Approximating Boundary-Triangulated Objects with Balls[*]

O. Aichholzer[†] F. Aurenhammer[‡] T. Hackl[†] B. Kornberger[†] M. Peternell[§] H. Pottmann[§]

Abstract

We compute a set of balls that approximates a given 3D object, and we derive small additive bounds for the overhead in balls with respect to the minimal solution with the same quality. The algorithm has been implemented and tested using the CGAL library [7].

1 Introduction

Representing a complex geometric object with primitives is a fundamental task in computational geometry and computer graphics. A general distinction is between exact representation, like slab decomposition or triangulation, and approximate representation, like spline boundary conversion or approximate covering. The choice of the primitives used depends on the application, for example, whether the object is to be processed for visualization or for subsequent manipulation.

In the present note, we deal with the problem of converting a 3D object into a set of balls. While such a representation may be less advantageous for visualization (than, e.g., a conversion into ellipsoids [4]) it is sometimes particularly convenient for further processing. The main applications we have in mind are computing Minkowski sums and detecting collisions.

Calculating the Minkowski sum of two nonconvex 3D objects is a complicated task with various applications to problems where proximity is involved [14]. For instance, planning a translational motion of a robot A in a workspace B can be based on constructing their Minkowski sum $A \oplus B$. A common approach is to decompose the (polyhedral) objects A and B into convex parts and adding them up pairwise. Using few but complex parts leads to the need of invoking Minkowski sum algorithms for general convex polyhedra [10, 8], and decomposition is a challenging question in itself even when tetrahedra are used [6, 18]. As an alternative, the objects to be summed may be approximately covered by balls. Calculating the Minkowski sum of two balls is trivial; the main objective is finding a small set of covering balls.

In collision detection, object handling is typically based on hierarchical representations. Common data structures in this context are so-called sphere-trees which, as an easy possibility, can be derived from octrees [17, 16]. This approach does not make explicit use of the geometry of the objects, however. Refined methods for constructing sphere-trees have been proposed [11, 5], utilizing the medial axes of the objects. Again, the problem of converting an object into a minimum number of balls arises. The reverse process of extracting the boundary of an approximating set of balls is treated in [3].

We develop an algorithm that takes as input a 3D object with triangulated boundary, and generates an almost-minimal set of balls that covers all the triangle endpoints, ensuring that no such point is covered with more than a user-specified offset ε. The quality of the approximation thus will also depend on the quality of the boundary mesh. If the object boundary is not covered completely, this can be achieved if desired, with a simple postprocessing step. Following known paths [1, 2], we first generate a candidate set of approximating balls centered on the triangles endpoints' Voronoi diagram. Correct labeling of balls as having their centers inside or outside the object becomes an issue, and we propose a simple though efficient labeling algorithm using the boundary triangulation. A method for reducing the candidate set of balls is then applied as an instance of the set covering problem. The heuristic we use allows us to determine how close to the optimum is the produced set of balls. Experimental results for practical data are described.

2 Ball generation

We specify the input object, A, as a bounded and interior-connected 3-manifold whose boundary is connected and triangulated. Thus A may have tunnels, but holes are disallowed. For each boundary triangle of A its orientation with respect to A is given. This is a common representation of an object, sufficiently general for many applications. Let P be the set of vertices of A, called *sample points* in the following. Our first aim is to produce a candidate set of balls which covers P and at the same time approximates A.

Call a ball $b \subseteq A$ *maximal* if there exists no ball $b' \subseteq A$ such that $b' \supsetneq b$. The medial axis of A

[*]Research supported by the FWF Joint Research Project 'Industrial Geometry' S9205-N12.

[†]Institute for Software Technology, Graz University of Technology, Austria, {oaich,thackl,bkorn}@ist.tugraz.at

[‡]Institute for Theoretical Computer Science, Graz University of Technology, Austria, auren@igi.tugraz.at

[§]Institute of Discrete Mathematics and Geometry, Vienna University of Technology, Austria, {peternell,pottmann}@geometrie.tuwien.ac.at is defined as the set of centers of all its maximal balls. As A is just the union of all maximal balls, a set of n sufficiently large balls centered close to the medial axis of A will serve the desired purpose. This observation has been made use of in various papers in computational geometry and computer graphics; see, for example, [11, 5] and [1, 2], respectively.

