Fast and Accurate Hausdorff Distance Calculation between Meshes

Michael Guthe Pavel Borodin Reinhard Klein
University of Bonn
Institute of Computer Science II
Römerstraße 164
D-53117, Bonn, Germany
guthe@cs.uni-bonn.de borodin@cs.uni-bonn.de rk@cs.uni-bonn.de

ABSTRACT

Complex models generated e.g. with a laser range scanner often consist of several thousand or million triangles. For efficient rendering this high number of primitives has to be reduced. An important property of mesh reduction – or simplification – algorithms used for rendering is the control over the introduced geometric error. In general, the better this control is, the slower the simplification algorithm becomes. This is especially a problem for out-of-core simplification, since the processing time quickly reaches several hours for high-quality simplification.

In this paper we present a new efficient algorithm to measure the Hausdorff distance between two meshes by sampling the meshes only in regions of high distance. In addition to comparing two arbitrary meshes, this algorithm can also be applied to check the Hausdorff error between the simplified and original meshes during simplification. By using this information to accept or reject a simplification operation, this method allows fast simplification while guaranteeing a user-specified geometric error.

Keywords

Mesh comparison, Hausdorff error measurement, mesh simplification.

1. INTRODUCTION

Today, polygonal meshes have become ubiquitous as three-dimensional geometric representation of objects in computer graphics and some engineering applications. They are used for rendering of objects in a broad range of disciplines like medical imaging, scientific visualization, computer aided design (CAD), movie industry, etc. New acquisition techniques allow the generation of highly detailed objects with a permanently increasing polygon count. The handling of huge scenes composed of these high-resolution models rapidly approaches the computational capabilities of any graphics hardware. Therefore, level-of-detail techniques become inevitable. In order to build such level-of-detail representations many simplification algorithms exist that produce high-quality approximations of complex models with a reasonable amount of polygons.

The Journal of WSCG, Vol.13, ISSN 1213-6964
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.

However, for many applications it is very important to have precise control over the geometric error introduced by simplification. The common way to provide an accurate error control, which can be used to calculate image space errors during visualization, is to measure the Hausdorff distance between the simplified and original meshes. However, this distance can only be approximated by sampling, and therefore, the better the accuracy is, the slower the measurement algorithm becomes. When used to steer simplification, the performance of the simplification algorithm is reduced accordingly.

The main contribution of this work is an efficient algorithm to measure and update the Hausdorff distance between a simplified mesh and the original model. The superior speed of our approach is mainly due to its ability to quickly determine regions of high geometric distance (or during simplification, regions where the distance is above the desired value) and adapt sampling there.

2. PREVIOUS WORK

Since mesh simplification is one of the fundamental techniques used for polygonal meshes, there is an extensive amount of different methods. Since there are detailed reviews of simplification algorithms (e.g. [Lue01]), we give only a short overview of the most related methods.

Rossignac and Borrel [Ros93] introduced the family of *vertex clustering* methods. Although very fast, their algorithm and its derivative methods (e.g. [Low97]) allow almost no control over the error (it is bound by the cell size), and the reduction rate is quite low in flat parts of the model.

Cohen et al. [Coh96] developed *simplification envelopes* to guarantee fidelity bounds while enforcing local and global topology. The simplification envelopes consist of two offset surfaces at some distance ε from the original surface. Since these envelopes are not allowed to self-intersect, ε is decreased at high curvature regions. By keeping the simplified surface inside these envelopes, the algorithm can guarantee a geometric deviation of at most ε, and additionally it checks that the surface does not self-intersect. While this algorithm has the advantage to guarantee a geometric error bound, it is quite slow and requires an orientable manifold for the construction of the offset surfaces. Zelinka and Garland [Zel02] modified this approach by using *permission grids* – spatial occupancy grids, where an operation is only performed if all cells that are intersected by the new triangles are allowed to be occupied. Although the algorithm is much faster than [Coh96] and doesn't need an orientable manifold mesh, the simplified model often contains much more triangles due to the discrete grid and the fact that the Manhattan distance is used instead of the Euclidean.

