Fast and Tight Fitting Bounding Spheres

Thomas Larsson*
School of Innovation, Design and Engineering
Mälardalen University
Sweden

Abstract

Bounding spheres are utilized frequently in many computer graphics and visualization applications, and it is not unusual that the computation of the spheres has to be done during run-time at real-time rates. In this paper, an attractive algorithm for computing bounding spheres under such conditions is proposed. The method is based on selecting a set of k extremal points along $s = k/2$ input directions. In general, the method is able to compute better fitting spheres than Ritter's algorithm at roughly the same speed. Furthermore, the algorithm computes almost optimal spheres significantly faster than the best known smallest enclosing ball methods. Experimental evidence is provided which illustrates the qualities of the approach as compared to five other competing methods. Also, the experimental result gives insight into how the parameter s affects the tightness of fit and computation speed.

CR Categories: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—Geometrical problems and computations; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

Keywords: bounding sphere, enclosing ball, extremal points, computational geometry, computer graphics

1 Introduction

Bounding spheres, also called enclosing balls, are often used to accelerate computations in computer graphics, GIS, robotics, and computational geometry. For example, they are commonly utilized to speed up operations such as picking, ray tracing, view-frustum culling, collision detection, motion planning, and range queries. Efficient computation of bounding spheres is therefore of great importance.

In 3D, the minimum sphere enclosing a set of points is uniquely defined by 2, 3, or 4 points. These points are called the set of support since they are extremal points lying on the surface of the sphere. There can be more than 4 points on the surface, but at most 4 points are needed to uniquely define the smallest sphere. Thus, a brute force computation of the minimal sphere can be performed by considering all possible combinations of 2, 3, and 4 points. For each combination, a minimal enclosing sphere is computed. Then the sphere is checked to see if it also encloses all other points. If this is the case, and the computed sphere is also the smallest valid bounding sphere found so far, it is kept as the currently smallest bounding sphere. When all the combinations have been processed, it is guaranteed that the minimal bounding sphere has been found. Of course, such a brute force $O(n^5)$ method is prohibitively slow.

Interestingly, Megiddo has presented a deterministic linear time algorithm. However, an actual implementation of this algorithm is likely to be slow due a large hidden constant in the time complexity [Megiddo 1983]. The randomization method by Welzl, however, runs in expected $O(n)$ time, and several robust implementations have been presented [Welzl 1991; Gärtner 1999; Eberly 2007].

*e-mail: thomas.larsson@mdh.se

Unfortunately, implementations based on Welzl's algorithm [1991] are still considered too slow in many real-time applications. As it appears, simpler and faster algorithms are often preferred, despite the fact that non-optimal bounding spheres are computed [Ritter 1990; Wu 1992; Ericson 2005]. Ritter's algorithm, for example, is an extremely fast two-pass linear time algorithm, which has become very popular [Ritter 1990]. Also, approximation methods based on core-sets have been proposed [Bâdoiu and Clarkson 2003; Kumar et al. 2003; Bădoiu and Clarkson 2008]. These methods are able to compute $(1 + \epsilon)$-approximations of the smallest enclosing sphere, and they have been reported to be considerably faster than the best known exact solvers.

We propose a fast and simple algorithm called the *Extremal Points Optimal Sphere* (EPOS) to compute tight fitting bounding spheres in worst-case $O(sn)$ time. The input variable s, which determines the number of directions (normal vectors) considered by the algorithm internally, can be varied as a mean to balance the trade-off between execution time and sphere quality. Using a small constant value of, e.g. $s = 3$, yields a highly efficient $O(n)$ algorithm, which computes tight fitting spheres in many cases. In fact, in this case, the algorithm runs approximately as fast as Ritter's method, while still producing bounding spheres of higher quality. Furthermore, like the Bădoiu-Clarkson algorithm, the proposed method is always capable of computing close to optimal spheres, while being significantly faster.

