Controlled Simplification of Genus for Polygonal Models

Jihad El-Sana Amitabh Varshney

State University of New York at Stony Brook

Abstract

Genus-reducing simplifications are important in constructing multiresolution hierarchies for level-of-detail-based rendering, especially for datasets that have several relatively small holes, tunnels, and cavities. We present a genus-reducing simplification approach that is complementary to the existing work on genus-preserving simplifications. We propose a simplification framework in which genus-reducing and genus-preserving simplifications alternate to yield much better multiresolution hierarchies than would have been possible by using either one of them. In our approach we first identify the holes and the concavities by extending the concept of α-hulls to polygonal meshes under the L_∞ distance metric and then generate valid triangulations to fill them.

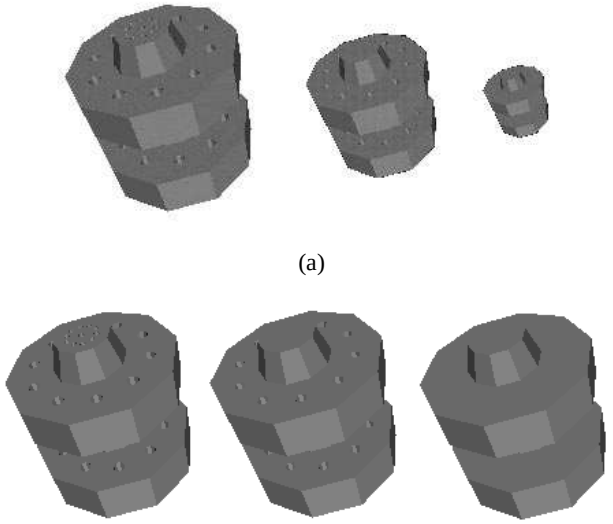
CR Categories and Subject Descriptors: I.3.3 **[Computer Graphics]:** Picture/Image Generation — Display algorithms; I.3.5 **[Computer Graphics]:** Computational Geometry and Object Modeling — Curve, surface, solid, and object representations.

1 Introduction

The recent growth in the complexity of three-dimensional graphics datasets has outpaced the advances in the graphics hardware. Several software and algorithmic solutions have been recently proposed to bridge the increasing gap between hardware capabilities and the complexity of the visualization datasets. These solutions are based on creating multiresolution hierarchies [38, 34, 33, 9, 26, 13, 23, 24, 39] and then using them with level-of-detail-based rendering [7, 10, 11, 20, 32, 28], visibility-based culling [1, 37, 22, 27, 21, 40], and image-based rendering [6, 5, 29, 36, 12]. The approach presented in this paper falls under the first category – creation of multiresolution hierarchies.

Level-of-detail-based rendering involves rendering perceptually less significant objects in a scene at lower levels of detail and perceptually more significant objects at higher levels of detail. Automatic creation of multiresolution hierarchies is a crucial first-step in any level-of-detail-based rendering system. Most algorithms for creating multiresolution hierarchies preserve the topology of the input object. For certain applications, such as molecular modeling, this is truly a desirable feature. For example, consider a molecular surface that shows a tunnel through the molecule. Such tunnels often act as atomic sieves aiding biochemical processes. Information about the presence of tunnels and cavities in a molecule is quite important in biochemistry applications such as rational drug design. Regardless of the degree of surface simplification, it is crucial for these applications that such topological features be preserved. Similarly, adaptive simplifications for finite-element structures and study of fine tolerances between nearby parts in mechanical CAD necessitates the use of genus-preserving simplifications.

However for certain applications, such as virtual reality, topology-preservation in multiresolution hierarchy creation is not a prerequisite. For example, consider a mechanical part as shown in Figure 1. As this part is moved farther away from the view-plane, a level-of-detail-based rendering algorithm will progressively substitute lesser and lesser detailed versions of the object. However, if the multiresolution hierarchy for this object is topology preserving, then even the simplest representation of this object will have all the holes. On the other hand, if we create a genus-reducing multiresolution hierarchy for this object we will be able to achieve a simpler representation without the holes. Our experience has been that such objects often occur in real-life engineering datasets.



(a)

(b)

Figure 1: Hierarchical simplifications of the genus

Topology-preservation criterion can actually be a costly constraint for a vast majority, if not all, of the objects that are used in an interactive walkthrough of a virtual environment. Most objects can be only simplified to a limited amount before it becomes necessary to switch to a simplified topological representation. We propose a method that will reduce the genus of an object in a *controlled* fashion. Our technique allows rejection of all holes, tunnels, and cavities in an object that are less than a user-specifiable threshold. We give the details of our approach in Section 3. In addition to two-manifold polygonal objects our approach also works in presence of some limited cases of non-manifold polygonal objects including those that have T-junctions and T-edges (shown in Figure 2). Such degeneracies are quite common in mechanical CAD datasets from the manufacturing industry as well as some scientific visualization datasets (such as the results of an incorrect marching cubes implementation). Like other multiresolution hierarchy generation algorithms, the running times for our approach depend on the choice of the object and the error tolerance. We have empirically observed the running times for our genus-reducing stages to be usually much lesser compared to the genus-preserving simplification stage times. The reason behind this is that for a typical object there are few candidate regions for genus-reducing simplifications

Contact address: Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400, Email: jihad|varshney@cs.sunysb.edu

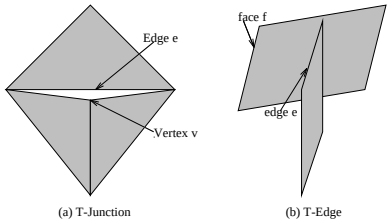


Figure 2: Permissible non-manifold degeneracies

and their rapid identification speeds up the entire process. Details of our implementation are given in Section 4.

