Visual Comput (2005)
DOI 10.1007/s00371-005-0355-6

ORIGINAL ARTICLE

Justin Jang
Peter Wonka
William Ribarsky
Christopher D. Shaw

Punctuated simplification of man-made objects

Published online: 22 November 2005
© Springer-Verlag 2005

J. Jang (✉)
Georgia Institute of Technology, GVU Center and College of Computing, Atlanta, GA 30332, USA
jang@cc.gatech.edu

P. Wonka
Arizona State University, PRISM and Dept. of Computer Science and Engineering, Tempe, AZ 85287, USA
peter.wonka@asu.edu

W. Ribarsky
UNC Charlotte, CS Dept. and Charlotte Visualization Center, Charlotte, NC 28223, USA
ribarsky@uncc.edu

C.D. Shaw
Simon Fraser University Surrey, School of Interactive Arts and Technology, 14-660 Central City Tower, 13450 102 Ave, Surrey, BC, Canada V3T 5X3
shaw@sfu.ca

Abstract We present a simplification algorithm for manifold polygonal meshes of plane-dominant models. Models of this type are likely to appear in man-made environments. While traditional simplification algorithms focus on generality and smooth meshes, the approach presented here considers a specific class of man-made models. By detecting and classifying edge loops on the mesh and providing a guided series of binary mesh partitions, our approach generates a series of simplified models, each of which better respects the semantics of these kinds of models than conventional approaches do. A guiding principle is to eliminate simplifications that do not make sense in constructed environments. This, coupled with the concept of "punctuated simplification", leads to an approach that is both efficient and delivers high visual quality. Comparative results are given.

Keywords Mesh simplification · Feature detection · Level-of-detail

1 Introduction

A great deal of current modeling and visualization effort is directed towards triangle meshes with high geometric detail. To maintain high frame rates during interactive visualization, a common strategy is to create different levels of simplification for one object and switch between these representations during runtime. The levels of simplifications are also called levels of detail (or LODs) of the model.

Simplification algorithms are often demonstrated on models from the Stanford scanning repository [31], which includes the well-known models of a bunny, a Buddha statue and Michelangelo's David. These models can be iteratively simplified, where each new level of simplification has one vertex less than the previous one [9, 12].

Although impressive results can be achieved for these models, there is still the lingering problem that current automatic simplification algorithms perform poorly on a large class of man-made objects. Often designers must create simplified versions along with the original versions for these models.

In contrast to the previously mentioned models, which are dominated by smooth differential surfaces, man-made objects are usually dominated by features. These models contain many sharp edges and for large parts of the model the triangular mesh is not an approximation of a smooth differential surface. Instead, the mesh represents the actual piece-wise linear surface. Examples include furniture, machine parts, electronic devices and buildings (see Fig. 1).

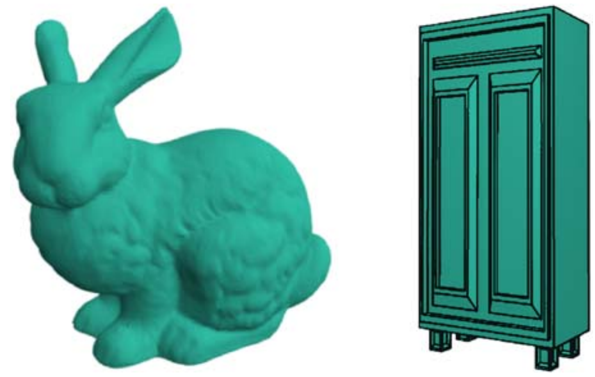


Fig. 1. Models with differential (*left*) and non-differential (*right*) surfaces

If we apply current smooth simplification methods to man-made models, resulting simplifications may deviate from the ideal. The following are weaknesses of per-vertex simplification schemes:

– Small features are merged into new larger ones. This is illustrated in Fig. 2a. Here, the larger features have characteristics not present in the smaller features. In this case, new face orientations are introduced.
– Many intermediate steps of the calculated simplifications are not correct (see Fig. 2b). The simplification of a wheel in Fig. 3 illustrates this problem. The simplification shown on the right has a low visual quality (in this case due to violation of symmetry (see also Fig. 2c) and is not very useful for most applications.
– It is not clear which intermediate simplification steps are meaningful.

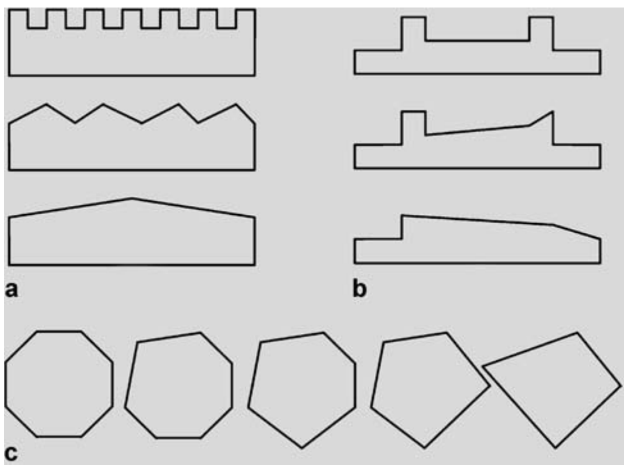


Fig. 2a–c. Problems with successive vertex merging. Small features merge into larger ones (**a**). In the window with frame (**b**) and the circle (**c**)some intermediate steps are intuitively not correct