The *vertex pair contraction* operation introduced at the same time by Popović and Hoppe [Pop97] and Garland and Heckbert [Gar97] has become the most common operation and is used in many simplification methods. In conjunction with the quadric error metric introduced in that work, it offers flexible control over the quality, still at very high reduction speed. However, the quadric metric mostly overestimates the real geometric error which results in non-optimal reduction rates and the need to measure the exact error after simplification.

Klein et al. [Kle96] first used the Hausdorff distance between the original and simplified mesh to control the simplification error, although with significant computational effort. In [Bor03a] Borodin et al. have produced high-quality results by combining *generalized pair contractions* – an extension of the vertex pair contraction – with the control of the distance between the original and simplified models during the whole simplification process.

In the area of mesh comparison, Cignoni et al. [Cig98] introduced the first method dedicated exclusively to measurement of errors on simplified surfaces, which allows to compare quality of different simplification methods. Another method, presented by Aspert et al. [Asp02], is more efficient in terms of speed at the cost of higher memory use. Both algorithms are based on sampling of the geometry of the two models to be compared, where the sampling density depends on the desired accuracy. In order to double the accuracy the number of samples needs to be multiplied by four. Therefore, these algorithms quickly become slow for higher accuracy.

3. TERMINOLOGY

First we define the distance $d(p,S')$ between a point p on a surface S and another surface S' as:

$$d(p,S') = \min_{p' \in S'} d(p,p'),$$

where $d(p,p')$ is the Euclidian distance between two points in E^3.

The geometric distance – also called one-sided or single-sided Hausdorff distance – between two surfaces S and S' is then defined as:

$$d(S,S') = \max_{p \in S} d(p,S')$$

Note, that this distance is not symmetric in general, i.e. $d(S,S') \neq d(S',S)$. The symmetrical Hausdorff distance is defined as:

$$d_s(S,S') = max\left(d(S,S'), d(S',S)\right)$$

This value gives more accurate measure of the distance between two surfaces by preventing the possible underestimation, which can appear if using only one-sided distances.

4. MESH COMPARISON

The main idea of our new mesh comparison algorithm is to adapt the sampling density used for distance calculation to the actual geometric deviation in the corresponding area. Hereby, the main goal is to draw samples only in those regions where the maximum distance between both objects is expected.

To achieve this, we first make two observations:

- Since the Hausdorff distance is defined as the maximum of the distances of all points on both meshes to the other mesh, we should avoid sampling in areas, where they are closer to each

other than the actual – yet unknown – Hausdorff distance. This can be achieved by comparing coarse voxelizations of the two objects, considering triangles within voxels of high distance first, and stopping comparison, when the already found distance is larger than the highest possible distance between remaining voxels.

- When processing triangles inside a voxel cell, we only need to subsample a triangle, if its geometric distance can be larger than the already found maximum. This can only happen, if any of its vertices is farther away from the other mesh than one of its interior points, or if any of these distances exceeds the maximum. Therefore, a tight upper bound of a triangle-to-mesh distance is required.

Data Structures

To quickly determine the regions of high geometric distance we sort the triangles of both meshes into two voxel grids respectively. Note, that later on in our algorithms – similarly to [Cig98] and [Asp02] – this grid is also used to quickly find the closest point on one of the meshes for a given sample point.

The grid dimensions depend on the objects' bounding boxes and the number of triangles. We aim to have 10 triangles per occupied cell in average. This can be achieved approximately by calculating the number of required cells for a cube tessellated with the same number of triangles as is in the larger mesh. This leads to a resolution $r = \sqrt{\frac{\#triangles}{10 \cdot 6}}$. To avoid memory problems we restrict ourselves to resolutions of 256^3.

To speed up finding voxels of high distances between both voxelizations we use an octree structure for each of them, build upon the entries within the grids. In order to get full octrees we allow only resolutions of $2^n \times 2^n \times 2^n$.