We view the simplification of an object of arbitrary topological type as a two-stage process – simplification of the topology (i.e. reduction of the genus) and simplification of the geometry (i.e. reduction of the number of vertices, edges, and faces). One can mix the execution of these two stages in any desired order. We have observed that genus reductions by small amounts can lead to large overall simplifications. This observation together with the fact that genus-reducing simplifications are usually faster (since they do less work) than genus-preserving simplifications, makes such an approach quite attractive for generating multiresolution hierarchies. The results of our approach are presented in Section 5.

2 Previous work

Most of the existing work on creation of multiresolution hierarchies for polygonal models works with the genus-preservation constraint and is complementary to our approach presented in this paper. Notable exceptions include algorithms by Rossignac and Borrel [33] and He *et al* [25].

Rossignac and Borrel's algorithm [33] subdivides the model by a global grid and uses a vertex clustering approach within each grid cell. All vertices that lie within a grid cell are combined and replaced by a new vertex. The polygonal mesh is suitably updated to reflect this. The nice properties of this approach are that it is quite fast and can work in presence of degeneracies often found in real-life polygonal datasets. This approach simplifies the genus if the desired simplification regions fall within a grid cell. However, it makes no guarantees about genus reduction and fine control over the genus simplification is not easy to achieve.

He *et al* [25] present an algorithm to perform a controlled simplification of the genus of an object. It is a good approach that is applicable to volumetric objects on a voxel grid. It can handle degeneracies. However, since it works in the volumetric domain, polygonal objects have to be first voxelized. This often results in an increased complexity that is related to the volume of the object, not to its original representation complexity. Besides, the results from this approach lead to a scaled-down version of the simplified object that needs to be scaled back up to maintain the same size as the original object. This is not difficult but it adds another stage to the overall simplification process.

The α-hull has been defined as a generalization of the convex hull of point-sets by Edelsbrunner, Kirkpatrick, and Seidel [15, 14]. Given a set of points P, a ball b of radius α is defined as an *empty* α*-ball* if $b \cap P = \phi$. For $0 \leq \alpha \leq \infty$, the α*-hull* of P is defined as the complement of the union of all empty α-balls [14]. Three-dimensional α-shapes have been defined on the Delaunay tetrahedralization $\mathcal{D}$ of the input points P by Edelsbrunner and Mücke [17]. Let b_T be the circumsphere of a k-simplex Δ_T and let its radius be ρ_T. Let $G_{k,\alpha}, 1 \leq k \leq 3$ be the set of k-simplices $\in \mathcal{D}$ for which b_T is empty and $\rho_T < \alpha$. An α-complex of P, denoted by C_α is the cell complex whose k-simplices are either in $G_{k,\alpha}$ or they bound the $(k+1)$-simplices of C_α. The α-shape of P denoted by S_α is the union of all simplices of C_α. Thus, an α-shape of a set of points P is a subset of the Delaunay triangulation of P. Edelsbrunner [14], has extended the concept of α-shapes to deal with weighted points (i.e. spheres with possibly unequal radii) in three dimensions. An α-shape of a set of weighted points P_w is a subset of the regular triangulation of P_w. Recent work by Bernardini and Bajaj [3] uses α-hulls to elegantly deal with point-sets that represent a sampling of mechanical CAD (polygonal or otherwise) models. Their work primarily deals with reconstruction of surfaces from unorganized sets of points, although it could potentially be used for genus-simplification of reconstructed surfaces if the input is either (a) a collection of unorganized set of points, or (b) a sampling of points from a given polygonal dataset. The existing work for α-hulls deals with only point- and sphere-based datasets.

Our research extends the concept of α-hulls to polygonal datasets for performing genus-reducing simplifications. The primary targets of our research are the interactive three-dimensional graphics and visualization applications where topology can be sacrificed if (a) it does not directly impact the application underlying the visualization and (b) produces no visual artifacts. Both of these goals are easier to achieve if the simplification of the topology is finely *controlled* and has a sound mathematical basis. In the next section we outline our approach that has these properties.

3 Our approach

The intuitive idea underlying our approach is to simplify the genus of a polygonal object by rolling a sphere of radius α over it and filling up all the regions that are not accessible to the sphere. This is the same as the underlying idea of α-hulls over point-sets. The problem of planning the motion of a sphere amidst polyhedral obstacles in three-dimensions has been very nicely worked out by Bajaj and Kim [2]. We use these ideas in our approach. Let us first assume that our polygonal dataset consists of only triangles; if not, we can triangulate the individual polygons [35, 8, 31]. Planning the motion of a sphere of radius α, say $S(\alpha)$, amongst triangles is equivalent to planning the motion of a point amongst "grown" triangles. Mathematically, a grown triangle $T_i(\alpha)$ is the Minkowski sum of the original triangle t_i with the sphere $S(\alpha)$. Formally, $T_i(\alpha) = t_i \oplus S(\alpha)$, where $\oplus$ denotes the Minkowski sum which is equivalent to the convolution operation. Thus, our problem reduces to efficiently and robustly computing the union of the grown triangles $T_i(\alpha)$. The boundary of this union, $\partial \bigcup_{i=1}^{n} T_i(\alpha)$, where n is the number of triangles in the dataset, will represent the locus of the center of the sphere as it is rolled in contact with one or more triangles and can be used to guide the genus simplification stage. We should point out here that our choice of the L_∞ metric over the L_2 metric buys us robustness and efficiency but sacrifices the rotational invariance of error tolerance in the conventional Euclidean (L_2) sense.

