Exact Computation of the Hausdorff Distance between Triangular Meshes

Raphael Straub

Universität Karlsruhe (TH), Karlsruhe, Germany

Abstract

We present an algorithm that computes the exact Hausdorff distance between two arbitrary triangular meshes. Our method computes squared distances for each point on each triangle of one mesh to all relevant triangles of the other mesh yielding a continuous, piecewise convex quadratic polynomial over domains bounded by conics. The maximum of this polynomial is the one-sided Hausdorff distance from one to the other mesh. We ensure the efficiency of our approach by employing a voxel grid for searching relevant triangles and an attributed half-edge data structure for representing the squared distance function.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Geometric algorithms, languages, and systems, I.3.5 [Computer Graphics]: Boundary representations, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction and Previous Work

The *Hausdorff distance* is a very intuitive and therefore commonly used distance measure between two subsets A and B of a metric space. It can be defined by means of the *one-sided Hausdorff distance*:

$$d_{\mathrm{H}'}(A,B) := \max_{\mathbf{p}\in A} \min_{\mathbf{q}\in B} \|\mathbf{p}-\mathbf{q}\| \ .$$

The norm $\|\cdot\|$ in this definition can be any norm, but commonly the Euclidean norm $\|\cdot\|_2$ is meant, as in this paper. Note that the one-sided Hausdorff distance is not symmetric. To transform it into a symmetric distance, the maximum of the two one-sided Hausdorff distances is taken which yields the Hausdorff distance:

$$d_{\mathrm{H}}(A,B) := \max\{d_{\mathrm{H}'}(A,B), d_{\mathrm{H}'}(B,A)\} \ .$$

This distance can be used for comparing two meshes. Probably, one of the first known mesh comparison tools is *Metro* [CRS98]. Metro samples one mesh uniformly or randomly and computes distances from sampled points to the other mesh to approximate the one-sided Hausdorff distance. The search of the closest point on the second mesh is accelerated by a uniform cell grid. In [ASCE02] Aspert et al. present a similar method based on regular sampling. Their tool *Mesh* also uses a uniform grid to speed up the closest triangle search. This grid is implemented in a memory efficient way, thus its memory footprint is smaller and it is faster than Metro. Guthe et al. [GBK05] use adaptive subsampling to accelerate the computation of approximate Hausdorff distances, especially for higher accuracies. A triangle of one mesh is subdivided only if its distance to the other mesh can exceed the current one-sided Hausdorff distance approximate. In addition, an octree voxel grid for efficiently finding nearest neighbors is used. In [Lla05], a randomized algorithm for finding the maximum distance from a finite point set to an arbitrary compact set is presented. This method is adapted for computing the exact Hausdorff distance between convex polyhedra. The main drawback of this method is that it is inapplicable for non-convex meshes, which are common in practical applications.

One area of application for Hausdorff distances is quality control for mesh simplification algorithms. Many mesh simplification algorithms, like in [HDD*93,RB93,GH97], allow no or only indirect control over the maximum Hausdorff distance between the original and the simplified mesh. In contrast, Klein et al. show in [KLS96] how to decide whether a mesh simplification step would exceed a user defined limit on the Hausdorff error and therefore should be rejected.

In this paper, we present an algorithm that computes the exact Hausdorff error between two arbitrary triangular

meshes. The key advantage over previous work is that the computation is done analytically, i. e., without any sampling, and therefore always gives an *exact* result. The explicit computation of the Hausdorff distance is possible due to the simple structure of the considered objects, which are piecewise linear. To accelerate our method, we use voxel grid techniques similar to the ones described in [CRS98, ASCE02, GBK05].

2. Basic Overview

Let $M = T_1 \cup \ldots \cup T_n$ and $M' = T'_1 \cup \ldots \cup T'_m$ be two triangular meshes with triangles $T_i = [\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i]$ defined as the convex hull, denoted by $[\cdot]$, of its corners. Then the one-sided Hausdorff distance between these meshes can be computed by regarding distances between points on their triangles:

$$d_{\mathrm{H}'}(M', M) = \max_{\mathbf{p} \in M'} \min_{\mathbf{q} \in M} \|\mathbf{p} - \mathbf{q}\| = \max_{T' \in \{T'_1, \ldots, T'_m\}} \underbrace{\max_{\mathbf{p} \in T'} \min_{T \in \{T_1, \ldots, T_n\}} \underbrace{\min_{\mathbf{q} \in T} \|\mathbf{p} - \mathbf{q}\|}_{=: d_T(\mathbf{p})}}_{=: d(\mathbf{p})} . \quad (1)$$

Obviously, it is sufficient to compute the maximum value of the distance function d for each triangle T' of the first mesh M' and take the maximum of these values. For each T', d can be computed as the minimum of all d_T, $T \in \{T_1, \ldots, T_n\}$.