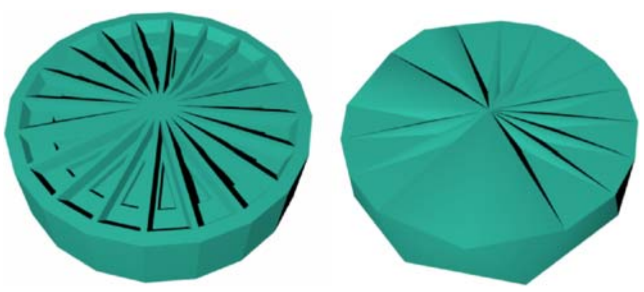


Fig. 3. A model of a wheel (*left*). The simplification on the *right* has low visual quality

In this paper, we take a different approach to the simplification of man-made objects. Our approach seeks to identify features in a model and removes them in a consistent manner. Note that our approach to handle features is different than previous approaches [16, 27]. In previous work, the main goal was to mark certain important parts of the model (features) and try to retain them as long as possible during simplification. However, the actual simplification of these features is again a triangle-by-triangle simplification. In contrast, our approach tries to identify features as clusters of triangles and removes a whole cluster at once in a consistent manner.

Our algorithm draws from ideas of the computer-aided manufacturing community, where designs have to be decomposed into meaningful semantic parts before they can be manufactured. We employ a loop-based feature detection algorithm to create a hierarchical tree that structures the model. To obtain different levels of detail, we remove several features of similar or correlated geometric importance together, rather than in a more continuous LOD fashion. We call this approach punctuated simplification. In this paper, we show that punctuated simplification typically leads to relatively few steps between the full and the simplest model. Further, the intermediate models have better visual quality than per-vertex intermediate models of similar complexity. The simplification tree can be used to extract a large number of possibly view-dependent levels of detail. These LODs can be precalculated or generated during runtime.

We believe that the idea of punctuated simplification is a new contribution to the world of simplification and will help to extend the applicability of automatic simplification algorithms to applications like CAD, computer games, urban [8, 15, 26] and architectural [33] simulation.

2 Related work

2.1 Simplification

There is an extensive literature on the simplification of polygonal models. We will not try to cover this broad literature but will rather focus on representative work most relevant to our approach. We refer the reader to recent surveys for a comprehensive discussion of simplification methods [23].

A variety of per-vertex algorithms have been developed for mesh simplification. These include algorithms that perform vertex merges [9], edge collapses [12], or vertex removals [18]. Some algorithms require manifold topology [12], while others are "topologically-tolerant" [9], but all work one vertex at a time. In addition, there are per-vertex algorithms that attempt to preserve appearance by considering not only errors in surface position caused by the simplification, but also errors due to changes in surface color and curvature [2, 10]. Although in principle a simplification algorithm could be constructed that considers all these aspects of appearance, in practice this is hard to do in an efficient and balanced way [14]. In practice, either geometric or color/texture aspects dominate.

There are also more general vertex merge algorithms based on various multi-vertex clustering mechanisms [20, 30]. These are rather insensitive to topology and in the most general case do not require mesh connectivity at all. The algorithms are fast, work well on large out-of-core meshes, and can produce drastic simplifications. However, it is difficult to specify the output in terms of number of polygons for the algorithm, and the results are not usually as visually pleasing as with per-vertex algorithms.

In general, the vertex merge, removal or clustering operations in all these approaches can be encoded in a tree, which can then be traversed in any order. This gives rise to view-dependent approaches where on-the-fly simplification occurs based on the current user viewpoint [13, 18]. Perspective and distance are taken into account, so that nearby geometry facing the viewer will have more detail than distant or oblique geometry.

Several extensions to the quadric-error approach [3, 9, 16, 27] allow the user to specify important parts of the model. In these extensions, the simplification process simultaneously considers the quadric error and the user specified importance to select candidates for simplification. However, these approaches do not address the problem of how to identify and consistently remove features, but rather determine the simplification order and thereby answer the question of when to remove features.

El-Sana and Varshney [6] present a topology-simplifying approach based on the concept of alpha-hulls. The approach is able to eliminate small holes and protuberances that can hinder and restrict extreme simplification. However, the approach can only deal with relatively small holes and protuberances with small gaps and ignores the size of the protuberances themselves.

Our approach provides a set of simplifications and an order to follow them through, but it does not do this as a sequence of per-vertex simplifications. Alternatively, we do not follow a clustering approach that uses some distance criterion for determining which vertices to merge. Rather, our punctuated simplification approach preserves planes, edges, and orientations until they are deemed candidates for removal, at which point they are removed all at once.

2.2 Feature detection

Feature detection starts with defining what is meant by the word "feature". The definition usually depends on the context of the application and is given only very broadly, as for example "a region of interest on the surface of a part" [28] (see Fig. 4). For the actual implementation of a feature detector, a more precise definition is necessary. A common solution is to give an enumerative list of features. As a consequence, most feature detectors are rule-based and each rule is able to detect a certain type of feature. For a survey of feature detection see [35].

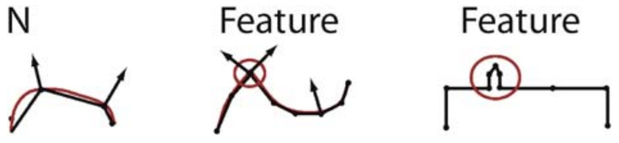


Fig. 4a–c. Three cases of mesh features. **a** Smooth surfaces: normals per vertex; the mesh is only seen as an approximation to the actual (smooth) surface; differential geometry applies. **b** Smooth surfaces with features: these are smooth surfaces that have sharp edges and corners. The edges and corners are called features. **c** Plane-dominant objects: the mesh is the actual geometry; normals are per polygon and not per vertex. A feature is a larger connected part of the mesh

Feature detectors can be based on convex decomposition [17, 34], topology of a dual face-edge graph [5], topology of a face-edge graph in combination with geometric tests [11, 24, 29], or loop detection on the geometry of the model [21, 22].

