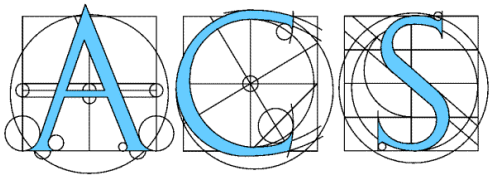

ACS

Algorithms for Complex Shapes
with Certified Numerics and Topology

On filter methods in CGAL's 2D curved kernel

Michael Kerber

ACS Technical Report No.: ACS-TR-243404-03

Part of deliverable:	WP-III/D4
Site:	MPI
Month:	36

Project co-funded by the European Commission within FP6 (2002–2006)
under contract nr. IST-006413

Abstract

We report on several filter techniques on top of the recently completed CGAL package `Curved_kernel_via_analysis_2`, a model for CGAL's ArrangementTraits_2 concept. Geometric predicates for two arcs are usually answered by analyzing the geometry of the underlying supporting curve pair. Such an analysis tends to be a time-consuming task, especially if arcs are defined by algebraic curves of arbitrary degree. Our refined model `Filtered_curved_kernel_via_analysis_2` tries to prevent the computation of such a curve pair analysis if possible by a filtered method to detect non-intersections. When instantiated with a proper model of the CurveKernel_2 concept, we show experimentally that fewer curve pair analyses are necessary.

1 Introduction

CGAL's new `Curved_kernel_via_analysis_2` (`CKvA`) [1] package provides geometric functionality for curved objects in the plane. For instance, it decomposes curves into its x-monotone segments, compares points in the plane lexicographically and computes the intersections of x-monotone arcs. By that, it constitutes a model for ArrangementTraits_2.

The `CKvA` itself is a generic package, the type of input objects is not specified a priori. Instead, it must be instantiated with a model of the CurveKernel_2 concept: this model specifies which input object are considered, and how points and (x-monotone) arcs are represented. Moreover, the model has to provide methods to analyze geometric properties of a single curve (`Curve_analysis_2`) and of a pair of curves (`Curve_pair_analysis_2`). The `CKvA` mainly relies on these analyses to answer queries on points and arcs.

Usually, the analysis of curves and curve pairs is a time-consuming task. In our applications, the input objects are typically algebraic curves, either of bounded degree, or even in full generality. A curve (pair) analysis therefore causes expensive symbolic computations and usually constitutes the bottleneck in geometric algorithm, no matter how carefully these analyses are implemented. In the worst case, n curve analyses and $\binom{n}{2}$ curve pair analyses are necessary for n input objects (of course, the results of such an analysis are cached for efficiency). Therefore, it is highly desirable to prevent as many such analyses as possible in efficient algorithms.

In this work, we report on a refined model of the `CKvA`, which we call `Filtered_curved_kernel_via_analysis_2`. The idea is to detect without using a curve pair analysis when two arcs do not intersect. For that we construct an approximating box that contains the arc and check whether the boxes of two arcs are disjoint. In that way, curve pair analyses are only necessary when two arcs really intersect, or when their distance is too small, so that the approximating boxes overlap. Also other functions in the `CKvA` have to be adapted to prevent curve pair analyses.

In particular, when used with a model of AlgebraicKernel_d_2 as CurveKernel_2 instance, one has to take care that numerical filters are used in the algebraic kernel as well. We report on experiments for arrangement computation

of arcs of algebraic plane curves. They show that indeed, the number of curve pair analyses is reduced when the `Filtered_curved_kernel_via_analysis_2` is used.

2 The filtered intersection predicate

The `Intersection_2` predicate is one of the must fundamental predicates in the `CKvA`: Given two (x-monotone) arcs, compute their intersection points. We derive a filtered predicate that can quickly detect when two arcs are not intersecting at all, at least in favorable situations.

Our approach is straight-forward: for an arc a, find an area A that is guaranteed to contain the whole arc. To detect non-intersections, simply check for disjointness of the areas of two arcs.

Intuitively, the smaller the area of an arc, the more accurate the filter works, but also, the more expensive its computation is expected to be. We take the simplest possible area, namely an axis-aligned box for each arc.

To compute a box A of an arc a that is part of an algebraic curve f, we proceed as follows: define a threshold t. Refine the x-coordinates of the endpoints up to precision t to compute the x-range of A. Compute the x-critical point of f (i.e., the intersections of f and $\frac{\partial f}{\partial x}$). For each one that lies also on a, approximate the y-coordinate up to precision t. Also refine the y-coordinate of the endpoints up to this precision. The smallest and largest y-coordinates then defines the y-range of A.