3.1 Alpha prisms

We refer to an α-grown triangle $T_i(\alpha)$ as an α*-prism*. For general polygons, computing a constant radius offsetting is a difficult operation in which several degeneracies might arise [2]. However, for triangles this is an easy, robust, and constant-time operation. Computing the union of $T_i(\alpha), 1 \leq i \leq n$ can be simplified by considering the offsetting operation in the L_1 or the L_∞ metrics. This is equivalent to convolving the triangle t_i with an oriented cube (which is the constant distance primitive in the L_1 or the L_∞ metrics, just as a sphere is in the L_2 metric). Offsetting a triangle t_i in the L_1 or the L_∞ metrics yields a $T_i(\alpha)$ that is a convex polyhedron (result of the convolution of an oriented cube with the triangle t_i). Example of a two-dimensional α-prism in the L_∞ distance metric is shown in Figure 3.

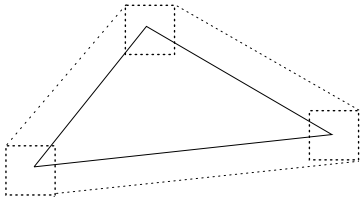


Figure 3: An α-prism around a triangle in two dimensions

We choose the L_∞ distance metric over the L_1 metric because the former results in axially-aligned unit-distance cubes, which are easier to deal with in union and intersection operations. We have efficiently implemented the construction of α-prisms for the L_∞ metric as outlined in Section 4.1.

3.2 Overview of our approach

Figure 4 gives the overview of the stages involved. For clarity, the overview in the figure is given in two dimensions; extension to three dimensions (which we have implemented) is straightforward. Let us consider the behavior of our approach in the region of a two-dimensional "hole" $abcde$ (shown unfilled). We first generate the α-prisms (Figure 4(b)), compute their union (Figure 4(c)), and generate a valid surface triangulation from the union (Figure 4(d)). The result of this process as shown for this example adds an extra triangle abe in the interior of the region $abcde$. This extra triangle is added in the region where the oriented L_∞ cube of side 2α could not pass through.

The computation of the union of the α-prisms, $\bigcup_{i=1}^{n} T_i(\alpha)$, under the L_∞ metric is simply the union of n convex polyhedra. The union of n convex polyhedra can be computed from their pairwise intersections. Intersection of two convex polyhedra with p and q vertices can be accomplished in optimal time O$(p+q)$ [4]. A simpler algorithm to implement takes time O$((p+q)\log(p+q))$ [30].

With our approach one could, in principle, compute the union of α-prisms over the entire polygonal object (and not just along the boundary of the region of interest as shown in Figure 4). This will result in automatic identification of the region $abcde$ as being partially inaccessible to the L_∞ cube of side 2α, and therefore a candidate for getting filled. In practice, we have observed that the regions on an object that actually participate in simplification of the genus are few. As a result, a lot of effort is spent in processing those regions of the object that ultimately do not result in any simplifications. To avoid this extra computation and rapidly identify such regions that can be partially or completely triangulated we use a heuristic as outlined next.

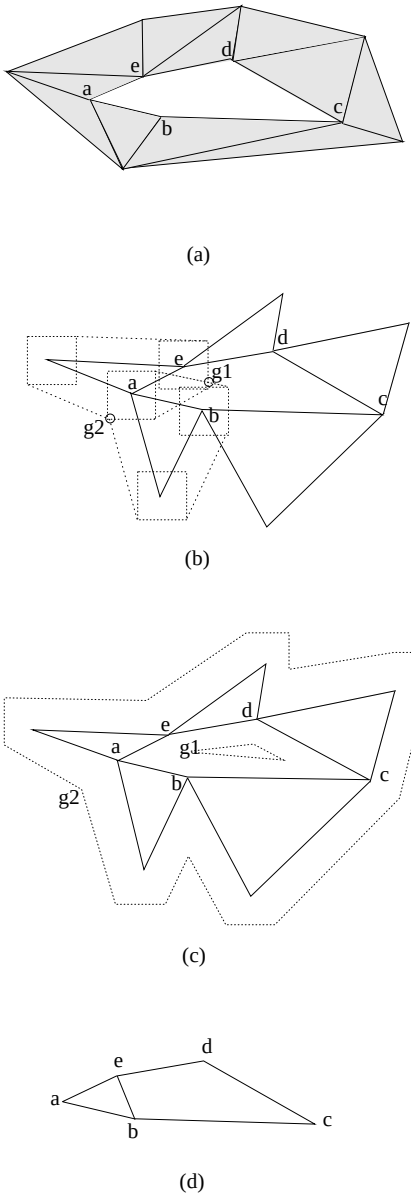


Figure 4: Overview of our approach

3.3 Determination of tessellation regions

We define a *tessellation region* on an object as the region that is partially or completely inaccessible to an L_∞ cube of side 2α. A tessellation region is a closed-connected region bounded by a *tessellation chain*. A tessellation chain consists of a linear sequence of edges – the *tessellation edges*. In Figure 4, the tessellation edges are $\{ab, bc, cd, de, ea\}$, the tessellation chain is $(abcdea)$, and the tessellation region is the unfilled polygonal region $(abcde)$.