Main Algorithm

Initially, we set the current Hausdorff distance to zero. We start traversing the octree structures of both meshes simultaneously, measuring the distance of each cell to all other cells on the same level in order to find the closest one in the other mesh. If for the current cell the closest other cell is found, we can calculate the minimum and maximum distances between two points inside these cells. If the minimum distance is larger than the current Hausdorff distance, we update the Hausdorff distance accordingly. If the maximum distance is less than or equal to the current Hausdorff distance, traversal of the subtree is skipped.

In order to consider cells containing triangles with larger distance first, the octree traversal is steered using a priority queue. This queue contains the already processed octree cells sorted by their maximum geometric distance.

When a leaf cell is reached during traversal, we collect all contained triangles and insert them into the same priority queue as the cells, again according to their maximum possible geometric distance. Depending on their minimum distance we again update the Hausdorff distance. To prevent multiple insertions of the same triangle into the priority queue, we mark triangles and process only those yet unmarked. The traversal and therefore the whole algorithm stops if either the queue becomes empty (e.g. when both meshes are identical) or the maximum possible distance of all remaining cells and triangles is less than the already found Hausdorff distance. The main algorithm to calculate the Hausdorff distance is shown in Fig. 1.

```
MinError=0
AddToQueue(RootCellA)
AddToQueue(RootCellB)
while(QueueNotEmpty)
   GetCellWithHighestMaxDistance
   UpdateMinError
   if(LeafNode)
      InsertTrianglesIntoQueue
   else
      InsertChildrenIntoQueue
return minError
```

Figure 1. Main algorithm to calculate the Hausdorff distance.

Cell-Based Distance

To quickly find the closest cell, when traversing the octree from a node to its children, we store all indices of occupied cells, for which the minimum distance was less than the maximum distance to the closest cell. Then we need to check only the children of these cells when calculating the distances of the cells' child nodes. Note, that for the root nodes calculating the closest cells and the distances is trivial.

To simplify the distance calculation, we use the bounding box of the union of both meshes to construct the grid. Furthermore, we restrict ourselves to cubic grid cells, which further simplifies the distance calculation to calculations based on the cell coordinates.

Distance of a Triangle

To calculate lower and upper bounds for the geometric distance between a triangle and the other mesh, we first need to calculate the distances of its vertices. If a vertex is inside the currently processed grid cell, we can use its stored closest cells to find candidate triangles for the next surface point in the other mesh. If it is outside the current cell, we descend the hierarchy again to find the occupied cells closest to the current vertex. Then we calculate distances to all triangles starting with those contained in the closest cell. When the distance to the closest point found so far is closer than the distance to the remaining cells, the distance of the currently processed vertex is found. To prevent multiple distance calculations for the same triangle, we store the indices of triangles and collect only the unprocessed triangles from each cell.

After the distances for the three vertices of the current triangle are calculated, we know that the minimum geometric distance of the triangle is the maximum of the vertex distances $\|V_i - P_i\|$, and the maximum geometric distance is at most the maximum of the vertex distances and the distances of the triangle barycentre B to the three vertex base points P_i (see Fig. 2).

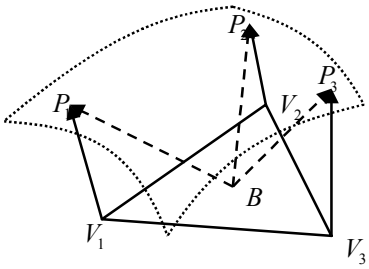

Figure 2. Minimum and maximum geometric distances of a triangle.

Therefore, we can determine the possible interval of the geometric distance d as:

$$d \geq \max_{i=1}^{3}\left(\|V_i - P_i\|\right) = d_{min}$$

$$d \leq max\left(H_{min}, \max_{i=1}^{3}\left(\|B - P_i\|\right)\right).$$

Additionally, no point on the triangle can be farther away from the other mesh than its vertices from any of the base points, and thus

$$d \leq \min_{i=1}^{3}\left(\max_{j=1}^{3}\left(\|V_i - P_j\|\right)\right).$$

If the closest points of all three vertices lie on the same triangle (see Fig. 3), the maximum vertex distance is already the geometric distance of the current triangle. Otherwise, the triangle is inserted into the priority queue. Note, that we have to take care about the fact that the closest point may lie on several triangles (if it falls onto an edge or into a vertex).

