Computing the Minimum Enclosing Circle of a Set of Planar Curves

Gill Barequet[1], Gershon Elber[1], and Myung-Soo Kim[2]

[1]Technion—Israel inst. of Technology, {barequet|gershon}@cs.technion.ac.il
[2]Seoul National University, mskim@cse.snu.ac.kr

ABSTRACT

The problem of computing the minimum enclosing circle of a point set is a classical problem in computational geometry. It is known to be an LP-type problem, hence it has a very efficient algorithm whose running time on average is linear in the number of points. In this paper we generalize this approach to smooth curves in the plane. We prove the LP characteristics of the problem and provide details of our implementation of the algorithm.

Keywords: Algorithms, planar curves, minimum spanning circle, LP-type problems.

1. INTRODUCTION

Computing the minimal enclosing circle of a given set of points in the plane is considered a classic problem of computational geometry [3]. (The original reference goes back to [11].) The optimal solution (a minimum-radius circle) can be found in $\Theta(n)$ expected time, where n is the number of points. In fact, the minimum enclosing sphere of n points in *any* dimension can be found in expected $\Theta(n)$ time by using the same method. Sharir and Welzl [10] further generalized the technique and introduced the notion of *LP-type* (linear-programming-like) problems. In a nutshell, one can apply the technique to an optimization problem in which adding a new constraint $\mathcal{C}$ does not change the solution $\mathcal{P}$, or the new solution $\mathcal{P}'$ is partially defined by $\mathcal{C}$, thereby reducing the number of possible configurations of constraints defining $\mathcal{P}'$. Amenta [2] later showed that special properties of these problems imply so-called Helly-type theorems. Fischer et al. [6, 7] analyzed minimum enclosing balls and how they can be computed. For example, the *sphere tree* supports a hierarchical modeling of complex 3-dimensional scenes using bounding spheres of various 3-dimensional objects in a scene [8]. It has been extensively used in 3-dimensional computer games [1, 5].

In this paper we show that the generalized problem, in which one aims to find the minimum enclosing circle of a set of smooth curves in the plane, is also an LP-type problem. (By smooth curves we mean polynomial curves of bounded degree.) This type of problem usually arises when there is a need to design efficient hierarchical data structures for storing a large amount of objects. These data structures support fast clipping, searching, collision detection, and other operations.

The paper is organized as follows. In Section 2 we prove the LP characteristics of the problem. We provide the full details of the algorithm in Section 3, and analyze its complexity in Section 4. We provide some experimental results in Section 5, and terminate with some concluding remarks in Section 6.

2. LP CHARACTERISTICS OF THE PROBLEM

We need the following central definition:

Definition 1 *A circle* touches *a curve that it fully encloses if either the circle is tangent to some internal point on the curve, or it contains either an endpoint of the curve (if it is open) or a C^1-discontinuity point of the curve.*

See Figure 1 for a few touching examples.

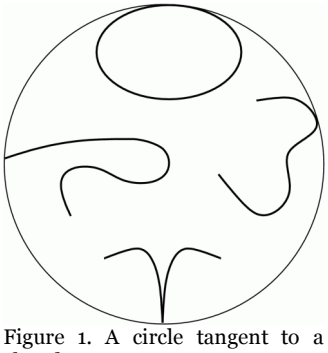

Figure 1. A circle tangent to a closed curve, tangent to an open curve, touching an endpoint of an open curve, and touching a C^1 discontinuity of a curve.

We will now show that computing the minimum enclosing circle of a set of smooth curves is LP-type. To do this we need to prove that given the minimum enclosing circle of a set of curves, if a new curve is entirely or partially outside the circle, then the minimum enclosing circle of all the curves (including the new one) is partially defined by the new curve. In other words, the new circle touches the curve at, at least, one point.