2 The EPOS Algorithm

A bounding sphere, S, of a point set P with n points is described by a center point, **c**, and a radius, r. As stated earlier, there is always a unique bounding sphere that is minimal. To compute it, we need to locate 2, 3, or 4 supporting points. The algorithm attempts to locate these points quickly or at least a good approximation of them. The pseudocode is given in Figure 1. It starts out by selecting a suitable subset E of P with k extremal points such that $4 < k < n$ (Line 2). Then the minimum sphere S' enclosing the point set E is computed using an exact solver, such as Gärtner's algorithm (Line 3). After this, a final iteration over all the points in P makes sure that the sphere is grown if necessary to include all points. Each time a point outside the current sphere is encountered, a new larger sphere enclosing the current sphere and the point is computed (Line 4). It is of course expected that $n > k$ from the beginning, otherwise there is no need to reduce the number of points passed to the exact solver (Lines 1, 5, 6). Since the minimum sphere of E is always computed, the resulting method is called the Extremal Points Optimal Sphere (EPOS) algorithm. Given that the points in E include the actual points of support of the optimal sphere, the algorithm is guaranteed to compute the minimum sphere.

Clearly, it is important that the extremal points in E are selected wisely. In addition, since speed is a major concern here, the selection method must be highly efficient as well. For these reasons, the extremal points E are selected by using a pre-determined normal set N with $s = k/2$ normals. For each normal, two extremal points are selected based on projecting all the input points $\mathbf{p}_i \in P$ on the normal, which is illustrated in Figure 2.

type	#op ($\mathbf{p} \cdot \mathbf{n}$)	#normals	normals
0 0 1	0	3	(1, 0, 0), (0, 1, 0), (0, 0, 1)
1 1 1	2	4	(1, 1, 1), (1, 1, -1), (1, -1, 1), (1, -1, -1)
0 1 1	1	6	(1, 1, 0), (1, -1, 0), (1, 0, 1), (1, 0, -1), (0, 1, 1), (0, 1, -1)
0 1 2	2	12	(0, 1, 2), (0, 2, 1), (1, 0, 2), (2, 0, 1), (1, 2, 0), (2, 1, 0) (0, 1, -2), (0, 2, -1), (1, 0, -2), (2, 0, -1), (1, -2, 0), (2, -1, 0)
1 1 2	3	12	(1, 1, 2), (2, 1, 1), (1, 2, 1), (1, -1, 2), (1, 1, -2), (1, -1, -2) (2, -1, 1), (2, 1, -1), (2, -1, -1), (1, -2, 1), (1, 2, -1), (1, -2, -1)
1 2 2	4	12	(2, 2, 1), (1, 2, 2), (2, 1, 2), (2, -2, 1), (2, 2, -1), (2, -2, -1) (1, -2, 2), (1, 2, -2), (1, -2, -2), (2, -1, 2), (2, 1, -2), (2, -1, -2)

Table 1: *The 49 normals used for efficient computation of extremal points. Here the normals have been divided into six groups based on the operation cost (add, sub, mul) of performing a simple dot product $\mathbf{p} \cdot \mathbf{n}$.*

EXTREMALPOINTSOPTIMALSPHERE(P, N)
input: $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ and $N = \{\mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_s\}$
output: A bounding sphere $S = \{\mathbf{c}, r)\}$
1. **if** $(n > (k \leftarrow 2s))$ **then**
2. $E \leftarrow$ FINDEXTREMALPOINTS(P, N)
3. $S' \leftarrow$ MINIMUMSPHERE(E)
4. $S \leftarrow$ GROWSPHERE(P, S')
5. **else**
6. $S \leftarrow$ MINIMUMSPHERE(P)

Figure 1: *Pseudocode for the EPOS algorithm.*

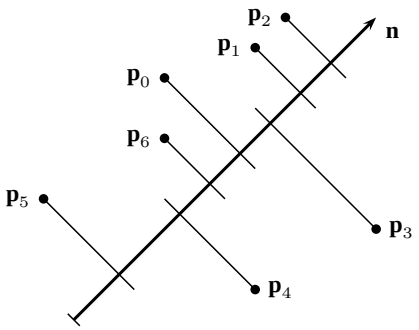

Figure 2: *To select two extremal points along a given normal $\mathbf{n}$, the input points $\mathbf{p}_i \in P$ are projected on the normal. The points with maximum and minimum projection values, $\mathbf{p}_2$ and $\mathbf{p}_5$ in this case, are selected as the extremal points.*

