# Infimaximal Frames
# A Technique for Making Lines Look Like Segments

Kurt Mehlhorn and Michael Seel

November 11, 2002

**Abstract**

Many geometric algorithms that are usually formulated for points and segments generalize easily to inputs also containing rays and lines. The sweep algorithm for segment intersection is a prototypical example. Implementations of such algorithms do, in general, not extend easily. For example, segment endpoints cause events in sweep line algorithms, but lines have no endpoints. We describe a general technique, which we call infimaximal frames, for extending implementations to inputs also containing rays and lines. The technique can also be used to extend implementations of planar subdivisions to subdivisions with many unbounded faces. We have used the technique successfully in generalizing a sweep algorithm designed for segments to rays and lines and also in an implementation of planar Nef polyhedra [Nef78, Bie95].

Our implementation is based on concepts of generic programming in C++ and the geometric data types provided by the C++ Computational Geometry Algorithms Library (CGAL).

## 1 Introduction

Many geometric algorithms that are usually formulated for points and segments generalize nicely to inputs containing also rays and lines. Do implementations generalize as easily? Let us consider two concrete examples: plane sweep for segment intersection and map overlay.

In the plane sweep algorithm for segment intersection a vertical line is swept across the plane from left to right. The intersections between the sweep line and the input segments are kept in a data structure, the Y-structure. The Y-structure is updated whenever the sweep line encounters a segment endpoint or an intersection point between two segments. The event points are kept in a priority queue, the X-structure. The sweep paradigm can clearly also handle rays and lines. Will an implementation generalize easily, e.g., does LEDA's implementation [MN99, Section 10.7] generalize? It does not. For example, the X-structure needs to be initialized with the endpoints of the segments, but what are the endpoints of lines and segments?

In Section 2 we will argue that the answer is not given by projective geometry (neither standard nor oriented). We will also argue that enclosing the scene in a fixed geometric frame and clipping rays and lines at the frame is an unsatisfactory solution. It excludes on-line algorithms, it requires non-trivial changes in the software structure, and it decreases the effectiveness of floating point filters. In Section 3 we propose infimaximal frames as a general technique for handling rays and lines. We propose to enclose the scene in a frame of infimaximal size and to clip rays and lines at the frame. Infimaximal frames support on-line algorithms, require no change in software structure, and cooperate well with floating point filters.

Our second example concerns map overlay. The texts [dBvKOS97, Section 2.3] and [MN99, Section 10.8] describe algorithms for maps with single unbounded faces, i.e., all faces (except one

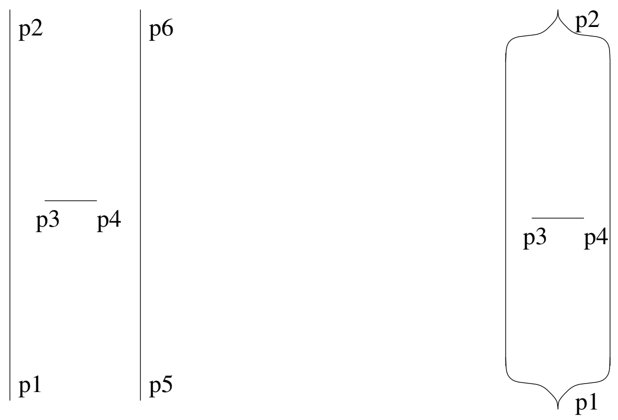


Figure 1: The left part shows a scene consisting of two parallel vertical segments and one horizontal segment. The sweep line encounters the endpoints in the order $p_1$, $p_2$, $p_3$, $p_4$, $p_5$, $p_6$. In the right part the vertical segments are extended to lines. In projective geometry, parallel lines share endpoints: Oriented projective geometry identifies $p_1$ with $p_5$ and $p_2$ with $p_6$ and standard projective geometry identifies all four points. In either case, there is no order on the endpoints which would allow to sweep the scene.

unbounded face per map) are bounded by simple closed polygons. Again the algorithms readily generalize to subdivisions with more than one unbounded face, e.g., Voronoi diagrams or arrangements of lines. Will implementations generalize? No, they do not. For example, the standard data structure for representing maps, namely doubly connected edge lists [PS85, dBvKOS97], assumes that all face cycles are closed and hence DCELs cannot even represent subdivisions with several unbounded faces in a direct way. Infimaximal frames offer a simple solution. Enclosing the scene in an infimaximal frame makes all faces (except the outside of the frame) finite and hence extends the use of DCELs to subdivisions with several unbounded faces.

This paper is structured as follows. In Section 2 we discuss projective geometry and the inclusion of input scenes in concrete geometric frames and argue that these approaches are insufficient. In Section 3 we introduce infimaximal frames and discuss their mathematics. In Section 4 we describe our implementation of infimaximal frames. Section 5 discusses our application experience. We report about the use of infimaximal frames in a sweep algorithm and in the implementation of Nef polyhedra and we compare the efficiency of our implementation of infimaximal frames with a realization of concrete geometric frames. We will see that there is no loss of efficiency and in some situations even a gain.