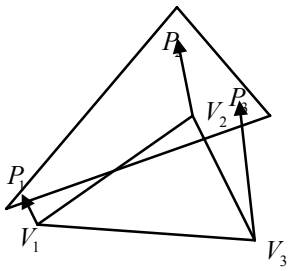

Figure 3. Exact geometric distance of a triangle.

When a triangle from the queue is processed, it is subdivided and the distances for its children are calculated. To prevent repeated calculation of the closest point/triangle for the same vertex, we calculate them for the three new vertices during subdivision. Then we only need to calculate the minimum and maximum possible distances before eventually storing the child triangles in the priority queue. The subdivision algorithm is shown in Fig. 4.

```
CalculateSubdivisionBasePoints
for(allChildTriangles)
   minDistance=max(vertexDistances)
   if(AllBasePointsOnSameTriangle)
      maxDistance=minDistance
   else
      maxDistance=max(barycenterDistances)
   InsertIntoQueue
```

Figure 4. Subdivision sampling algorithm.

Note, that calculating the base points and checking if they all lie on the same triangle is also necessary, when a leaf cell is processed in order to add all contained triangles to the queue.

5. APPLICATION TO SIMPLIFICATION

To control the Hausdorff error during simplification, only the part of the mesh affected by the current operation needs to be considered. Therefore, the affected triangles of the simplified mesh are directly inserted into the queue, and the error measurement for the original model is restricted to the region around these triangles using their common bounding box. Since the error of neighbouring triangles in the original model may also be affected, we need to extend this bounding box by the current Hausdorff error.

Furthermore, it is not necessary to calculate the exact geometric error, but only to check if it is below a user-specified threshold. Therefore, we do not need

to insert cells or triangles, for which the maximum possible distance is below this threshold, into the queue, and thus refine sampling only in regions, where the error may be above this value. Analogously, if the minimum error found so far is above this threshold, we can immediately stop the calculation and reject the simplification operation. When calculating the geometric error of a triangle, we can also immediately stop searching for the base points P_i as soon as we found one that is closer than the desired error minus the maximum length of the two edges adjacent to the current vertex (according to the triangle inequation no vertex can be farther from a point than the distance of any vertex to this point plus the distance to this vertex).

The fact that only an accept/reject decision is required to decide, if a simplification operation will be performed, allows for some additional simple tests to quickly find an answer in most cases.

The simplification algorithm delivering the best trade-off between speed and quality of the simplified model is the one based on the quadric error metric [Gar97]. Choosing this simplification algorithm as base for our method, we get the additional advantage: the error quadric gives an (admittedly sometimes largely overestimated) upper bound for the Hausdorff error and can thus be used as a criterion to accept an operation without further tests.

Then two additional simple tests are possible to quickly reject an operation. First, the distance of the new vertex to the simplified mesh before the current edge collapse operation is calculated. If this exceeds twice the desired Hausdorff error ε, the operation can be rejected. Note, that exceeding of 2ε is required due to possible configurations similar to the one shown in Fig. 5.

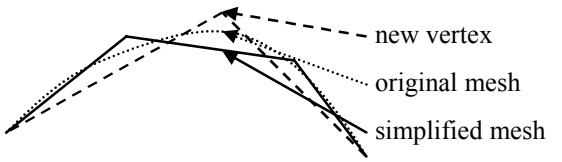

Figure 5. Quick reject tests.

If the operation passed this test, the distance from the new vertex to the original mesh is calculated. If this exceeds the specified threshold, the operation is also rejected. These two tests have the advantage that they quickly reject many operations and no update of the grid is required for their calculation.

When an operation passed these two tests without being rejected, the grid and octree of the simplified model are updated. If the operation has not been accepted by the quadric test, the Hausdorff distance between the updated meshes is calculated. When the operation is rejected by the Hausdorff error check, the vertex is split again, updating the grid and octree of the simplified mesh, and the operation is removed from the simplification queue. The overall pipeline of the error-checking algorithm is shown in Fig. 6.

