge tree, Hoppe proceeds in a top-down fashion by minimizing an energy function first defined in [9]. Hoppe uses screen-space projection and orientation of the polygons to guide the run-time view-dependent simplification. Like the approach of Xia *et al*, this approach also requires constraints to prevent mesh foldovers. However, unlike [19], Hoppe [10] empirically observes that for some distance metrics (such as the energy minimization function described there), the vertex-split/edge-collapse constraints limited to only the four faces f_α, f_β, f_γ, and f_δ as shown in Figure 2 are adequate. However, this in general is not a sufficient requirement for other distance metrics, such as the shortest-edge-first, for which the entire neighborhood has to be stored as a constraint for vertex-split/edge-collapse. In Section 4.2 we propose to define these constraints in an implicit manner thereby obviating the need to store them explicitly.

Guéziec *et al* [7] have developed a surface partition algorithm for a progressive encoding scheme for surfaces in the form of a directed acyclic graph (DAG). The DAG represents the partial ordering of the edge collapses with path compression. De Floriani *et al* [4] have introduced the multi-triangulation(MT). The change of level of detail in MT is achieved through a set of local operators that affect fragments of the mesh. The dependencies between the fragments of the mesh are used to construct a DAG of these fragments. This DAG is used at run time to guide the change of the resolution of each fragment.

Schilling and Klein [17] have introduced a refinement algorithm that is texture dependent. They measure the texture distortion in the simplified mesh by mapping the triangulation into the texture space and then measuring the error at vertices and edge intersections. In the vertex hierarchy they store the sequence of the simplification operations and the texture distortion with each

operation. Klein *et al* [11] have developed an illumination-dependent refinement algorithm for multiresolution meshes. The algorithm stores maximum deviation from Phong interpolated normals and introduces correspondence between the normals during the simplification algorithm. In order to avoid aliasing artifacts they recompute the normals at the vertices. In the vertex hierarchy they store the geometric error and maximum normal deviation at each triangle.

2.3 View-Dependent Topology Simplifications

Luebke and Erikson [13] use a scheme based on defining a *tight octree* over the vertices of the given model to generate hierarchical view-dependent simplifications. In a tight octree, each node of the octree is tightened to the smallest axis-aligned bounding cube that encloses the relevant vertices before subdividing further. If the screen-space projection of a given cell of an octree is too small, all the vertices in that cell are collapsed to one vertex. Adaptive refinement is performed analogously. Marshall *et al* [14] have developed a view-dependent topology simplification algorithm based on a clustering approach and simplification metric. The simplification metric minimizes changes to the final image rather than changes to the input model.

3 Overview

We present a technique for performing geometry and genus simplifications in a view-dependent manner. We first construct a hierarchy of vertex-pair collapses to construct a *view dependence* tree. We would like to note here that the view dependence tree differs from trees constructed in the previous literature [10, 19] in that it allows genus simplifications and it does not store any explicit constraints. Details of how we construct the view dependence tree are given in Section 4. In general for n vertices $O(n^2)$ vertex pairs are candidates for collapse. In our current implementation we only consider $O(n \log n)$ candidate vertex pairs by constructing an octree and considering only the nearest neighbors across adjacent cells as candidates.

We have tried several distance metrics and have found that the combination of the vertex coordinates with the normals yields the most acceptable results. We discuss this multi-attribute metric further in Section 5. Almost all view-dependent simplification criteria make use of vertex normals. We discuss how tests for backfacing regions, view-frustum, foveation, and local illumination can be performed in a natural fashion by using our distance function. Our algorithm that makes use of these criteria results in a visually better view-dependent simplification of a scene, than purely Euclidean-distance metrics. We discuss these criteria further in Section 5.

4 View Dependence Tree Construction

A view dependence tree is a generalization of the merge tree introduced by Xia *et al* in two important ways. First, a view dependence tree is capable of performing

genus-reducing simplifications whereas a merge tree can only perform genus-preserving geometric simplifications. Second, a view dependence tree does not store any explicit constraints. Instead implicit constraints are used to ensure runtime consistency in the generated triangulations. We next describe these two important differences.

4.1 Simplifying Genus

An edge collapse combines two vertices that are connected by an edge. A vertex-pair collapse is a generalization of an edge collapse that combines any two vertices. For a dataset with n vertices, $O(n^2)$ vertex pairs are possible. An algorithm that selects from amongst these in a sorted order would take time $O(n^2 \log n)$ – too slow for most practical applications.