Our approach is most closely related to loop-based feature detection [21, 22]. The idea is to couple the detection of edge loops that potentially contain a feature together with geometric tests to verify its existence. (Note that what we call the feature here is not the edges in the loop, but the mesh partition bounded by the loop.) We use an adaptation of these loop-based feature detectors in our implementation.

3 Overview

As input to our algorithm we consider triangle meshes that represent a topological 2-manifold with boundary. Given a model that contains non-triangular polygonal faces, the model can be triangulated first and then processed by our algorithm.

Our algorithm accepts models with holes and multiple non-connected parts, but we do not alter the topology of the model during simplification, so the simplification procedure preserves holes and keeps unconnected parts separate. Our algorithm accepts models with self-intersections, but our goal is not to repair erroneous input models. Self-intersections and other errors in the input model can result in unwanted results during simplification.

We do not attempt here to deal with large models. While a lot of simplification research has been devoted to handling large meshes, applications such as games and urban visualization often call for a large number of simple meshes as opposed to a few complex ones. In certain situations, it may be necessary to display drastically simplified meshes with visible approximation error. Thus, it is important to ensure the quality of the coarser approximations. Furthermore, man-made objects, especially constructed objects like buildings and furniture, generally contain more planar or near planar surfaces than organic forms. In approximating a shape, flat regions require much fewer linear facets than curved regions. Therefore, plane-dominant models normally contain fewer polygons than those with an abundance of smooth or curvy regions.

3.1 Algorithm overview

The algorithm has three major parts. We will briefly describe these parts and then give more details about these parts in the next section.

1. Feature extraction – we employ a rule-based loop-finding method to detect the boundaries of a feature on the surface of the model. This feature induces a partition of the mesh in two parts.
2. Hierarchical partitioning – using the feature extraction method, we organize the features (mesh partitions) hierarchically.

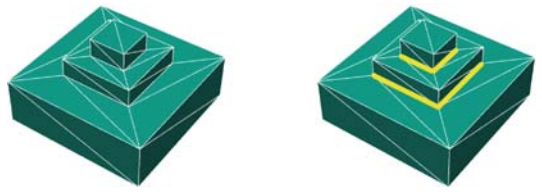


Fig. 5. A simple model (*left*) and two detected loops on the surface of the model shown in yellow (*right*). Not all possible loops are shown

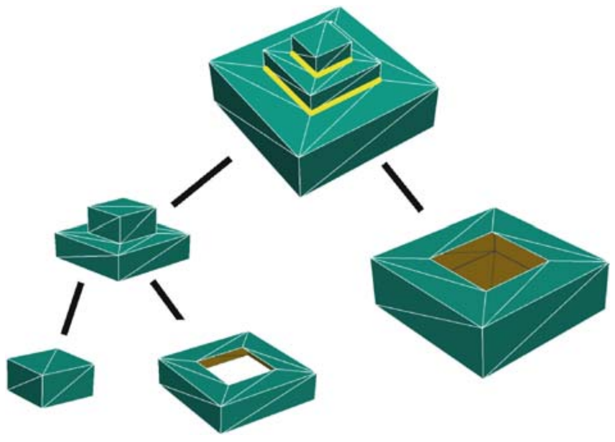


Fig. 6. The partitioning of the model from the previous figure in a hierarchical tree

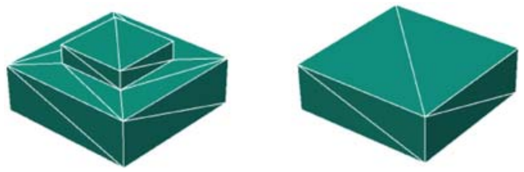


Fig. 7. Two possible simplifications extracted from the tree

3. Simplification – we use the hierarchy to simplify the model.

These three steps are demonstrated on a simple example in Fig. 5, Fig. 6, and Fig. 7.

4 Feature extraction

Our approach can be considered a general framework for simplification. This framework incorporates explicit feature identification and treatment into a system for generating simplified meshes. A feature is any subset of the mesh that can be detected by a set of rules or procedures. (The loop-finding method described here is just one procedure that could be used.) Thus, a feature can be arbitrarily complex. For each feature, there is a corresponding simplification operation. This operation can be fairly general in behavior, so we prefer to call it a simplification treatment. Thus, both the identification and treatment of features are defined procedurally in the framework. The framework is flexible, since it can fall back on a traditional simplification algorithm where features are not present.

Loop-based feature detection is based on finding closed polylines on the surface of a model (a loop). The segments of the polyline are typically edges of the triangulation. In this section, we propose a taxonomy for loop detectors. We classify detectors according to a) the type of edges they detect, b) how many planes are involved in defining the feature, c) the tolerance to noise, and d) the number of loops specifying a feature (ability to detect topological features).

Edge type. Edges can be concave, convex, planar, or virtual. A concave edge is an edge of the triangulation where the adjacent faces form an angle of less than 180 degrees. A convex edge is an edge of the triangulation where the adjacent faces form an angle of more than 180 degrees. A planar edge is an edge of the triangulation where the two adjacent faces are coplanar. A virtual edge is a line segment that crosses a face of the triangulation. Virtual edges are helpful to make the loop finder more independent of the actual triangulation. Fig. 8 illustrates the first three cases.