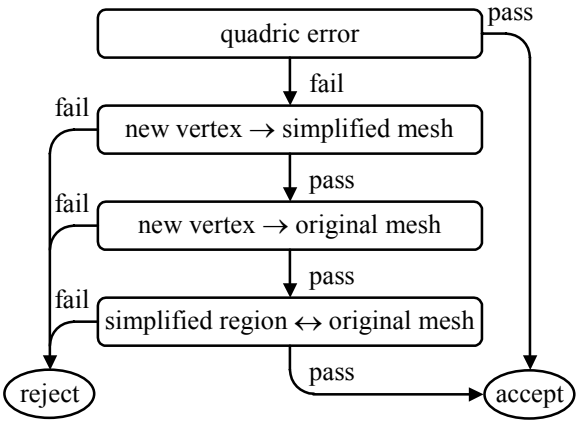

Figure 6. Error testing pipeline.

If the simplification queue is empty, all possible collapse operations that do not exceed the specified Hausdorff error have been performed.

6. RESULTS

Since our algorithm is applicable to both, measuring distances between meshes and controlling the introduced Hausdorff error during simplification, we compare it to previous approaches in both fields. We ran all tests on a PC with an Athlon 3000+ and 2 GB of main memory.

Mesh Comparison

To demonstrate the advantages of our algorithm, we compare its computation time with the two standard tools for measuring the Hausdorff distance: Metro [Cig98] (version 4.0) and MESH [Asp02] (version 1.12). The models used for evaluation are shown in

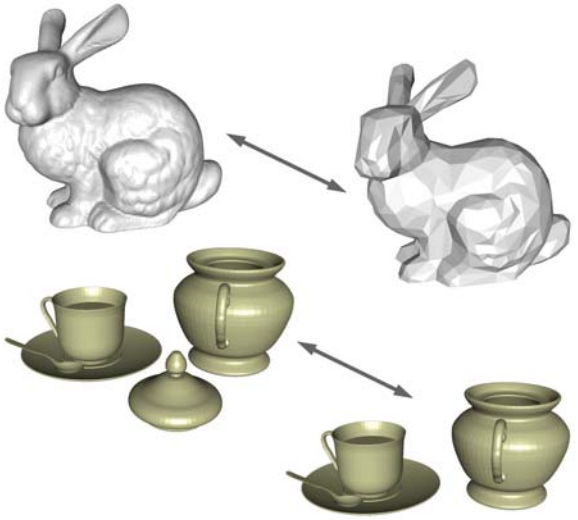

Figure 7. Models used for mesh comparison.

Fig. 7; the numbers of their vertices and triangles are listed in Tab. 1.

Model	# triangles	# vertices
Bunny (orig.)	69,451	34,834
Bunny (simpl.)	1,001	553
Coffee set	69,696	34,860
Without lid	60,936	30,480

Table 1. Models used for mesh comparison.

Tab. 2 shows the comparison in computation time of the three algorithms with an accuracy of 0.01% of the model diameter.

	Metro	MESH	Our alg.
Bunny	1,406 sec	395 sec	2.7 sec
Coffee set	13,008 sec	1,396 sec	2.1 sec

Table 2. Computation times of error-measuring algorithms.

At this accuracy our algorithm is several orders of magnitude faster than Metro and MESH, since we sample the mesh surface densely in regions of high geometric distance only. This is especially visible, when comparing the coffee set with and without lid, as shown in Fig. 8, where only samples in the region of the highest Hausdorff distance were taken.