Following the approach in [2], we consider the Voronoi diagram, $V(P)$, of the given point sample P. For a point $p \in P$, let π_p be an *(inner) pole* of p, that is, a vertex of the region of p in $V(P)$ that lies inside A and maximizes $\|p - \pi_p\|$. In contrast to arbitrary Voronoi vertices, the value of poles is that they are located near the medial axis of A if the sample P is sufficiently dense [1]. (Let us temporarily ignore the fact that π_p does not exist if all region vertices for p happen to lie outside of A.) If we take, for each $p \in P$, the (closed) Delaunay ball with center π_p and radius $\|p - \pi_p\|$ then the resulting set, B, of balls covers P. Also, B approximates A with a quality which depends on that of the boundary mesh. Observe that B is optimal in the sense that any other set of balls which 'touches' P and is of the same cardinality will be inferior to B in approximating A.

Identifying poles among the vertices of a Voronoi diagram is a problem in itself. In [2], for the sake of subsequent power crust construction, two vertices per sample point are identified using an angle criterion that works for dense samplings. (At most) one of these two vertices is located in A and is the pole we are looking for. We need to exactly find the poles, as balls centered outside A will lead to a violation of any approximation property. To this end, we utilize the given triangular mesh that bounds A (which is not part of the input in [2]). After having computed all the vertices of $V(P)$, we use ray shooting to determine their location with respect to A. Among those lying inside A, one vertex per region which is at maximal distance from the respective sample point is selected. This direct method will work correctly regardless of the quality of the sample mesh.

The outcome of the ray shooting procedure is crucial, hence a correct and efficient implementation is needed. To decide $u \in A$ for a vertex u of point p's Voronoi region, we use the ray $\overrightarrow{up}$ and determine the first point of intersection of $\overrightarrow{up}$ with the boundary of A. Clearly, if this point is p, then decision can be made locally from the orientations of the incident triangles. Otherwise, the boundary triangle hit first is not incident to p, and we adopt the following strategy for finding it. Let S_u be the sphere with center u and radius $\|p - u\|$. Define a *critical sphere* S_p centered at p, as below. Let L be the length of the longest edge of the boundary mesh, and put $R = \frac{1}{\sqrt{3}}L$. If $\|p - u\| \geq R$ then choose S_p so as to intersect S_u in a circle of radius R. Otherwise, define the radius of S_p as $\sqrt{\|p - u\|^2 + R^2}$. Consult Figure 1.

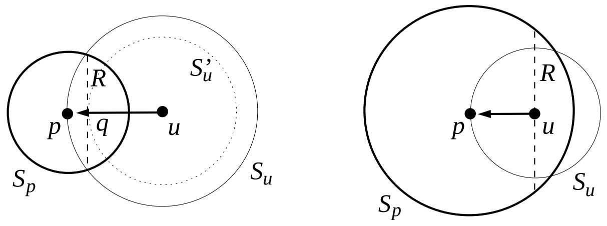

Figure 1: Defining the critical sphere S_p

Lemma 1 *If the ray $\overrightarrow{up}$ intersects a boundary triangle Δ of A then at least one endpoint of Δ is enclosed by S_p.*

Proof. Let Δ be intersected by $\overrightarrow{up}$. Consider the smallest disk, D_Δ, that contains Δ. All edges of Δ are of length at most L, so the radius of D_Δ is at most $R = \frac{1}{\sqrt{3}}L$. We treat the case $\|p - u\| \geq R$ first. As the (Delaunay) sphere S_u does not enclose any endpoint of Δ, we get that Δ intersects $\overrightarrow{up}$ between p and the center, q, of the circle $S_u \cap S_p$. On the other hand, assuming that no endpoint of Δ is enclosed by S_p implies that Δ intersects $\overrightarrow{up}$ between q and u, a contradiction. Now let $\|p - u\| < R$. In this case, the plane normal to $\overrightarrow{up}$ at u intersects S_p in a circle of radius R. Assuming that S_p encloses no endpoint of Δ now implies that Δ avoids the ray $\overrightarrow{up}$ altogether, a contradiction again. $\square$

