Linear Time Mesh Simplification with Reliable Error–Bounds

J. Vorsatz, K. Kähler, L. Kobbelt and H.-P. Seidel

Max-Planck Institut für Informatik, Computer Graphics Group
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
Email:{vorsatz,kaehler,kobbelt,hpseidel}@mpi-sb.mpg.de

Abstract

This paper proposes an incremental mesh reduction scheme that provides fine grained control over the approximation error while being linear in time in the number of vertices. We use the well–known half–edge collapse as our atomic decimation operation but we differ from previously reported methods in that we use a sequence of oriented bounding-boxes to track the decimation error instead of retaining the complete vertex information of the original model. This reduces storage costs and computation time while still providing a reliable upper bound for the deviation from the original data. We also propose different error accumulation strategies which makes the algorithm adaptable to different application scenarios.

1 Introduction and Prior Work

Even though the speed of computers increased drastically over the years, they will probably never be able to keep up with the need for more accurate and thus larger data-sets. This is especially true for polygonal objects, which became increasingly popular for representing arbitrary geometric objects. In particular, triangle meshes have become commonplace for describing highly detailed surfaces. These large meshes typically result from 3D acquisition techniques, e.g. laser range scans, MRI volumetric imaging combined with iso-surface extraction or computer-aided design. In order to display or modify the meshes within reasonable response times, it is necessary to remove redundant information. Hence numerous mesh reduction algorithms have been proposed in the literature during the last decade. The general goal is to take a triangulated surface as input and generate a simplified version of it while meeting some target criteria, e. g. a prescribed face–count. In CAD applications in particular, a maximal geometric deviation from the source model has to be guaranteed.

In addition to schemes that globally change the connectivity [12, 10], several methods that perform vertex removal incrementally have won more and more recognition. Pioneering work in this field has been done by [11] where they retriangulate the hole resulting from vertex removal and record the error with respect to some approximating plane. This error function is used to determine the removal order. Many other similar schemes have subsequently been proposed. Basically they differ in the kind of error estimation and in the type of retriangulation, but they all have in common, that the error metric and/or a "quality criterion" steers the decimation order of the vertices.

Hamann [5], for example aims for a low triangle count in locally flat regions and thus utilizes a curvature criterion based on the current decimation level. This memory–efficient method does not maintain any geometric history and therefore cannot take global approximation error bounds into account. To be able to do so, the more recent simplification algorithms maintain a geometric history. The most notable examples are [6, 4, 9, 3, 7]. Specifically [6] is worth mentioning in this context, since it introduces the *edge collapse* (see Fig. 1). After the vertex removal, the position of the other involved vertex is optimized with respect to the error functional. This leads to *Progressive Meshes*, a versatile multiresolution representation for triangle meshes.

Kobbelt et. al. [7] propose the *half–edge collapse* (see Fig. 1), a simplification of the edge collapse, where one vertex is removed by 'dragging' it along an edge of the mesh. To meet the error–bound almost exactly a list of all removed vertices is maintained and the intuitive one–sided Hausdorff distance is used. Having such an intuitive measure is an enormous advantage e.g. for the simplification of geometric models in the area of CAD. But as all point–based methods, it is not linear in runtime and the storage requirements depend on the complexity of the input mesh.

[9, 3] track the approximation error through accumulated squared distances to sets of planes or error quadrics respectively. Hence the evaluation cost of the error metric does not depend on the size of the input mesh and the scheme is still able to stay within a global error bound. However, the interpretation of these error criteria might become difficult.

The algorithm presented by Cohen and coworkers [1] is able to meet almost all goals. It keeps track of surface deviation from the original surface by enclosing removed vertices in axis–aligned error boxes (instead of storing them explicitly and taking all of them into account to find the next collapse). They merge these boxes after every decimation step in such a way that they are able to propagate the error to successive levels. But, due to the nature of these bounding volumes, the resulting approximation error is only a very conservative estimate.

In this paper we present a simplification algorithm that uses the half–edge collapse due to its simplicity, local control, and full reversibility. Our method is similar to [1], but we use oriented bounding boxes to achieve tighter error bounds. We also investigate different strategies for propagating and merging these error volumes.

The main features of our approach are:

- The error metric measures the real world geometric deviation from the source model.
- The runtime is linear in the number of vertices so that the algorithm does not slow down towards the end of the decimation process.
- The memory requirements are lower than for previously reported schemes. This makes our scheme applicable for larger models.
- The tradeoff between meeting exact error bounds and optimized runtime/storage–costs can be controlled by the user.