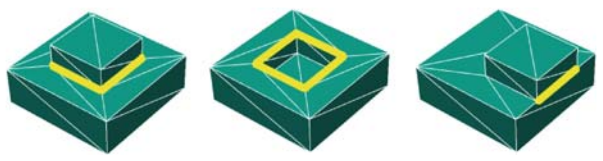


Fig. 8. Three edge types, highlighted in yellow. Left – concave. Middle – convex. Right – planar

Number of planes. The number of planes involved in defining a feature greatly contributes to the complexity of the detector. Typically, one plane means that the feature is contained in a plane of the model. For two planes, the feature is located at an edge. For three planes the feature is typically located at a corner. However, other configurations are possible for features of three or more planes. Fig. 9 illustrates two of these possibilities.

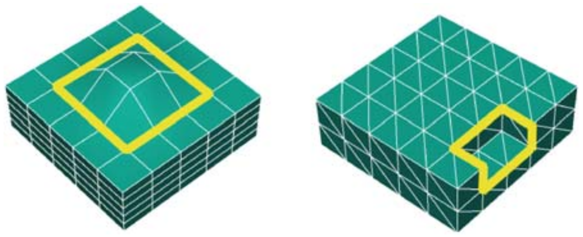


Fig. 9. A feature on a plane (*left*) and a feature on an edge (*right*)

Noise tolerance. The tolerance to noise defines the robustness of the detector. Typically some tolerance is required for all detectors to compensate for numerical imprecision, but scanned data often has a significant level of noise that requires different approaches.

Number of loops. Some feature detection algorithms are also able to detect topological features, such as holes in the model. To be able to detect these features, a single loop is no longer sufficient. To classify these features we can use the number of loops that are necessary to specify the feature.

5 Algorithm details

5.1 Loop finding

We have implemented a greedy, recursive loop finder that is able to detect planar loops. Although faster or more robust algorithms might be found, the focus of the current work was not to improve existing feature detectors.

The loop finder recursively traverses halfedges until a loop is found. A halfedge is a directional edge with a head and a tail vertex. (For details about the halfedge data structure, see [1].) We start by selecting a seed halfedge. To add a halfedge to the loop, the next halfedge emanates from the tail of the last halfedge. We use the following restrictions:

1. The first halfedge h_1, cannot be a planar edge or already belong to a loop.
2. The second halfedge h_2 cannot be collinear with the first halfedge.
3. A halfedge h_i $(i > 2)$ has to lie in the plane formed by h_1 and h_2. This plane is called the loop plane.
4. A halfedge h_i $(i > 2)$ cannot have two adjacent faces that are both coplanar with the loop plane.
5. A halfedge h_i $(i > 2)$ can only extend or close the loop. It is not allowed to touch or cross the loop.

We still need some geometric tests to verify the loop. For example, we discard loops that bound a flat polygon. In the end, it turns out that the branching factor for this constrained search is pretty small and the loop detection only takes a few seconds for the models we used for our tests.

5.2 Hierarchical partitioning

In support of the subsequent simplification phase, our algorithm generates a hierarchy of feature partitions. Given a triangular mesh M as a set of triangles, any given loop L induces a partitioning of M into two subsets, M1 and M2. This binary partition forms the basis of the hierarchy, which emerges as a binary tree of mesh partitions.

We need to answer the following questions to build the tree:

1) Given two mesh partitions M1 and M2 we must decide which mesh is considered to be the feature. This is important, because the feature and the rest of the mesh are treated differently during the simplification phase. We employ a simple heuristic H for the decision. We compute the extent, H(M) = max(d(p1, p2)), where d is the

Euclidian distance metric and p1, p2 are vertices of the mesh. The mesh partition with the smallest non-zero extent is considered the feature. We call this the interior partition. Note that the other partition, called the exterior partition, might be an empty mesh.

2) To build the tree we always select the partitioning with the largest interior partition where the exterior partition is not empty. We choose to use the extent of the mesh as a heuristic. This heuristic ensures that the features are properly nested.

We then construct the tree with a recursive procedure (see Fig. 10.) For the sake of discussion, we choose to position the interior partition as the left child and the exterior partition as the right child. An example tree is given in Fig. 6. For models with many nested features, the tree may contain long runs of branching from the left child node (the internal partition node). For models with many identical features, the tree may contain long runs of branching from the right child node (the external partition node).

```
buildtree(M)
  L = findloops(M)
  For p in L
    [M1, M2] = partition(M, p)
    I = minextent(M1, M2)
    E = maxextent(M1, M2)
    If extent(I) > extent(bestI) then
      bestI = I
      bestE = E
    Endif
  Endfor
  M.left = buildtree(bestI)
  M.right = buildtree(bestE)
```

Fig. 10. Simplified pseudocode of the recursive procedure for constructing the hierarchy

5.3 Simplification

The submesh tree can be used in more than one way to guide simplification. For example, the tree can guide the construction of a sequence of static level-of-detail (LOD) representations.

We need the following three procedures to implement the simplification:

1) Simplify(M, L): For a feature mesh M that is bound by a loop L, we need a simplification treatment that generates a simplified version of the mesh M. We call this Simplify(M, L). The simplification treatment for a planar feature is typically hole-filling [4] (that is triangulation of an arbitrary polygon), but more complex operations are possible [29].

2) Coalesce(M): We need a procedure to reduce the number of coplanar triangles in a mesh. We call this Coalesce(M). We choose to use the framework of Garland and Heckbert [9] for this task. By selecting an error threshold close to zero and specifying relevant loops as constraints, we can achieve the desired effect.

3) ST(M, L): We need a cost function to determine how much error the simplification of a feature introduces. We choose to use a simple metric calculating the surface area of the mesh M minus the surface area of the simplified version of M, that is ST(M, L). Depending on the amount of semantic information available for the model, the cost function can be made more interesting. Along with geometric characteristics, factors such as importance, semantic sensibility, and physical plausibility can be incorporated into such a metric.

The process for generating static LODs is as follows.