By Lemma 1, the first triangle hit by $\overrightarrow{up}$ can be detected once the subset Q of P enclosed by the critical sphere S_p has been reported. This spherical range search problem has a practically efficient implementation based on kd-trees. Observe that the radius of S_p is $\Theta(L)$; it lies between R and $\sqrt{2} \cdot R$. Thus only $O(1)$ points will be reported if P obeys a minimum distance of $c \cdot L$ for some constant c.

To facilitate later decisions $v \in A$, we make use of the following property, which holds if the (practically more relevant) case $\|p - u\| \geq \frac{1}{\sqrt{3}}L$ occurs for the present vertex u. See Figure 1, left-hand side.

Lemma 2 *Let S_u' be the sphere centered at u and passing through the center of the circle $S_u \cap S_p$. All Voronoi vertices v enclosed by S_u' have the same relative position to A as the vertex u.*

Proof. By the bound on the smallest containing disk for a mesh triangle, no such triangle intersects the sphere S_u'. Thus S_u' either lies completely inside or completely outside of A. Also, S_u' does not enclose A because S_u' is empty of points from P, as is S_u. $\square$

The output is a set of Delaunay balls for P whose union approximates the object A. We observed a runtime linear in $|P|$ in all our examples. In particular, the number of vertices of $V(P)$ stayed below $9 \cdot |P|$. Each produced ball covers at least four points in P with its boundary. There may be uncovered points (at rare cases), due to the lack of their poles. Such points are added to the set as balls of radius zero. In order to be able to delete a large fraction of balls later on, we increase redundancy in covering by enlarging the radius of each ball by a user-specified constant ε. Each point in P is now covered with an offset of at most ε. It will turn out that the obtained set of balls is highly redundant even for small offsets ε.

3 Reduction algorithm

Let B_ε denote the set of ε-offset balls produced in Section 2. We aim at finding a subset of B_ε of minimal cardinality that still covers the set P of sample points. This is an instance of the classical set covering problem, shown to be NP-complete in [13]. We are going to describe a hybrid heuristic that reduces B_ε almost to the optimum in many cases, and that allows to bound the produced overhead.

In a first step, we arrange P in a kd-tree and determine which points of P are covered by which balls in B_ε. The result is stored in an incidence matrix, where entry (i, j) is put to 1 or 0 depending on whether the i-th ball contains the j-th point. Standard reduction rules are then applied iteratively to the rows and columns of this matrix: (1) If column j has exactly one entry 1, say (i, j), then delete row i and all columns having entry 1 in that row. Sphere i has to be taken for any solution. (2) If row i_1 is dominated (in 1s) by row i_2 then delete row i_1. All points covered by sphere i_1 are also covered by sphere i_2. (3) If column j_1 dominates column j_2 then delete column j_1. Every solution that covers point j_2 has to cover point j_1 as well. We speed up these operations by using hashing techniques, and avoid excessive storage by sparse matrix representation.

If the reduced matrix is nonempty, we try to decompose it into independent submatrices. These are given by the connected components in the corresponding (bipartite) incidence graph. For each submatrix M, if small enough, the respective set covering problem is solved exactly, using branch-and-bound [15] for its integer programming formulation: Minimize

$$\sum x_j \text{ w.r.t. } M^T \cdot x \geq (1, \ldots, 1)^T, \; x_j \in \{0, 1\}.$$

Whereas up to this point optimality is ensured, we have to resort to a heuristic for approximating subproblems too large for exact solving. To this end, we next choose one ball covering the most yet uncovered points. When applied repeatedly, this is a simple greedy algorithm, yielding an $O(\log n)$ approximation [12]. In fact, the set covering problem is unlikely to be approximable beyond a factor of $c \log n$ in polynomial time; see [9]. Here n denotes the number of covering sets (i.e., balls in our case). Our algorithm, after each single greedy pick, runs through the steps before again. Figure 2 displays its flow chart.