In the remainder of the paper we describe the algorithm in detail and introduce four merging strategies. In Section 3 we validate the results we achieved and give an outlook for further investigations in Section 4.

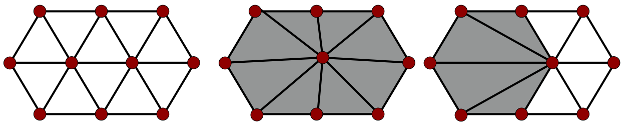

Figure 1: general edge collapse (center) and half edge collapse operations; the support of each operator is shown in grey

2 The Algorithm

2.1 Point Accumulation

A standard incremental mesh simplification algorithm has the following basic structure:

```
For all vertices v in mesh:
  For all vertices v' in star(v)
    p := calc_prio_for_collapse(v, v')
    insert (v,v',p) into queue

while (queue not empty)
```

```
first queue element -> (v,v')
collapse(v, v')
For all edges (w,w') in affected region:
  p := calc_prio_for_collapse(w,w')
  update_queue_entry(w,w',p)
```

In the context of this paper, we are interested in the `calc_prio_for_collapse()` function, where the cost of a candidate operation – part of which is the approximation error – is computed.

To keep track of the approximation error, the triangles of the current (simplified) surface have to be compared to the original surface. If we adopt the argument that only the actual *vertices* of the original mesh hold the data we need to preserve – as is the case in 3D scans, where the surface is merely connecting the measured data points – it is appropriate to restrict to measuring the one–sided Haussdorf distance, i.e. the distance from those surface points to the current surface (and not vice versa).

To avoid calculating distances from all removed vertices to all remaining triangles, we assign each removed vertex to the closest triangle in the affected mesh area. Existing vertex assignments from previous simplification steps have to be reconsidered with each change of geometry. To determine the triangle a point has to be assigned to, we employ the *plane sweep* technique [7], which is fast and only rarely errs in assigning a vertex to the closest triangle – this is still safe, since the estimation is conservative.

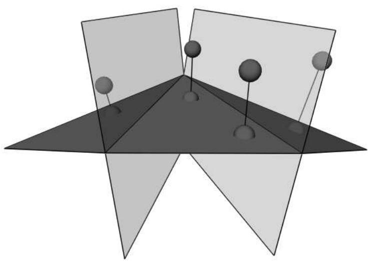

Figure 2: Plane sweep to assign vertices to nearest triangles

The basic observation is now that this method of repeatedly collecting and assigning removed vertices to triangles tends to slow down the decimation process due to the increasing amount of per–vertex computations necessary to assert the one–sided Hausdorff distance criterion. In the extreme case, where a mesh of n vertices is reduced to one triangle, in the last simplification step $n-3$ vertices have to be tested against this polygon.

2.2 Box Accumulation

Thus, within an incremental framework where we remove one vertex at a time, we have to make the cost of one removal operation constant to achieve overall linear running time.

The main idea of our method is to approximate the point cloud representing the removed vertices by bounding volumes, where ideally only one of these volumes is assigned to a triangle. Instead of reassigning points to the closest triangle in a collapse operation, we merge and assign these volumes, without retaining the original points.

The obvious tradeoff here is between a close fit to the original point cloud (directly linked to the approximation error estimate) and the computational overhead.

Our requirements for a suitable bounding volume representation are:

- low storage
- fast on–the–fly computation
- merge operation in constant time
- good approximation properties

Since the areas where surface points are removed during simplification are usually relatively flat, we opt for oriented bounding boxes, which provide reasonable approximation quality, are easy to implement and don't cause too much computational overhead. We consider axis–aligned boxes unsuitable, because they don't align to the surface, resulting in a much too conservative approximation bound.

When a vertex is removed, we store a bounding box enclosing it, which initially has its center at the vertex position and no extent. It then grows or is discarded through the subsequent merging operations. Figure 3 shows a

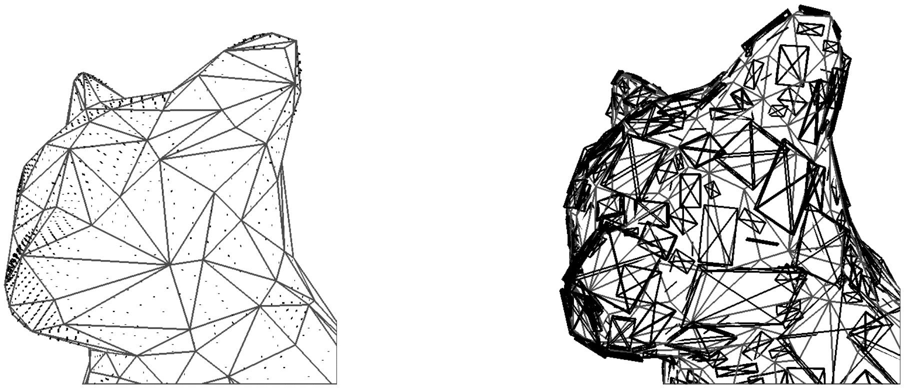

Figure 3: Left: Simplification with point assignment; right: same model, but accumulating oriented bounding boxes

closeup of simplification collecting points and boxes, resp.

