Reliable Distance and Intersection Computation Using Finite Precision Geometry

Katja Bühler[1], Eva Dyllong[2], and Wolfram Luther[2]

[1] VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH
Donau-City-Strasse 1, A-1220 Wien, Austria
buehler@vrvis.at
http://www.vrvis.at
[2] University of Duisburg-Essen, Department of Computer Science
Lotharstr. 65, D-47048 Duisburg, Germany
{dyllong, luther}@informatik.uni-duisburg.de
http://www.informatik.uni-duisburg.de

Abstract. In this paper we discuss reliable methods in the field of finite precision geometry. We begin with a brief survey of geometric computing and approaches generally used in dealing with accuracy and robustness problems in finite precision geometry. Moreover, two reliable geometric algorithms based on these approaches are presented. The first one is a new distance algorithm for objects modeled in a common octree. The results are exact and include good bounds on all subdivision levels. Using smoother enclosures on the highest level, a link is provided to well-known algorithms for convex and non-convex objects.
We discuss the general concept and advantages of special bounding volumes with representations directly connected to the representation of the enclosed object: Implicit and parametric Linear Interval Estimations (I)LIEs are roughly speaking, just thick planes enclosing the object. They are constructed using Taylor models or affine arithmetic. The particular structure of (I)LIEs allows the construction of effective hierarchies of bounding volumes and the development of effective intersection tests for the enclosed object with rays, boxes and other LIEs. In addition, a fast reliable intersection test for two LIEs is presented in detail.

1 Introduction

Geometric algorithms are widely used in robotics, computer graphics, computer aided design or any simulations of a virtual environment. Common representations for objects are constructive solid geometry models (CSG-models), boundary representation models (B-Rep-models) or tessellations (e.g. octrees). Single surfaces or surface patches are mostly represented in parametric or implicit form, or as subdivision surfaces. The choice of the appropriate representation is dependent on the application.

Because exact modeling of an object is very time consuming and can be carried out only in certain special cases, polyhedral structures are recommended for path planning in robotics. Octrees are often used for scene reconstruction

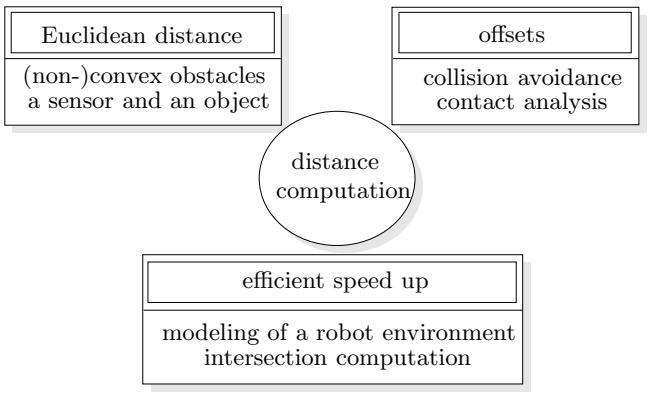

Fig. 1. Applications of distance computation.

from sensor data. Parametric surfaces are an important tool for objects which are located near the robot. In the field of contact analysis and path planning, efficient distance and intersection algorithms play a decisive rule in most simulations.

Distance algorithms are most frequently used in robotics (see Figure 1) and also in computer games not only to determine the distance between two obstacles in the environment of a robot or between a sensor point and an object, but also to obtain the results of difficult geometric comparisons without actually doing them. If we know that two surfaces are too far apart to intersect, we do not need the more expensive intersection calculations. Here bounding volumes are a common technique, which relies on a hierarchical model representation of the two surfaces using axis-aligned bounding boxes (AABBs), oriented bounding boxes (OBBs), parallelepipeds, discrete-orientation polytopes (DOPs), spheres, or new concepts of parameterized bounding volumes such as Linear Interval Estimations (LIEs) [7] or Implicit Linear Interval Estimations (ILIEs) [8]. Hierarchies of bounding volumes provide a fast way to perform collision detection even between complex models. The determination of the offset to a surface is another example of a problem which can be formulated in terms of distance computation. Hierarchical algorithms are also applied in computer graphics to perform point- or box-surface incidence tests and ray-surface or surface-surface intersections. Here, it is of interest not only to test whether an intersection exists, but also to compute the (exact) intersection set. Some applications for such algorithms are, for instance, the rendering of implicit and parametric surfaces, the voxelization of implicit objects, the computation of surface-surface intersections, and visibility computations.

The methods mentioned here represent only a small selection of the geometric algorithms and structures commonly applied in the field of object modeling, contact analysis and path planning.

Usually, they are sophisticated algorithms designed and proven to be correct for objects defined over the domain of real numbers which can only be approxi-

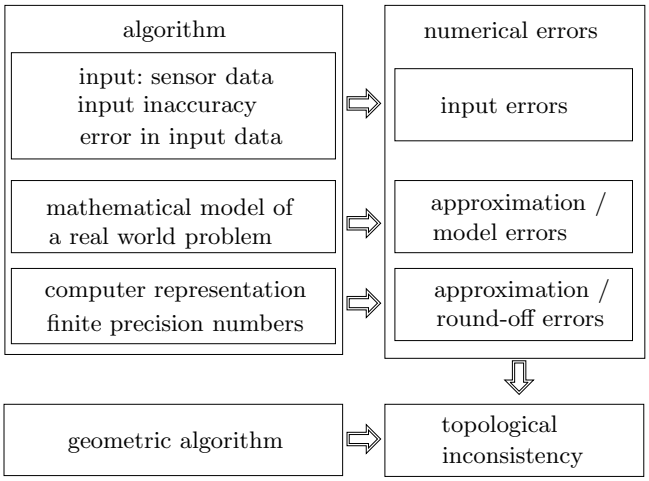

Fig. 2. Accuracy and robustness problems.

mated on the computer. Due to rounding errors many implementations of geometric algorithms simply compute the wrong results for input values for which they are supposed to work. Numerical non-robustness in scientific computing is a well-known and widespread phenomenon. The implementation of an algorithm is in general considered robust if its output is always the correct response to some perturbation of the input, and stable if the perturbation is small.

Although non-robustness is already an issue in a purely numerical computation, it is more intractable in a geometric one. To appreciate why the robustness problem is especially hard for geometric computation, we need to understand what makes a computation geometric. Geometric computation involves not only numerical computations but also combinatorial structures as well as certain non-trivial consistency conditions between the numerical and combinatorial data. Consequently, in purely numerical computations a result becomes unusable when there is a severe loss of precision. In geometric computations errors become serious when the computed result leads to inconsistent states of the program or is qualitatively different from the true result, e.g. combinatorial structure is wrong. Accordingly, a loss of robustness related to geometric algorithms must always be understood in both its numerical and its topological meanings (see Figure 2).

Researchers trying to create robust geometric software use one of two approaches. The first is some form of exact computation in which every numerical quality is computed exactly (explicitly, if possible) and which relies on big number packages and use filters to make this approach viable. Alternatively, they can continue to use floating-point or some other finite precision arithmetic, and try to make their computation robust.

Although exact computation is a safe method of achieving robustness, it is somewhat inefficient for most robotic applications. Exact geometric compu-

tation requires that every evaluation is correct, which can be achieved either by computing every numeric value exactly (e.g. using exact integer or rational arithmetic) or by employing some implicit or symbolic representation that allows values to be computed exactly. But an exact computation is only possible whenever all numeric values are algebraic or if the result of the geometric algorithm depends only on the signs of some quantities to be known (such information's can be obtained with adaptive methods). Furthermore, the cost of an arithmetical operation is no longer constant, as in the case of floating-point arithmetic, but depends upon its context and increases due to geometric constructions in which a new geometric structure is produced from an old one. Because of this perceived performance cost, the exact geometric computation does not appear to be widely used in robotics. Besides, in most robotic applications the input data are arbitrary real numbers (e.g. sensor data) which have to be cleaned up into exact values (e.g. an inexact input point can be viewed as the center of a small ball) before being fed to the exact algorithm.
On the other hand, the common alternative to exact computation, finite precision geometry, is faster, readily available, and widely used in practice; however exactness and robustness are no longer guaranteed. Here, correct and verifiable geometric reasoning using finite precision arithmetic is demanded.