1. Pick the lowest cost feature (interior partition) node with mesh M and loop L.
2. Calculate Simplify(M, L) and store the simplified version with the node.
3. Collapse nodes. A node can be collapsed if the feature child (the interior partition node) has been simplified and the other child (the exterior partition node) does not have any further children. To collapse a node we:
   (a) combine the meshes M1 and M2 of the children to obtain M = M1 + M2,
   (b) call Coalesce(M) on the combined geometry and store it with the node.
4. Repeat steps 2–3.

At any time the union of all leaf nodes can be calculated to obtain a valid level-of-detail of the model. To obtain a discrete set of static LODs, we propose the following methods: (a) find the peaks in a histogram of [errors incurred, faces simplified]; (b) find the zero crossings of derivatives of [errors incurred, faces simplified]; (c) round to logarithmic steps; or (d) use thresholds. For our results, we used a moving threshold. That is, when the error passes $e+t$, where e is the last error incurred by the last LOD (initially zero) and t is the threshold, we grab the current LOD and update e. Note that, in general, applying any of these methods to a traditional simplification sequence like a quadric simplification will not produce the same results as applying them to the hierarchically partitioned mesh, which has eliminated meaningless or incorrect simplification steps.

6 Results

In this section, we demonstrate that the proposed method gives good results and that the consideration of features is in fact crucial to get meaningful simplifications for man-made objects. We demonstrate our results by comparing our simplifications with the algorithms proposed by Garland and Heckbert (quadrics/Qslim) [9] and Lindstrom and Turk (memoryless simplification) [19]. We did not try to optimize our implementation for speed, but the simplification times are still reasonable. To give a rough estimate,

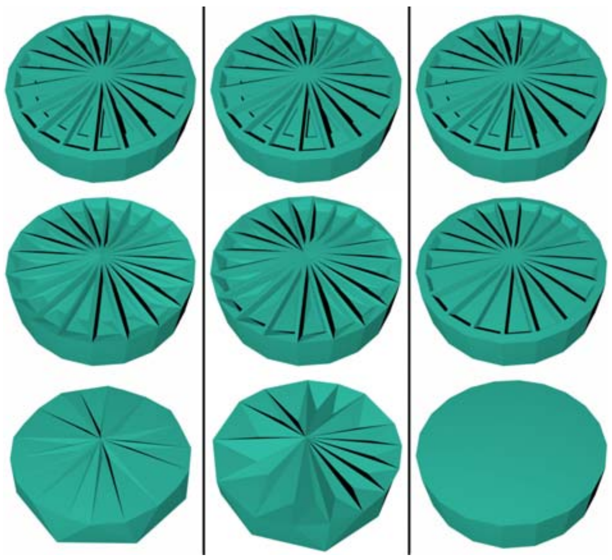


Fig. 11. Three levels-of-detail obtained with Qslim (*left*), memoryless simplification (*center*), and our algorithm (*right*). The three LODs contain 558 (*top row*), 318 (*second row*), and 76 (*bottom row*) triangles. Our algorithm automatically extracts these levels-of-detail. In contrast to the other two methods, both simplifications of our method make sense and have good visual quality. Note that the LODs in the top row are geometrically identical to the original (560 triangles)

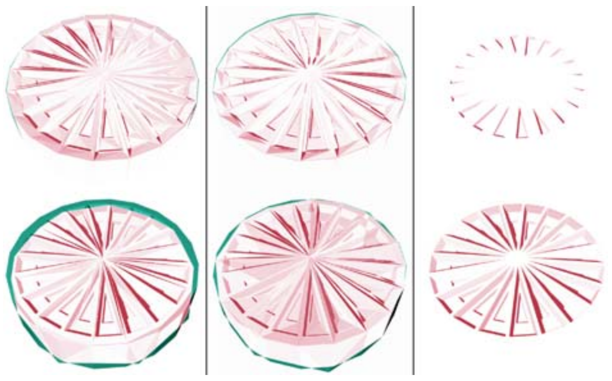


Fig. 12. Difference images (negative image) of the second and third rows in Fig. 11 with respect to the original (i.e. *top row*)

the simplification time is under 5 seconds for the models shown here.

The first model is the model of the wheel shown in Fig. 3. Figure 11 shows three LODs obtained using the three simplification approaches. See the figure caption for a description of the results. Figure 12 shows the image differences for each of these LODs.

The second model used to illustrate our method is an armoire (see Fig. 13 and Fig. 14). We compare again against the original model.

Figure 15 shows a LOD sequence of a window model produced with our method.

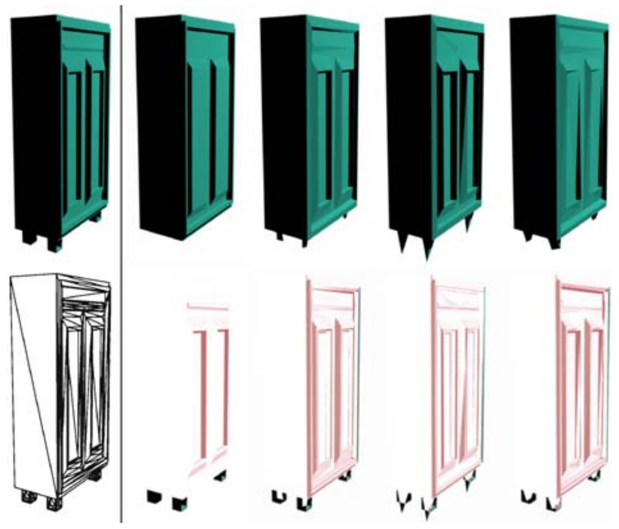


Fig. 13. *Top left*: the original armoire model with 476 polygons. *Bottom left*: wireframe of the original. *Top row*: selected simplifications using our algorithm (94 triangles), Qslim (94 triangles), memoryless simplification (94 triangles), and Maya (96 triangles) [25]. *Bottom row*: corresponding difference images (negative image) of the simplifications to the original