More formally, we prove the following:

Theorem 1 *Let S be a set of curves in the plane, and let $C(S)$ be the minimum enclosing circle of S. Also, let $c \notin S$ be an additional curve. Then, either*

1. *The curve c is completely inside (possibly touching) $C(S)$, in which case $C(S \cup \{c\}) = C(S)$; or*
2. *The curve c is partially or completely outside $C(S)$, in which case c touches $C(S \cup \{c\})$.*

The proof follows step by step the proof of Lemma 4.15 in [3], which makes the same assertion for points.

Proof:

1. $C(S)$ fully contains c. Assume to the contrary that some circle a whose radius is less than that of $C(S)$ contains $S \cup \{c\}$. In particular, a contains S, which contradicts the minimality of $C(S)$.

2. Clearly, $C(S \cup \{c\}) \neq C(S)$ and the two circles must intersect; see Figure 2. Let $a_0 = C(S)$ and $a_1 = C(S \cup \{c\})$ with the respective center points P_0 and P_1. Denote by Q one of the intersection points of a_0 and a_1. Define a continuous deformation between a_0 and a_1 as follows: the center of a_t (for $0 \le t \le 1$) is $P_t = (1-t)P_0 + tP_1$, and its radius is $D(P_t,Q)$, where $D(\cdot,\cdot)$ is the Euclidean distance. Since c is partially or fully outside a_0, and c is fully contained in a_1, by continuity there must be some $0 < t^* \le 1$ such that a_{t^*} contains c and touches it. The set S is contained entirely in both circles a_0 and a_1, therefore, S is contained in $a_0 \cap a_1$, which, by definition, in turn is contained in a_t (for any $0 \le t \le 1$). Also, the radius of any circle a_t (for any $0 < t < 1$) is strictly less than that of a_1. Therefore, we must have $t^* = 1$; otherwise, we have a contradiction to the minimality of a_1.

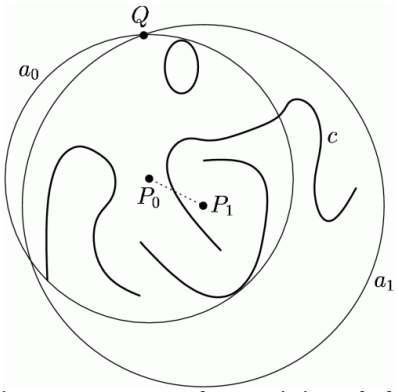

Figure 2. LP-type characteristics of the problem (see Theorem 1).

□

Note: The uniqueness of the minimum enclosing circle of the set S is shown by a similar argument of a continuous family of circles. The reader is referred to [3, §4.7, Lemma 4.15] for the exact details.

3. THE ALGORITHM

3.1 High-Level Description

After establishing the LP characteristics of the problem, we apply the well-known machinery to solve it in a time that is linear in n, the number of curves. We use the randomized algorithm of [3, §4.7, pp. 85—88] that originally finds the minimum enclosing circle of a planar point set. Assume for the moment that the minimum enclosing circle of the set of curves is defined by points the belong to different curves.

In the upper level of the algorithm, we iteratively compute the minimum enclosing circle of the first i curves, where i goes from 2 to n (see function MinCircle in Figure 3). The minimum enclosing circle a_1 of the curve c_1 either touches it at two diametrical points or at three points. In the ith step, we check whether the curve c_i lies inside a_{i-1}, the minimum enclosing circle of the first i-1 curves. (The description of this "black box" is found in Section 3.2.) If so, then $a_i = a_{i-1}$. Otherwise, if c_i lies partially or entirely outside a_{i-1}, we need to re-compute a_i, but now it is guaranteed by Theorem 1 that a_i touches c_i.