To generate the candidate vertex pairs more rapidly we construct an octree over the vertices of the object. For each cell C_i of the octree we include the closest pair of vertices $(\mathbf{P_1}, \mathbf{P_2})$ such that $\mathbf{P_1}$ lies in C_i, and $\mathbf{P_2}$ lies in C_j, where C_i and C_j share a common subdividing plane Π_{ij}. This results in $O(n \log n)$ candidate vertex pairs. We have found that this method works better than selecting vertex pairs based on Delaunay tetrahedralizations or grid-based methods.

Once the vertex pairs have been selected, these together with all the edges of the model are considered for possible collapses to build the view dependence tree in the shortest-edge-first order. The distance metric that we have used to compute the shortest edge is given by Equation 1 in Section 5. The resulting view dependence tree is constructed much along the lines of a merge tree, except for handling of constraints that is discussed next.

4.2 Implicit Constraints

In construction of a view dependence tree we keep track of the identification numbers of the vertices. If the model has n vertices at the highest level of detail they are assigned vertex-ids $0, 1, \ldots, n-1$. Every time a vertex pair is collapsed to generate a new vertex, the id of the new vertex is assigned to be one more than the greatest vertex-id thus far. This process is continued till the entire view dependence tree has been built. The order of the selection of vertex pairs to collapse is made on the basis of the following criteria:

1. *Shortest Distance First:* We store all candidate vertex pairs in a priority heap and select the vertex pair that has the shortest distance based on some distance metric. For our current implementation we use a multi-attribute distance metric defined by Equation 1.
2. *Avoid Mesh Foldover:* If the vertex pair collapse occurs along an edge (i.e. is non-genus-reducing), and performing this collapse does not result in the normals of any of the final triangles from being "flipped" with respect to the pre-collapse triangles then we flag this collapse as valid, otherwise we go back and test criteria (1) above with the next shortest edge.

For certain applications in which long and thin sliver triangles are not desirable an additional test can be added to the above list that will flag a vertex pair collapse as invalid if it results in creation of sliver triangles. All of the above tests are done during the preprocessing stage; they are too costly to be performed at runtime to determine view-dependent triangulations. The outcomes of these tests are represented in the sequence in which the vertices are collapsed during the preprocessing and are reflected in the vertex ids.

4.3 Runtime Traversal

The list of vertices that are used for display at any frame i is defined as the set of *active vertices* for that frame. Active vertices for display in frame $i+1$ are determined by collapsing or splitting from amongst the active vertices for frame i based on the view-dependent criteria. The list of triangles that are displayed in frame i comprise the set of *active triangles* for that frame. The determination of active triangles for frame $i+1$ proceeds in an interleaved fashion with the determination of active vertices for frame $i+1$ from frame i. Every time a display vertex of frame i collapses or splits in frame $i+1$ we simply delete and add appropriate triangles to the list of active triangles. For frame 0 we initialize the lists of active vertices and active triangles to be the entire set of vertices and triangles, respectively, in the model.

Before a vertex is split or collapsed at runtime we make a few simple checks based on vertex ids to ensure the consistency of the generated triangulations and avoid mesh foldovers. These can be simply stated as:

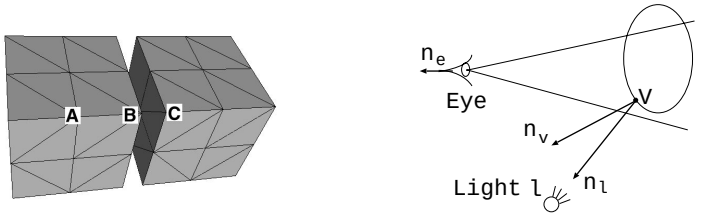
- *Vertex Split:* A vertex p can be safely split at runtime if its vertex-id is greater than the vertex-ids of all its neighbors.
- *Vertex Pair Collapse:* A vertex pair (p, c) can be collapsed if the vertex-id of the resulting vertex is less than the vertex-ids of the parents of the union of the neighbors of vertices p and c.

The two above checks can be efficiently implemented by storing two values with every active vertex – (a) the maximum vertex-id of all its neighbors and (b) the minimum vertex-id of the parents of all its neighbors. During each frame each active vertex is visited once to evaluate its potential to split or collapse. If an active vertex passes the view-dependent tests outlined in Section 5 these two values stored locally at the vertex can be used to determine whether it will be safe to split/collapse. These values are updated only when an active vertex or one of its neighbors actually splits or collapses.