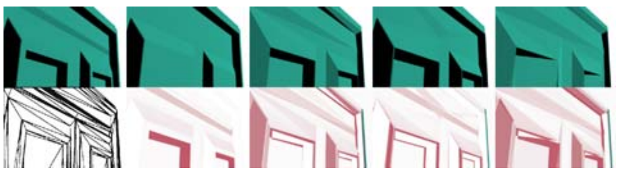


Fig. 14. Close-up of the armoire. All images correspond to those in Fig. 13. Notice that triangle-shaped artifacts appear on the Qslim, memoryless, and Maya [25] results. Also notice the difference in size and angle of the bevel on the armoire doors

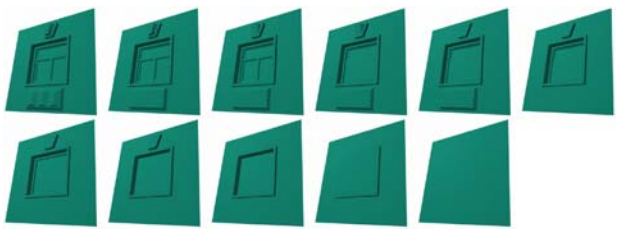


Fig. 15. A sequence of simplifications of a window model is automatically extracted with our algorithm. Entire features are removed per step, while the rest of the model is retained

7 Discussion

Figures 16–18 compare the max, mean, and RMS errors of three approaches, Qslim/quadric simplification (QS), memoryless simplification (MS), and our punctuated simplification approach (PS). Forward errors (original-to-sim-

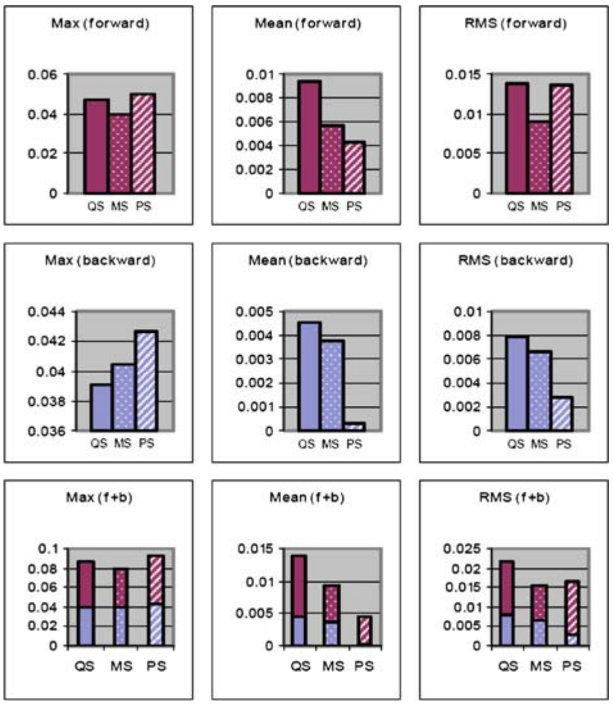


Fig. 16. Max, mean, and RMS error values for the wheel model simplifications of 318 triangles. Forward (original-to-simplified), backward (simplified-to-original), and forward plus backward errors are shown for Qslim (QS – *left*), memoryless simplification (MS – *middle*), and our method (PS – *right*)

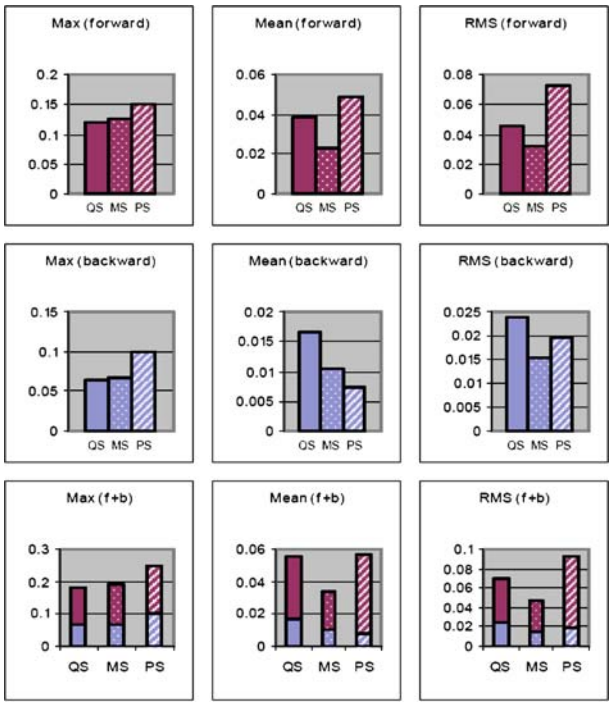


Fig. 17. Max, mean, and RMS error values for the wheel model simplifications of 76 triangles. Forward, backward, and forward plus backward errors are shown for Qslim (QS), memoryless simplification (MS), and our method (PS)

plified) and backward errors (simplified-to-original) along with forward plus backward errors are shown. Notice that for the models tested, the visual quality of the simplifications is not fully represented or revealed by the metrics. The difference images seem to suggest that punctuated simplification is better than the other approaches. However, the max, mean, and RMS metrics give mixed results and even results counter to what one gets from visual examination. This confirms that different metrics, such as one based on perception [32], are sometimes necessary for evaluating simplification quality, especially for constructed models like these. Furthermore, even a straightforward measure of RMS image difference cannot account for qualitative inaccuracies, such as violation of symmetry (Fig. 11) or the creation of misrepresentative shapes (Fig. 14).