Accordingly, we now invoke a secondary function that performs the same task (namely, finding the minimum enclosing circle of a set of curves), with the only restriction that the sought-after circle touches one specific curve q

(see function MinCircleWithCurve in Figure 3). Again, we add one curve at a time, and check whether the newly-added curve is contained in the previously-computed circle. If this is not the case, we need to re-compute the circle, but this time we are guaranteed that the new circle touches both q and the newly-added curve.

```
Algorithm MinCircle (S)
        Input:   A set S of n curves in the plane.
        Output:  The minimum-radius circle that fully contains S.
    begin
        Compute a random permutation c1, ..., cn of the curves in S.
        Let a1 be the smallest circle enclosing c1.
        for i = 2, ..., n do
            if ci ∈ ai-1
                then ai := ai-1;
                else ai := MinCircleWithCurve ({c1, ..., ci-1}, ci).
            end if
        end for
        return an.
    end MinCircle

Function MinCircleWithCurve (S, q)
        Input:   A set S of n curves in the plane and a curve q, such that there exists an enclosing circle of S
                 that touches q.
        Output:  The minimum-radius circle that fully contains S and that touches q.
    begin
        Compute a random permutation c1, ..., cn of the curves in S.
        Let a1 be the smallest circle enclosing {c1,q}.
        for i = 2, ..., n do
            if ci ∈ ai-1
                then ai := ai-1;
                else ai := MinCircleWithTwoCurves ({c1, ..., ci-1}, q, ci).
            end if
        end for
        return an.
    end MinCircleWithCurve

Function MinCircleWithTwoCurves (S, q1, q2)
        Input:   A set S of n curves in the plane and two curves q1, q2, such that there exists an enclosing
                 circle of S that touches q1 and q2.
        Output:  The minimum-radius circle that fully contains S and that touches q1 and q2.
    begin
        Compute a random permutation c1, ..., cn of the curves in S.
        Let a0 be the smallest circle enclosing {q1,q2}.
        for i = 1, ..., n do
            if ci ∈ ai-1
                then ai := ai-1;
                else ai := the minimum enclosing circle of q1, q2, and ci.
            end if
        end for
        return an.
    end MinCircleWithTwoCurves
```

Figure 3. Computing the minimum-radius enclosing circle of a set of curves in the plane (following closely the algorithm of [3] for enclosed points.

Finally, we invoke a tertiary function that performs the same task, this time with the restriction that the sought-after circle touches two specific curves q_1, q_2 (see Function MinCircleWithTwoCurves in Figure 3). Again, we add one curve at a time, and check whether the newly-added curve is contained in the previously computed circle. If

not, we need to re-compute the circle, this time with the knowledge that the new circle touches all of q_1, q_2, as well as the newly-added curve. We compute this circle with a "black box" that is described in Section 3.2.

We do not need to alter the algorithm to accommodate cases in which the minimum enclosing circle is defined by two or three points that lie on the same curve. The invariant of Theorem 1 remains true. That is, if the ith curve lies (partially or entirely) outside the minimum enclosing circle of the first through the (i-1)st curve, then the new enclosing circle must touch the ist curve. The algorithm only needs to consider all possible multi-touching cases.

The entire algorithm is shown in Figure 3. The algorithm was broken into three levels only for clarity of exposition. In fact, the three functions can be implemented as a single function, which also receives an input parameter that specifies how many curves are known to be touched by the enclosing circle at the current level of calling to the function.

In the above description of the algorithm we neglected two possible cases:

1. The minimum enclosing circle is defined by two curves, in which case it either touches one of the curves at two different points, or it touches each curve once and the touching points are diametrical;
2. The minimum enclosing circle is defined by one curve, in which case it either touches the curve at three different points, or it touches the curve at two diametrical (and hence distinct) points.

To accommodate all possible cases, the primitive "black box" that computes the enclosing circle must check all cases in increasing order of complexity. First, it checks for a tri-tangency (a circle defined by one curve), then a bi-tangency (a circle defined by two curves), and finally, a simple tangency (a circle defined by three curves). Naturally, the circle needs to enclose the defining curves. The correctness of the algorithm follows directly from the LP characteristics of the problem.

3.2 Primitive Functions

We now describe the functions that the algorithm employs to operate on curves.

- **CurveInCircle**: Answers whether a curve $C(t)$ is inside a circle (P, R), centered at P and having a radius R.
- **MinEncCircleTan2Curves**: Computes the minimum enclosing circle (P, R) of two given curves $C_1(t)$ and $C_2(r)$, if it exists. (Such a circle does not exist if the minimum-radius circle that encloses both curves touches only one of them.)
- **MinEncCircleTan3Curves**: Computes the minimum enclosing circle (P, R) of three given curves $C_1(t)$, $C_2(r)$, and $C_3(s)$, if it exists.