Let $T = [\mathbf{a}, \mathbf{b}, \mathbf{c}]$ be a triangle of M. In the following, we consider only the squared distances d_T^2 instead of the distances d_T in order to simplify notation and computation. We subdivide the plane of T' into seven regions, where each region contains either the points $\mathbf{p}$ closest to the interior of T or closest to one of the edges or one of the corners of T, see Fig. 1. These regions are the *Voronoi regions* of T, its edges, and its corners, restricted to the plane of T'. They may be empty and are obtained by projecting T and six half-lines emanating from the three corners orthogonally to one of the two adjacent edges into the plane of T', where the projection direction is orthogonal to T.

Clearly, d_T^2 is a continuous function that is quadratic over each of these seven regions:

$$d_T^2(\mathbf{p}) = \begin{cases} \left(\dfrac{\det[\mathbf{p}-\mathbf{a} \quad \mathbf{b}-\mathbf{a} \quad \mathbf{c}-\mathbf{a}]}{\|(\mathbf{b}-\mathbf{a}) \times (\mathbf{c}-\mathbf{a})\|} \right)^2 & \text{if } \mathbf{p} \in R_{\mathbf{abc}} \\ \left(\dfrac{\|(\mathbf{p}-\mathbf{a}) \times (\mathbf{b}-\mathbf{a})\|}{\|\mathbf{b}-\mathbf{a}\|} \right)^2 & \text{if } \mathbf{p} \in R_{\mathbf{ab}} \\ \vdots & \\ \|\mathbf{p}-\mathbf{a}\|^2 & \text{if } \mathbf{p} \in R_{\mathbf{a}} \\ \vdots & \end{cases} .$$

The quadratic functions d_T^2 consists of are non-negative and

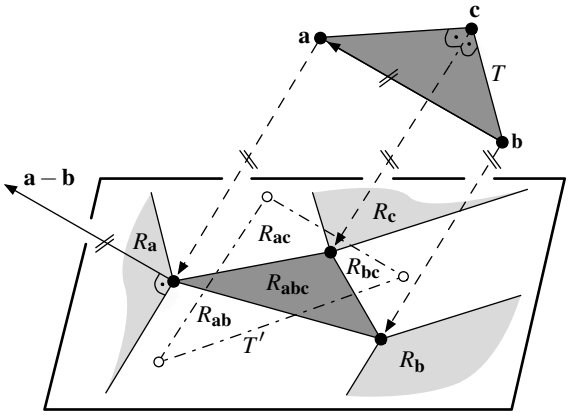

Figure 1: *The triangle T is projected perpendicularly to T into the plane of T' dividing it into seven distinct regions $R_{\mathbf{abc}}$, $R_{\mathbf{ab}}$, $R_{\mathbf{bc}}$, $R_{\mathbf{ac}}$, $R_{\mathbf{a}}$, $R_{\mathbf{b}}$, and $R_{\mathbf{c}}$. The boundaries of the regions $R_{\mathbf{a}}$, $R_{\mathbf{b}}$, and $R_{\mathbf{c}}$ lie on planes through the corners of T each perpendicular to one of the two adjacent edges of T. For example, the boundary between $R_{\mathbf{a}}$ and $R_{\mathbf{ab}}$ lies on the plane through $\mathbf{a}$ perpendicular to $\mathbf{a} - \mathbf{b}$.*

therefore they are convex, since non-negative quadratic functions are either constant or represent convex parabolic cylinders or elliptic paraboloids. All other types of quadrics do not represent functions at all or only functions with negative values.

Now, to obtain d^2 (cf. Equ. (1)) we have to compute the minimum of the $d_{T_i}^2$ ($i = 1, \ldots, n$) over T'. The minimum of two functions $d_{T_i}^2$ and $d_{T_j}^2$ is piecewise quadratic, where the domains of the quadratic pieces are bounded by the boundaries of the regions shown in Fig. 1 and the conics given by the equation

$$d_{T_i}^2(\mathbf{p}) = d_{T_j}^2(\mathbf{p}) \quad \text{or} \quad d_{T_i}^2(\mathbf{p}) - d_{T_j}^2(\mathbf{p}) = 0 .$$

These domains are the intersections of the Voronoi regions of all triangles, edges, and vertices of M (cf. [Mil93]) with T'. Since a convex function reaches its maximum at the boundary of its domain, the squared distance function $d^2 = \min_{i=1}^n d_{T_i}^2$ has its maximum on one of these conics or line segments. Figure 2 shows the graph of a squared distance function for two triangles.