By using only integer values, with as many zero and one values as possible for the x, y, and z components of the normals, and still making sure that the normal directions are reasonably uniformly distributed over all possible normals of a hemisphere, the projection of the points using the dot product can be computed more efficiently. The normals used are listed in Table 1. As can be seen, up to 49 normals are used. The cost is measured as the number of arithmetical operations required to compute a dot product using the different types of normals. By unrolling the loop over the normals when projecting the points in P, the unnecessary operations for each computed dot product is easily eliminated. Also, by varying s, the relation between tightness of fit and computational speed is easily controlled. Currently, we have tested specialized implementations of the algorithm using $s \in \{3, 7, 13, 49\}$, which corresponds to selecting $k \in \{6, 14, 26, 98\}$ extremal points in E. The corresponding implementations are hereafter called EPOS-6, EPOS-14, EPOS-26, and EPOS-98.

3 Experimental Results

The experimental evaluation of the EPOS algorithm includes EPOS-6, EPOS-14, EPOS-26, and EPOS-98, as well as five other competing methods. The exact solver by Gärtner with the code on the web is used[1]. The simple but high quality approximation method by Bădoiu-Clarkson, based on core-sets is also used [Bâdoiu and Clarkson 2003] with 100 iterations used for the main loop. Furthermore, three very simple and extremely fast methods are included. These are Ritter's method [Ritter 1990], and the obvious methods of choosing either the mid point of an axis-aligned bounding box (AABB), or the average point, as the center of the sphere. These three methods have in common the fact that they make only two passes over the input points. The algorithms have been implemented in C++, compiled under Microsoft Visual Studio 2005 using the default release mode setting, and run single-threaded on a laptop PC with an Intel T2600 CPU, 2.16 GHz, with 1 GB of memory.

Rather than using random point sets, the data sets used in the benchmarks here are the vertices of 10 polygonal meshes, which is believed to be characterizing for a large number of models used in interactive computer graphics applications. Rendered images of the test models together with their bounding spheres are shown in Figure 3. In Table 2, the results from the benchmarks are presented for all models. All approximation algorithms give significant speedups compared to Gärtner's exact method. In many cases, the approximation methods are two or three orders of magnitudes faster.

The average center method is always the fastest, but it also produces bad fitting spheres in many cases, with a radius of up to twice the length of the minimum. Among the used data sets, the frog model is the worst with 26.36% increase in radius. Ritter's and the AABB center methods are also very fast, but they may also compute loose fitting spheres. For example, consider the tetrahedron model where the increase in radius is 23.28% and 26.93%, respectively. Ritter's algorithm also computes loose fitting spheres with more than a 10% increase in radius for the frog, chair, and knot models.

The proposed EPOS algorithm, on the other hand, manages to compute tight fitting spheres efficiently. The worst cases encountered over all test models were radius increases of 0.05%, 0.21%, 0.91%, and 8.94% for EPOS-98, EPOS-26, EPOS-14, and EPOS-6, respectively. Also, the last mentioned method has a comparable execution speed to both Ritter's method and the AABB center algorithm, but clearly yields tighter fitting volumes.

Furthermore, with the approximation accuracy used here for the Bădoiu-Clarkson method, it becomes significantly slower than all the tested variants of the EPOS method, and still it computes looser

[1] www.inf.ethz.ch/personal/gaertner/miniball.html

Model: frog, $N_v = 4010$, $N_s = 3$				
Algorithm	radius	inc(%)	time(ms)	speed-up
Gärtner	0.59903	0.00	207.65	1.00
Bădoiu-Clarkson	0.59997	0.16	4.73	43.91
EPOS-98	0.59903	0.00	1.42	146.35
EPOS-26	0.59903	0.00	0.36	572.21
EPOS-14	0.60019	0.19	0.22	926.80
EPOS-6	0.61349	2.41	0.12	1804.11
AABB center	0.63922	6.71	0.10	2053.30
Ritter	0.65965	10.12	0.10	2058.99
Average center	0.75696	26.36	0.07	2892.20