5 Unifying Geometry and Topology

One of the important issues in combining genus simplification with geometry simplification is quantifying and prioritizing the changes in the genus of an object relative to changes in its geometry. Consider for example the object shown in Figure 3(a). Using simple Euclidean-distance-based metrics we find that the

distance between A and B is larger than that between B and C. The decision to collapse vertices B and C will be topology-modifying whereas the collapse of A and B will be a geometry simplification.



(a) Geometry and topology simplification (b) View-dependent simplifciaction

Fig. 3. View-dependent topology simplifications

Currently there are two techniques for deciding which collapse to perform first: (a) *Geometry First:* Perform as many genus-preserving geometry simplifications as possible. When it is no longer possible to perform any geometry simplifications any further for a given error bound, perform genus-reducing simplification. This is the approach taken in [18, 5]; (b) *Equal Preference:* Treat genus-preserving and genus-reducing simplifications on an equal basis and make decisions based only by the spatial distance in R^3. This is the approach taken in [6, 13].

We would like to be able to prioritize genus-preserving and genus-reducing simplifications between the two above extremes. We observe that genus-reducing simplifications are almost always characterized by a large difference in the normals of the surfaces that are merged. This suggests the use of the following distance function between two points $\mathbf{P_0} = (\mathbf{v_0}, \mathbf{n_0})$ and $\mathbf{P_1} = (\mathbf{v_1}, \mathbf{n_1})$:

$$dist(\mathbf{P_0}, \mathbf{P_1}) = \frac{\|\mathbf{v_0} - \mathbf{v_1}\|_2}{(1 + \mathbf{n_0}.\mathbf{n_1}) + \epsilon} \tag{1}$$

where $\mathbf{v_i} = (x_i, y_i, z_i)$ and $\mathbf{n_i} = (n_{i_x}, n_{i_y}, n_{i_z})$ denote the position and normal respectively of point $\mathbf{P_i}$ and ϵ is a user-specified preference factor that prioritizes the genus-preserving and genus-reducing simplifications. As an example, when $\epsilon = 0.1$, genus-preserving geometry simplifications (that involve collapsing two similarly oriented vertices) would be preferred over distances 21 times greater than genus-reducing simplification (that involve collapsing two oppositely oriented vertices).

As shown in Figure 3(b) most of the view-dependent simplifications can be cast in terms of computing the distance between two points with coordinate positions and normals. Let us assume that the eye is located at position $\mathbf{e}$ and has associated with it a vector $\mathbf{n_e}$ that is the oppositely oriented with respect to the look-at vector.

– *Triangle Position and Orientation:* This involves computing the distance between $(\mathbf{v}, \mathbf{n_v})$ and $(\mathbf{e}, \mathbf{n_e})$. If the triangle is backfacing, this will yield a

large value assisting in backface simplification. If the triangle is near the center of the screen and facing the viewer, Equation 1 yields a small value of the distance assisting in implementing foveation, i.e. high detail in the direction the eye is looking at. Foveation can be used to mimic the high visual acuity at the center of the human retina [1].

– *Lighting-based Adaptive Simplification:* If the triangle under consideration is front facing we use the minimum of the distance between $(\mathbf{v}, \mathbf{n_v})$ and $(\mathbf{l}, \mathbf{n_l})$ and the distance between $(\mathbf{v}, \mathbf{n_v})$ and $(\mathbf{e}, \mathbf{n_e})$ to compute the requisite level of detail for illumination.

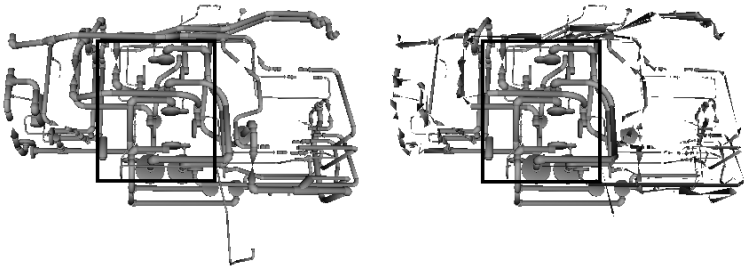
6 Results and Discussion

We have implemented our approach on a SGI Onyx2 with Infinite Reality graphics and tested it on several models. The times for preprocessing appear in Table 1. We would like to point out that our code has not yet been particularly optimized. The results of our approach for the above models appear in the Figures 1– 6.