Once the maximum of d^2 on the boundaries of its domains is computed, the maximum of these values for all triangles $T'_1, \ldots, T'_m$ is the one-sided Hausdorff distance $d_{\mathrm{H}'}(M', M)$. Algorithm 1 gives a basic overview of the complete algorithm for the one-sided Hausdorff distance. Finally, to obtain the wanted Hausdorff distance, the other one-sided Hausdorff distance $d_{\mathrm{H}'}(M, M')$ can be determined analogously.

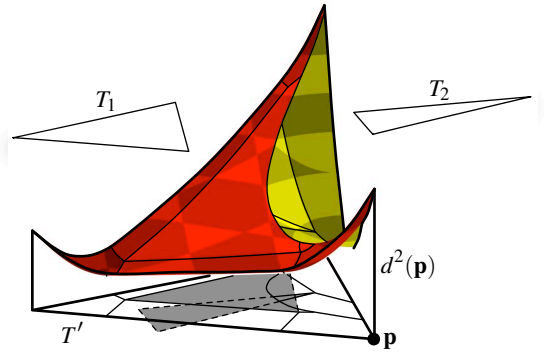

Figure 2: *The graph of the piecewise quadratic squared distance d^2 to two triangles T_1 and T_2 over a triangle T'. The red and yellow parts of the graph belong to the points that are nearest to T_1 and T_2, respectively. The projections of T_1 and T_2 into the plane of T' are depicted in gray.*

Algorithm 1 Computation of the one-sided Hausdorff distance $d_{\mathrm{H}'}(\{T'_1,\ldots,T'_m\},\{T_1,\ldots,T_n\})$

1: $d^2_{\mathrm{H}'} := 0$
2: **for all** $T' \in \{T'_1,\ldots,T'_m\}$ **do**
3: $\quad d^2 := \infty$
4: $\quad$ **for all** $T \in \{T_1,\ldots,T_n\}$ **do**
5: $\quad\quad$ compute d^2_T by projecting T into T'
6: $\quad\quad d^2 := \min\{d^2, d^2_T\}$
7: $\quad$ **end for**
8: $\quad d^2_{\mathrm{H}'} := \max\left\{d^2_{\mathrm{H}'}, \max_{\mathbf{p}\in T'} d^2(\mathbf{p})\right\}$
9: **end for**
10: **return** $\sqrt{d^2_{\mathrm{H}'}}$

3. The Algorithm in Detail

In this section, we explain how our one-sided Hausdorff distance computation can be implemented. Especially, we go into the details of the squared distance function computation and the selection of relevant triangles.

3.1. Computing the Squared Distance Function

We store the squared distance functions d^2 in a half-edge data structure (cf. [BSBK02]), which is commonly used for meshes. Here, the edges of the mesh represent the conic boundaries of the domains and the faces represent the domains. Hence, the edges have the equation of the boundary conic and the faces the equation of d^2 in the corresponding domain as attributes. This attributed mesh data structure enables us to split and merge faces and to find adjacent domains in constant time.

The squared distance function d^2 over a triangle T' is the minimum over all squared distance functions d^2_T ($T \in \{T_1,\ldots,T_n\}$). As this minimum is computed incrementally over all relevant triangles T (cf. line 6 in Algorithm 1), we assume in the following that we already have a squared distance function d^2 in our attributed mesh data structure. Our goal is now to insert the distance function d^2_T of a new triangle T into our data structure so that it represents the minimum of d^2 and d^2_T.

The projection of T into T' divides the attributed mesh into at most seven regions (cf. Fig. 1). For each domain, d^2 is compared to the new distance function d^2_T for all overlapping regions by computing the conic $Q(\mathbf{p}) := d^2(\mathbf{p}) - d^2_T(\mathbf{p}) = 0$. This conic is then trimmed by the boundaries of the domain and the corresponding region. The remaining part of the conic, if existent, yields a new edge and the region of all $\mathbf{p}$ where $Q(\mathbf{p}) > 0$, i. e., $d^2_T(\mathbf{p}) < d^2(\mathbf{p})$, yields a new face in the domain mesh. Potential resulting adjacent faces where d^2 is the same polynomial can be merged subsequently.

As the global maximum of the squared distance function d^2 over T' is achieved on the boundary of a domain, this maximum is the maximum of all local maxima on the edges and the values of d^2 at the vertices of the domain mesh. To determine the local maxima on the edges we parameterize d^2 over each of its boundary conics $Q(\mathbf{p}) = 0$ by using a rational quadratic parameterization $\mathbf{f} : \mathbf{R} \to \mathbf{R}^3$, $Q(\mathbf{f}(t)) = 0$ ($t \in \mathbf{R}$) of Q (cf. [Far95, pp. 61f]). The resulting parameterization $d^2(t) := d^2(\mathbf{f}(t))$ is a rational quartic polynomial in t whose local maxima can be computed in a straightforward manner.