Model: golfball, $N_v = 100722$, $N_s = 4$				
Algorithm	radius	inc(%)	time(ms)	speed-up
Gärtner	0.50110	0.00	4343.12	1.00
Bădoiu-Clarkson	0.50216	0.21	116.98	37.13
EPOS-98	0.50112	0.00	29.51	147.16
EPOS-26	0.50114	0.01	7.56	574.18
EPOS-14	0.50145	0.07	4.80	903.91
EPOS-6	0.50155	0.09	2.52	1720.50
AABB center	0.50197	0.17	2.37	1830.28
Ritter	0.51531	2.83	2.27	1916.47
Average center	0.50189	0.16	1.72	2518.04

Model: horse, $N_v = 48485$, $N_s = 2$				
Algorithm	radius	inc(%)	time(ms)	speed-up
Gärtner	0.62897	0.00	1128.45	1.00
Bădoiu-Clarkson	0.62897	0.00	56.26	20.06
EPOS-98	0.62897	0.00	14.43	78.18
EPOS-26	0.62897	0.00	3.72	303.60
EPOS-14	0.62899	0.00	2.34	481.96
EPOS-6	0.63023	0.20	1.23	918.03
AABB center	0.63517	0.99	1.15	979.71
Ritter	0.63476	0.92	1.11	1017.47
Average center	0.65200	3.66	0.83	1359.13

Model: hand, $N_v = 327323$, $N_s = 2$				
Algorithm	radius	inc(%)	time(ms)	speed-up
Gärtner	0.52948	0.00	6086.75	1.00
Bădoiu-Clarkson	0.52992	0.08	390.04	15.61
EPOS-98	0.52948	0.00	95.94	63.44
EPOS-26	0.52949	0.00	24.78	245.60
EPOS-14	0.52949	0.00	15.81	385.02
EPOS-6	0.52949	0.00	8.38	726.53
AABB center	0.53029	0.15	7.93	767.26
Ritter	0.52949	0.00	7.60	800.43
Average center	0.61943	16.99	5.94	1024.20

Model: bunny, $N_v = 32875$, $N_s = 3$				
Algorithm	radius	inc(%)	time(ms)	speed-up
Gärtner	0.64321	0.00	1680.20	1.00
Bădoiu-Clarkson	0.64792	0.73	38.07	44.13
EPOS-98	0.64328	0.01	9.69	173.36
EPOS-26	0.64415	0.15	2.49	675.77
EPOS-14	0.64423	0.16	1.58	1062.04
EPOS-6	0.65017	1.08	0.83	2029.13
AABB center	0.67296	4.63	0.78	2164.99
Ritter	0.67694	5.24	0.74	2285.09
Average center	0.74940	16.51	0.56	2992.21

Model: tetrahedron, $N_v = 32770$, $N_s = 4$				
Algorithm	radius	inc(%)	time(ms)	speed-up
Gärtner	0.61237	0.00	3361.89	1.00
Bădoiu-Clarkson	0.61237	0.00	37.90	88.71
EPOS-98	0.61237	0.00	9.71	346.26
EPOS-26	0.61237	0.00	2.49	1349.41
EPOS-14	0.61237	0.00	1.58	2123.15
EPOS-6	0.61237	0.00	0.83	4054.59
AABB center	0.77728	26.93	0.78	4331.90
Ritter	0.75494	23.28	0.74	4522.37
Average center	0.61238	0.00	0.56	5990.06

Model: teapot, $N_v = 32922$, $N_s = 3$				
Algorithm	radius	inc(%)	time(ms)	speed-up
Gärtner	0.50313	0.00	2043.18	1.00
Bădoiu-Clarkson	0.50314	0.00	38.70	52.79
EPOS-98	0.50322	0.02	9.88	206.76
EPOS-26	0.50322	0.02	2.54	805.91
EPOS-14	0.50322	0.02	1.60	1274.82
EPOS-6	0.50322	0.02	0.83	2450.14
AABB center	0.51913	3.18	0.78	2612.96
Ritter	0.50322	0.02	0.74	2761.96
Average center	0.54038	7.40	0.57	3581.62