Given a candidate vertex v that has to be ranked before updating the priority queue, the point–accumulation algorithm evaluates the quality of the mesh for all possible half edge collapses in the star of v, where the main load is on assigning and evaluating distances to collected vertices:

```
v_set := assigned_vertices_in_star(v)
v_set.add(v)

For all edges (v,v') in star(v)
  t := triangulation_if_collapse(v,v')
  assign_vertices(v_set,t)
  error := max_distance(v_set,t)
```

This procedure is now modified to work with oriented bounding boxes. Other than points, these boxes do have spatial extent, i.e. a box can overlap multiple neighboring triangles. If we want to use the "assignment to closest triangles" approach, some extra bookkeeping is necessary and we introduce a reference count for each bounding box to enable proper sharing.

The only two steps in the outline of the evaluation algorithm given above that change considerably are:

- reassigning and adjusting collected boxes instead of vertices;
- computing distances with respect to boxes.

The main problem in reassigning bounding boxes is how they should be merged into larger boxes, since we attempt to keep the number of boxes per triangle constant. The following section discusses this in detail.

2.3 Merging Strategies

Given a set of bounding boxes collected from $star(v)$, we started by employing the *plane sweep* technique in a way similar to the point–based method to determine to which triangle(s) a box should be assigned. If a box is completely enclosed between two adjacent planes, it will be assigned to only one triangle; if it is intersected by one or more planes, it will be shared (a sufficiently large box can be shared between a great number of triangles).

The assigned boxes are then merged; this operation is crucial, in that we have to make a decision towards more precise approximation error estimates or lower number of retained bounding volumes. Since the distance of a box to a triangle is measured as the maximum distance to the corners, the growth of the boxes is an important factor.

We have implemented three merging strategies in conjunction with the plane sweep method so far:

Contained: Only non–shared bounding boxes assigned to the same triangle are merged. This is the "safest" method and introduces the lowest degree of overestimation. Unfortunately the number of boxes per triangle still increases considerably over the run time of the algorithm, because many boxes positioned across edges are never merged.

Star: This strategy merges all bounding boxes together which are completely contained in $star(v)$. This results in a lower number of boxes, since boxes crossing the inner edges of the updated cell can be merged. Boxes on the cell boundary still remain unmergeable. The overestimation is higher with this method, because we are merging boxes across generally non–planar regions of the mesh.

Intersect: Here *all* bounding boxes that are at least partially assigned to a triangle are merged into one. Only one bounding box is "owned" by any triangle. This leads to rapidly growing bounding boxes and thus very low approximation quality.

Figure 4 shows how these strategies apply to a 2D half edge collapse. The differences in size and number of resulting bounding boxes are clearly visible.

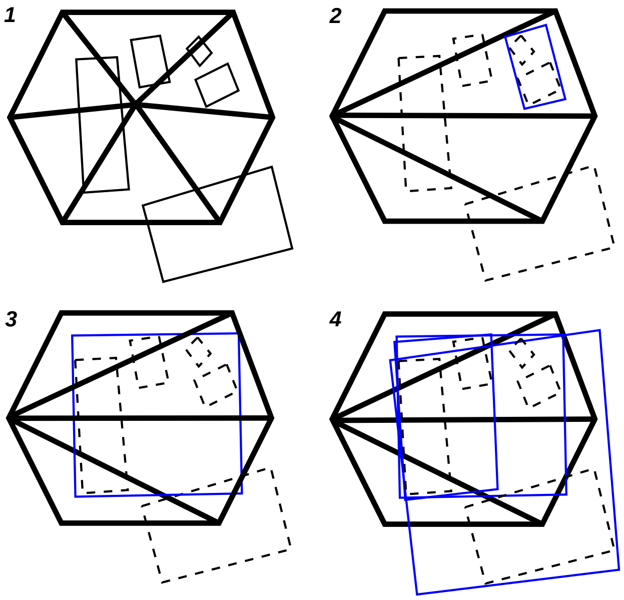

Figure 4: Different merging strategies applied to the support of the half edge collapse operator; 1) before collapse; 2) contained; 3) star; 4) intersect

We found that the "star" method provides the most reasonable quality/performance tradeoff. More sophisticated merging strategies are imaginable, e.g. merging boxes which share the same edge to alleviate the abovementioned problem of an increasing number of small boxes crossing triangle edges.