Dataset	Vertices	Triangles	Preprocessing Time	
			View-Dependent Tree	Octree
Bunny	35947	69451	10.3 s	1.4 s
Pipes	107754	206352	58.1 s	87.5 s
AMR	173042	339444	1 m 55 s	5 m 10 s
Torp	464720	736516	12 m 46 s	19 m 12 s

Table 1. Preprocessing times for various models

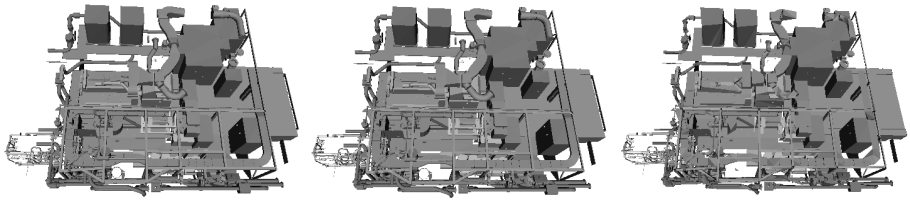
Figure 1 shows the results of lighting-dependent genus simplification. Figure 4 shows the results of view-frustum-guided simplifications. The yellow rectangle in the center of each image shows the outline of the screen-space projection of the view frustum. As can be seen the detail outside the view frustum has been considerably simplified in Figure 4(b). Figure 5 shows the original and two progressively lower levels of detail of the Auxilliary Machine Room dataset from a notional submarine dataset. We would like to point out that on an average traversal of the view dependence tree takes up only $8 - 10\%$ of the time to draw each frame; rest of the time is being taken up in drawing the triangles that have been determined. Figure 6 shows view-dependent simplifications for a procedurally generated model of pipes. Most of the simplification in this case is because of genus-reducing simplifications. This might be an appropriate place to compare our work with that of Luebke and Erikson's [13] as two instances of view-dependent topology simplification work. In Luebke and Erikson's [13] approach fine control over the simplification of the topology is not easy to achieve. For example, a hole that exists on the border of three or more cells may become larger as result of the collapse of the vertices of the adjacent cells. Since our primitive operation of a vertex-pair collapse is simpler than that of collapsing



(a) Original model: 86.5K triangles (b) Simplified model: 21K triangles

Fig. 4. View-frustum-guided genus simplification

all vertices in an octree cell, we believe that our method will be able to provide a finer level of control for more realistic rendering. However, identification of candidate vertex pairs takes more time than simply constructing an octree over input vertices.



(a) Original 340K tris (b) 140K tris (c) 49K tris

Fig. 5. Three LODs from the view dependence tree for Auxilliary Machine Room

We have observed several advantages in making the split and collapse constraints implicit and storing the values to be checked locally:

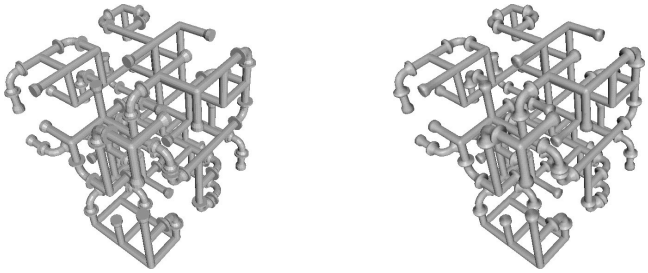
1. *Local Accesses:* Explicit constraints the way they are stored in previous work [10, 19] result in several non-local accesses resulting in unnecessary paging for large datasets or on computers with less memory. Implicit constraints overcome these drawbacks and suggest possibilities for developing external memory algorithms for view-dependent visualization of datasets that do not even fit into the main memory of the visualization workstation.
2. *Change-Sensitive Processing:* Explicit constraints need to visit every neighbor of an active vertex to determine whether or not it can split/collapse. This might result in visiting of a node several times, once from each of its neighbors, often unnecessarily. With implicit constraints, the way we have defined them, the algorithm needs to visit a node of the view dependence tree only when its associated active vertex actually splits or collapses. Thus the processing time is now proportional to the actual number of changes as opposed to potential changes.