## 2 Alternative Approaches

We discuss projective geometry and the inclusion of the scene in a concrete geometric frame. We argue that projective geometry is unable to solve our problem and that the inclusion in a concrete geometric frame is unsatisfactory.

## 2.1 Projective Geometry

Projective geometry provides points at infinity and hence, at first sight, seems to solve all our problems. There are two versions of projective geometry: the standard version [Cox87] and the oriented version of Stolfi [Sto91]. In the standard version, there is one point at infinity for every family of parallel lines, and in the oriented version, there are two points at infinity for every family of parallel lines. Neither version allows to sweep the configuration shown in the right part of Figure 1. In this configuration, a finite segment lies between two vertical parallel lines. Since the finite segment lies completely to the right of the left vertical line, the left vertical line should be swept before the finite segment. Similarly, the right vertical line should be swept after the segment. However, parallel lines share endpoints in projective geometry and hence there is no way to define a sweep order on the endpoints of lines and segments. We conclude that projective geometry is unable to solve our problem.

## 2.2 Inclusion in a Concrete Geometric Frame

The following argument is typically used to show that an algorithm designed for segments can also handle rays and lines:

> Enclose the input scene in a large enough frame and clip rays and lines at the frame. Solve the problem for segments and translate back to rays and segments. The frame must be large enough such that no interesting geometry is lost. Adding and removing the frame are simple pre- and postprocessing steps which do not affect the asymptotic running time of the algorithm.

We next argue that inclusion of the input scene in a concrete geometric frame is a bad implementation strategy.

- The frame must be large enough so that no interesting geometry is lost and hence the frame size can only be chosen once the input is completely known. Thus on-line algorithms are excluded. Also, merging different scenes is non-trivial, if constructed with different frame sizes. It requires to change representations of points.

- Implementations have to be changed in a non-trivial way. We first need to make a pass over the data to determine an appropriate frame size. Next we clip rays and lines at the frame and replace them by segments. Then we run the algorithm for segments. Finally, we need to translate back.

- A large concrete frame size makes floating point filters ineffective. In the exact computation paradigm of computational geometry [OTU87, KLN91, Yap93, YD95, Sch], all geometric predicates are evaluated exactly. Floating point filters are used to make exact computation efficient [FvW96, MN94, BFS98]. Floating point filters are most effective when point coordinates are small. Clipping rays and lines on a concrete frame introduces points with large coordinates which make filters less effective. Observe that in an arrangement of lines a single intersection with large coordinates will force the use of a large frame. Also observe, that lines with $k$ bit coefficients may intersect in points whose coordinates require $2k$ bits.

It may seem that frame size can be changed dynamically. For example, one could define the frame size as a variable. Whenever a ray or line needs to be clipped, the current value of the variable is taken as the frame size, and whenever interesting geometry happens outside the current frame, the value of the variable is increased. Also, when the frame size is increased, the coordinates of all points on the frame must be changed in order to maintain consistency and hence the approach incurs a large overhead in time if the frame size needs to be adopted frequently. Infimaximal frames avoid this overhead.

# 3 Infimaximal Frames

We propose to use a frame of infimaximal size. More precisely, we enclose the scene in a square box with corners $NW(-R,R)$, $NE(R,R)$, $SE(R,-R)$, and $SW(-R,-R)$. We leave the value of $R$ unspecified and treat $R$ as an infimaximal number, i.e., a large positive number which is, finite but larger than the value of any concrete real number. Infimaximal numbers are the counterpart of infinitesimal numbers as, for example, used in symbolic perturbation schemes [EM90].

Before we go into details, we argue that this proposal overcomes the deficiencies of the concrete frame approach.

- Since the value of $R$ is infimaximal, no interesting geometry lies outside the frame and on-line problems cause no difficulties. All scenes are constructed with the same infimaximal frame and hence merging scenes causes no problems.
- Implementations do not have to be changed at all. We will define new point classes and segment classes (extended points and extended segments, respectively). Extended points are either standard points or points on our infimaximal frame and extended segments are spanned by extended points. Thus extended segments can model standard segments, rays and lines. Many LEDA [LED] and CGAL [CGA] algorithms can operate on the new point and segment classes without any change, see Section 5.1 for examples.
- Filters stay effective up to larger input bit sizes. Point coordinates are polynomials in $R$ and the evaluation of geometric predicates amounts to determining the sign of the highest non-zero coefficient. These coefficients are smaller than the corresponding values arising in the computation with a fixed frame size, see Section 4 for details.

## 3.1 Frame Points and Extended Points

We define extended points in terms of their position with respect to the square box.