Though the run–time behavior of the decimation algorithm using these strategies shows improvements over the point accumulation algorithm, linearity is still not achieved: All of these methods, especially the third one, produce successively larger bounding boxes over the iterations of the decimation algorithm, which in turn means that these boxes cover and are shared between many triangles. So, even though the number of boxes actually belonging to a triangle can be small, the number of shared boxes that have to be tested against the surface doesn't really remain constant.

Due to these intermediate results, we came up with a simpler method which does guarantee linear time behavior of the decimation process. Every triangle is initially assigned an empty bounding box. When a vertex is removed, the box belonging to each of the triangles in the affected region is resized and realigned to enclose the previous volume and the removed vertex. The boxes belonging to the triangles that are deleted in the collapse operation are also added to the remaining triangles in the cell. This method causes some overlap of the bounding boxes, but asserts that each current triangle's box encloses all of the original surface points that have to be compared against the triangle (see figure 5). An additional advantage of the method is that we don't have to perform plane sweeps anymore, since assignment is fixed.

2.3.1 Merging Details

Since we are restricted to constant time per merge operation, we cannot compute the union of two oriented boxes from the sum of the original point clouds they enclose (this would take up more and more time as the point clouds grow), which of course would give the best possible result. Instead we only compute a new oriented box from the corner

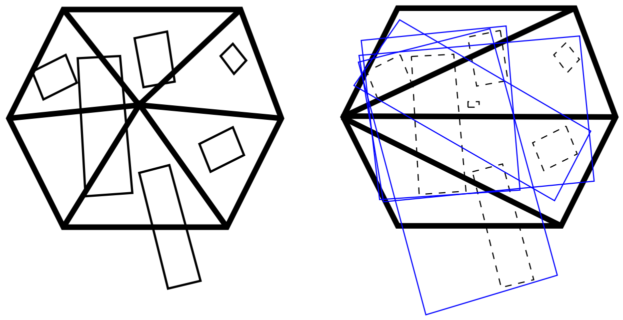

Figure 5: Merging strategy without plane sweep

points of the boxes which are to be merged. If the number of boxes in a merge operation is constant, the computational overhead for merging is constant as well.

To compute an oriented bounding box for m given corner points $p_0, \ldots, p_{m-1}$, we first build the 3×3 so–called *covariance matrix* C, which is a well–known statistical tool and is used here to measure the tendency of the x,y, and z coordinate values to vary together:

$$C_{i,j} = \sum_{k<m} (p_{ki} - \bar{p}_i)(p_{kj} - \bar{p}_j)$$

where $\bar{p}$ is the centroid of the point set. The eigenvectors of this matrix form an orthogonal frame and the size of the box is then determined by projecting the points onto these axes. Extra care has to be taken for special cases where not all three eigenvectors exist, e.g. when the points lie in a plane.

Computing the oriented bounding boxes is more expensive than assigning and measuring distances to only a few vertices. As can be seen from figure 6, in our current implementation the method amortizes at decimation rates below approx. 2%, when (on average) more than 50 vertices have been assigned to each remaining triangle. Since all these calculations are operating on 3×3 matrices only, they can in principal be carried out much more efficiently [2].

One could simply store and sum up the covariance matrices for efficiency[3], but this way we would include some points into the equation multiple times, and the alignment of the resulting boxes would be biased towards the "center of mass" rather than the geometrical configuration. Either way, we still have to compute the eigenvectors and determine the size of the box, which accounts for the main portion of the computational overhead.

3 Results

Figure 6 shows a plot of the decimation of a typical CAD model (part of a car hood) with 540960 triangles downto 560 triangles and serves to demonstrate how the vertex accumulation algorithm breaks in at very high simplification levels as a result of the massive number of vertices being assigned to each triangle. The bounding box method on the other hand continues in a linear fashion.

An evaluation of the implemented merging strategies shows the expected tradeoff between accuracy and performance: The strategy that remains closest to linear runtime behavior delivers the worst approximation behavior. The most accurate strategy exhibits a clearly visible loss of performance, though the asymptotic behavior of the algorithm is still better than with the point accumulation technique.