The details of these functions are given below.

3.2.1 Is a curve inside a circle?

Consider a parametric curve $C(t)$ and a circle (P, R), centered at P and of radius R. $C(t)$ is inside the circle if and only if

$$\| C(t) - P \| \leq R, \quad \forall t,$$

or

$$F(t) = \| C(t) - P \|^2 - R^2 \leq 0, \quad \forall t. \tag{1}$$

Given a piecewise rational curve $C(t)$, Equation (1) is also rational, and hence one can examine all the coefficients of $F(t)$. If all the coefficients are nonpositive, then, due to the convex-hull property of the NURBs representation, $F(t)$ must be nonpositive as well. Otherwise, one needs to compute the zero set of $F(t)$. If the zero-set is empty, then the curve is either completely inside or completely outside the circle, in which case a single curve evaluation could reveal its inclusion status.

3.2.2 Minimum enclosing circle defined by two points

The enclosing circle can be defined by two touching points. Assume first that both points are tangency points (see, for example, Figure 4(a)). In such cases, and due to the minimality of the circle, it is tangent to the curves at two *antipodal* points. Hence, given two curves $C_1(t)$ and $C_2(r)$, the minimal enclosing circle (P, R), which touches the two curves, if it exists, must satisfy the following constraints:

$$\begin{aligned} \left\langle C_1'(t), C_1(t) - \frac{C_1(t)+C_2(r)}{2} \right\rangle &= 0, \\ \left\langle C_2'(t), C_2(r) - \frac{C_1(t)+C_2(r)}{2} \right\rangle &= 0. \end{aligned} \tag{2}$$

Having the two tangencies at antipodal points, Equations (2) ensure that the circle and the curves are indeed tangent.

Having two equations in two unknowns (t and r), Equations (2) can be solved using the multivariate piecewise-rational solver described in [4]. For all solution pairs (t_i, r_i) of Equations (2), we examine whether the circle $\mathcal{C}_i$, centered at $(C_1(t_i)+C_2(r_i))/2$ and of radius $||(C_1(t_i)-C_2(r_i))/2||$, contains the two curves.

If one tangency constraint is replaced by a real touching constraint (that is, that the enclosing circle passes through an endpoint or a C^1-discontinuity point of a curve), then we have one equation in one unknown. For example, let $C_2(r_0)$ be such a touching point on the second curve. Then, we need only solve

$$\left\langle C_1'(t), C_1(t) - \frac{C_1(t)+C_2(r_0)}{2} \right\rangle = 0,$$

and check, for each solution t_i, the validity of the circle centered at $(C_1(t_i)+C_2(r_0))/2$ and of radius $||(C_1(t_i)-C_2(r_0))/2||$. Likewise, if we have two real touching constraints, say, at $C_1(t_0)$ and $C_2(r_0)$, we need not solve any equation but only check the validity of the circle centered at $(C_1(t_0)+C_2(r_0))/2$ and of radius $||(C_1(t_0)-C_2(r_0))/2||$.

Note that **MinEncCircleTan2Curves** can easily identify cases in which the two touching points are contributed by the same curve. This is accomplished by invoking this function with the same parameter twice, namely, using $C_1(r)$ instead of $C_2(r)$. In such a case the function makes sure that the two tangency points are distinct, namely, by requiring that $t \neq r$.

3.2.3 Minimum enclosing circle defined by three points

The enclosing circle can be defined by three touching points. Assume first that all three points are tangency points (see, for example, Figure 4(b)). Hence, given three curves $C_1(t)$, $C_2(r)$, and $C_3(s)$, the minimum enclosing circle (P, R), if exists, must satisfy the following constraints:

$$\begin{aligned} \left\langle P - C_1(t), P - C_1(t) \right\rangle &= \left\langle P - C_2(r), P - C_2(r) \right\rangle, \\ \left\langle P - C_1(t), P - C_1(t) \right\rangle &= \left\langle P - C_3(s), P - C_3(s) \right\rangle, \end{aligned} \tag{3}$$

making sure that the center of the circle, P, is at the same distance from the three touching points.