**Definition 1:** A *frame point* or *non-standard point* is a point on one of the four frame boundaries. A point on the left frame boundary has coordinates $(-R, f(R))$, where $f$ is a function with $|f(R)| \leq R$ for all sufficiently large $R$. Points on the other frame boundaries are defined analogously, i.e., points on the right boundary have coordinates $(R, f(R))$, points on the lower boundary have coordinates $(f(R), -R)$, and points on the upper boundary have coordinates $(f(R), R)$. A *standard point* is simply a point in the affine plane and has coordinates $(x,y)$ with $x, y \in \mathbb{R}$. An *extended point* is either a standard point or a non-standard point.

Although the definition above makes sense for arbitrary function $f$, we restrict ourselves to linear functions in $R$ in this paper, as this will suffice to model endpoints of rays and lines.

## 3.2 The Endpoints of Segments, Rays, and Lines

The endpoints of a segment are standard points, a ray has a standard endpoint and a non-standard endpoint and a line has two non-standard endpoints.

Consider a line $\ell$ with line equation $ax+by+c=0$. If $b=0$, the endpoints of the line are $(-c/a, \pm R)$. If $b \neq 0$, we have $y = mx+n$, where $m=-a/b$ and $n=-c/b$. If $|m|<1$, the line has endpoints $(\pm R, \pm mR+n)$, if $|m|>1$, the line has endpoints $(\pm R/m - n/m, \pm R)$, and if $|m|=1$, the line has endpoints $(-R, -mR+n)$ and $(R-nm, mR)$ if $sign(n) = sign(m)$, and has endpoints $(-R-nm, -mR)$ and $(R, mR+n)$ if $sign(n) \neq sign(m)$. The non-standard endpoint of a ray is determined similarly. We see that the coordinates of the endpoints of a line are simple linear functions in $R$, our infimaximal.

The endpoints of more complex geometric objects can be determined similarly, but the coordinate expressions become more complex. For example, the parabola $(y-x)^2 = 5x$ intersects the upper frame boundary in point $(R+5/2-\sqrt{5R+25/4}, R)$ and the right frame boundary in point $(R, R-\sqrt{5R})$. To obtain these solutions either set $x=R$ or $y=R$, calculate the solutions to the squared equations and ignore the points that are not part of the square frame consisting of all non-standard points.

We use the coordinate approach described above also in our implementation, i.e., the coordinates of frame points are linear functions in $R$ (recall that we are only dealing with lines and segments). We have also explored an alternative implementation strategy, namely to store a frame point as a reference to the underlying geometric object plus an indicator which selects the appropriate frame point. We found that this strategy leads to heavy case switching within geometric predicates as a point has four different representations (a standard point, a ray tip, the left endpoint of a line, and the right endpoint of a line) and hence a predicate operating on four points, e.g., the *side_of_circle* predicate defined below, would have to cope with up to $2^4$ cases.

The coordinate approach treats all cases uniformly. Moreover, it incurs no significant runtime penalty as we will see in Section 4.

### 3.3 Predicates on Extended Points

The flow of control in geometric algorithms is determined by the evaluation of geometric predicates. Important predicates are the lexicographic order of points (*compare_xy*), the orientation of a triple of points (*orientation*), and the incircle test for a quadruple of points (*side_of_circle*). Many[1] geometric predicates can be evaluated by computing the sign of a simple function defined on the coordinates of the points involved.

For example, the lexicographic order is simply a cascaded comparison of coordinates (sign of their difference), the orientation of three points is defined by

$$\text{orientation}(p1, p2, p3) = \text{sign} \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}$$

and the side of circle predicate is given by

$$\text{side\_of\_circle}(p1, p2, p3, p4) = \text{sign} \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ x_1^2+y_1^2 & x_2^2+y_2^2 & x_3^2+y_3^2 & x_4^2+y_4^2 \end{vmatrix}$$

What happens when we apply these predicates to extended points? The value of the predicate will become the sign of a function in $R$. If this sign is independent of $R$ for all large values of $R$, the value of the predicate is well defined. We can qualify such predicates as follows.

[1] The authors know of no predicate that is not.

**Lemma 3.1:** If a geometric predicate is defined as the sign of a polynomial in point coordinates and the coordinates of extended points are polynomials in $R$, the value of the predicate when applied to the extended points is well-defined.

*Proof.* Assume that the predicate is defined as the sign of a polynomial $P$. Substituting the point coordinates into $P$ gives us a polynomial in $R$. For sufficiently large values of $R$, the sign of this polynomial is given by the sign of the highest non-zero coefficient. $\square$

We give an example. Consider the orientation of two standard points $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ and one non-standard point $p_3 = (-R, mR + n)$ on the left frame segment. We obtain:

$$\begin{aligned} \text{orientation}(p1, p2, p3) = \text{sign} \quad ( \quad & [m(x_2 - x_1) + (y_2 - y_1)]R \\ + \quad & [n(x_2 - x_1) + (x_1 y_2 - x_2 y_1)] \quad ) \end{aligned}$$

If the coefficient of $R$ is non-zero, its sign determines the orientation, and if the coefficient of $R$ is zero, the constant term determines the orientation. The latter is the case if the non-standard point is the endpoint of a line parallel to the line through $p_1$ and $p_2$. Note that the result is stable as long as $R$ is a large enough number.