This paper aims to present new methods for the design of accurate and robust finite precision geometric algorithms which yield reliable results despite rounding errors caused by the limited precision of the computation. It begins with a short overview of the most common reliable techniques in the field of finite precision geometry: interval arithmetic or affine arithmetic, approaches which reduce the effect of overestimation caused by interval evaluations, Taylor models, and the exact scalar product.
Section 3 proposes a new algorithm for distance computation between octrees based on the use of the exact scalar product. Another center of interest in this section is the development of efficient and accurate algorithms for distance calculation between a sensor point fixed on a robot and a target or obstacle (or obstacles) in a complex environment. An accurate distance algorithm for convex and non-convex polyhedra with a priori error bounds of the computed values is provided. Robust solutions to these geometric problems are used in collision-free path planning if a given end-effector is moving amid a collection of (un)known obstacles from an initial to a desired final position as well as in dealing with the resulting contact problems. The advantages of the special structure of (implicit) linear interval estimations computed using Taylor models and affine arithmetic are demonstrated in Section 4, followed by a detailed discussion of robust intersection and enumeration algorithms for implicit and parametric surfaces based on spatial subdivision. Finally, Section 5 summarizes the results.

2 Handling of Robustness Problems

Because there is no general theory on how to deal with them, the handling of robustness problems in finite precision geometry takes a number of different

approaches. In order to avoid inconsistent decisions these fall into two categories. The first places higher priority on topological and combinatorial data, while the second emphasizes numerical data.

The topology-oriented approach leads to robust algorithms which never crash and compute output with essential combinatorial properties, but the computed numerical values do not necessarily correspond to the real solution of the geometric problem being addressed. Typically a topology-oriented algorithm does not treat sign computations producing sign zero. In those cases where the numerical value of a sign computation is zero, it will be replaced by a positive or negative value, whichever is consistent with the current topology. For this reason the topology-oriented approach is not suitable for certain computations, such as determining the real distance points between two objects.

In such cases numerical approaches are more appropriate. Their typical strategies are based on an association of tolerances to geometric objects in order to represent uncertainties. The representation of a value by an approximation and an error bound or an interval is a numerical analogue of these strategies. In this context the term *interval geometry* can also be found [33].

2.1 Interval Arithmetic

Approximation and error bounds define an interval that contains an exact value. In interval arithmetic the real numbers are stored as intervals with floating-point endpoints. Computations on the numbers are performed as sets of computations on the interval bounds, e.g. $[a, b] + [c, d] = [a + c, b + d]$. Interval arithmetic is the most common technique providing reliable solutions for many numerical problems. Unfortunately, overestimation resulting from standard interval evaluations is an often criticized drawback of interval arithmetic. See Alefeld and Herzberger [1] for further reading.

2.2 Epsilon Geometry

Another method closely related to interval arithmetic is epsilon geometry, which was defined by Guibas, Salesin and Stolfi [21] and uses an epsilon predicate instead of a Boolean value to obtain information on how much the input satisfies the predicate. An epsilon predicate returns an interval that identifies a region over which the predicate is definitely true, definitely false or simply uncertain. So far, epsilon geometry has been applied only to a few basic geometric predicates. Moreover, it is not clear how to handle the regions of uncertainty.

2.3 Affine Arithmetic

Affine arithmetic, first proposed by Comba and Stolfi [11], is an extension to interval arithmetic which reduces the effect of overestimation by taking into account the dependencies of the uncertainty factors of input data, approximation and rounding errors. In this way, error expansion can often be avoided and tighter

bounds on the computed quantities achieved.
When using this approach, each numerical number is stored as an affine form

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \ldots + x_n\varepsilon_n, \tag{1}$$

where $\epsilon_i \in [-1, 1]$ denotes a *noise symbol* representing one source of error or uncertainty. x_0 is the *central value* of the affine form and the x_i are *partial deviations*. For each new source of error a new noise symbol ϵ_i is introduced and added to the affine form.

Each interval can be expressed as an affine form, but an affine form can only be approximated by an interval as it carries much more information. An interval describes only the general uncertainty of the data, whereas affine arithmetic splits this uncertainty into specific parts. Thus, a conversion from affine forms to intervals in most cases implies a loss of information.

Let $\hat{x} := x_0 + x_1\epsilon_1 + x_2\epsilon_2 + ... + x_n\epsilon_n$ be the affine form of the fuzzy quantity x. x lies in the interval

$$[\hat{x}] := [x_0 - \xi, x_0 + \xi]; \qquad \xi := \sum_{i=1}^{n} |x_i|$$

$[\hat{x}]$ is the smallest interval enclosing all possible values of x.

Let $X = [a, b]$ be an interval representing the value x. Then x can be represented as the affine form

$$\hat{x} = x_0 + x_k\epsilon_k$$

with $x_0 := (b + a)/2;\ x_k := (b - a)/2$.

Affine arithmetic is slower than standard interval arithmetic, but in cases where there might be error correlation from one computation step to the next, this approach is beneficial.

2.4 Arithmetical approaches

Certain approaches might be described as being based primarily on arithmetical - as opposed to geometric - considerations. A highly precise evaluation of arithmetical expressions provides a solid tool for the solution of various geometric problems. The idea of arithmetical approaches is to isolate the basic operations (primitives) which have to be handled in a numerically correct way, where the manner in which the respective operands are represented is crucial. The primitives have to be implemented in such a way that they yield a result which is as close as it can be to the best possible machine representation. The computational depth of geometric algorithms has to be kept low to control the propagation of round-off errors.

Since scalar products occur frequently and are important basic operations in many geometric computations, it is advantageous to perform the scalar product calculation with the same precision as the basic arithmetical operations. Using the exact scalar product delays the onset of qualitative errors and improves the robustness of the implementation. Other arithmetical approaches, like the permutation of operations combined with random rounding (up and down), can also be used [33].

2.5 Taylor models

The idea of this approach is the representation of a (multivariate) function as a Taylor polynomial plus an interval that encloses the range of the remainder: the *Taylor model* of the function.

Definition 1. *Let be* $f \in C^{n+1}(D)$; $D \subset \mathbb{R}^m$ *and* $\mathbb{B} \in I\mathbb{R}^m$ *an interval box with* $\mathbb{B} \subset D$. *Let* T *be the Taylor polynomial of order* n *of* f *around the point* $\boldsymbol{x}_0 \in \mathbb{B}$.

- *An interval* I *with* $\forall \boldsymbol{x} \in \mathbb{B} : \; f(\boldsymbol{x}) - T(\boldsymbol{x}) \in I$ *is called an* n-th order Remainder Bound of f on $\mathbb{B}$.
- *A pair* (T, I) *is called an* n-th order Taylor model of f.
- *A set of all remainder bounds is called the* Remainder Family, *the optimal enclosure of the remainder is called the* Optimal Remainder Bound.

Thus, a Taylor model is a polynomial of n-th order enclosing the approximated function on the interval box $\mathbb{B}$.

Berz and Hofstätter [5] define an arithmetic for Taylor models based on uni- and bivariate arithmetical operators and basic functions. It turns out that these methods are similar to interval arithmetic for the case $n = 0$.

Taylor models have a remarkable feature with respect to the quality of the approximation and its convergence: If $\mathbb{B}$ decreases, I will decrease in size as the $(n+1)$-st power of the size of the box $\mathbb{B}$.

3 Accurate Distance Algorithms

Obstacles are often modeled or reconstructed from sonar and visual data leading to uncertain information. Descriptions based on polyhedral or hierarchical octree structures lead to a considerable reduction of data, which makes effective storing and processing possible. First, we will deal with objects represented by an octree in three dimensions and then with a more general n-tree in higher dimensions.

Octrees are very suitable for building environments where obstacles must be taken into account when considering collision-free path planning as they enable the location of free and occupied regions based on accurate distance calculations.

In Figure 3 a non-convex object is represented by an axis-aligned, level-three octree. The round nodes are gray because they have white and black leaves. Since the octree is constructed through the subsequent division of boxes, all constructed nodes are boxes whose boundary representations can be computed using an appropriate fixed-point arithmetic.