3.2. Selecting Relevant Triangles

Our main algorithm consists of two nested loops (cf. lines 2 and 4 in Algorithm 1) over all triangles of the meshes M' and M. We should avoid computing the squared distance functions d^2 for all pairs of triangles of M' and M as the number of triangles in each mesh can exceed some millions and the computation of d^2 will become time-consuming.

Since for each triangle $T' \subseteq M'$ we are only interested in the distance to the nearest triangles $T \subseteq M$, we start projecting near triangles into T' and stop as soon as we are sure that all other triangles of M are too far away to change d^2. To speed up this process we use a voxel grid as a spatial search structure. In a preprocessing step all triangles of M and M' are inserted into all voxels they intersect, where each voxel has two lists, one for the triangles of each of the two meshes.

The size of the voxel grid is determined by the size $l \times w \times h$ of the bounding box of $M \cup M'$ and the number $n + m$ of triangles. Assuming that the voxels are approximately cubical with edge length a and that all triangles have the same size and are equally distributed on the surface of the bounding box, we have

$$k = \frac{a^2(n+m)}{2(lw+lh+wh)}$$

triangles per voxel. In practice, we select, e. g., $k = 1$ and compute the corresponding approximate size a of the voxels through the equation above.

The selection of relevant triangles T for which to compute the squared distance function d_T^2 starts with one triangle of M in the voxel nearest to one voxel of T'. The nearest voxel is found using a breadth-first search strategy starting with the voxels of T'. In each iterative step, after the update of the squared distance function d^2 (cf. line 6 in Algorithm 1) the maximum $d_{\max} := \max_{\mathbf{p} \in T'} d^2(\mathbf{p})$ is computed. For all following steps only triangles T in voxels that have at least one point nearer than $d_{\max}$ to a point in a voxel of T' are considered. All other triangles are farther than $d_{\max}$ away from T', so they do not contribute to the minimum of all $d_{T_i}^2$.

4. Conclusion and Future Work

The algorithm presented in this paper computes the exact Hausdorff distance between triangular meshes. An implementation and subsequent tests on real world examples are planned for the future. It is obvious that our method can be easily employed for arbitrary piecewise linear surfaces, as it does not make use of the mesh connectivity and non-triangular meshes can be triangulated before. A straightforward extension of our method would be to compute the mean squared distance by taking the integral of the squared distance function over a triangle divided by its area.

5. Acknowledgments

We would like to thank Raphael Diziol for creating the visualization of the squared distance function and Bertrand Klimmek for proof-reading this paper.

References

[ASCE02] ASPERT N., SANTA-CRUZ D., EBRAHIMI T.: Mesh: Measuring errors between surfaces using the Hausdorff distance. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)* (2002), vol. 1, pp. 705–708.

[BSBK02] BOTSCH M., STEINBERG S., BISCHOFF S., KOBBELT L.: Openmesh – a generic and efficient polygon mesh data structure. In *OpenSG Symposium* (2002).

[CRS98] CIGNONI P., ROCCHINI C., SCOPIGNO R.: Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (1998), 167–174.

[Far95] FARIN G. E.: *NURB Curves and Surfaces*. A K Peters, Wellesley, MA, 1995.

[GBK05] GUTHE M., BORODIN P., KLEIN R.: Fast and accurate Hausdorff distance calculation between meshes. In *Proceedings of the 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)* (2005), pp. 41–48.

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), pp. 209–216.

[HDD*93] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Mesh optimization. *Computer Graphics 27* (1993), 19–26.

[KLS96] KLEIN R., LIEBICH G., STRASSER W.: Mesh reduction with error control. In *IEEE Visualization '96* (1996), Yagel R., Nielson G. M., (Eds.), pp. 311–318.

[Lla05] LLANAS B.: Efficient computation of the Hausdorff distance between polytopes by exterior random covering. *Computational Optimization and Applications 30*, 2 (2005), 161–194.

[Mil93] MILENKOVIC V.: Robust construction of the Voronoi diagram of a polyhedron. In *Proceedings of the Fifth Canadian Conference on Computational Geometry* (Aug. 1993), Lubiw A., Urrutia J., (Eds.), pp. 473–478.

[RB93] ROSSIGNAC J. R., BORREL P.: Multi-resolution 3d approximations for rendering complex scenes. In *Geometric Modeling in Computer Graphics* (1993), Falcidieno B., Kunii T. L., (Eds.), Springer Verlag, pp. 455–465.