**Corollary 3.1:** The values of the predicates *compare_xy*, *orientation*, and *side_of_circle* are well defined for endpoints of segments, rays and lines.

*Proof.* Predicates *compare_xy*, *orientation*, and *side_of_circle* are defined as signs of polynomials in point coordinates and the coordinates of endpoints of segments, rays, and lines are linear polynomials in $R$, our infimaximal. $\square$

## 3.4 Extended Segments

We introduce extended segments as a unified view of segments, rays, and lines. Recall that our goal is to extend programs written for segments to rays and lines. Thus we need a unified view of segments, rays, and lines.

A segment is defined by a pair of (standard) points. An extended segment is defined by a pair of extended points. We have to be a bit more careful. We cannot give meaning to every pair of extended points, but only to those pairs which correspond to a segment, ray, or line.

**Definition 2 (extended segment):** A pair of distinct extended points $(p_1, p_2)$ defines an *extended segment (esegment)* if one of the following conditions holds:

1. both points are standard points.
2. both points are non-standard and are the endpoints of a common line.
3. one point is standard and one is non-standard and they are the endpoints of a common ray.
4. both points are non-standard points and lie on the same frame box segment.

Extended segments defined by items 1) to 3) are called standard and extended segments defined by item 4) are called non-standard. Standard esegments correspond to objects of affine geometry — affine segments, rays and lines, non-standard segments do not. For every fixed value of $R$, a non-standard segment corresponds to a well-defined geometric object.

Taking any two extended points to form an extended segment causes the following problem. Consider two extended points at the tips of two rays that have antipodal direction but where the underlying lines are not identical. Then we cannot give any reasonable meaning to the affine combination of two such points. Such an object covers a whole stripe of points in between the two lines containing the rays depending on the size of $R$. Therefore we loose the one-dimensional character of segments, rays and lines.

### 3.5 Intersections of Extended Segments

An extended segment represents any one of the following four possibilities: a standard segment, a ray, a line, or a segment on one of the frame boundaries. We define the intersection point of two esegments as follows. If for every fixed sufficiently large value of $R$, the corresponding geometric objects intersect in a single point, this point is the point of intersection. Otherwise, the intersection is undefined. Observe that if the intersection lies on the frame for every sufficiently large value of $R$, the intersection is indeed one of our frame points and hence this definition makes sense.

We next show that the standard analytical methods for handling intersections of segments apply to extended segments.

Consider two non-trivial segments $s_1 = (p_1, q_1)$ and $s_2 = (p_2, q_2)$ where $p_1$, $p_2$, $q_1$, $q_2$ are affine points and $\ell_1$ and $\ell_2$ are their underlying lines. The segments intersect in a single point if and only if [2] the endpoints of $s_i$ do not lie on the same side[3] of $\ell_{3-i}$ for $i = 1,2$. Thus, the test whether two segments intersect amounts to four evaluations of the orientation predicate. We have already argued that the orientation predicate works for extended points and hence the test whether two segments intersect does so.

Consider next the computation of the intersection point $p$ of $s_1$ and $s_2$. The coordinates of $p$ are rational functions $r_x$ and $r_y$ of the coordinates of $p_1$, $p_2$, $q_1$, and $q_2$. Rational expressions $E_x$ and $E_y$ in the coordinates of $p_1$ to $q_2$ representing functions $r_1$ and $r_2$ are well known and easily obtained. Simply derive the line equations for $\ell_1$ and $\ell_2$ and then solve a linear system to obtain $E_x$ and $E_y$[4]:

$$\ell_i \equiv \quad a_i x + b_i y + c_i = 0$$

where

$$a_i = y_{pi} - y_{qi},\ b_i = x_{qi} - x_{pi},\ c_i = x_{pi} y_{qi} - x_{qi} y_{pi}$$

Then the point of intersection $p$ is defined by the expressions:

$$E_x = (b_1 c_2 - b_2 c_1)/(a_1 b_2 - a_2 b_1),\ E_y = (a_2 c_1 - a_1 c_2)/(a_1 b_2 - a_2 b_1)$$

What is the situation for two extended segments? The intersection point is an extended point and for every fixed value of $R$, let $r_x(R)$ and $r_y(R)$ be the rational expressions in $R$ that form the coordinates of the intersection point. If the intersection point is a standard point, $r_{x,y}(R)$ does not depend on $R$ — $R$ can be eliminated from the expression as such a point is purely affine — and if the intersection point is non-standard[5], one of the functions $r_{x,y}(R)$ is the identity function and the other has absolute value at

[2] For simplicity, we are ignoring the possibility that the underlying lines are identical and the two segments share an endpoint. The discussion is easily extended to also handle this situation.

[3] An oriented line has three sides: left, on, and right.