8 Conclusions

In this paper, we described the punctuated simplification approach for simplifying man-made objects. We argued that previous simplification algorithms are in fact mainly applicable to models that are dominated by smooth surfaces and that for another large class of objects (that we

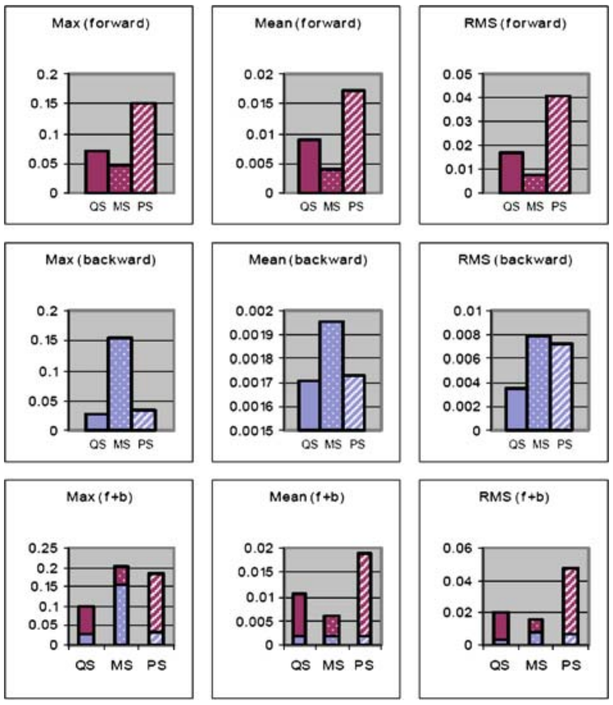


Fig. 18. Max, mean, and RMS error values for the armoire model simplifications of 94 triangles (from LOD 5 of 10). Forward, backward, and forward plus backward errors are shown for Qslim (QS), memoryless simplification (MS), and our method (PS)

call man-made objects) they often fail to calculate meaningful results. We demonstrated that the recognition and consistent removal of features is essential to obtain good perceptual quality for the simplified models. We presented an initial algorithm to attack this problem and gave a visual comparison to previous methods.

We believe the simplification of man-made objects is an essential problem, because these models are at the heart of many visualization applications.

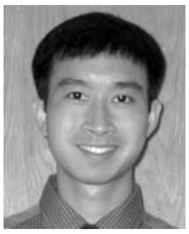
For future research, we envision several ways to extend the implementation of the basic approach. Similar to other level-of-detail algorithms, we plan to handle texturing and view-dependent levels-of-details. Additionally, we think that integration with other traditional simplification algorithms would be important to obtain a complete system for simplification.

The major drawback of the current approach is that general and robust feature detection is still a challenge. We expect to study this question in the future.

Acknowledgement This work is supported by the Department of Defense's MURI program, administered by the Army Research Office. We would also like to acknowledge the support of NSF and FWF grant number J2329-N04.