Equation (3) is equivalent to the following linear equations in $P = (P_x, P_y)$:

$$\begin{aligned} 2\left\langle P, C_1(t) - C_2(r) \right\rangle &= \| C_1(t) \|^2 - \| C_2(r) \|^2, \\ 2\left\langle P, C_1(t) - C_3(s) \right\rangle &= \| C_1(t) \|^2 - \| C_3(s) \|^2 . \end{aligned}$$

Using Cramer's rule, one can symbolically solve for $P_x(t,r,s)$ and $P_y(t,r,s)$ as rational functions of t, r, and s. We then substitute the rational trivariate expression $P(t,r,s)$ into the following orthogonality constraints at the circle-curve tangency points:

$$\begin{aligned} \left\langle P - C_1(t), C_1'(t) \right\rangle &= 0, \\ \left\langle P - C_2(r), C_2'(r) \right\rangle &= 0, \\ \left\langle P - C_3(s), C_3'(s) \right\rangle &= 0. \end{aligned} \tag{4}$$

Now we have three equations in three variables and we get discrete solutions of (t,r,s) in general. Once again, Equations [4] can be solved using the multivariate piecewise-rational solver of [4]. All the solution circles are tested to see if they contain all of the three curves and the minimum of all valid solutions is chosen. Note that this circle is not necessarily the real minimum since the latter can be tangent to only two of the curves while strictly containing the third curve.

As with antipodal points, one, two, or three tangency constraints can be replaced by real touching constraints. The effect on Equations (3) will simply be the substitution of the known endpoint or a C^1-discontinuity point of a curve ($C_1(t_0)$, $C_2(r_0)$, and/or $C_3(s_0)$) for the respective unknown. This does not affect our ability to solve these equations symbolically. In addition, each replacement of a tangency constraint by a real touching constraint will result in losing one equation out of Equations (4), so that we will have k equations in k unknowns, for $0 \le k \le 3$. The validity of all the solution circles (only one circle in the special case of three real touching constraints and hence no equations) needs, as above, to be verified.

Note that **MinEncCircleTan3Curves** can easily identify cases in which the three touching points are contributed by only one or two curves. This is accomplished by invoking this function using the same curves repeatedly. In such a case the function makes sure that the three tangency points are distinct.

4. COMPLEXITY ANALYSIS

Denote by n the number of curves, by m the maximum number of segments of a single curve, and by d the maximum degree of the polynomials describing the curves. Let $I(m,d)$ be the time needed to decide whether or not such a curve is fully contained in a given circle, and let $M(m,d)$ be the time needed to compute the minimum enclosing circle of one, two, or three such curves.

The main algorithm, MinCircle, performs n iterations, in each of which it either decides in $I(m,d)$ time whether or not the minimum-radius circle (computed so far) has to change. Occasionally, it will also call the function MinCircleWithCurve. In the worst case, the main algorithm can call the latter function $\Theta(n)$ times, in case all of the second through the nth curves require an update of the enclosing circle. Similarly, the function MinCircleWithCurve performs $k = O(n)$ iterations (where k is the size of the curve set it receives as a parameter). In each iteration it decides in $I(m,d)$ time whether or not to re-compute the enclosing circle by calling the function MinCircleWithTwoCurves. As in the main procedure, calling the latter function can occur in $\Theta(k)$ iterations. The same holds for the function MinCircleWithTwoCurves. Its running time is, therefore, $\Theta(k\ (I(m,d)+M(m,d))) = O\ (n\ (I(m,d)+M(m,d)))$, where k is the size of the curve set it receives as a parameter. Overall, in the worst case, the entire algorithm requires $\Theta(n^3\ (I(m,d)+M(m,d)))$ time. Note that $I(m,d)$ and $M(m,d)$ do not depend at all on n. These are the running times of "black boxes" that solve the respective problems in numerical methods. Their time complexities depend solely on m, d, and on the convergence parameters. When the latter are fixed or assumed constant, we have $I(m,d) = O(m\ d^2)$ and $M(m,d) = O(m\ d^6)$. If we further fix m and d, then $I(m,d)$ and $M(m,d)$ can be regarded as constants. Even if we take into consideration curve endpoints and C^1-discontinuity points, the asymptotic running time of the algorithm will still not change. Hence, the worst-case running time of a practical implementation of our algorithm will be $\Theta(n^3)$.