[4] Note that the rational expressions are not unique. One can easily expand the quotient by an arbitrary factor.

[5] intersect a line or ray with the non-standard esegment that contains an endpoint.

most $R$ for sufficiently large $R$ — simply because we consider the intersection with an axis-orthogonal box of equal side-length $2R$.

We may also apply the rational expressions $E_x$ to $E_y$ to the coordinates of the endpoints of the esegments $p_1$ to $q_2$. By inserting the coordinate polynomials of the extended points, we obtain representations for rational functions in $R$, our infinitesimal. For every fixed value of $R$, we have $r_{x,y}(R) = E_{x,y}(R)$ and hence the rational functions must simplify to the canonical representation of non-standard points. We have thus shown:

**Lemma 3.2:** Let *intersection*$(p1, p2, q1, q2)$ be the partial function that returns the coordinates of the intersection point of the segments $s(p_1, p_2)$ and $s(q_1, q_2)$. Then *intersection* is correct when applied to extended segments.

# 4 Implementation

We implemented extended points in C++ and used them together with CGAL and LEDA. We report about the use in Section 5.1. Our implementation went through three versions.

In the first version, we used the representation alluded to at the end of Section 3.2. An extended point was represented by a reference to the underlying geometric object plus an indicator which selects a frame point. Predicates and geometric constructions used case switching based on the representation. We soon realized that this approach is too cumbersome and that the complicated control structure of our predicates makes it difficult to ensure correctness.

Versions two and three use the coordinate representation based on arithmetic in a polynomial ring. We implemented a number type *RPolynomial* modeling $\mathbb{Z}[R]$ the ring of polynomials in a variable $R$. Our type offers the ring operations $+$, $-$, $*$, polynomial division and the gcd operation, as described in [Coh93, Knu98], and a sign function. The sign function returns the sign of the highest non-zero coefficient.

We obtained Extended points and segments by combining our number type *RPolynomial* with the geometry kernel of CGAL. The geometry kernels of CGAL are parameterized by an arithmetic type. Objects may either be represented by their Cartesian or their homogeneous coordinates. For an introduction of homogeneous coordinates and their advantages please consult the description of the rational geometry kernel of LEDA[MN99].

We use the homogeneous kernel as it only requires a ring type (the Cartesian kernel requires a number type that is a field). We instantiated two dimensional homogeneous points with our number type *RPolynomial*, and added some additional construction code for points from standard points and oriented lines. No work was required for the geometric predicates as predicate evaluation amounts to sign computation of arithmetic expressions and we defined the sign function of our ring type according to Section 3.3.

A slight modification was required for the intersection code. The CGAL kernel does not automatically cancel common factors in the representation of points, i.e., it is not guaranteed that the gcd of the homogeneous coordinates of a point is equal to one[6]. For our situation this implied that point representations could contain redundant polynomial factors and hence non-linear polynomials. Correctness was not impaired, but running times were miserable. We remedied the situation by insisting that representations are always in their reduced form, i.e., whenever a point is constructed the gcd of the homogeneous coordinates is computed and a common factor is canceled. This ensures that the polynomials in point representations stay linear as argued in Section 3.5.

[6] It cannot do so since the notion of gcd does not make sense for every ring type.

Our second implementation has the strong appeal of very modular programming and thereby its correctness was very simple to achieve. We have used it heavily as a backup checker for the more elaborate techniques used in our third version.

In the third version we optimized the representation of points and the evaluation of predicates. This forced us to write our own classes *epoint* and *esegment* and to write code for the evaluation of predicates. When we store points we exploit that the normalizing coordinate is an integer (and never a polynomial of degree one). In order to optimize the evaluation of predicates, we derived closed form expressions for the polynomials arising in the predicates and incorporated filter technology. For example the *orientation predicate* of three extended points in homogeneous representation $p_i = (m_{xi}R + n_{xi}, m_{yi}R + n_{yi}, w_i)$ $i = 1,2,3$, amounts to computing the sign of a quadratic polynomial $A \cdot R^2 + B \cdot R + C$ in our infimaximal $R$, where

$$\begin{aligned}
A &= (m_{x1}\,w_3\,m_{y2} - m_{x1}\,w_2\,m_{y3} + m_{x3}\,w_2\,m_{y1} - m_{x2}\,w_3\,m_{y1} - m_{x3}\,w_1\,m_{y2} + m_{x2}\,w_1\,m_{y3}) \\
B &= (m_{x1}\,w_3\,n_{y2} - m_{x1}\,w_2\,n_{y3} + n_{x1}\,w_3\,m_{y2} - m_{x2}\,w_3\,n_{y1} - n_{x1}\,w_2\,m_{y3} + m_{x2}\,w_1\,n_{y3} - \\
&\quad n_{x2}\,w_3\,m_{y1} + m_{x3}\,w_2\,n_{y1} + n_{x2}\,w_1\,m_{y3} - m_{x3}\,w_1\,n_{y2} + n_{y3}\,w_2\,m_{y1} - n_{y3}\,w_1\,m_{y2}) \\
C &= n_{x2}\,w_3\,n_{y1} + n_{x1}\,w_3\,n_{y2} + n_{x2}\,w_1\,n_{y3} - n_{x1}\,w_2\,n_{y3} + n_{y3}\,w_2\,n_{y1} - n_{y3}\,w_1\,n_{y2}
\end{aligned}$$

We use a two stage filtering scheme to determine the sign of the coefficients. We first evaluate the coefficients using interval arithmetic (CGAL number type *Interval_nt*). If interval arithmetic is not able to determine the sign, our code resorts to multi-precision integer arithmetic. The gain in runtime can be seen in section 5.2.