There are three main stages in determination of the tessellation regions. We first determine the global set of tessellation edges that forms the boundaries of all the regions that need to be tessellated on an object. We next organize the tessellation edges into a set of tessellation chains. A tessellation chain is defined as a sequence of connected tessellation edges whose every vertex is adjacent to exactly two tessellation edges, except the first and the last vertices which can have a different degree of tessellation edges. The third stage involves orienting each tessellation chain so that the tessellation region lies on its left side (assuming that the polygons are oriented counterclockwise).

Determination of tessellation edges In principle, it is possible to determine all tessellation regions (and hence tessellation edges) from the union of α-prisms over the entire polygonal object. However, this is rather slow and we have observed that in practice for most mechanical CAD models the genus-reducing simplifications occur in well-defined regions. To efficiently determine the tessellation edges we currently use the heuristic that genus-reducing simplifications usually occur in the neighborhood of sharp edges. We define a *sharp edge* to be one for which the dihedral angle between the neighboring triangles is greater than some threshold β; we currently use the constant $\beta = 70^0$. We have found this to be a good heuristic for mechanical CAD models.

Determination of tessellation chains After we have determined the tessellation edges, they are initially stored as an *edge soup* – an unorganized collection of edges. We organize the tessellation edges into tessellation chains by using a linear-time depth-first scanning strategy. At the end of this stage we have tessellation chains as shown in Figure 5 that may be closed (chain *a*) or open (chains *b* and *c*).

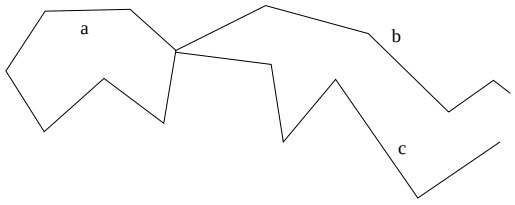


Figure 5: Tessellation chains

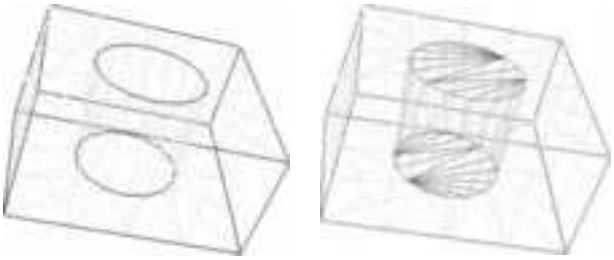
Orienting tessellation chains For each tessellation chain we need to determine the side on which the tessellation region will lie. Once this side has been determined, we orient the tessellation chain such that the tessellation region lies on its left. If we were to use the global information from a completely computed α-hull of the polygonal object, determination of the orientation would be quite easy, although expensive. To speed-up this stage, we use the heuristic that the tessellation regions have a larger variation of the surface normal. This heuristic works quite well for mechanical CAD models for which almost all genus-reducing simplifications occur in holes, tunnels, or sharp concavities in regions that are otherwise flat. Orientation of the tessellation chains helps in tiling as well as removal of the internal triangles as discussed next.

3.4 Generating a valid surface triangulation

We consider those vertices u_i of the union of α-prisms, $\bigcup_{i=1}^{n} T_i(\alpha)$, that belong to two or more α-prisms. For each vertex u_i, we consider every pair of α-prisms $T_j(\alpha)$ and $T_k(\alpha)$ to which u_i belongs. If we center the L_∞ cube of side 2α at u_i, it will touch two edges (say e_l and e_m), one from each triangle t_j and t_k. Let the edge $e_l = (v_{l_0}, v_{l_1})$, and the edge $e_m = (v_{m_0}, v_{m_1})$ For example in Figure 4, $u_i = g_1$, $e_l = ab$, and $e_m = ae$. If the L_∞ cube of side 2α centered at the point u_i contains a pair of non-adjacent vertices v_p and v_q out of $\{v_{l_0}, v_{l_1}, v_{m_0}, v_{m_1}\}$ (in this example, vertices b and e), then we add the edge (v_p, v_q) (in this example, we add edge be) to the set of diagonal edges that will be used to generate the surface triangulation. In this example if the L_∞ cube at g_1 did not contain either of the vertices b or e, we would not have added any diagonal edge.

At the end of this stage we have the superset of the diagonals required for the final triangulation as well as a collection of oriented tessellation chains. The diagonals and the oriented tessellation chains are used in the surface reconstruction approach as outlined in [19]. The objective function that we use for optimization in the surface reconstruction is minimizing the length of the diagonal edges used (shortest edge first). This gives us a unique and consistent triangulation.

As a result of our triangulation above, some triangles that were initially on the surface of the object now become interior as shown in Figures 6 and 10. We identify and remove the internal triangles as follows. All the tessellation regions that we covered have oriented tessellation boundaries. Walking along the direction of the tessellation chain, we classify the left side as interior (i.e. the region that has just been covered) and the right side as exterior (this assumes counter-clockwise orientation of polygons). We start from a triangle belonging to the input object that lies to the left (i.e. the interior) of an oriented tessellation chain and spread out in a breadth-first manner until we hit one of the hole edges or another triangle that has been similarly traversed. All triangles that are thus traversed are marked as interior and eliminated. In Figure 10(c) the patched triangles are shown in green and the interior triangles that are eliminated are shown in yellow.