3.1 An Accurate Distance Algorithm for Octrees

The distance calculation between two objects represented by a common octree which has depth N and extra color information in gray nodes is based on a simple computation of the distance between two boxes.
First, we establish a procedure $dist^2(Q_1, Q_2)$ for the rectilinear axis-aligned

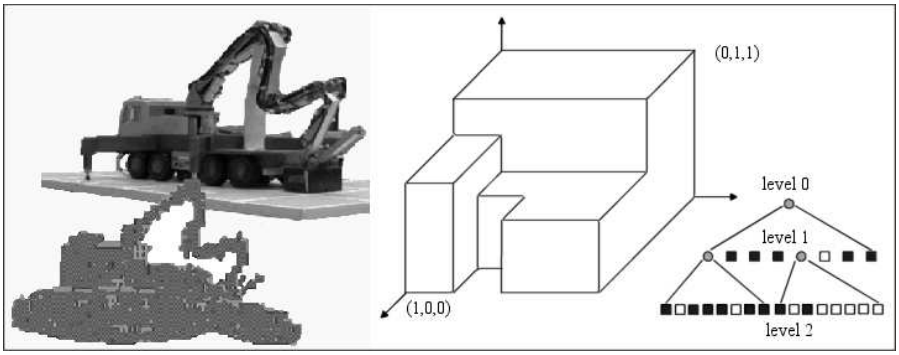

Fig. 3. Octree representing a non-convex object.

boxes Q_1, Q_2 described by a vertex point with the smallest coordinates and the length of three edges:

$$Q_1 : [X_1, X_2, X_3, h_1, h_2, h_3] = I_1 \times I_2 \times I_3$$
$$Q_2 : [Y_1, Y_2, Y_3, k_1, k_2, k_3] = J_1 \times J_2 \times J_3$$

We introduce a case-selector determining by where the first box lies with respect to the other (outside below or above, cutting):

$$c_n := \begin{cases} (Y_n - X_n - h_n)^2, Y_n > X_n + h_n \\ (X_n - Y_n - k_n)^2, X_n > Y_n + k_n \\ 0, \text{ otherwise} \end{cases}, \quad n = 1, 2, 3.$$

The following cases appear (including also the other cases surface to vertex etc.):

- Intersection:

$$I_1 \cap J_1 \neq \emptyset \wedge I_2 \cap J_2 \neq \emptyset \wedge I_3 \cap J_3 \neq \emptyset \Longrightarrow dist^2 = 0$$

- Surface to surface (the distance vector may move on opposite facets; l, m, n pairwise disjoint):

$$I_l \cap J_l \neq \emptyset \wedge I_m \cap J_m \neq \emptyset \wedge I_n \cap J_n = \emptyset \Longrightarrow dist^2 = c_n$$

- Edge to edge (the distance vector may move on opposite edges):

$$I_l \cap J_l \neq \emptyset \wedge I_m \cap J_m = \emptyset \wedge I_n \cap J_n = \emptyset \Longrightarrow dist^2 = c_n + c_m$$

- Vertex to vertex:

$$I_1 \cap J_1 = \emptyset \wedge I_2 \cap J_2 = \emptyset \wedge I_3 \cap J_3 = \emptyset \Longrightarrow dist^2 = c_1 + c_2 + c_3$$

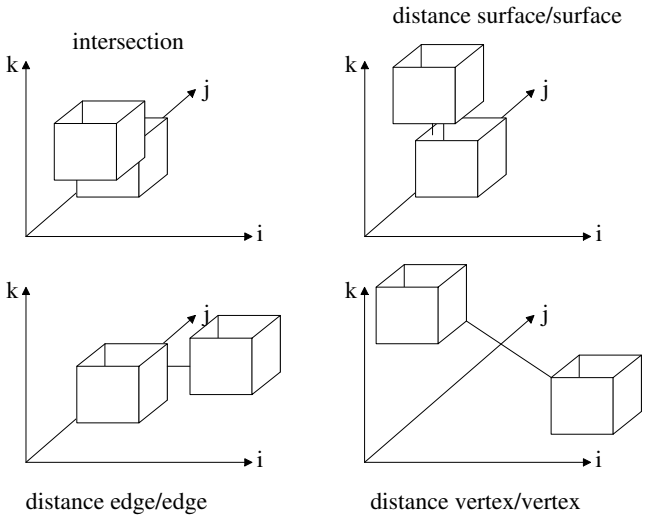

Fig. 4. Various examples of positioning two boxes.

If the entries X_1, X_2, X_3, X_1+h_1, X_2+h_2, X_3+h_3, Y_1, Y_2, Y_3, Y_1+k_1, Y_2+k_2, Y_3+k_3 are machine numbers, the square of the distance can be calculated up to 1 *ulp* with the aid of the exact scalar product. If a fixed point arithmetic is used, the results are exact.
We will now assume that the octree represents two objects, a white (w) and a black (b) one, and that the leaves are integrally white or black depending on the represented object or red (r) for the free space. We further assume that the octree has no bw-boxes which would yield $dist^2 = 0$.

The second part of our algorithm computes the distance between the two objects using the distance formulae between two cubes from part one:

- Initialize the lists LB, LW, LG, the distance $D = 3$, and boxes $W = [0,0,0,0,0,0]$ and $B = [1,1,1,0,0,0]$.
  /*The lists LB and LW are void, LG contains the unit cube. LB contains actual black boxes, LW contains actual white boxes, LG contains gray boxes of the i-th level */

- For all levels $i = 0, 1, ..., N$ /* N depth of the octree */ do

  /* Step 1: Fill lists LW, LB */

  For all children Q of all boxes of size 2^{-i} on level i

  /* Update LG */
  If $Q = white$ then
  { $Q \rightarrow LW$; For all $T \in LB$ do

if $(dist^2(Q,T) < D)$
then { $D := dist^2(Q,T); W := Q; B := T$ }
}
else if $Q = black$ then
{ $Q \rightarrow LB$; For all $T \in LW$ do
if $(dist^2(Q,T) < D)$
then { $D := dist^2(Q,T); W := T; B := Q$ }
}
/* Two or more different kinds of subboxes*/
else if $Q = gray$ then $Q \rightarrow LG$;

/* Step 2: Drop all irrelevant boxes; define $\min(\emptyset) = 0$*/

For all $T \in LB, T \neq B$ do

For all $Q \in LG$ with attribute wr or bwr calculate $dist^2(Q,T)$;
$dist^2_{wr} := \min\left\{dist^2(Q,T) | Q \text{ has attribute wr}\right\}$;
$dist^2_{bwr} := \min\left\{dist^2(Q,T) | Q \text{ has attribute bwr}\right\}$;
if $dist^2_{wr} > D$ and $dist^2_{bwr} > 3 \cdot 2^{-2i-2}$ then drop T in LB;

For all $T \in LW, T \neq W$ do

For all $Q \in LG$ with attribute br or bwr calculate $dist^2(Q,T)$;
$dist^2_{br} := \min\left\{dist^2(Q,T) | Q \text{ has attribute br}\right\}$;
$dist^2_{bwr} := \min\left\{dist^2(Q,T) | Q \text{ has attribute bwr}\right\}$;
if $dist^2_{br} > D$ and $dist^2_{bwr} > 3 \cdot 2^{-2i-2}$ then drop T in LW;
return D.

3.2 Remarks

This algorithm can be modified to return a list of all solutions. To this end, it is necessary to establish a list of pairs of boxes with the same temporary distance. The algorithm provides good upper and lower bounds: the temporary distance D is an upper bound, but we may use $D = 3 \cdot 2^{-2i}$ if there is a bwr-box on level i. It is also possible to compute lower bounds. To this behind determine the greatest level i with bwr-boxes. Replace on an arbitrary level $j \geq i$ all br-boxes with black boxes and all wr-boxes with white boxes. Then apply the algorithm to return D as a lower bound.
The algorithm works in any higher dimension when the definition of the case-selector is generalized to arbitrary dimensions.
On level i we find $2^{6(i+1)}/3$ as an upper bound for the number of box comparisons and distance calculations. Thus, in the worst case, overall complexity is $O\left(2^{6(N+1)}\right)$. If we do not drop irrelevant black and white boxes, the complexity is bounded by the product of the number of black and white boxes.
On the highest level tighter (convex) enclosures of the objects inside the boxes can be used to obtain better bounds for D. Then the simple distance computations in the first step are replaced by an algorithm for convex objects.

For an implementation it is not necessary to create the lists LB, LW, LG. All work can be done on the underlying data structure by traversing the octree in a certain manner and using appropriate flags in the nodes.