# 5 Discussion

We come to the evaluation of our results. We discuss functionality in Section 5.1 and efficiency in Section 5.2.

## 5.1 Applications

We describe two applications, the first one being used in our second.

The first application is the computation of arrangements of segments, rays, and lines. We use the sweep algorithm of LEDA [MN99] together with epoints and esegments. Only minimal changes of code were required.

The second application is an implementation of planar Nef polyhedra; in fact, this application made us think about the problem of making rays and lines look like segments. A planar Nef polyhedron is any point set that can be obtained from the open halfspaces by a finite number of set complement and set intersection operations. The set of Nef polyhedra is closed under the Boolean set operations and under the topological operations boundary, closure, and interior. Figure 2 shows two Nef polyhedra enclosed into a frame. We view Nef polyhedra as embedded into a infimaximal frame. This makes all faces (except for the outside of the frame) bounded and allows us to represent planar Nef polyhedra by the attributed plane maps[7] of LEDA. The position of a vertex is given by an epoint and all edges of a Nef polyhedron correspond to esegments. All faces have circular closed face cycles.

[7] An plane map is a bidirected graph in which the edges incident to each node are cyclically ordered [MN99, Section 8.4]. Each node, edge and face carries a mark bit indicating whether it is contained in the Nef polyhedron or not.

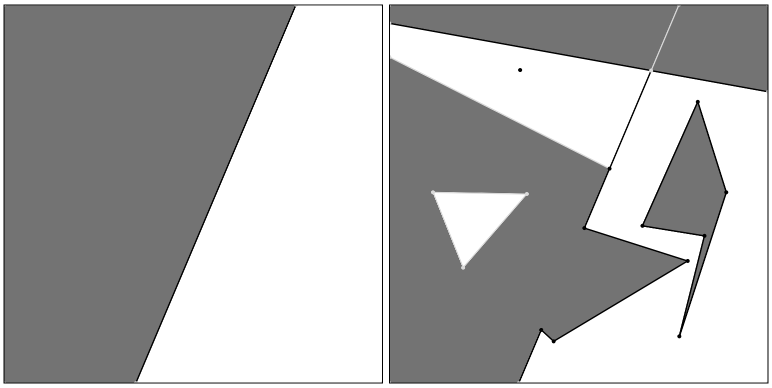

Figure 2: The left part shows a halfspace pruned in the frame. The right part shows a complicated Nef polyhedron consisting of diverse faces and low dimensional features (a ray taken from a face and an isolated vertex). All vertices are embedded via extended points. All points on the square boundary are non-standard points.

We implemented an overlay engine for plane maps that works for both the bounded affine scenario as well as the unbounded Nef structure. The engine is based on the generic sweep algorithm mentioned above; instantiating it with the an affine geometry kernel (e.g. LEDA's or CGAL's) makes it work for bounded maps and instantiating it with extended points and segments makes it work for Nef polyhedra.

Binary operations on two Nef polyhedra $N_1$, $N_2$ are realized in three phases: we first calculate the overlay of $N_1$ and $N_2$ and transfer the marks of the input objects to the overlay objects; we then use a binary selection predicate on the marks, and we finally simplify the obtained structure towards a minimal representation. The third step is as described in [RO90] or [MN99, Section 10.8].

The details of the implementation of Nef polyhedra can be found in [Pla].

### 5.2 Efficiency

We present runtime results. We performed the following experiments. Starting from $n$ random halfspaces (the coefficients of the boundary line are random integers in $[0,n]$ and one of the halfspaces is chosen at random) we built a Nef polyhedron by symmetric difference operations. We put the halfspaces at the leaves of a balanced binary tree and formed the symmetric difference of the two children at each internal node. In the root we obtained a Nef polyhedron $P$ of complexity $\Theta(n^2)$. The polyhedron is essentially the arrangement defined by the boundary lines; a vertex, edge, face of the arrangement belongs to $P$ iff it is contained in an even number of the halfspaces[8]. We then took two Nef polyhedra obtained in this way and formed their *intersection* and *union*. We performed this experiment for different values of $n$ ranging from 10 to 200. Table 1 shows the results. The line marked $\oplus$ presents the complexity and times for the recursive synthesis of the Nef polyhedron. The columns labeled #V, #E, and #F give the actual number of nodes, edges and faces of the resulting arrangement (deviation from the formulas above are due to degeneracies). The lines labeled $\cap$ and $\cup$ present the results of the corresponding binary operation.