This method also applies in a compactified sense if one (or both) of the endpoints are infinite: such points can be interpreted as $(\alpha, \pm\infty)$, $(\pm\infty, \pm\infty)$, or $(\pm\infty, \beta)$, and the x- and y-range can be set accordingly. If the endpoint has x-coordinate $\pm\infty$, its y-coordinate can be computed using the method `Curve_analysis_2::asymptotic_value_of_arc`.

With this, the filtered intersection predicate now works as follows: Given two arcs a_1 and a_2, check their x-ranges for disjointness. If disjoint, they do not intersect. If they overlap, compute the arcs a_1' and a_2' that are restricted to the overlapping region. Compute the approximations A_1 of a_1' and A_2 of a_2'. If they do not overlap, the arcs do not intersect. If they overlap, use the exact predicate to check for intersections.

3 Other filtered predicates

3.1 `Compare_y_near_boundary_2`

Another predicate form the `CKvA` is `Compare_y_near_boundary_2`: given two arcs a_1 and a_2 that are unbounded in x-direction, say they both go to $+\infty$. Then, the predicate returns the y-order of the arcs "just before" $+\infty$ (this is well defined for algebraic arcs since they can only change their order finitely many times).

The usual strategy is to perform a curve pair analysis of the two corresponding curves, and look at the rightmost status line of the analysis to de-

termine the order of the arcs. To prevent this exact method, we again use `Curve_analysis_2::asymptotic_value_of_arc`: let $(+\infty, \beta_1)$ be the endpoint of a_1, and $(+\infty, \beta_2)$ be the endpoint of a_2, where $\beta_1, \beta_2 \in [-\infty, +\infty]$. If there are not equal, say $\beta_1 < \beta_2$, it follows by basic calculus that the arc a_1 must be below the arc a_2 close at $+\infty$. If $\beta_1 = \beta_2$, the exact method must be used.

3.2 Comparisons

The comparison functor do not provide filter techniques by themselves; it simply calls the compare function of the exact `CKvA`, which in turn calls the suitable compare function of the AlgebraicKernelWithAnalysis_2 model. However, to design a meaningful filtered kernel, the underlying AlgebraicKernelWithAnalysis_2 must provide filtered implementation of these predicates. The `Filtered_algebraic_curve_kernel_2`, presented in [3] implements such filtered predicates: In order to compare two covertical points lexicographically, it first approximated their y-coordinates up to some threshold, and checks the intervals for disjointness. Only if this fails, the more expensive method with a curve pair analysis is used.

Also, the functor `Is_on_2(p,f)` that checks whether a point p lies on the curve f is filtered in this way: the sign of $f(p)$ is approximated using interval arithmetic, so that points away from the curve can be identified easily.

4 Implementation

The filtered curved kernel is available inside the CGAL library as the class `Filtered_curved_kernel_via_analysis_2` inside the `Curved_kernel_via_analysis_2` package. It has the following interface:

```
template < class CurvedKernelViaAnalysis_2 >
    class Filtered_curved_kernel_via_analysis_2;
```

CurveKernelViaAnlysis_2 must be a model of the corresponding concept. Its functors are used whenever the filtered methods do not apply. The underlying CurveKernel_2 model is defined implicitly in the CurvedKernelViaAnalysis_2 model.

5 Experiments

Our experiments were performed on an AMD Dual-Core Opteron(tm) 8218 (1 GHz) multi-processor platform with a total memory of 32 GB, running Debian Etch. The program was compiled using gcc-4.1.2 with compiler flags `-O2` and `-DNDEBUG`. The internal CGAL release 3.4-I-302 has been used for the benchmarks. This release was configured using boost-1.33.1, gmp-4.2 and mpfr-2.2.0. Also, the program requires the AlciX library plus other parts of Exacus. A tarball with the files used is available on demand. AlciX internally requires the NTL-library (version 5.4).

The EXACUS part is necessary since the `Algebraic_curve_kernel_2` instance that we use as CurveKernel_2 model needs certain parts from EXACUS (compare [3]). Since the `CKvA` is a model of the `Arrangements_traits_2` concept, we can use it to compute arrangements of arbitrary algebraic plane curves with it (this technique has also been described in [2]). This dependency on EXACUS is temporary; the merge of the relevant parts into CGAL is on its way.