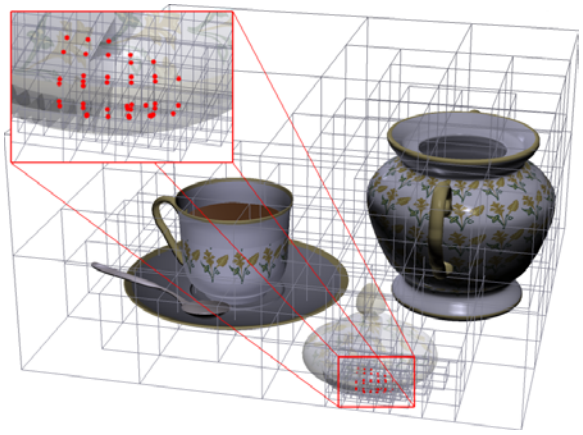

Figure 8. Visited octree cells and taken samples for coffee set scene with and without lid.

Fig. 9 shows a detailed plot of the computation times of the three algorithms, when comparing the simplified bunny with the original model, using different accuracies ranging from 1% of the bounding box diameter (practically useless) to 0.001% (very accurate).

It is clearly visible, that in contrast to both Metro and MESH, the computation time of our algorithm depends only very little on the desired accuracy. Note, that comparing the meshes with accuracy higher than 0.01% was not possible using MESH,

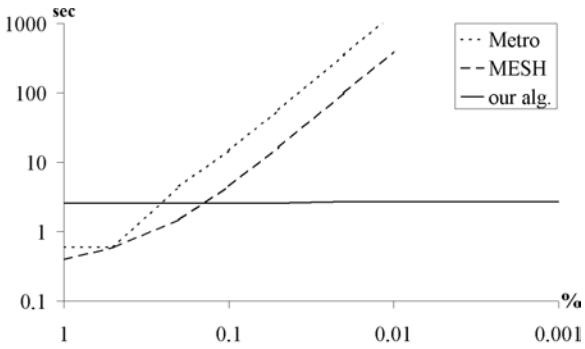

Figure 9. Computation times of error-measuring algorithms.

since it ran out of memory and Metro needs more than a day to compare the simplified and original bunny at 0.001% .

Error Control

In the field of error control during simplification, we compare our method with two simplification algorithms that guarantee a user-specified geometric error: *simplification envelopes* [Coh96] and *high-quality simplification* [Bor03a] (using the out-of-core simplification [Bor03b], when necessary). For comparison, we use different scanned objects from the Stanford 3D Scanning Repository [Sta3D] and the Digital Michelangelo Project [DigMi] shown in Fig. 10 and Tab. 3.

Figure 10. Models used for simplification.

Tab. 4 compares the computation times of the two mentioned simplification algorithms with our approach. For all models and algorithms the same simplification errors (1% and 0.1% of the model diameter) were used. The Hausdorff distance of 1%

Model	# triangles	# vertices
Bunny	69,451	34,834
Dragon	871,414	437,645
Buddha	1,087,474	543,652
David 2mm	7,227,031	3,614,098

Table 3. Models used for simplification.

is especially interesting for out-of-core simplification using hierarchical partitioning (e.g. [Bor03b]), since it is close to the resolution of $\frac{\varepsilon}{128}$ used for each octree cell.

	[Coh96]	[Bor03a]	our alg.
$\varepsilon = 1\%$			
Bunny	1:12	1:25	0:52
Dragon	n.a.	27:58	6:48
Buddha	n.a.	25:27[1]	12:37
David 2mm	n.a.	3:01:43[1]	1:06:22
$\varepsilon = 0.1\%$			
Bunny	0:46	0:46	1:28
Dragon	n.a.	15:37	14:59
Buddha	n.a.	24:08[1]	21:13
David 2mm	n.a.	3:00:03[1]	1:51:56

Table 4. Computation times of simplification algorithms.

Note, that the simplification envelopes restricts only the geometric error from the simplified model to the original, which is sufficient for rendering, but may cause inaccuracies for other applications like collision detection. Similarly, the high-quality simplification guarantees an upper bound for the geometric error from the original to the simplified model only, and thus may close large holes in the model, which is not always desired. Additionally, the accuracy is low, since only samples at vertex positions are taken. If out-of-core simplification is used, the error is only guaranteed to lie between $\frac{4}{5}\varepsilon$ and ε. This means that a more aggressive simplification would be possible without exceeding the threshold.