3.3 Examples

The first example concerns a level-three quadtree. In executing the algorithm the white box on the right-hand side is dropped. The result is found in the second and third quadrant. By applying a convex hull algorithm on the set of extreme vertices we find simple convex enclosing sets.

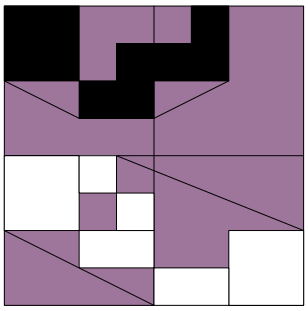

Fig. 5. Quadtrees and convex hull of two objects.

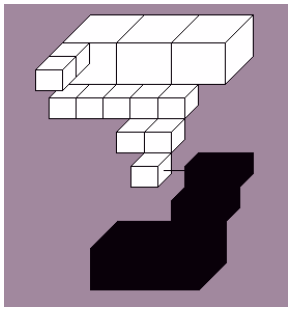

Fig. 6. Octree with two objects on level three.

The convex hull of the extreme vertices is shown in Figure 5. The distance remains unchanged. In the next example (see Figure 6) the algorithm eliminates the boxes near the boundary $z = 1$ with respect to the coordinate system shown in Figure 3.

3.4 Convex Hulls

Now let us turn our attention to the objects obtained by representing three-dimensional convex sets S by octrees to apply distance theorems for this kind of sets. If the sets are non-convex, they can be split into convex parts. Building the octree corresponds to a certain kind of rasterization. So the question arises whether the objects are digital convex. If we replace each box on the highest level with its center point x we obtain sets of grid points S_Δ. This approach allows us to apply results from digital convexity (d.c.):

Theorem 1 (see [16]). *A digital set $S_\Delta \subseteq Z^d$, the set of all d-dimensional vectors whose components have integer values, is digital convex if and only if for each point of $x \in Z^d \setminus S_\Delta$ there is a hyperplane with normal vector x' and distance α to the origin such that $x \cdot x' = \alpha$ and $y \cdot x' > \alpha$ for all $y \in S_\Delta$. If for each boundary point x of S_Δ there is a hyperplane such that $x \cdot x' = \alpha$ and $y \cdot x' \geq \alpha$ for all $y \in S_\Delta$ then S_Δ is digital convex.*

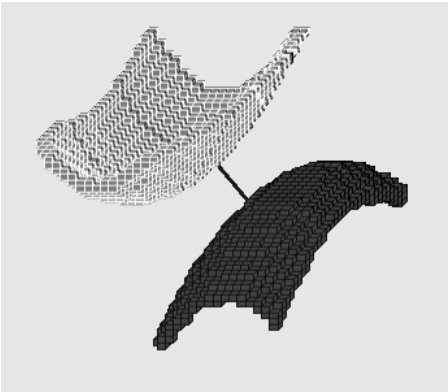

Fig. 7. Parabolic objects - level 5.

This theorem finds its analogous result due to Tietze in the context of continuous convexity.

Unfortunately, Tietze's theorem, which says that the condition $x \cdot x' = \alpha$ and $y \cdot x' \geq \alpha$ should be verified only locally when deriving continuous convexity, does not hold in the digital world. For this reason, a test for digital convexity cannot be done in a time proportional to the number of neighbors and boundary points, as was shown by a counter example given in Eckhardt [16]. However, if the set S_Δ is simply connected and all the boundary points fulfill the interior point condition, i.e., each point $x \in \partial S_\Delta$ has at least two 8-neighbor points belonging to S_Δ and these points are all connected in the 4-neighborhood topology, then the result of Tietze's theorem holds true.

A simpler way to proceed is to use the concept of extreme vertices of boxes on the boundary. A vertex is said to be an extreme vertex if none of the adjacent boxes belongs to the object. In the case of a quadtree there are three neighboring boxes; for octrees there are seven boxes. The convex hull of all extreme vertices is constructed to obtain an enclosure of the object. Obviously, the convex hull also contains the original set S.

Then we can apply our distance algorithms for convex sets and obtain lower bounds for the distances. This approach also opens the way to dynamic algorithms for moving objects. It is well known that rotational motions of octrees lead to an unwanted wrapping-effect, which can be avoided by using the convex hulls of the objects [25].

3.5 Accurate Distance Algorithms for Convex and Non-Convex Polyhedra

Generally, distance algorithms focus on objects represented by convex polyhedra, which are defined as the convex hull of points in three-dimensional space. Although these approaches can be applied for convex polytopes (bounded polyhedra) in three-dimensional space, a wider class of objects is permitted since it

is also possible to treat conveniently non-convex shapes as a union of convex polytopes.

There are two main classes of distance algorithms for convex polyhedral models. In the first class algorithms are based on Voronoi regions, like the Lin-Canny (LC) algorithm [24] and its software implementations, such as I-Collide [10], V-Clip [27], or SWIFT++ [17]. Another class is the simplex-based Gilbert-Johnson-Keerthi (GJK) algorithm [19] and its various extensions, including non-convex objects [29] and proximity queries with collision detection [4].

One drawback of the original LC algorithm is that it does not readily handle penetrating polyhedra; a second is its lack of robustness when applied to models in degenerate configurations. The GJK-like algorithms are more robust than LC; they can also handle penetration cases. Nonetheless, with GJK-like algorithms, computations generally require more floating-point operations. The collision detection library Q-Collide [9] was spawned from I-Collide, which replaces LC with the GJK algorithm for low-level collision detection. A numerical comparison of some derivations of GJK and LC algorithms was done in [20].

Although the GJK algorithm is widely used in robotics, there has been no verification of the computed results. For this reason, we have implemented an interval version of the GJK distance algorithm for tracking the distance between convex polyhedra which is adapted to sensor-based input data [15].

We are also interested in simple accurate algorithms to calculate the distance between two objects, such as points, collections of axis-aligned boxes, (non-)convex polyhedra or NURBS-surfaces with interval vertices. Accurate finite precision algorithms have been developed based on suitable projections and using controlled rounding and the exact scalar product whereby a verified enclosure of the solution is ensured [12].

If the end-effector or the sensor is taken to be a single moving point, an efficient distance algorithm, which does not rely on convex properties and thus is applicable to non-convex polyhedral surfaces has been developed [13]. Under the same assumption the problem has been solved for the more difficult case of NURBS-defined solids based on subdivision techniques and using an algorithm for the solution of nonlinear polynomial systems proposed by Sherbrooke and Patrikalakis [30]. The extension of this algorithm introduces interval arithmetic, the interval version of the convex hull algorithm, and a modified Simplex algorithm. The new solver allows a verification of obtained results [14] using new criteria to guarantee the existence of zeros within the calculated inclusions [18].

Our algorithm to compute the distance between a point and a non-convex polyhedron does not require decomposing the polyhedron into convex parts or iteration and yields the result with high accuracy [13]. It is possible to derive the explicit absolute or relative errors in a real distance point and the distance value to the (non-)convex polyhedron as well as the computed approximations of these values.

3.6 An Accurate Distance Algorithm between a Point and a (Non-)Convex Polyhedron

Given a point $\boldsymbol{y}$ outside a non-degenerated polyhedron P bounded by $\partial P := \{S_i, i = 1, \ldots, m;\ \boldsymbol{v}_j, j = 1, \ldots, n\}$ with m facets and n vertices.
In the following, the vertices belonging to the facet S_i are denoted by $\boldsymbol{s}_{ik}$, $k = 1, \ldots, t_i$, $t_i > 2$, given in counter-clockwise order, and by $[\boldsymbol{s}_{ik}, \boldsymbol{s}_{i(k+1)}]$, the edges of the facet S_i; $k = 1, \ldots, t_i$, $\boldsymbol{s}_{i(t_i+1)} := \boldsymbol{s}_{i1}$.

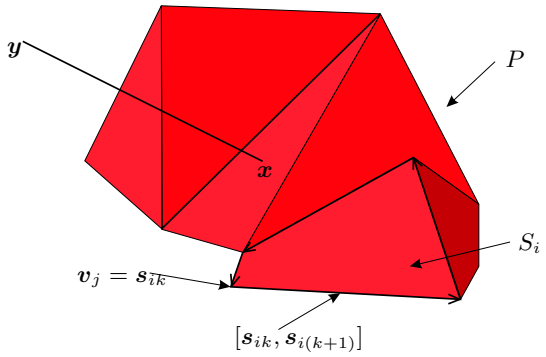

Fig. 8. A point $\boldsymbol{y}$ and a non-convex polyhedron P.