(a) (b)

Figure 6: Determination of internal triangles

4 Implementation details

4.1 Efficiently constructing alpha prisms

The α-prism is the convex hull of the three axially aligned L_∞ cubes, each of side 2α units, centered at each of the three vertices of a triangle. There are several sets of coplanar points for this case and as a result the traditional methods of computing convex hulls do not work robustly here. One solution to this robustness problem is to perform perturbation of all the input points [16, 18]. Although such approaches guarantee removal of all degeneracies, they require multi-precision arithmetic which is slow. Besides, such perturbations lead to generation of unnecessary sliver triangles in the union of the α-prisms. To overcome these limitations, we decided to adopt a different, although specialized, approach That works quite efficiently and robustly. We next outline the basic idea of this approach.

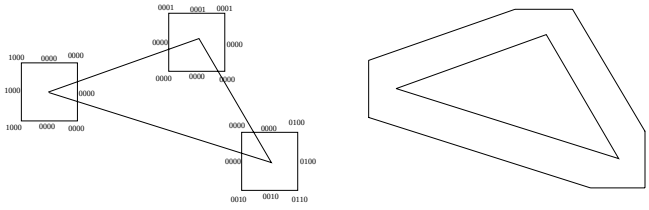


Figure 7: Fast computation of alpha prisms

We associate a 6-bit vector with each vertex, edge, and face of the three L_∞ cubes. Each of the six bits of the bit-vector associated with a vertex v is 1 if and only if the corresponding property holds amongst all of the 24 vertices : {minimum x, maximum x, minimum y, maximum y, minimum z, maximum z}. Thus, bit[0] of a vertex will be 1 if its x-coordinate has the minimum value amongst all the 24 vertices; it will be 0 otherwise. The edge bit-vector is computed as the logical bit-wise AND of its two vertices' bit-vectors. The face bit-vector is the logical bit-wise AND of its four edges' bit-vectors. The α-prism can be computed by using the information from these vertex, edge, and face bit-vectors. For instance, all the cubes' faces that have non-zero bit-vectors belong to the α-prism. Details of this approach are outside the scope of this paper; they will be described elsewhere. The computation of the

α-prism using this strategy for the two-dimensional case (where we use 4-bit vectors) is shown in Figure 7. Notice that only those edges of the L_∞ squares that have non-zero bit-vectors are a part of the two-dimensional α-prism (analogous to the faces of the L_∞ cubes in three dimensions).

4.2 Efficiently intersecting alpha prisms

The α-*neighborhood* of a triangle t_i is defined as the set of all triangles t_j such that the intersection of their α-prisms is non-null, $T_i(\alpha) \bigcap T_j(\alpha) \neq \phi$. Every such pair of α-prisms is defined to be α-*close* to each other. The first stage in generating a valid surface triangulation is to identify the pairs of α-prisms that are α-close to each other.

For practical values of α, which are relatively small, the average size of the α-neighborhood of a triangle has been empirically observed to be a small constant. We use a global grid to speed up the determination of the α-neighborhood for each triangle t_i. Each side of the cubical grid is 2α units, to minimize the number of neighboring cubes that have to be considered for determining the α-neighborhood of a given triangle.

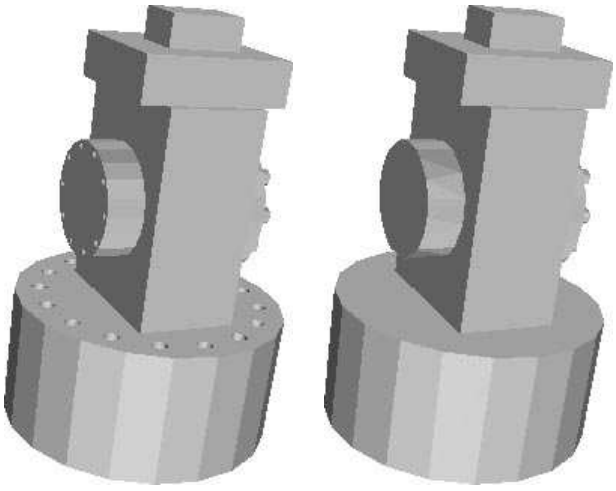
5 Results and discussion

We have implemented the approach outlined above and have obtained very encouraging results. We ran our implementation on one R10000 processor of SGI Challenge for all the results reported here. The various mechanical parts on which we tested our algorithm and reported our results are shown in Figures 8– 12.

We chose the Simplification Envelopes [9] approach to perform our genus-preserving simplifications. All error tolerances (α and ϵ) given in this section are specified as the percentage of the diagonal of the bounding box of the object. We first simplified the given objects using the genus-preserving approach with a tolerance of ϵ. Then on this simplified object we compared the results of the following two simplification strategies:

(a) genus-reducing simplifications with a tolerance of α followed by genus-preserving simplifications with a tolerance of ϵ

(b) genus-preserving simplifications with a tolerance of $\alpha + \epsilon$



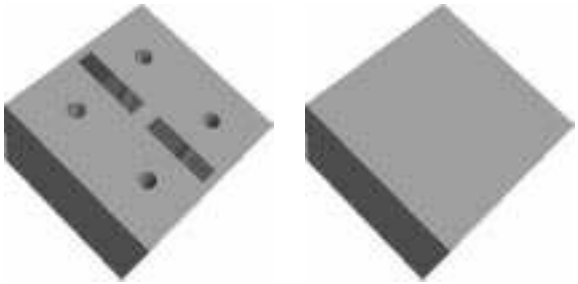
(a) 5180 triangles (b) 1438 triangles

Figure 8: Genus reduction for an industrial CAD part

The results of our comparisons are given in Table 1. As can be seen, performing genus-reducing simplification with genus-preserving simplifications vastly improves the overall simplification ratio. The timings in the following table are in units of seconds, minutes, and hours and are accordingly marked by s, m, and h, respectively. Not only did we observe the genus-reducing simplifications improved the overall simplification ratio substantially, we also observed that the times taken by the genus-reducing stage were reasonably small. Part of the reason behind the latter is our use of heuristics as given in Section 3.3 that we have observed work quite well for mechanical CAD datasets.