The computation time of the simplification envelopes is similar to the one of the high-quality simplification, but the algorithm requires orientable manifold meshes, and therefore worked only for the bunny model. Although our algorithm guarantees the Hausdorff distance to be below a specified threshold, the performance is even better than the simplification envelopes and the high-quality simplification for larger models and/or simplification errors.

[1] Out-of-core simplification [Bor03b].

7. CONCLUSION

We have presented an efficient algorithm to measure the geometric distances and the Hausdorff distance between two meshes. Our approach is much faster than existing algorithms for reasonable accuracies (i.e. less than 0.01% of the model diameter), since it needs to refine sampling only in regions of high distance and thus hardly depends on the required accuracy. This is accomplished by using a bi-hierarchical search algorithm to quickly find regions of possibly high geometric distances.

Furthermore, we have shown that our algorithm can also be applied to increase performance, efficiency, and accuracy of error-bounded simplification by using a chain of simple accept/reject tests to quickly determine, if exact evaluation of the Hausdorff distance is necessary. Instead of measuring the distance, we can stop traversing the hierarchy, when the minimum possible error is above the desired threshold, or the maximum possible is below. Using this technique, our approach is up to four times as fast as comparable algorithms when drastically simplifying the model.

8. ACKNOWLEDGEMENTS

We thank the Stanford 3D Scanning Repository and the Digital Michelangelo Project for providing us with the models. The coffee set model is courtesy of Renzo Del Fabbro.

9. REFERENCES

[Asp02] Aspert, N. Santa-Cruz, D., and Ebrahimi T. MESH: measuring errors between surfaces using the Hausdorff distance. Proc. of the IEEE International Conference on Multimedia and Expo, pp. 705-708, 2002.

[Bor03a] Borodin, P., Gumhold, S., Guthe, M., and Klein, R. High-quality simplification with generalized pair contractions. Proc. of GraphiCon '03, pp. 147-154, 2003.

[Bor03b] Borodin, P., Guthe, M., and Klein, R. Out-of-core simplification with guaranteed error tolerance. Proc. of Vision, Modeling and Visualisation '03, pp. 309-316, 2003.

[Cig98] Cignoni, P., Rocchini, C., and Scopigno, R. Metro: measuring error on simplified surfaces. Computer Graphics Forum, vol. 17, no. 2, pp. 167-174, 1998.

[Coh96] Cohen, J., Varshney, A., Manocha, D., Turk, G., Weber, H., Agarwal, P., Brooks, F., and Wright, W. Simplification envelopes. Computer

Graphics (Proc. of SIGGRAPH ’96) 30, pp. 119-128, 1996.

[DigMi] The Digital Michelangelo Project. http://www-graphics.stanford.edu/projects/mich.

[Gar97] Garland, M. and Heckbert, P. S. Surface simplification using quadric error metrics. Computer Graphics (Proc. of SIGGRAPH ’97) 31, pp. 209-216, 1997.

[Kle96] Klein, R., Liebich, G., and Straßer, W. Mesh reduction with error control. Proc. of IEEE Visualization ’96, pp. 311-318, 1996.

[Low97] Low, K.-L. and Tan, T.-S. Model simplification using vertex-clustering. Proc. of Symposium on Interactive 3D Graphics, pp. 75-81, 1997.

[Lue01] Luebke, D. A Developer’s Survey of Polygonal Simplification Algorithms. IEEE Computer Graphics and Applications, 21(3), pp. 24-35. 2001.

[Pop96] Popović, J. and Hoppe, H. Progressive simplicial complexes. Computer Graphics (Proc. of SIGGRAPH ’97) 31, pp. 217-224, 1997.

[Ros93] Rossignac, J. and Borrel, P. Multi-resolution approximations for rendering. Modeling in Computer Graphics, pp. 455-465, 1993.

[Sta3D] The Stanford 3D Scanning Repository. http://www-graphics.stanford.edu/data/3dscanrep.

[Zel02] Zelinka, S. and Garland, M. Permission grids: practical, error-bounded simplification. ACM Transactions on Graphics, 21(2), pp. 1-25, 2002