We are searching for the shortest straight line segment $[\boldsymbol{y}, \boldsymbol{x}]$ between point $\boldsymbol{y}$, which is any point outside of polyhedron P, and this polyhedron with $\boldsymbol{x} \in \partial P$.
At the beginning, before starting the distance algorithm, we calculate the correctly rounded cross product

$$\boldsymbol{n}_{i2} = (\boldsymbol{s}_{i2} - \boldsymbol{s}_{i1}) \times (\boldsymbol{s}_{i3} - \boldsymbol{s}_{i2}) = \boldsymbol{s}_{i2} \times \boldsymbol{s}_{i3} + \boldsymbol{s}_{i1} \times \boldsymbol{s}_{i2} - \boldsymbol{s}_{i1} \times \boldsymbol{s}_{i3}$$

with $\boldsymbol{x} \times \boldsymbol{y} := (x_2 \cdot y_3 - x_3 \cdot y_2, x_3 \cdot y_1 - x_1 \cdot y_3, x_1 \cdot y_2 - x_2 \cdot y_1)$ for $\boldsymbol{x} = (x_1, x_2, x_3)$, $\boldsymbol{y} = (y_1, y_2, y_3)$, and a normal vector $\boldsymbol{n}_i = \boldsymbol{n}_{i2} / \sqrt{\boldsymbol{n}_{i2} \cdot \boldsymbol{n}_{i2}})$ for all $i = 1, \ldots, m$.
Then E_i denotes the plane described by

$$E_i : \ \boldsymbol{x} \cdot \boldsymbol{n}_i - \boldsymbol{s}_{i1} \cdot \boldsymbol{n}_i = 0.$$

For all scalar product computations the algorithm uses the exact scalar product followed by rounding (to nearest):

A: We calculate the distances between point $\boldsymbol{y}$ and each plane E_i

$$l_i := \boldsymbol{y} \cdot \boldsymbol{n}_i - \boldsymbol{s}_{i1} \cdot \boldsymbol{n}_i.$$

We store the sign of l_i, $i = 1, ..., m$ for future use. There is at least one $l_i > 0$ therefore the set $I := \{i \mid l_i > 0\}$ is not empty, and we can form the set J of all $j \in \{1, \ldots, n\}$ with $\exists_{i \in I} \boldsymbol{v}_j \in S_i$ and the set K of all pairs (s, r) with

$$\exists_{i \in I} \exists_k [\boldsymbol{v}_s, \boldsymbol{v}_r] = [\boldsymbol{v}_{ik}, \boldsymbol{v}_{i(k+1)}],\ s, r \in \{1, \ldots, n\}.$$

Then, for all $i \in I$, the projections onto E_i can be accurately calculated:

$$\boldsymbol{x}_i := \boldsymbol{y} - l_i \cdot \boldsymbol{n}_i.$$

Next, we have to decide whether $\boldsymbol{x}_i$ is in S_i. For that purpose we calculate the number of intersections of the ray $\boldsymbol{x}_i + t'(\boldsymbol{m} - \boldsymbol{x}_i)$, $t' \geq 0$, suitable $\boldsymbol{m} \in E_i$, with edges $[\boldsymbol{s}_{ik}, \boldsymbol{s}_{i(k+1)}]$, $k = 1, \ldots, t_i$, avoiding vertices, by solving a system of two equations with two variables. These equations result from setting the first derivatives of the function

$$f(t', t'') = \|\boldsymbol{s}_{ik} + t''(\boldsymbol{s}_{i(k+1)} - \boldsymbol{s}_{ik}) - \boldsymbol{x}_i - t'(\boldsymbol{m} - \boldsymbol{x}_i)\|^2$$

in the variables t'' and t' to zero. If $\boldsymbol{x}_i$ belongs to the polygonal surface S_i, i.e. if the number of intersections is odd, we remove all edges from K belonging to S_i and calculate for the remaining $(s, r) \in K$ the scalar products

$$w_{si} := (\boldsymbol{y} - \boldsymbol{x}_i) \cdot (\boldsymbol{v}_s - \boldsymbol{x}_i) \quad \text{and} \quad w_{ri} := (\boldsymbol{y} - \boldsymbol{x}_i) \cdot (\boldsymbol{v}_r - \boldsymbol{x}_i)$$

and, if $w_{si} \leq 0$ and $w_{ri} \leq 0$, we redefine $K := K \backslash \{(s, r)\}$. Then we set a distance-point $\boldsymbol{x} := \boldsymbol{x}_i$ and the distance $d := l_i$ or update them (if there are points with the same distance, the result of the algorithm will be a list of them), and stop the algorithm if $K = \emptyset$.

B: If $K \neq \emptyset$ after step A, then we have to decide for all edges with $(s, r) \in K$, whether the projection of $\boldsymbol{y}$ to the line

$$\boldsymbol{u}(t) := \boldsymbol{v}_s + t\,(\boldsymbol{v}_r - \boldsymbol{v}_s)$$

meets a point with parameter $0 \leq t \leq 1$. To do so, we form the accurately calculated scalar products

$$\kappa := (\boldsymbol{y} - \boldsymbol{v}_s) \cdot (\boldsymbol{v}_r - \boldsymbol{v}_s) \quad \text{and} \quad \mu := (\boldsymbol{v}_r - \boldsymbol{v}_s) \cdot (\boldsymbol{v}_r - \boldsymbol{v}_s).$$

If $\kappa < 0$ or $\kappa > \mu$, then the projection ray does not meet the section between $\boldsymbol{v}_s$ and $\boldsymbol{v}_r$. Otherwise, the projection point on $[\boldsymbol{v}_s, \boldsymbol{v}_r]$ is given by

$$\boldsymbol{x}_{sr} := \boldsymbol{v}_s + \frac{\kappa}{\mu}\,(\boldsymbol{v}_r - \boldsymbol{v}_s)$$

and the square of the distance by

$$d_{sr}^2 := \frac{((\boldsymbol{v}_r - \boldsymbol{v}_s) \times (\boldsymbol{y} - \boldsymbol{v}_r)) \cdot ((\boldsymbol{v}_r - \boldsymbol{v}_s) \times (\boldsymbol{y} - \boldsymbol{v}_r))}{\boldsymbol{v}_r \cdot \boldsymbol{v}_r - \boldsymbol{v}_r \cdot \boldsymbol{v}_s - \boldsymbol{v}_s \cdot \boldsymbol{v}_r + \boldsymbol{v}_s \cdot \boldsymbol{v}_s}.$$

We replace J by $J \backslash \{s, r\}$. Using the projection point $\boldsymbol{x}_{sr}$ we calculate for all $j \in J$ the scalar products

$$w_{jsr} := (\boldsymbol{y} - \boldsymbol{x}_{sr}) \cdot (\boldsymbol{v}_j - \boldsymbol{x}_{sr})$$

and if $w_{jsr} \leq 0$ we set $J := J \backslash \{j\}$.
If the projection point is the nearest distance point, we update $\boldsymbol{x}$ with $\boldsymbol{x}_{sr}$ and d with $\sqrt{d_{sr}^2}$. We stop the algorithm if $J = \emptyset$.

C: If $J \neq \emptyset$ after step B, we compare the distance values of the paths joining point $\boldsymbol{y}$ and each vertex-point $\boldsymbol{x}_j$, $j \in J$, with the distance found so far and update d and $\boldsymbol{x}$ if necessary.

The accurate distance algorithm works in linear time $O(Cn)$ with an order-constant C depending on the number of successful projections onto facets and edges. Furthermore, it can be used to determine the local distance between a point and any polyhedral surface described by its vertices and oriented facets.

3.7 Error Discussion

Let $\varepsilon_l \leq 2^{-52}$ be the rounding error in the floating-point number space $S := (\mathcal{B}, l, em, eM)$ characterized by its base $\mathcal{B}$, mantissa length l and $[em, eM]$ the smallest and largest allowable exponents. Then, for the error estimation of a calculated distance point $\boldsymbol{x} = (x_1, x_2, x_3)$ and the distance value $d = \|\boldsymbol{y} - \boldsymbol{x}\|$ in the cases discussed in steps A, B and C it can be shown [12] that the results in Table 1 are valid.