Model: chair, $N_v = 7260$, $N_s = 4$				
Algorithm	radius	inc(%)	time(ms)	speed-up
Gärtner	0.63776	0.00	482.73	1.00
Bădoiu-Clarkson	0.63777	0.00	8.45	57.10
EPOS-98	0.63788	0.02	2.30	209.50
EPOS-26	0.63792	0.03	0.59	819.32
EPOS-14	0.64359	0.91	0.37	1288.55
EPOS-6	0.69474	8.94	0.19	2482.68
AABB center	0.64737	1.51	0.18	2747.14
Ritter	0.73014	14.49	0.17	2870.35
Average center	0.73279	14.90	0.13	3814.45

Model: knot, $N_v = 1440$, $N_s = 4$				
Algorithm	radius	inc(%)	time(ms)	speed-up
Gärtner	0.54180	0.00	5.69	1.00
Bădoiu-Clarkson	0.54372	0.35	1.68	3.39
EPOS-98	0.54193	0.02	0.64	8.90
EPOS-26	0.54262	0.15	0.16	36.24
EPOS-14	0.54259	0.14	0.09	61.72
EPOS-6	0.54382	0.37	0.05	122.69
AABB center	0.59055	9.00	0.04	151.99
Ritter	0.61211	12.98	0.04	151.99
Average center	0.54181	0.00	0.03	216.66

Model: tiger, $N_v = 30892$, $N_s = 3$				
Algorithm	radius	inc(%)	time(ms)	speed-up
Gärtner	0.51397	0.00	238.16	1.00
Bădoiu-Clarkson	0.51704	0.60	35.74	6.66
EPOS-98	0.51424	0.05	9.11	26.15
EPOS-26	0.51507	0.21	2.34	101.85
EPOS-14	0.51507	0.21	1.48	160.55
EPOS-6	0.52531	2.21	0.78	306.44
AABB center	0.56327	9.59	0.73	327.26
Ritter	0.53835	4.74	0.78	306.77
Average center	0.55602	8.18	0.53	450.59

Table 2: *Experimental results obtained for 10 models (polygonal meshes) with N_v vertices. N_s is the number of supporting points for the minimum enclosing ball. For each algorithm, the resulting radius and execution time are given. For convenience, the increase in radius and speed-up compared to the exact solver by Gärtner are also given.*

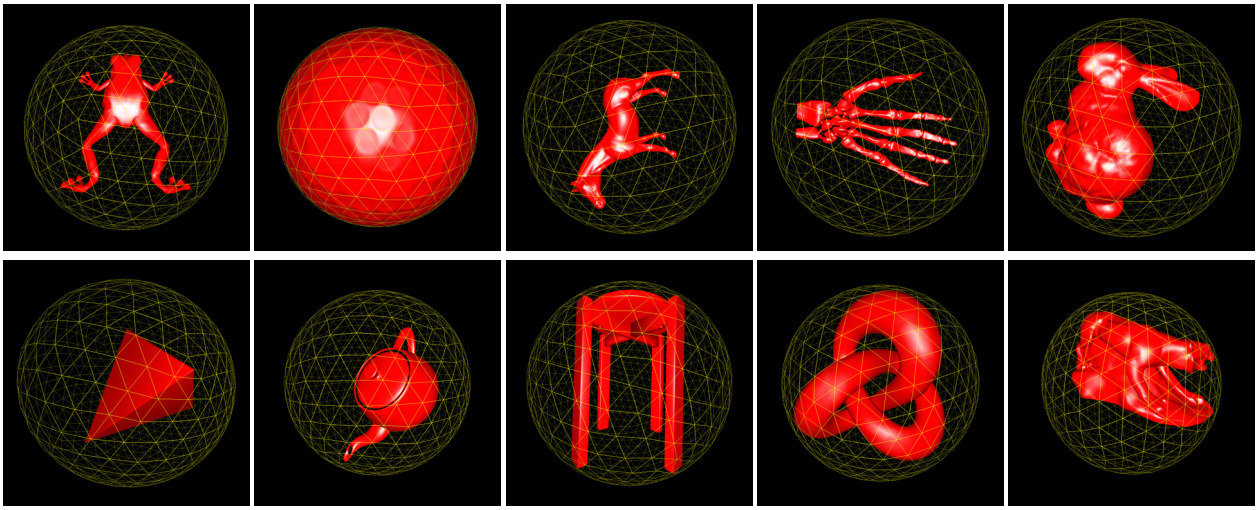

Figure 3: *Visualization of the used models. The bounding spheres shown here in wireframe were computed using the EPOS-26 algorithm.*

fitting spheres than e.g. EPOS-98 and EPOS-26 for six out of 10 test models (frog, golfball, hand, bunny, knot, and tiger).