The average case is much more favorable. We will now show that the expected number of invocations of the primitive black boxes is only $\Theta(n)$. For the following analysis assume that both $I(m,d)$ and $M(m,d)$ are constant, ignoring the multiplicative factor of $(I(m,d) + M(m,d))$ in the running times. We already know that the expected

running time of the lowest-level function MinCircleWithTwoCurves is $\Theta(k)$, where k is the size of its first parameter (the set of curves). Let us then estimate the expected running time of the function MinCircleWithCurve. Assume that its first parameter is also a set of k curves. Then it performs k-1 steps, in each of which it either spends a constant time on checks and assignments, and optionally calls the function MinCircleWithTwoCurves. At the ith step (for $1 \le i \le k$) the probability of the latter event occurring is *at most* $2/i$. This is verified by a simple backward-analysis argument: Let a_i be the circle *after* the ith step. Discard the curve c_i and run the algorithm *backward*. The circles a_i and a_{i-1} are different only if c_i was one of the at-most three defining curves of a_i. One of the at-most three curves is known (q), so the probability of the event is at most $2/i$. (If only one or two curves defined a_i, then the probability of the event $a_i \neq a_{i-1}$ would be strictly less than $2/i$.)

Now, the total expected running time of the function is at most $\sum_{i=2}^{k} ((2/i)\, O(i)) = O(k)$.

A similar analysis holds for the expected running time of the main algorithm, MinCircle. This time the probability of calling the function MinCircleWithCurve at the ith step is at most $3/i$, following a similar argument. The total expected running time of the algorithm is, thus, at most $\sum_{i=2}^{n} ((3/i)\, O(i)) = O(n)$.

Taking into account the time needed to execute the two black boxes, one that checks whether a curve is contained in a circle, and the other that computes the minimum enclosing circle of at most three curves, we conclude that the expected running time of the entire algorithm is $O(n)$. A trivial matching lower bound of $\Omega(n)$ time is obviously required for just reading n curves. Therefore, the expected running time of the algorithm is $\Theta(n)$, which is optimal.

The amount of space used by the algorithm is clearly $\Theta(n\ m\ d)$, which is the space needed to store all the curves. At all times the algorithm holds no more than a constant number of circles needed to decide whether the minimum enclosing circle of three curves touches one, two, or all the three of them.

5. EXPERIMENTAL RESULTS

We have implemented the algorithm described in this paper (see Figure 3) for computing the minimum enclosing circle of a set of curves. The software was implemented in IRIT [9]. It consists of about 600 lines of code built over the functionality of this library.

Our initial implementation assumes that the curves are closed and C^1 continuous everywhere. That is, the software does not check cases in which the enclosing circle touches endpoints or C^1-discontinuity points of the curves. Extending our software to cover these cases is straightforward since these points can be pre-computed before running the main algorithm.

	Number of Curves	Degree of Curve	Number of Control Points	Number of Curve-in-Circle Tests	Number of 2-Cirlce Touching Tests	Number of 3-Circle Touching Tests	Running Time (Min:Sec)
(a)	56	3	3	2,968	29	34	11.6
(b)	16	3	4	3,412	17	25	15.5
(c)	12	3	4	4,546	16	26	20.6
(d)	13	3	3	2,259	21	23	9.3
(e)	15	3	5	8,795	18	27	39.7
(f)	35	3	3	421	5	3	1.3
(g)	35	3	5	9,277	28	36	46.7
(h)	22	4	5	3,083	20	22	56.5
(i)	23	4	5	8,597	32	49	2:15.0
(j)	35	3	5	278	11	7	3.7
(k)	43	4	5	952	8	6	15.6
(l)	25	5	5	5,322	28	35	3:31.3

Table 1. Statistics of the test cases shown in Figure 4.