References

1. Botsch, M., Steinberg, S., Bischoff, S., Kobbelt, L.: OpenMesh – a generic and efficient polygon mesh data structure. OpenSG Symposium (2002)
2. Cohen, J., Olano, M., Manocha, D.: Appearance-preserving simplification of polygonal models. In: Proceedings ACM SIGGRAPH'98, pp. 115–122 (1998)
3. Coors, V.: Feature-preserving simplification in web-based 3D-GIS. First International Symposium on Smart Graphics, New York (2001)
4. de Berg, M., Van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry. Springer, Berlin Heidelberg New York (2000)
5. De Floriani, L.: Feature extraction from boundary models of three-dimensional objects. IEEE Trans Pattern Anal Mach Intell **11(8)**: 785–798 (1989)
6. El-Sana, J., Varshney, A.: Topology simplification for polygonal virtual environments. IEEE Trans Visual Comput Graph **4(2)**:133–144 (1998)
7. Erikson, C., Manocha, D., Baxter III, W.V.: HLODs for faster display of large static and dynamic environments. In: Proceedings ACM Symposium on Interactive 3D Graphics, pp. 111–120 (2001)
8. Früh, C., Zakhor, A.: 3D model generation for cities using aerial photographs and ground level laser scans. In: Proceedings IEEE Computer Vision and Pattern Recognition, pp. 31–38 (2001)
9. Garland, M., Heckbert, P.: Surface simplification using quadric error metrics. In: Proceedings ACM SIGGRAPH '97, pp. 209–216 (1997)
10. Garland, M., Heckbert, P.: Simplifying surfaces with color and texture using quadric error metrics. In: Proceedings IEEE Visualization'98, pp. 263–269 (1998)
11. Gavankar, P., Henderson, M.R.: Graph-based extraction of protrusions and depressions from boundary representations. Comput Aided Des **22(7)**:442–450 (1990)
12. Hoppe, H.: Progressive meshes. In: Proceedings ACM SIGGRAPH'96, pp. 99–108 (1996)
13. Hoppe, H.: View-dependent refinement of progressive meshes. In: Proceedings ACM SIGGRAPH'97, pp. 189–198 (1997)
14. Jang, J., Ribarsky, W., Shaw, C., Wonka, P.: Appearance-preserving view-dependent visualization. In: Proceedings IEEE Visualization, pp. 473–480 (2003)
15. Jepson, W., Liggett, R., Friedman, S.: Virtual modeling of urban environments. Presence **5(1)**:72–86 (1996)
16. Kho, Y., Garland, M.: User-guided simplification. In: Proceedings ACM Symposium on Interactive 3D graphics, pp. 123–126 (2003)
17. Kim, Y.S.: Recognition of form features using convex decomposition. Comput Aided Des **24(9)**:461–476 (1992)
18. Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L.F., Faust, N., Turner, G.A.: Real-time, continuous level of detail rendering of height fields. In: Proceedings ACM SIGGRAPH'96, pp. 109–118 (1996)
19. Lindstrom, P., Turk, G.: Fast and memory efficient polygonal simplification. In: Proceedings IEEE Visualization'98, pp. 279–286
20. Low, K.L., Tan, T.S.: Model simplification using vertex clustering. In: Proceedings ACM Symposium on Interactive 3D Graphics, pp. 75–82 (1997)
21. Lu, Y., Gadh, R., Tautges, T.J.: Feature decomposition for hexahedral meshing. In: Proceedings ASME Ddesign Automation Conference (1999)
22. Lu, Y., Gadh, R., Tautges, T.: Volume decomposition and feature recognition for hexahedral mesh generation. 8th International Meshing Roundtable, SAND99-2288, Sandia National Laboratories, pp. 269–280 (1999)
23. Luebke, D.: A developer's survey of polygonal simplification algorithms. IEEE Comput Graph Appl **21(3)**:24–35 (2001)
24. Marefat, M., Kashyap, R.L.: Geometric reasoning for recognition of three-dimensional object features. IEEE Trans Pattern Anal Mach Intell **12(10)**:949–965 (1990)
25. Maya® version 6.0, April 9 (2004)
26. Parish, Y., Mueller, P.: Procedural modeling of cities. In: Proceedings ACM SIGGRAPH 2001, pp. 301–308 (2001)
27. Pojar, E., Schmalstieg, D.: User-controlled creation of multiresoltion meshes. In: Proceedings ACM Symposium on Interactive 3D Graphics, pp. 127–130 (2003)
28. Pratt, M., Wilson, P.R.: Requirements for support of form features in a solid modeling system. Technical Report R-85-ASPP-01, CAM-I Inc, Arlington Texas, June (1985)
29. Ribelles, J., Heckbert, P., Garland, M., Stahovich, T., Srivastava, V.: Finding and removing features from polyhedra. In: Proceedings ASME Design Engineering Technical Conference, September (2001)
30. Rossignac, J., Borrel, P.: Multi-resolution 3D approximations for rendering complex scenes. In: Geometric Modeling in Computer Graphics, pp. 455–465, Springer (1993)
31. The Stanford 3D Scanning Repository. http://graphics.stanford.edu/data/3Dscanrep/ (2004)
32. Williams, N., Luebke, D., Cohen, J., Kelley, M., Schubert, B.: Perceptually guided simplification of lit, textured meshes. In Proceedings ACM Symposium on Interactive 3D Graphics, pp. 113–121 (2003)
33. Wonka, P., Wimmer, M., Sillion, F., Ribarsky, W.: Instant architecture. In: Proceedings ACM SIGGRAPH 2003, pp. 669–678 (2003)
34. Woo, T.: Feature extraction by volume decomposition. In: Proceedings Conference CAD/CAM Technology in Mechanical Engineering, MIT (1982)
35. Wu, M.C., Liu, C.R.: Analysis on machined feature recognition techniques based on B-rep. Comput Aided Des **28(8)**:603–616 (1996)



JUSTIN JANG is a Ph.D. student in the Graphics, Visualization, and Usability Center at the Georgia Institute of Technology. He received his B.S. degree (2000) in Computer Engineering from Auburn University. His research interests include shape analysis, geometry processing, and visualization.

PETER WONKA joined the computer science faculty of Arizona State University (ASU) as Assistant Professor in 2004 after two years as a post-doctorate researcher at the Georgia Institute of Technology. He received his Ph.D. from the Vienna University of Technology in 2001. His research interests include various topics in computer graphics, especially real-time rendering and procedural modeling. Peter Wonka is a member of the PRISM lab at ASU.

WILLIAM RIBARSKY holds the Bank of America Endowed Chair in Information Technology at UNC Charlotte and is the founding director of the Charlotte Visualization Center. He received a Ph.D. in physics from the University of Cincinnati. His research interests include visual analytics, 3D multimodal interaction, bioinformatics visualization, virtual environments, visual reasoning, and interactive visualization of large-scale information spaces. Dr. Ribarsky is the former Chairman and a current Director of the IEEE Visualization and Graphics Technical Committee. He also chairs the Steering Committees for the IEEE Visualization Conference and the IEEE Virtual Reality Conference, the leading international conferences in their fields. Dr. Ribarsky co-founded the Eurographics/IEEE visualization conference series (now called EG/IEEE EuroVis) and led the effort to establish the Virtual Reality Conference series. Dr. Ribarsky has published 100 scholarly papers, book chapters, and books. He has received competitive research grants and contracts from NSF, ARL, ARO, ONR, AFOSR, DARPA, NASA, NIMA, and several companies.

CHRISTOPHER D. SHAW is an Associate Professor in the School of Interactive Arts and Technology at Simon Fraser University Surrey. Previously, he was a Senior Research Scientist in the College of Computing at the Georgia Institute of Technology. His research and teaching efforts over the years have focused on areas that require a broad integrative knowledge of human-computer interaction, human perception, and computing. His research efforts in virtual environments resulted in the development of the MR Toolkit, which is VR software that has been licensed at over 600 research sites worldwide since 1994. Shaw received a B.Math degree from the University of Waterloo in 1986, an MSc. in Computing Science from the University of Alberta in 1988, and a Ph.D. in Computing Science from the University of Alberta in 1997. He is an associate editor of IEEE Transactions on Visualization and Computer Graphics.