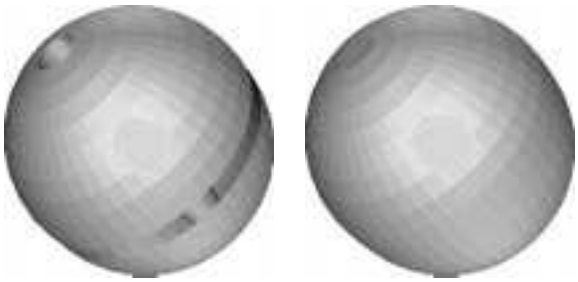
(a) Original (b) Simplified

Disk



(a) Original (b) Simplified

Box



(a) Original (b) Simplified

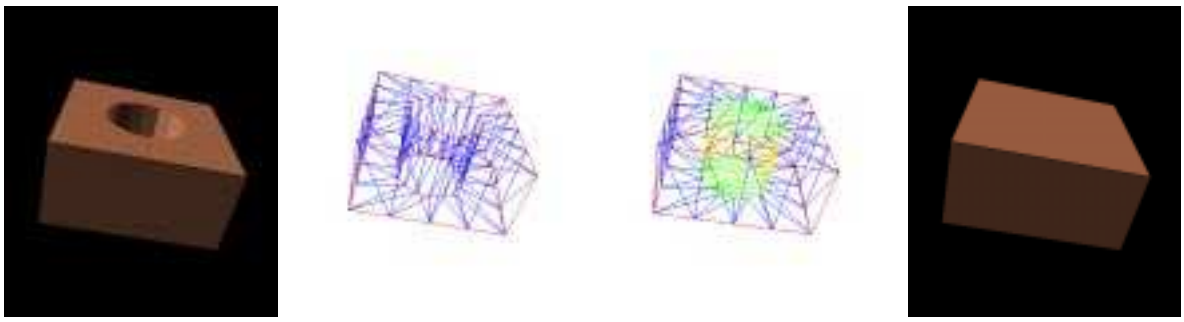
Sphere

Figure 9: Genus-reducing simplifications for various objects

An example of a deeper level-of-detail hierarchy can be seen in Figure 11. Here the object complexity was reduced as follows: (a) original object had 44892 triangles, (b) genus-preserving simplification with tolerance $\epsilon = 0.5$ resulted in 24532 triangles, (c) further genus-reducing simplification with tolerance $\alpha = 2.0$ resulted in 24427 triangles (but with only one hole remaining), (d) further genus-preserving simplification with tolerance $\epsilon = 1.0$ resulted in 2685 triangles, (e) further genus-reducing simplification with tolerance $\alpha = 5.0$ resulted in 2681 triangles (but with no remaining holes), (f) further genus-preserving simplification with tolerance $\epsilon = 2.0$ resulted in 878 triangles. Such high levels of simplifications would not have been easy to achieve without using genus-reducing simplifications.