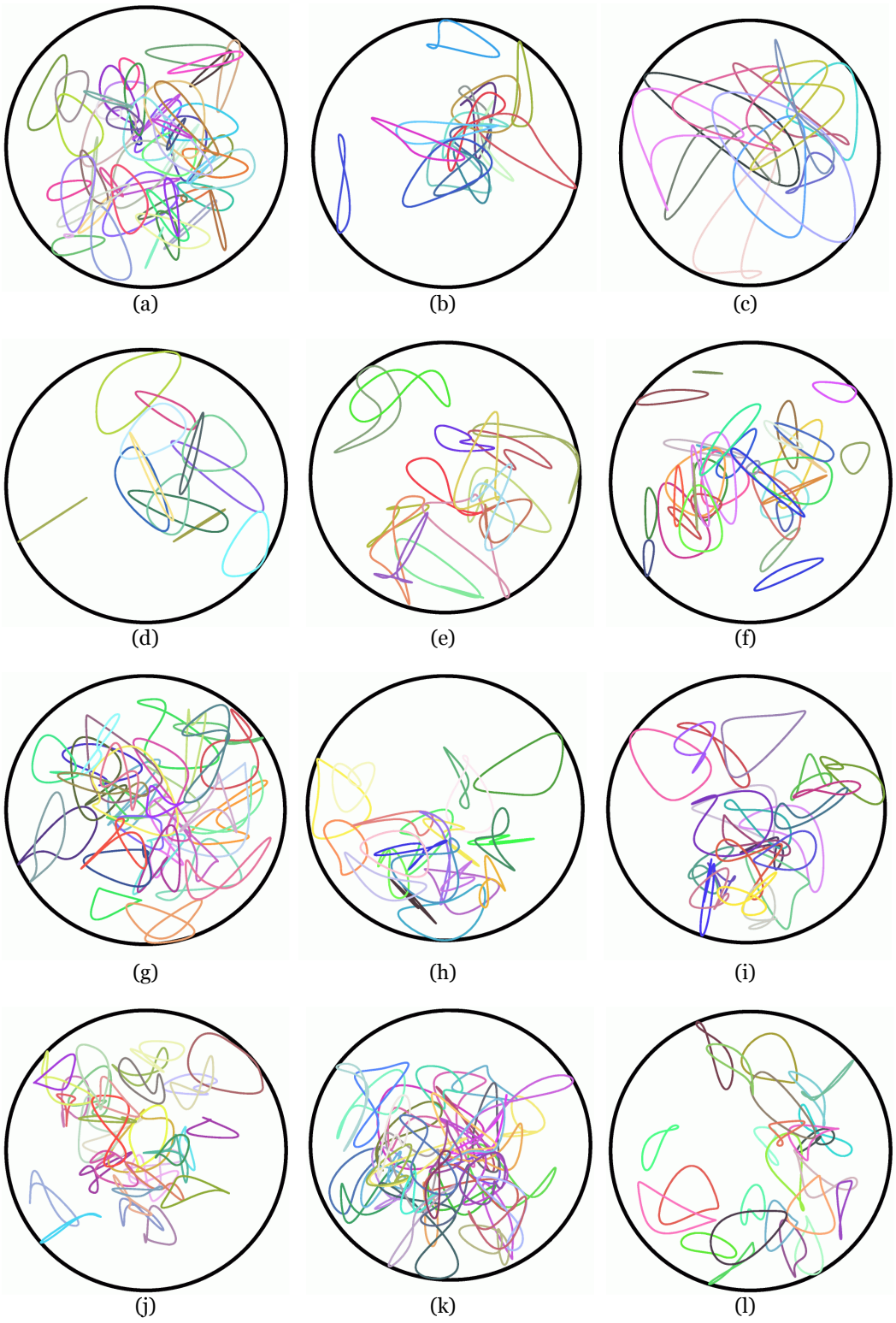

Figure 4. Minimum enclosing circles of planar C^1 curves.