We start with our experiments by computing the arrangements of random curves of degree 5 and bitsize 50 per coefficient. We compare the unfiltered version of the `CKvA` (which uses an instance of `Algebraic_curve_kernel_2` without filtered comparison methods) with the filtered version (which uses an instance of `Filtered_algebraic_curve_kernel_2` with filtered comparison methods as explained in Section 3.2). The filtered kernel always uses an approximation threshold of $\frac{1}{100}$, that means, the coordinates are approximated up to precision $\frac{1}{100}$, and the exact method is used if the filtered predicate was not successful. We show both the overall running time to compute the arrangement, and the number of curve pair analyses that was performed (CPA). For the filtered kernel, we also count the number of *filter failure* (ff), that means, how often the filtered intersection predicate did not succeed although the arcs do not intersect.

Table 1: Arrangements of algebraic curves of degree 5 with 50 bit coefficients

			unfiltered		filtered		
# curves	nodes	edges	# CPA	time	# CPA	time	ff
10	211	372	45	1.99	55	3.24	60
20	656	1238	190	4.12	210	9.26	151
30	1365	2630	435	8.55	465	19.23	249
40	2408	4654	780	15.00	820	34.48	346
50	4129	8056	1225	25.64	1275	56.73	506

Table 1 shows that our filter techniques are not really useful for that examples: No curve analysis was prevented, and the total running time increases. This is not surprising though, because two curves of degree 5 always have a real intersection point by Bezout's Theorem, and so the filtered intersection predicate cannot be successful for all arc pairs. The running times show that the filter techniques do not have negligible running time, and the number of curve pair analyses even increases. This is not surprising either, because the filtered intersection predicate requires the computation of x-critical point which triggers a curve pair analysis of a curve with its derivative.

To demonstrate the strength of our filtered kernel, we slightly change the input: Instead of computing the arrangement of the whole curve, we choose one random x-monotone arc of each curve, and compute the arrangements of these arcs. One expects much fewer intersection points in this case.

As Table 2 shows, the number of curve pair analyses is indeed reduced when the filtered kernel is used. However, we still observe that the overhead caused by the filter is still so big that the overall performance is worse. This is mainly due to the rather low-degree curves considered here; when we use curves of

Table 2: Arrangements of arcs of algebraic curves of degree 5 with 50 bit coefficients

			unfiltered		filtered		
# curves	nodes	edges	# CPA	time	# CPA	time	ff
50	175	264	300	4.23	220	6.31	49
100	724	1252	932	12.70	786	17.24	108
150	1307	2322	1696	25.02	1415	33.21	173
200	2645	4930	3116	41.88	2667	56.48	263
250	4007	7556	4471	60.95	3918	83.58	338
300	5131	9716	5887	76.59	5179	107.35	439

degree 9 instead, the reduction of CPA's in fact also improves the performance, as Table 3 demonstrates. The effect becomes even more significant if the degree of the input curves is pushed up higher, since the influence of the curve pair analyses on the total running time becomes more significant.

Table 3: Arrangements of arcs of algebraic curves of degree 9 with 50 bit coefficients

			unfiltered		filtered		
# curves	nodes	edges	# CPA	time	# CPA	time	ff
20	37	32	42	12.05	34	12.67	8
40	84	84	138	33.53	97	32.07	28
60	185	236	291	66.49	183	52.79	34
80	276	376	444	104.00	305	82.61	69
100	433	642	660	142.82	471	126.70	100

6 Outlook

Our implementation for the filtered kernel is in an experimental status, still we have seen that our strategy helps to improve several kernel operations. However, we want the filtered kernel also to be practically useful for input curves of smaller degree. For that, we need to reduce the overhead caused by the filtered predicates themselves, in particular the intersection predicate. That means we need more efficient ways to approximate the y-interval of x-monotone arcs.

A better approximation of arcs than the current axis-aligned box representation might reduce the number of filter failures. However, it is not clear how this approximation can be done without worsening the performance of the intersection filter.

Finally, a real improvement would be to filter also cases where two arcs do have intersections, without performing a curve pair analysis. We are currently investigating possibilities to include such methods into our model.

References

[1] E. Berberich and P. Emeliyanenko. CGAL's curved kernel via analysis. Technical Report ACS-TR-123203-04, Algorithms for Complex Shapes with certified topology and numerics, Max Planck Institut für Informatik, Saarbrücken, Germany, 2008.

[2] A. Eigenwillig and M. Kerber. Exact and efficient 2D-arrangements of arbitrary algebraic curves. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA08)*, 2008. 122–131.

[3] P. Emeliyanenko and M. Kerber. An implementation for the 2D algebraic kernel. Technical Report ACS-TR-363602-01, Algorithms for Complex Shapes with certified topology and numerics, Max Planck Institut für Informatik, Saarbrücken, GERMANY, 2008.