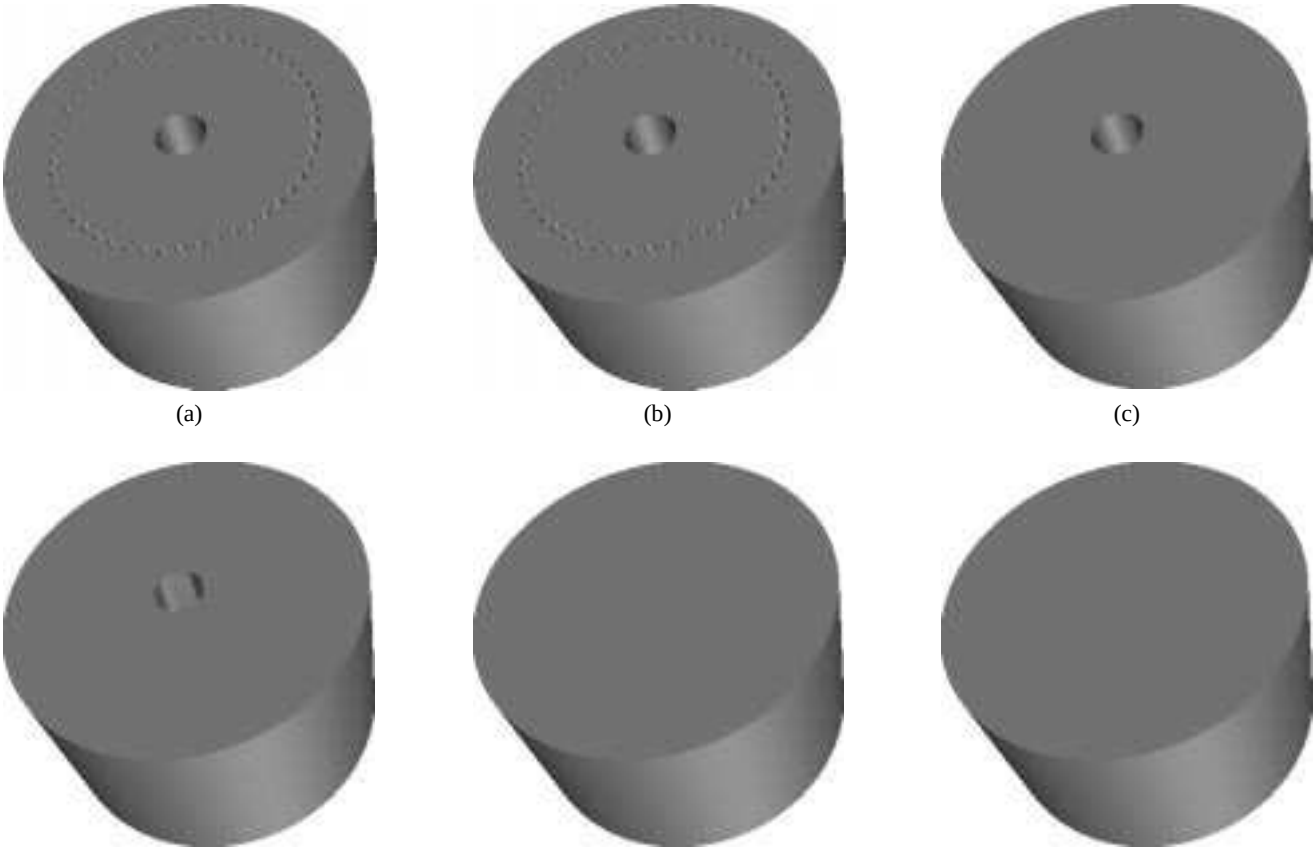
Original Object		Genus-Preserving Simplification			Genus-Reducing and Genus-Preserving Simplification					Genus-Preserving Simplification			Ratio of Simplifcation Improvement
Name	Num Tris	ϵ	Time	Num Tris	Genus Reducing		Genus Preserving		Num Tris	α + ϵ	Time	Num Tris	
					α	Time	ϵ	Time					
Disk	752	1.0	7.9 s	554	2.0	0.3 s	1.0	8.4 s	146	3.0	14.6 s	462	3.09
Box	1612	1.0	30 s	166	2.0	0.3 s	1.0	2.0 s	14	3.0	4.2 s	138	9.86
Sphere	2058	5.5	100 s	432	11.0	0.3 s	5.5	7.3 s	136	16.5	19.2 s	162	1.19
Block	17644	0.4	24 m	2054	0.8	5.6 s	0.4	20 s	78	1.2	280 s	1622	20.79
Fixture	74396	0.3	5:30 h	10806	0.6	129 s	0.3	42 s	62	0.9	11:22 m	7300	117.7

Table 1: Results of genus-reducing and genus-preserving simplifications



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

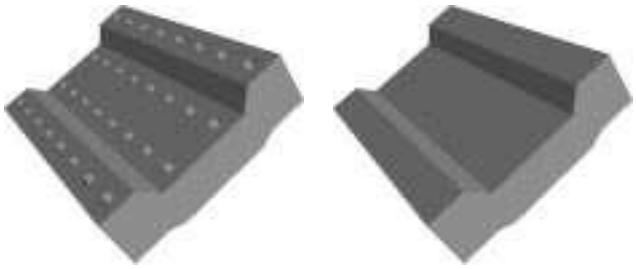
Figure 10: Overview of the genus-reducing simplification



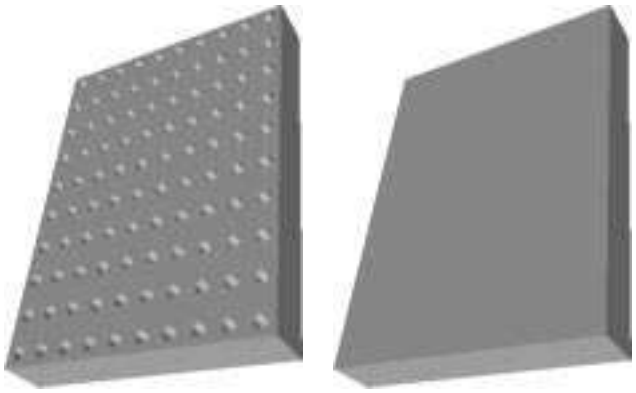
(a) (b) (c)

(d) (e) (f)

Figure 11: Alternating genus-preserving and genus-reducing simplifications



(a) Original (b) Simplified Block



(a) Original (b) Simplified Fixture

Figure 12: Genus-reducing simplifications

6 Conclusions and future work

We have presented an approach that performs a controlled simplification of the genus of polygonal objects. This approach works quite well in conjunction with the traditional genus-preserving simplification approaches and yields substantially lower representation complexity objects than otherwise possible. Our approach can also work in presence of limited kinds of degeneracies, such as T-junctions and T-edges, that are quite widespread in real-life mechanical CAD datasets.

We currently use heuristics that rapidly detect holes and other cavities commonly found in CAD models. We plan to further explore other heuristics that will rapidly determine the tessellation regions on other kinds of scientific visualization datasets. Our approach currently works only for polygonal objects. Its generalization to objects described by higher-order algebraic patches will be an interesting area for future work.

Our approach does not at present handle certain degeneracies such as coincident polygons, self-intersecting meshes, and meshes with inconsistent orientations of polygons. Alpha hulls have been shown to work quite well for reconstructing surfaces from unorganized collections of points. It should be possible to extend the current work and merge it with that line of research to be able to handle the above degeneracies as well.