Figure 4 provides some examples of our rather comprehensive experimentation with the algorithm. The respective experimental data are summarized in Table 1. The running times were measured on a modern PC workstation.

5. CONCLUSION

In this paper we present an optimal (expected) solution for the problem of computing the minimum-radius enclosing circle of a set of curves in the plane. The solution is inherently the same as for computing the minimum spanning circle (or sphere) of a set of points; however, the implementation details of our solution are significantly more complicated. The expected running time of the algorithm is linear in the number of input curves, but also contingent on the number of segments per curve and their maximum degree.

Theoretically, our approach applies in any dimension. For example, it can be used for computing the minimum enclosing sphere of a set of curves and/or surfaces (possibly with boundaries) in three dimensions. Unfortunately, the complexity of the two black boxes, one that checks whether a curve or a surface patch is contained in a sphere, and the other that computes the minimum enclosing sphere of one, two, three, or four such entities, will increase significantly. In addition, considering open curves and surfaces in space, as well as accommodating for multi-touching situations, will complicate significantly any implementation of the algorithm. Nevertheless, its asymptotic expected running time will remain linear in the number of enclosed entities. We thus leave this generalization to future research.

6. ACKNOWLEDGEMENTS

Work on this paper by all authors has been supported in part by Israel's Ministry of Science Infrastructure Grant 01-01-01509. Work on this paper by the first and second authors has been supported in part by AIM@SHAPE, a grant of the European Commission 6th Framework. Work on this paper by the second author has been supported in part by the Israel Science Foundation Grant 857/04. Work by the first and third authors has been supported in part by a grant for Korean-Israeli Research Cooperation.

7. REFERENCES

[1] T. Akenine-Moeller and E. Haines, *Real-Time Rendering*, 2nd Ed., A.K. Peters, Natick, MA, 2002.
[2] N. Amenta, Helly-type theorems and generalized linear programming, *Discrete & Computational Geometry*, 12 (1994), 241-261.
[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry, Algorithms, and Applications* (2nd ed.), Springer-Verlag, Berlin, 2000.
[4] G. Elber and M.-S. Kim, Geometric constraint solver using multivariate rational spline functions, *Proc. 6th ACM/IEEE Symp. on Solid Modeling and Applications*, Ann Arbor, MI, 1-10, June 2001.
[5] D.H. Eberley, *3D Game Engine Design*, Morgan Kaufmann, San Francisco, CA, 2001.
[6] K. Fischer and B. Gärtner, The smallest enclosing ball of balls: Combinatorial structure and algorithms, *Proc. 19th ACM Symp. on Computational geometry*, San Diego, CA, 292-301, June 2003.
[7] K. Fischer, B. Gärtner, and M. Kutz, Fast smallest-enclosing-ball computation in high dimensions, *Proc. 11th Ann. European Symp. on Algorithms*, Budapest, Hungary, *Lecture Notes in Computer Science*, 2832, Springer-Verlag, 630-641, September 2003.
[8] P.M. Hubbard, Approximating polyhedra with spheres for time-critical collision detection, *ACM Trans. on Graphics*, 15 (1996), 179-210.
[9] IRIT 9.0 User's Manual, The Technion—IIT, Haifa, Israel, 2002. Available at http://www.cs.technion.ac.il/~irit.
[10] M. Sharir and E. Welzl, A combinatorial bound for linear programming and related problems, *Proc. 9th Symp. on Theoretical Aspects of Computer Science*, Cachan, France, *Lecture Notes in Computer Science*, 577, Springer-Verlag, 569-579, February 1992.
[11] E. Welzl, Smallest enclosing disks (balls and ellipsoids), in: *New Results and New Trends in Computer Science* (H. Maurer, ed.), *Lecture Notes in Computer Science*, 555, Springer-Verlag, 359-370, 1991.