Table 1. Absolute or relative errors in the distance point and value

Step:	Error estimations
A:	$x_v = X_v + \delta_{1,v}(11.032\|\boldsymbol{y}\| + 10.032\sigma_i)\varepsilon_l$
	$d = D + 4.27\delta_1'(\|\boldsymbol{y}\| + \sigma_i)\varepsilon_l$ (a point to a surface)
B:	$x_v = X_v + \delta_{2,v}(2.505\|\boldsymbol{y}\| + 14.515\sigma_i)\varepsilon_l$
	$d = D(1 + 3.003\delta_2'\varepsilon_l)$ (a point to an edge)
C:	$x_v = X_v$, $d = D(1 + 1.76\delta_3'\varepsilon_l)$ (a point to a point)

$v = 1, 2, 3,\ \sigma_i := \max_k \|\boldsymbol{s}_{ik}\|,\ |\delta_{j,v}| \leq 1,\ |\delta_j'| \leq 1,\ j = 1, 2, 3,$
$D = \|\boldsymbol{y} - \boldsymbol{X}\|,\ \boldsymbol{X} \in \partial P$ the real distance point

3.8 Example

The algorithm was implemented in C++ using the library Profil/BIAS [23]. Figure 9 shows the ASCII input file of a non-convex polyhedron. The input file consists of two parts: the fourteen vertex points of the polyhedron in a Cartesian coordinate system as geometric information and their positions on its nine faces as topological information. The corresponding program layout for the point $\boldsymbol{y}$ lying outside of the polyhedron in the origin of the Cartesian coordinate system is shown on the opposite side of the figure.

4 Reliable Intersection Algorithms

In the previous section accurate distance algorithms widely used for path planning in robotics were described. In computer graphics, it is important to know

```
polyhedron
 -1.0     6.0     0.0
 -4.0     0.0     0.0
  0.0    -4.0     0.0
  4.0     0.0     0.0
  1.0     6.0     0.0
  1.0    -1.0     0.0
 -1.0    -1.0     0.0
 -1.0     6.0    -2.0
 -4.0     0.0    -2.0
  0.0    -4.0    -2.0
  4.0     0.0    -2.0
  1.0     6.0    -2.0
  1.0    -1.0    -2.0
 -1.0    -1.0    -2.0
E
1  2  3  4  5  6  7
14  13  12  11  10  9  8
9  2  1  8
10  3  2  9
11  4  3  10
12  5  4  11
13  6  5  12
14  7  6  13
8  1  7  14
```

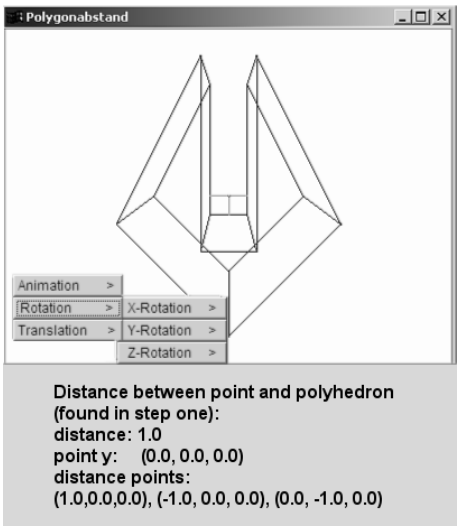

Fig. 9. Distance computation: the input file and program layout.

not only whether two objects intersect, but also where they intersect. Direct ray-tracing of parametric surfaces, rendering and voxelization of implicit curves, surfaces and volumes, as well as the computation of intersection curves are common tasks.

4.1 On Bounding Volumes and Subdivision

If a direct solution to the problem is not possible (which is generally the case), the application of a divide-and-conquer strategy is a widespread approach. A common technique for reducing the computational complexity of intersection problems is to subdivide the complex object into simpler objects and to simplify the shape using bounding volumes. Divide-and-conquer approaches to solve object-object intersection problems find by definition all possible intersections, but due to the piecewise enclosure of the solution information on the overall topology of the intersection gets lost. Postprocessing steps like connectivity determination and sorting are necessary to restore this information. Solutions for this problem can be found in classical literature on computational geometry and e.g. in [2]. Classical bounding volumes are simple solids, such as axis-aligned or -oriented bounding boxes, parallelepipeds, polyhedra or spheres. In general they are computed using range analysis methods based on sampling, exploiting convex hull properties of control points, evaluation of derivatives, or applying affine or interval arithmetic. Bounding volumes should be a reliable enclosure of the object, which is not the case if sampling techniques are used to construct the bounding volume. The direct application of interval or affine arithmetic to compute a bounding volume produces reliable bounds, but these bounds overestimate the object because functional dependencies are not taken into account, or

are lost during conversion from affine forms to intervals. Axis-aligned bounding boxes are easy to compute and intersect easily with other axis-aligned bounding boxes or rays; thus, they are well-suited for rapidly providing an insight into the structure of an environment with obstacles and targets. However, in most cases they significantly overestimate curves and surface patches. Therefore, in subdivision-based algorithms many more steps are necessary to reach precision than when using the much better fitting parallelepipeds. On the other hand, an intersection test for two parallelepipeds, for instance, is very complex and time-consuming. Furthermore, all classical bounding volumes are solids, i.e. they provide information only on the location of the whole object. Yet, especially for intersection algorithms for parametric objects, in order to accelerate the computation it would be interesting to be able to derive information on the location of the intersection of the enclosed objects in parameter space from the intersection of two bounding volumes. To summarize, the ideal bounding volume provides a tight and reliable enclosure of the object, is easily calculated, and intersects easily with other, similar bounding volumes.

4.2 Linear Interval Estimations

To overcome problems connected with classical bounding volumes, another form of enclosing objects satisfying the requirements for the ideal bounding volume listed above has been introduced for parametric and implicit objects: *Linear Interval Estimations* [7, 8] are defined as the linear approximation of the representation of the enclosed object combined with an interval estimation of the approximation error. An LIE is just a thick (hyper)plane, that can be understood as a continuous linear set of axis parallel bounding boxes. Furthermore, the representation of an LIE corresponds to the representation of the object. This means in the parametric case that the LIE can be parameterized in such a way that its parameterization corresponds to the parameterization of the enclosed object. Each point of the object is enclosed by an "interval point" (an interval box) of the LIE with the same parameters. In the case of the intersection of two LIEs this construction allows direct conclusions on the location of intersections of the two enclosed objects in object *and* parameter space. This characteristic of LIEs is the most significant difference to other common bounding volumes.

But LIEs are also easy to compute and usually provide much tighter enclosures than common solid bounding volumes. If reliable methods are used to compute the LIE, it also provides a reliable enclosure of the patch. Furthermore, the diameter of the interval part of the LIE contains information about the flatness of the patch and its extension has been proven to be a good termination criterion for subdivision-based algorithms. The linear structure of the LIE reduces the intersection problem of parametric or implicit objects to the solution of (constrained) linear equation systems, which can, in general, be solved much more easily than the original problem.

4.3 Parametric LIEs

Parametric objects are widely used in computer graphics and computer aided geometric design. Bézier, B-Spline, and NURBS curves, surfaces and volumes are standard representations used for effective and exact modeling and representation of smooth objects. A general parametric object S over a rectangular parameter domain can be defined as follows:

$$S : \left\{ \begin{array}{ll} \mathbb{I} = \prod_{i=1}^{m} I_i \in \mathbb{IR}^m & \rightarrow \mathbb{R}^n \\ \boldsymbol{x} & \mapsto \boldsymbol{f}(\boldsymbol{x}) := (f_1(\boldsymbol{x}), ..., f_n(\boldsymbol{x}))^T \end{array} \right\}$$

The corresponding linear interval estimation enclosing the object described above must fulfill the following requirements:

Definition 2. A linear map $\mathbb{L} : \mathbb{I}^* \in \mathbb{IR}^m \rightarrow \mathbb{R}^n$,

$$\mathbb{L}(\boldsymbol{x}^*) := \mathbb{p} + \sum_{i=1}^{m} x_i^* \, \boldsymbol{v}_i, \quad \boldsymbol{x}^* = (x_1^*, ..., x_m^*) \in \prod_{i=1}^{m} I_i^* = \mathbb{I}^* \in \mathbb{IR}^m \tag{2}$$

with $\mathbb{p} \in \mathbb{IR}^n$, $\boldsymbol{v}_i \in \mathbb{R}^n, i = 1, ..., n$ is called a *linear interval estimation (LIE)* of the parametric object $\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^n$; $\boldsymbol{x} \in \mathbb{I} \in \mathbb{IR}^m$ iff there exists a valid reparameterization

$$\phi : \left\{ \begin{array}{l} \mathbb{I} \rightarrow \mathbb{I}^* \\ \boldsymbol{x} \mapsto \phi(\boldsymbol{x}) := \boldsymbol{x}^* \end{array} \right.$$

of $\mathbb{L}$ so that for all $\boldsymbol{x} \in \mathbb{I}$ holds $\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{L}(\phi(\boldsymbol{x})) = \mathbb{L}(\boldsymbol{x}^*)$.