Acknowledgements

The first author has been supported by a Fulbright/Israeli Arab Scholarship. This work has been supported in part by the National Science Foundation CAREER award CCR-9502239. The CAD object in Figure 8 is a part of the dataset of a notional submarine provided to us by the Electric Boat Division of General Dynamics. We would like to acknowledge the valuable suggestions made by the anonymous referees that helped us in improving the presentation.

References

[1] J. M. Airey. *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations*. PhD thesis, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC 27599-3175, 1990.

[2] C. Bajaj and M.-S. Kim. Generation of configuration space obstacles: the case of a moving sphere. *IEEE Journal of Robotics and Automation*, 4, No. 1:94–99, February 1988.

[3] F. Bernardini and C. Bajaj. Sampling and reconstructing manifolds using alpha-shapes. Technical Report CSD-97-013, Department of Computer Sciences, Purdue University, 1997.

[4] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

[5] S. Chen. Quicktime VR – an image-based approach to virtual environment navigation. In *Computer Graphics Annual Conference Series (SIGGRAPH '95)*, pages 29–38. ACM, 1995.

[6] S. Chen and L. Williams. View interpolation for image synthesis. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 279–288, August 1993.

[7] J. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[8] K. Clarkson, R. E. Tarjan, and C. J. Van Wyk. A fast Las Vegas algorithm for triangulating a simple polygon. *Discrete Comput. Geom.*, 4:423–432, 1989.

[9] 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 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 119 – 128. ACM SIGGRAPH, ACM Press, August 1996.

[10] M. Cosman and R. Schumacker. System strategies to optimize CIG image content. In *Proceedings of the Image II Conference*, Scottsdale, Arizona, June 10–12 1981.

[11] F. C. Crow. A more flexible image generation environment. In *Computer Graphics: Proceedings of SIGGRAPH'82*, volume 16, No. 3, pages 9–18. ACM SIGGRAPH, 1982.

[12] L. Darsa, B. Costa, and A. Varshney. Navigating static environments using image-space simplification and morphing. In *Proceedings, 1997 Symposium on Interactive 3D Graphics*, pages 25 – 34, 1997.

[13] T. D. DeRose, M. Lounsbery, and J. Warren. Multiresolution analysis for surface of arbitrary topological type. Report 93-10-05, Department of Computer Science, University of Washington, Seattle, WA, 1993.

[14] H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1760, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

[15] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, IT-29(4):551–559, July 1983.

[16] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 118–133, 1988.

[17] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, January 1994.

[18] I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Eighth Annual Symposium on Computational Geometry*, pages 74–82, Berlin, Germany, June 1992. ACM Press.

[19] H. Fuchs, Z. M. Kedem, and S. P. Uselton. Optimal surface reconstruction from planar contours. *Commun. ACM*, 20:693–702, 1977.

[20] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of SIGGRAPH 93 (Anaheim, California, August 1–6, 1993)*, Computer Graphics Proceedings, Annual Conference Series, pages 247–254. ACM SIGGRAPH, August 1993.

[21] N. Greene. Hierarchical polygon tiling with coverage masks. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 65 – 74. ACM Siggraph, ACM Press, August 1996.

[22] N. Greene and M. Kass. Hierarchical Z-buffer visibility. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 231–240, 1993.

[23] A. Guéziec. Surface simplification with variable tolerance. In *Proceedings of the Second International Symposium on Medical Robotics and Computer Assisted Surgery, MRCAS '95*, 1995.

[24] B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11:197–214, 1994.

[25] T. He, L. Hong, A. Varshney, and S. Wang. Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–184, June 1996.

[26] H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 99 – 108. ACM SIGGRAPH, ACM Press, August 1996.

[27] D. Luebke and C. Georges. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In *Proceedings, 1995 Symposium on Interactive 3D Graphics*, pages 105 – 106, 1995.

[28] P. W. C. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 Symposium on Interactive 3D Computer Graphics*, pages 95–102, 1995.

[29] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics Annual Conference Series (SIGGRAPH '95)*, pages 39–46. ACM, 1995.

[30] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci.*, 7:217–236, 1978.

[31] A. Narkhede and D. Manocha. Fast polygon triangulation based on seidel's algorithm. *Graphics Gems 5*, pages 394–397, 1995.

[32] J. Rohlf and J. Helman. IRIS performer: A high performance multiprocessing toolkit for real–Time 3D graphics. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 381–395. ACM SIGGRAPH, July 1994.

[33] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering. In *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, June–July 1993.

[34] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *Computer Graphics: Proceedings SIGGRAPH '92*, volume 26, No. 2, pages 65–70. ACM SIGGRAPH, 1992.

[35] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51–64, 1991.

[36] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4–9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 75–82. ACM SIGGRAPH, ACM Press, August 1996.

[37] S. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics: Proceedings of SIGGRAPH'91*, 25, No. 4:61–69, 1991.

[38] G. Turk. Re-tiling polygonal surfaces. In *Computer Graphics: Proceedings SIGGRAPH '92*, volume 26, No. 2, pages 55–64. ACM SIGGRAPH, 1992.

[39] 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*, 3, No. 2:171 – 183, June 1997.

[40] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. In *Proceedings of SIGGRAPH '97 (Los Angeles, CA)*, Computer Graphics Proceedings, Annual Conference Series. ACM SIGGRAPH, ACM Press, August 1997.