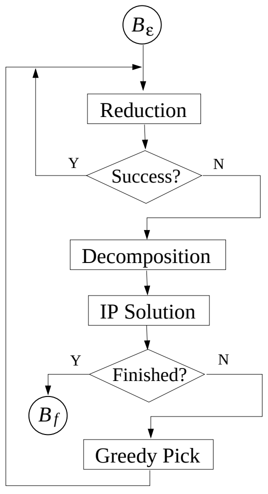

Figure 2: Control flow of the algorithm

There are several ways of checking the overhead in balls for the produced set, B_f. An obvious though quite effective way is to solve the linear program above under the relaxation $x_j \geq 0$. Unfortunately, even when being applied after the first reduction and decomposition, this method becomes very time consuming for larger offsets ε, and it also suffers from numerical problems. We therefore consider two alternative methods where a lower bound is easy to obtain.

Theorem 3 *Let n^* be the size of the optimal solution. If the algorithm takes k greedy picks then*

$$|B_f| \leq n^* + k \; .$$

Proof. After the last greedy pick, b_k, let $\tilde{B}$ be the set of balls still to choose from. Assume that the optimal solution for $\tilde{B}$ uses m balls. Taking no more greedy picks, our algorithm solves $\tilde{B} \cup \{b_k\}$ with $m + 1$ balls, whereas the optimal solution for $\tilde{B} \cup \{b_k\}$ uses at least m balls. The theorem follows by induction. $\square$

To get another bound, consider any set B of balls which covers P. For a ball $b \in B$ put $\alpha(b) = |P \cap b|$. Further, for a point $p \in P$, define its share as

$$\mathit{share}(p) = \frac{1}{\max_{p \in b \in B} \alpha(b)} \; .$$

Let $b(p)$ be a ball that achieves the maximum in the denominator above. Under the (ideal) assumption that B is a disjoint covering of P (and hence covers each point p with $b(p)$ and no other ball), $\mathit{share}(p)$ expresses the amount that p contributes to $|B|$. Thus, for any solution B, we have the lower bound

$$\left\lceil \sum_{p \in P} \mathit{share}(p) \right\rceil \leq |B| \; .$$

Let now $\hat{B}$ denote the set of balls being selected before the first greedy pick. Clearly, the size of the optimal solution for $B_\varepsilon \setminus \hat{B}$ differs from that for B_ε by exactly $|\hat{B}|$. If we fix the shares for the points in P with respect to the set $B_\varepsilon \setminus \hat{B}$, then the following holds.

Theorem 4 *The overhead in the set B_f is at most*

$$|B_f| - |\hat{B}| - \lceil \sum_{p \in P} share(p) \rceil \ .$$

4 Experimental results

We applied the ball generation method to various data sets, including the two benchmark examples below. In the following tables, the first column displays the allowed offset in percent of the (longest edge of the) bounding box. The second column shows the number of balls produced by a pure greedy algorithm, as opposed to our algorithm, shown in column three. The remaining columns list the three bounds for the overhead, described in Section 3. The LP bound, though mostly dominant, turned out to be too time expensive for the software [15] at entries '•'.