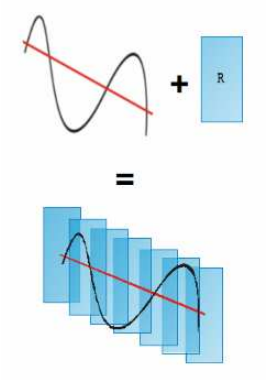

Fig. 10. Sketch of the construction of LIEs.

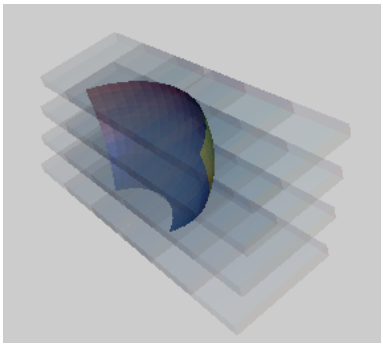

Fig. 11. Discrete representation of an LIE based on affine arithmetic.

Computation. The general recipe for constructing parametric LIEs is quite simple (see also Figure 10 and 11):

1. Compute a linear approximation of the object.
2. Estimate and enclose the approximation error with an interval vector.
3. Reparameterize the linear approximation so that it corresponds to the parameterization of the object.

Two different methods of computing LIEs have been proposed [7]. One is based on first order Taylor models and is straight forward:

Theorem 2. *Let $(T, \mathbb{J})$ be a first order Taylor model of the function $\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{I}$. Then*

$$\mathbb{L}(\boldsymbol{x}) = T(\boldsymbol{x}) + \mathbb{J}$$

is an LIE of $\boldsymbol{f}$.

$\mathbb{L}(\boldsymbol{x})$ can be written in the form $\mathbb{p} + \sum_{i=1}^{m} x_i \frac{\partial}{\partial x_i}\boldsymbol{f}(\boldsymbol{x}_0)$ with $\mathbb{p} := \boldsymbol{f}(\boldsymbol{x}_0) - \sum_{i=1}^{m} x_i^0 \frac{\partial}{\partial x_i}\boldsymbol{f}(\boldsymbol{x}_0) + \mathbb{J}$, which already corresponds to the parameterization of the object. Thus, in this case, a reparameterization of the LIE is not necessary.

The second method exploits the intrinsic structure of affine arithmetic:

Theorem 3. *Let $\boldsymbol{f} : \mathbb{I} \in I\!R^m \rightarrow I\!R^n$ be C^0 over $\mathbb{I}$ and $\mathbb{I} := \prod_{k=1}^{m} I_k$ with $rad(I_k) > 0, k = 1, ..., m$. $\tilde{x}_k := x_0^k + x_1^k \epsilon_k; \quad \epsilon_k \in [-1, 1]$ denote the affine forms corresponding to $I_k, k = 1, ..., m$ and $\tilde{\boldsymbol{x}} = (\tilde{x}_1, ..., \tilde{x}_m)^T$.*

$$\boldsymbol{f}(\tilde{\boldsymbol{x}}) = \tilde{\boldsymbol{f}}(\epsilon_1, ..., \epsilon_m, \gamma_1, ..., \gamma_l) = \boldsymbol{f}^0 + \sum_{k=1}^{m} \boldsymbol{f}^k \epsilon_k + \sum_{i=1}^{l} \boldsymbol{r}^i \gamma_i$$

with $\epsilon_k, \gamma_i \in [-1, 1]$ and $\boldsymbol{f}^k, \boldsymbol{r}^i \in I\!R^n$ for all $k = 1, ..., m, i = 1, ..., l$, denotes the evaluation of $\boldsymbol{f}$ with $\tilde{\boldsymbol{x}}$. Furthermore let be $\mathbb{p} := \boldsymbol{f}^0 + \mathbb{q}$ with $\mathbb{q} := \left[-\sum_{i=1}^{l} |\boldsymbol{r}^i|, \sum_{i=1}^{l} |\boldsymbol{r}^i|\right]$ and $|\boldsymbol{r}^i| := (|r_1^i|, ..., |r_n^i|)^T, \ i = 1, ..., l$. Then

$$\mathbb{L}_\epsilon(\epsilon_1, ..., \epsilon_m) := \mathbb{p} + \sum_{k=1}^{m} \boldsymbol{f}^k \epsilon_k; \quad \epsilon_k \in [-1, 1], k = 1, ..., m \tag{3}$$

is an LIE of $\boldsymbol{f}$.

The evaluation of the function describing our object with respect to the affine forms representing the parameter domain, is equivalent to the computation of a point symmetric polytope enclosing our object. The term

$$\boldsymbol{f}^0 + \sum_{k=1}^{m} \boldsymbol{f}^k \epsilon_k \tag{4}$$

describes a subset of the polytope that is a linear approximation of the input object with respect to the input error symbols and therefore also with respect to the original parameters. The sum $\sum_{i=1}^{l} \boldsymbol{r}^i \gamma_i$ describes for each point of (4) an enclosure of approximation and rounding errors introduced during the evaluation, that is estimated in the theorem by the interval vector $\mathfrak{q}$.

Formula (3) describes the combination of (4) with the error estimation $\mathfrak{q}$ which is after Definition 2 a LIE of $\boldsymbol{f}$ with respect to the input error symbols $\epsilon_k, k = 1, ..., m$ and the parameter domain $[-1, 1]^m$. To reestablish the direct connection between the parameterization of the original object and the parametrization of the corresponding LIE a reparameterization of $\mathbb{L}_\epsilon$ is necessary. The map ϕ describes the correspondence between the two parameterizations:

$$\phi : \begin{cases} \mathbb{I} := \prod_{k=1}^{m} [a_k, b_k] & \rightarrow & [-1, 1]^m \\ \boldsymbol{x} & \mapsto & \phi(\boldsymbol{x}) := (\alpha_1 x_1 - \beta_1, ..., \alpha_m x_m - \beta_m) = (\epsilon_1, ..., \epsilon_m) \end{cases}$$

with $\alpha_k := \frac{2}{b_k - a_k}$ and $\beta_k := \frac{b_k + a_k}{b_k - a_k}, k = 1, ..., m$. Finally $\mathbb{L}(\boldsymbol{x}) := \mathbb{L}_\epsilon(\phi^{-1}(\epsilon_1, ..., \epsilon_m))$ describes the parametrization of the LIE with respect to the same parameters and parameter domain of the enclosed object.

Intersection. All intersection problems for LIEs with boxes, rays or other LIEs can be described as a system of constrained linear interval equations of the form $A\boldsymbol{x} = \mathbb{b}$, where A is a thin matrix, $\mathbb{b}$ is an interval vector and $\boldsymbol{x}$ is the vector of unknown parameters constrained by their predefined domains:

Let be $\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^n; \boldsymbol{x} \in \mathbb{I} \in \mathbb{IR}^m$ and $\boldsymbol{g}(\boldsymbol{y}) \in \mathbb{R}^n; \boldsymbol{y} \in \mathbb{J} \in \mathbb{IR}^k$ two parametric objects in n-space with their respective LIEs $\mathbb{F}(\boldsymbol{x}) = \mathbb{f}_0 + \sum_{i=1}^{m} x_i \boldsymbol{f}_i$ and $\mathbb{G}(\boldsymbol{y}) = \mathbb{g}_0 + \sum_{j=1}^{k} y_j \boldsymbol{g}_j$. The intersection of $\mathbb{L}$ and $\mathbb{G}$ can be described as the solution of the system of linear equations

$$\sum_{i=1}^{m} x_i \boldsymbol{f}_i - \sum_{j=1}^{k} y_j \boldsymbol{g}_j = \mathbb{g}_0 - \mathbb{f}_0$$

with the constraints $\boldsymbol{x} \in \mathbb{I}$ and $\boldsymbol{y} \in \mathbb{J}$. A detailed discussion about how an enclosure of a linear system of interval equations can be computed is found in a number of books and articles (see, for example, [3, 28]). In addition to an enclosure of the solution in object space, an enclosure of the solution in parameter space is also needed for effective parameter domain pruning during a subdivision procedure. If interval arithmetic is applied, these enclosures can be generated partly as a byproduct of the intersection algorithms; additional steps might be necessary to compute tight solutions for all parameters.