We show the running times for four different implementations. The first two columns show the running times of our second and third implementation of epoints and esegments. The column (marked

[8] The exact number (assuming non-degeneracy and taking the objects on the frame into account) of vertices, edges, and faces is $n(n-1)/2+2n+4$, $n(n+1)+2n+4$, and $n(n+1)/2+2$, respectively.

| #lines | op | #V | #E | #F | naive | filtered | rgpn | rgp |
|---|---|---|---|---|---|---|---|---|
| 10 | $\oplus$ | 67 | 121 | 56 | 0.64 | 0.065 | 0.075 | 0.125 |
| 10 | $\cap$ | 162 | 217 | 61 | 0.74 | 0.09 | 0.08 | 0.93 |
| 10 | $\cup$ | 160 | 213 | 60 | 0.75 | 0.09 | 0.07 | 0.92 |
| 20 | $\oplus$ | 230 | 437 | 209 | 2.31 | 0.26 | 0.285 | 2.3 |
| 20 | $\cap$ | 622 | 831 | 241 | 2.89 | 0.36 | 0.3 | 19.2 |
| 20 | $\cup$ | 577 | 744 | 205 | 2.93 | 0.36 | 0.3 | 19.6 |
| 30 | $\oplus$ | 499 | 964 | 467 | 4.01 | 0.48 | 0.575 | -1 |
| 30 | $\cap$ | 1375 | 1821 | 514 | 7 | 0.92 | 0.8 | -1 |
| 30 | $\cup$ | 1410 | 1894 | 540 | 7.55 | 0.9 | 0.76 | -1 |
| 40 | $\oplus$ | 862 | 1680 | 820 | 8.7 | 1.05 | 1.25 | -1 |
| 40 | $\cap$ | 2395 | 3171 | 900 | 12.8 | 1.67 | 1.43 | -1 |
| 40 | $\cup$ | 2509 | 3370 | 960 | 12.9 | 1.71 | 1.42 | -1 |
| 50 | $\oplus$ | 1326 | 2600 | 1275 | 12.1 | 1.54 | 1.83 | -1 |
| 50 | $\cap$ | 3828 | 5111 | 1470 | 21.1 | 2.88 | 2.41 | -1 |
| 50 | $\cup$ | 3809 | 5073 | 1455 | 20.7 | 2.85 | 2.35 | -1 |
| 100 | $\oplus$ | 5149 | 10195 | 5048 | 49.8 | 6.76 | 8.29 | -1 |
| 100 | $\cap$ | 15188 | 20296 | 5826 | 92.6 | 13.5 | 13.1 | -1 |
| 100 | $\cup$ | 15088 | 20073 | 5731 | 92.4 | 13.6 | 13.4 | -1 |
| 150 | $\oplus$ | 11476 | 22799 | 11325 | 146 | 21.2 | 24.2 | -1 |
| 150 | $\cap$ | 34223 | 45801 | 13214 | 217 | 33.7 | 39.8 | -1 |
| 150 | $\cup$ | 33717 | 44785 | 12881 | 217 | 34.1 | 40.2 | -1 |
| 200 | $\oplus$ | 20302 | 40401 | 20100 | 207 | 33.8 | 40 | -1 |
| 200 | $\cap$ | 60043 | 79905 | 22909 | 415 | 66 | 101 | -1 |
| 200 | $\cup$ | 60551 | 80886 | 23352 | 410 | 67.1 | 100 | -1 |

Table 1: Running times

as *naive*) uses CGAL's homogeneous kernel instantiated with the ring type *RPolynomial* which in turn uses LEDA's multi-precision integer arithmetic. Column *filtered* shows the running times for version three which uses filtered predicates as described in Section 4. A comparison of the first two columns shows that our third implementation is much superior to the second.

We also wanted a comparison with the approach based on a concrete frame. LEDA offers a generic polygon type *rat_gen_polygon* which is closed under regularized boolean operations (a regularized boolean operation discards isolated lower dimensional features by replacing the true result of a boolean operation by the closure of the interior of the result) and allows only a single unbounded face. Regularized boolean operations lead to a simpler topological structure than general boolean operations. We enclosed our halfspaces into a concrete geometric frame (whose size we inferred from the computation of the Nef polygon) and then executed the same set of operation. *Rat_gen_polygons* also use LEDA's sweep for the overlay of two maps. The last column (labeled *rpg*) shows the running times of *rat_gen_polygons*. A dash indicates that the experiment was not run due to the excessively large running times. The excessively bad behavior of *rat_gen_polygon* surprised us. We traced it to the fact that LEDA does not normalize point representations automatically. Normalization means cancelling out common factor of nominators and denominators in the coordinate representation of points. We added a normalization step after each binary operation. The resulting running times are shown in column *rpgn*.