3. *Memory Savings:* Implicit constraints require only two integers to be stored per node of the view dependence tree as opposed to a pointer to every vertex in the neighborhood. This results in a modest savings of about 30% in the storing of the tree.

7 Conclusions and Future Work

We have presented the concept of view-dependence trees to perform genus-reducing simplifications for large polygonal datasets. These trees are more compact, faster to navigate, and easier to build than prior work. Further, we have also introduced a distance metric that can be used for prioritizing genus-reducing simplifications with respect to geometry-reducing simplifications in a view-dependent manner. This metric is particularly useful in that it is able to unify the distance function being used in genus-preserving versus genus-reducing simplifications with the other criteria in defining view-dependent simplifications.

We see scope for future work in designing external memory algorithms for visualization of datasets whose sizes exceed that of the main memory, by taking advantage of the localized and compact structure of view dependence trees. Also, there is scope for better defining the normal and coordinate values of a new vertex as a result of vertex pair collapse by using methods similar to those of Garland and Heckbert [6]; we currently use the average normal and average coordinate values of the two vertices.



(a) Original model: 25.7K triangles (b) Simplified model: 7.2K triangles

Fig. 6. View-dependent simplification of the procedural pipe model

Acknowledgements

This work has been supported in part by the NSF grants: CCR-9502239, DMI-9800690, ACR-9812572 and a DURIP instrumentation award N00014970362. Jihad El-Sana has been supported in part by the Fulbright/Arab-Israeli Scholarship and the Catacosinos Fellowship for Excellence in Computer Science. The figures 1, 4, and 5 show mechanical parts from the dataset of a notional submarine provided to us by the Electric Boat Division of General Dynamics.

References

1. E.-C. Chang and C. Yap. A wavelet approach to foveating images. In *Proceedings of the 13th International Annual Symposium on Computational Geometry (SCG-97)*, pages 397–399, New York, June4–6 1997. ACM Press.
2. P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54, February 1998. ISSN 0097-8493.
3. J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. Brooks, Jr., and W. V. Wright. Simplification envelopes. In *Proceedings of SIGGRAPH '96*, pages 119 – 128. ACM SIGGRAPH, ACM Press, 1996.
4. L. De Floriani, P. Magillo, and E. Puppo. Efficient implementation of multi-triangulation. In *Proceedings Visualization '98*, pages 43–50, 1998.
5. J. El-Sana and A. Varshney. Topology simplification for polygonal virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 4, No. 2:133–144, 1998.
6. M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH '97 (Los Angeles, CA)*, pages 209 – 216. ACM SIGGRAPH, ACM Press, August 1997.
7. A. Gueziec, F. Lazarus, G. Taubin, and W. Horn. Surface partitions for progressive transmission and display, and dynamic simplification of polygonal surfaces. In *Proceedings VRML 98, Monterey, California, February 16–19*, pages 25–32, 1998.
8. T. He, L. Hong, A. Varshney, and S. Wang. Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–184, 1996.
9. H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, pages 99 – 108. ACM SIGGRAPH, ACM Press, 1996.
10. H. Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH '97*, pages 189 – 197. ACM SIGGRAPH, ACM Press, 1997.
11. R. Klein, A. Schilling, and W. Straßer. Illumination dependent refinement of multiresolution meshes. In *Computer Graphics International*, June 1998.
12. P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *Proceedings Visualization '98*, pages 279–286, 1998.
13. D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of SIGGRAPH '97 (Los Angeles, CA)*, pages 198 – 208. ACM SIGGRAPH, ACM Press, August 1997.
14. D. Marshall, A. Campbell, and D. Fussell. Model simplification using directional clustering. In *Visual Proceedings*, page 174. ACM/SIGGRAPH Press, 1997.
15. J. Popović and H. Hoppe. Progressive simplicial complexes. In *Proceedings of SIGGRAPH '97*, pages 217 – 224. ACM SIGGRAPH, ACM Press, August 1997.
16. J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering. In *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, June–July 1993.
17. A. Schilling and R. Klein. Texture-dependent refinement for multiresolution models. In *Computer Graphics International*, June 1998.
18. W. Schroeder. A topology modifying progressive decimation algorithm. In *IEEE Visualization '97 Proceedings*, pages 205 – 212. ACM/SIGGRAPH Press, 1997.
19. J. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, pages 171 – 183, June 1997.