4 Conclusions and Future Work

As shown by the experimental results, there is a huge performance gap between exact solvers and approximation methods. Non-optimal bounding spheres can be computed several orders of magnitude faster than minimum spheres. Therefore, their use is widespread in time-critical situations. On the other hand, in many applications bounding spheres can be pre-computed. In this case, minimum bounding spheres are often preferable.

The proposed EPOS algorithm is fast, robust, and it computes high-quality spheres. Therefore, it may be suitable for real-time graphics and visualization applications. For example, the EPOS-6 algorithm can with advantage replace Ritter's method in many cases, since in general it computes tighter fitting spheres at roughly the same speed. In other interactive contexts, the EPOS-14, EPOS-26, or even the EPOS-98 may be applicable, since they clearly produce tighter fitting spheres. However, they are approximately 2, 3, and 12 times slower than the EPOS-6, respectively, in all the experiments.

Interesting future work includes designing a parallel version of the EPOS algorithm which utilizes loop parallelization methods on many core architectures, as well as modern vectorized instructions sets, such as Intel's SIMD SSE. Furthermore, computing the smallest enclosing ball of balls is a similar problem, and it would be interesting to modify the EPOS algorithm to handle this case as well [Fischer and Gärtner 2003].

Also, since the EPOS algorithm generalizes to the n-dimensional case, experimental evaluation for other dimensions than 3D would be interesting. In 2D, the EPOS algorithm can be used with advantage to solve the smallest enclosing circle problem. Good results are expected by using the following fixed normal set when selecting extremal points: (0, 1), (1, 0), (1, 1), (1, -1), (1, 2), (2, 1), (2, -1), and (1, -2). In dimensions $d \geq 4$, a procedure for automatic generation of appropriate normals would be preferable. Unfortunately, the number of needed normals to compute tight fitting balls grows superlinearly with the dimension d.

Finally, the EPOS algorithm can be modified to dynamically determine a more advantageous normal set by taking the actual distribution of the input points into consideration (cf. [Wu 1992]). This may be worthwhile if fewer normals can be selected to reach the same quality of the computed spheres as in the present solution.

References

BÂDOIU, M., AND CLARKSON, K. L. 2003. Smaller core-sets for balls. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 801–802.

BĂDOIU, M., AND CLARKSON, K. L. 2008. Optimal core-sets for balls. *Computational Geometry: Theory and Applications 40*, 1, 14–22.

EBERLY, D. H. 2007. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics, Second Edition*. Morgan Kaufmann.

ERICSON, C. 2005. *Real-Time Collision Detection*. Morgan Kaufmann.

FISCHER, K., AND GÄRTNER, B. 2003. The smallest enclosing ball of balls: combinatorial structure and algorithms. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, ACM, New York, NY, USA, 292–301.

GÄRTNER, B. 1999. Fast and robust smallest enclosing balls. In *ESA '99: Proceedings of the 7th Annual European Symposium on Algorithms*, Springer-Verlag, London, UK, 325–338.

KUMAR, P., MITCHELL, J. S. B., AND YILDIRIM, E. A. 2003. Approximate minimum enclosing balls in high dimensions using core-sets. *Journal of Experimental Algorithmics 8*.

MEGIDDO, N. 1983. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing 12*, 759–776.

RITTER, J. 1990. An efficient bounding sphere. In *Graphics Gems*, A. Glassner, Ed. Academic Press, 301–303.

WELZL, E. 1991. Smallest enclosing disks (balls and ellipsoids). In *New Results and Trends in Computer Science, Lecture Notes in Computer Science 555*, H. Maurer, Ed. Springer, 359–370.

WU, X. 1992. A linear-time simple bounding volume algorithm. In *Graphics Gems III*, D. Kirk, Ed. Academic Press, 301–306.