A comparison of columns *filtered* and *rpg* shows that the implementation described in this paper is slightly faster than LEDA's *rat_gen_polygons* for the synthesis step and slightly slower for the union and intersection. For $n = 200$ we are faster for all three operations. Our explanation is as follows (we admit that we are not completely sure whether our explanation is the right one): the use of a concrete geometric frame forces us to use large coordinate values for the frame points. The coefficients of our polynomials are smaller (much smaller in the early synthesis steps). For $n = 200$ the coordinate values in the geometric frame become so large that the filter in LEDA's rational geometry kernel start to loose its effectiveness. The filter in epoints stays effective.

It is also interesting to compare columns *naive* and *rpg*. For $n = 10$, *rpg* is superior, for $n = 20$, the two implementations are about the same, and for larger $n$, *naive* wins by a large margin due to the fact that we reduce the polynomial coordinate representation of naive epoints on construction by a polynomial division. Remember that the *rpg* code does not use any reduction.

# 6 Conclusion

We have shown that our approach to a dynamic frame offers one solution to the problem of non-compact structures. Its main advantage is the transparent handling of geometric configurations that appear in the standard affine plane but also on the frame when introducing extended points. Available program code for standard affine problems can be easily used as long as the used geometric predicates are extensible to extended points and extended segments. The application of our framework separates the geometric issues of the frame addition from the control structure of the implemented algorithm. On the other side our runtime results show that infimaximal frames can be used without introducing a relevant runtime overhead compared to standard affine geometry.

# References


[BFS98] C. Burnikel, S. Funke, and M. Seel. Exact arithmetic using cascaded computation. In *Proceedings of the 14th Annual Symposium on Computational Geometry (SCG'98)*, pages 175–183, 1998.

[Bie95] H. Bieri. Nef Polyhedra: A Brief Introduction. *Computing Suppl. Springer-Verlag*, 10:43–60, 1995.

[CGA] CGAL (Computational Geometry Algorithms Library). `www.cgal.org`.

[Coh93] H. Cohen. *A Course in Computational Algebraic Number Theory*, volume Graduate Texts in Mathematics 138. Springer Verlag, New York Berlin Heidelberg, 1993.

[Cox87] H.S.M. Coxeter. *Projective geometry*. Springer, 2nd ed. 1974. corr. 2nd print. 1994 edition, 1987.

[dBvKOS97] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry : Algorithms and Applications*. Springer, Berlin, 1997.

[EM90] H. Edelsbrunner and E.P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, January 1990.

[FvW96] S. Fortune and C. van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Transactions on Graphics*, 15:223–248, 1996. preliminary version in ACM Conference on Computational Geometry 1993.

[KLN91] M. Karasick, D. Lieber, and L.R. Nackman. Efficient Delaunay triangulation using rational arithmetic. *ACM Transactions on Graphics*, 10(1):71–91, January 1991.

[Knu98] D.E. Knuth. *The Art of Computer Programming*, volume Volume 2, Seminumerical Algorithms, 3rd edition. Addison-Wesley, 1998.

[LED] LEDA (Library of Efficient Data Types and Algorithms). `www.mpi-sb.mpg.de/LEDA/leda.html`.

[MN94] K. Mehlhorn and S. Näher. The implementation of geometric algorithms. In *Proceedings of the 13th IFIP World Computer Congress*, volume 1, pages 223–231. Elsevier Science B.V. North-Holland, Amsterdam, 1994.

[MN99] K. Mehlhorn and S. Näher. *LEDA, A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[Nef78] W. Nef. *Beiträge zur Theorie der Polyeder mit Anwendungen in der Computergraphik.* Herbert Lang & Cie AG, Bern, 1978.

[OTU87] T. Ottmann, G. Thiemt, and C. Ullrich. Numerical stability of geometric algorithms. In Derick Wood, editor, *Proceedings of the 3rd Annual Symposium on Computational Geometry (SCG '87)*, pages 119–125, Waterloo, ON, Canada, June 1987. ACM Press.

[Pla] Planar nef polyhedron specification. `www.mpi-sb.mpg.de/~seel/Nef\_polyhedra/`.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry : An Introduction*. Springer, 1985.

[RO90] J.R. Rossignac and M.A. O'Connor. SGC: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In M. J. Wozny, J. U. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 145–180. Elsevier Science Publishers B.V., North Holland, 1990.

[Sch] S. Schirra. Robustness and precision issues in geometric computation. to appear, preliminary version available as MPI report.

[Sto91] J. Stolfi. *Oriented projective geometry : a framework for geometric computations*. Academic Press, 1991.

[Yap93] C.K. Yap. Towards exact geometric computation. In *Proceedings of the 5th Canadian Conference on Computational Geometry (CCCG'93)*, pages 405–419, 1993.

[YD95] C.K. Yap and T. Dubé. The exact computation paradigm. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 452–492. World Scientific, Singapore, 2nd edition, 1995.