Figure 7 shows the same model (a 3D scan of a wooden toy elk) reduced with the original vertex collection vs. our linear bounding box strategy, where the same surface distance error tolerance of 3.0 was specified. The actual maximum distances of the original surface points to the decimated model were measured independently afterwards and are also given. As expected, the box method error estimate is more conservative on the rounded parts of the surface (e.g. the wheels on the elk model), while decimation of the flat parts performs essentially just as well.

In summary the results show that the proposed method is advantageous by comparison to the point–accumulation algorithm for situations of extreme simplification, especially considering that the current implementation has not yet been fully optimized. For moderate reduction ratios the approximation error

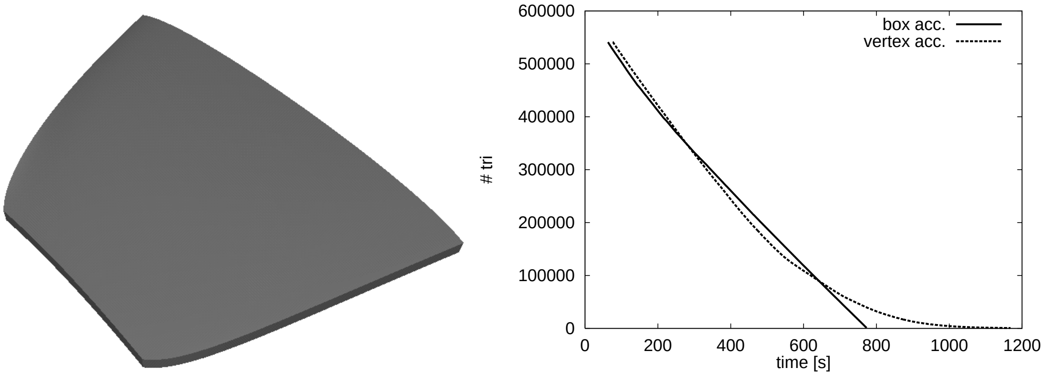

Figure 6: Reduction rates of a densely sampled car hood part using standard vertex accumulation vs. linear bounding box method

Figure 7: Toy elk model simplified using different strategies: left: original (149274 triangles); center: point accumulation method (6.94%, $\epsilon_{max} = 0.699$); right: linear box method (8.9%, $\epsilon_{max} = 0.405$)

is comparable to the old method, but varies widely with drastic reduction. The error is very sensitive to the bounding box merging strategy chosen, indicating that more work in that part of the algorithm would be worthwhile.

4 Conclusion

We presented a new algorithm for incremental mesh decimation using the estimated one–sided Hausdorff distance error measure that is running in linear time in the number of vertices.

The advantages of our method lie in extreme simplification of large data sets, where the linear runtime behavior leads to noticeable time savings while still having a useful bound on the approximation error.

Currently, our implementation is restricted by the size of the main memory, so that the advantage in not having to keep the complete geometric information through the decimation process cannot be exploited (since the mesh has to be loaded completely into memory first). In combination with "out–of–core" methods [8] we expect to be able to handle much larger data–sets, while retaining the advantageous properties of our algorithm. Our scheme should apply well to component–wise processing of the data since the global error information is stored in the bounding boxes with minimal memory consumption.

Since the standard vertex collection algorithm performs almost linear, and even a bit faster on low to medium reduction levels, we would like to investigate a hybrid scheme, where the decimation process starts by collecting vertices and then switches to the bounding box method. By introducing boxes in a later stage of the process, we expect much tighter error bounds.

References

[1] Jonathan Cohen, Dinesh Manocha, and Marc Olano. Simplifying polygonal models using successive mappings. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization ´97*, pages 395–402. IEEE, November 1997.

[2] Robert L. Cromwell. Efficient eigenvalues for visualization. In Paul Heckbert, editor, *Graphics Gems IV*, pages 193–198. Academic Press, Boston, 1994.

[3] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[4] André Guéziec. Surface simplification with variable tolerance. In *Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, pages 132–139, November 1995.

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

[6] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[7] Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. A general framework for mesh decimation. In *Graphics Interface*, pages 43–50, June 1998.

[8] P. Lindstroem. Out–of–core simplification of large polygonal models.

[9] Remi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3):C67–C76, C462, September 1996.

[10] J. Rossignac and P. Borrel. Multiresolution 3D approximation for rendering complex scenes. In *Second Conference on Geometric Modelling in Computer Graphics*, pages 453–465, June 1993. Genova, Italy.

[11] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. volume 26, pages 65–70, July 1992.

[12] Greg Turk. Re-tiling polygonal surfaces. volume 26, pages 55–64, July 1992.