Offset %	Greedy	Hybrid	LP	Thm 3	Thm 4
.00001	4326	4085	3	0	3
.0001	3653	3209	24	62	51
.001	3564	3159	21	55	44
.01	3151	2836	17	42	40
.1	1458	1304	10	46	58
1	168	135	12	31	68
2	73	55	•	8	31
3	45	34	3	8	22

Table 1: Bunny model, $|B_\varepsilon| = 8820$

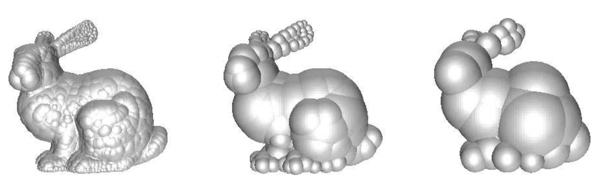
Figure 3: Bunny for offsets .00001%, 1%, and 3%

Offset %	Greedy	Hybrid	LP	Thm 3	Thm 4
.00001	16009	15092	39	105	55
.0001	13235	11699	326	1008	863
.001	12502	11193	317	996	864
.01	8995	8142	226	768	828
.1	2749	2378	221	612	928
1	321	255	•	78	162
2	136	107	•	25	74
3	86	65	•	12	37

Table 2: Dragon model, $|B_\varepsilon| = 34636$

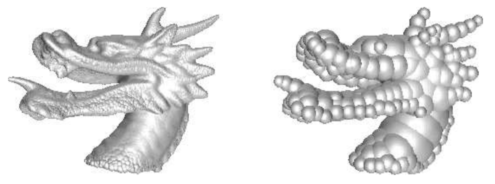
Figure 4: Dragon for offsets .00001% and 1%

References

[1] N. Amenta, M. Bern. *Surface reconstruction by Voronoi filtering.* Discrete & Computational Geometry 22 (1999), 481-504.

[2] N. Amenta, S. Choi, R.K. Kolluri. *The power crust, unions of balls, and the medial axis transform.* Computational Geometry: Theory and Applications 19 (2001), 127-153.

[3] F. Aurenhammer. *Improved algorithms for discs and balls using power diagrams.* J. Algorithms 9 (1988), 151-161.

[4] S. Bischof, L. Kobbelt. *Ellipsoid decomposition of 3D-models.* Proc. 1st IEEE Symp. 3D Data Processing Visualization and Transmission, 2002, 480-488.

[5] G. Bradshaw, C. O'Sullivan. *Adaptive medial-axis approximation for sphere-tree construction.* ACM Transactions on Graphics 23 (2004), 1-26.

[6] B. Chazelle. *Triangulating a nonconvex polytope.* Discrete & Computational Geometry 5 (1990), 505-526.

[7] `http://www.CGAL.org`

[8] E. Fogel, D. Halperin. *Exact and efficient construction of Minkowski sums of convex polyhedra with applications.* 8th Workshop Alg. Eng. Exper., Alenex'06, 2006.

[9] U. Feige. *A threshold of ln n for approximating set cover.* Proc. 28th Ann. ACM Symp. on Theory of Computing, 1996, 314-318.

[10] K. Fukuda. *From the zonotope construction to the Minkowski addition of convex polytopes.* J. Symbolic Computation 38 (2004), 1261-1272.

[11] P.M. Hubbard. *Approximating polyhedra with spheres for time-critical collision detection.* ACM Transactions on Graphics 15 (1996), 179-210.

[12] D.S. Johnson. *Approximation algorithms for combinatorial problems.* J. Computer and System Sciences 9 (1974), 256-278.

[13] R.M. Karp. *Reducibility among combinatorial problems.* R.E. Miller, J.W. Thatcher (eds.), Plenum Press, New York, 1972, 85-103.

[14] M.C. Lin, D. Manocha. *Collision and proximity queries.* J.E. Goodman, J. O'Rourke (eds.), Handbook of Discrete and Computational Goemetry, 2nd Ed., CRC, 2004, 787-807.

[15] `http://lpsolve.sourceforge.net/5.5/`

[16] C. O'Sullivan, J. Dingliana. *Collisions and perception.* ACM Transactions on Graphics 20 (2001), 151-168.

[17] I.J. Palmer, R.L. Grimsdale. *Collision detection for animation using sphere-trees.* Computer Graphics Forum 14 (1995), 105-116.

[18] J. Ruppert, R. Seidel. *On the difficulty of triangulating three-dimensional nonconvex polyhedra.* Discrete & Computational Geometry 7 (1992), 227-253.