We are proposing an effective algorithm for computing the intersection of two LIEs of surface patches in 3-space. Its goal is to compute an interval line that encloses the intersection of two LIEs as narrowly as possible as well as to locate the parameter domains $\tilde{\mathbb{I}} \subseteq \mathbb{I}$ and $\tilde{\tilde{\mathbb{J}}} \subseteq \mathbb{J}$ defined in Theorem 4, which enclose the parameter values of the interval intersection line as narrowly as possible. The derivation of the algorithm follows a geometric approach similar to the intersection of two parallelograms in space:

1. For each parallelogram compute the intersection points of the four border lines with the carrying plane of the other parallelogram.

2. Intersect all four line segments formed by the intersection points of parallel lines (see Figure 12).

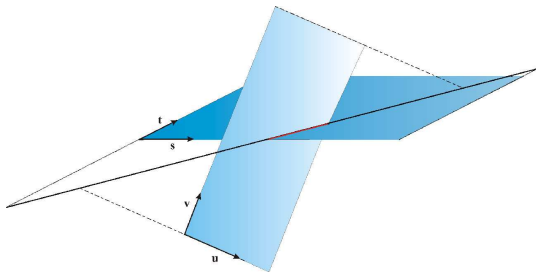

Fig. 12. Intersection of two parallelograms in space

Let be

$$\mathbb{L}_1(u,v) = \mathbb{p} + u\boldsymbol{y}_1 + v\boldsymbol{y}_2; \qquad (u,v) \in I_u \times I_v = \mathbb{I} \tag{5}$$

$$\mathbb{L}_2(s,t) = \mathbb{q} + s\boldsymbol{w}_1 + t\boldsymbol{w}_2; \qquad (s,t) \in J_s \times J_t = \mathbb{J} \tag{6}$$

two LIEs in $\mathbb{R}^3$. Equating (5) and (6) yields

$$(\boldsymbol{y}_1 \quad \boldsymbol{y}_2 \quad -\boldsymbol{w}_1 \quad -\boldsymbol{w}_2) \begin{pmatrix} u \\ v \\ s \\ t \end{pmatrix} = \boldsymbol{r}; \quad \boldsymbol{r} \in \mathbb{r} := \mathbb{q} - \mathbb{p}. \tag{7}$$

Under the assumption that each triple of vectors of $\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{w}_1, \boldsymbol{w}_2$ is linearly independent, the solution set of this underdetermined system is either the empty set, an interval point, or an interval line enclosing the intersection of $\mathbb{L}_1$ and $\mathbb{L}_2$.

Following the geometric approach, the four line segments containing the intersection can be computed in the following way: For each line segment assume that one parameter of (s,t,u,v) is fixed and apply Cramer's rule to the corresponding 3×3 sub-matrix to solve the system for the two parameters that belong to the other LIE. For example, if we consider t the fixed parameter, (7) changes to

$$(\boldsymbol{y}_1 \quad \boldsymbol{y}_2 \quad -\boldsymbol{w}_1) \begin{pmatrix} u \\ v \\ s \end{pmatrix} = \boldsymbol{r} + t\boldsymbol{w}_2; \quad \boldsymbol{r} \in \mathbb{r} := \mathbb{q} - \mathbb{p}. \tag{8}$$

Applied to all parameters this yields the following equations:

$$\begin{array}{ll} S_1(u) = \frac{1}{\alpha}(\,K + \beta\, u) & U_1(s) = \frac{1}{\beta}(\,-K + \alpha\, s) \\ S_2(v) = \frac{1}{\gamma}(\,L - \beta\, v) & U_2(t) = \frac{1}{\delta}(\,M - \alpha\, t) \\ T_1(u) = \frac{1}{\alpha}(\,M - \delta\, u) & V_1(s) = \frac{1}{\beta}(\,L - \gamma\, s) \\ T_2(v) = \frac{1}{\gamma}(\,N + \delta\, v) & V_2(t) = \frac{1}{\delta}(\,-N + \gamma\, t) \end{array}$$

where

$$\begin{array}{lll} \alpha := |\boldsymbol{y}_2 & -\boldsymbol{w}_1 & -\boldsymbol{w}_2| \\ \beta := |\boldsymbol{y}_1 & \boldsymbol{y}_2 & -\boldsymbol{w}_2| \\ \gamma := |\boldsymbol{y}_1 & -\boldsymbol{w}_1 & -\boldsymbol{w}_2| \\ \delta := |\boldsymbol{y}_1 & \boldsymbol{y}_2 & -\boldsymbol{w}_1| \end{array} \qquad \begin{array}{lll} K := |\boldsymbol{y}_2 & \mathbb{r} & -\boldsymbol{w}_2| \\ L := |\boldsymbol{y}_1 & \mathbb{r} & -\boldsymbol{w}_2| \\ M := |\boldsymbol{y}_2 & -\boldsymbol{w}_1 & \mathbb{r}| \\ N := |\boldsymbol{y}_1 & -\boldsymbol{w}_1 & \mathbb{r}| \end{array}$$

The equations above can be combined to the four intersection lines $\mathbb{g}_i, i = 1, 2$ and $\mathbb{h}_j, j = 1, 2$ parameterized using the same parameters as the LIEs.

$$\begin{array}{ll} \mathbb{g}_1(u) := \begin{pmatrix} S_1(u) \\ T_1(u) \end{pmatrix};\ u \in I_u & \mathbb{g}_2(v) := \begin{pmatrix} S_2(v) \\ T_2(v) \end{pmatrix};\ v \in I_v \\ \mathbb{h}_1(s) := \begin{pmatrix} U_1(s) \\ V_1(s) \end{pmatrix};\ s \in J_s & \mathbb{h}_2(t) := \begin{pmatrix} U_2(t) \\ V_2(t) \end{pmatrix};\ t \in J_t \end{array} \tag{9}$$

Notice that computing K, L, M and N during the first expansion of the determinant according to the elements of $\mathbb{r} = (R_1, R_2, R_3)^T$ avoids overestimation. In this case, each interval $R_i, i = 1, ..., 3$ appears only once in the expression, and the result is an exact enclosure. Furthermore, all occurring matrices are thin and assumed to be regular, which implies that the enclosure of the solution and the solution are identical [28]. Thus, $\mathbb{g}_i, i = 1, 2$ and $\mathbb{h}_j, j = 1, 2$ are optimal enclosures of the non-constrained problem.

The following theorem clarifies how an enclosure of the intersecting line segment and interval enclosures of the intersection in the parameter domains can be computed:

Theorem 4. *Let $\mathbb{L}_1$ and $\mathbb{L}_2$ be the LIEs defined by equations (5) and (6), and $\mathbb{g}_1, \mathbb{g}_2, \mathbb{h}_1, \mathbb{h}_2$ the interval lines defined by equations (9). If each triple of the vectors $\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{w}_1, \boldsymbol{w}_2$ is linearly independent, an enclosure of the intersection of $\mathbb{L}_1$ and $\mathbb{L}_2$ is provided by each of the interval line segments $\mathbb{g}_1(u),\ u \in \tilde{I}_u$, $\mathbb{g}_2(v),\ v \in \tilde{I}_v$, $\mathbb{h}_1(s)\ s \in \tilde{\tilde{J}}_s$ and $\mathbb{h}_2(t)\ t \in \tilde{\tilde{J}}_t$, where*

$$\tilde{\mathbb{J}} := \mathbb{g}_1(I_u) \cap \mathbb{g}_2(I_v) \cap \mathbb{J}$$

$$\tilde{\mathbb{I}} := \mathbb{h}_1(\tilde{J}_s) \cap \mathbb{h}_2(\tilde{J}_t) \cap \mathbb{I}$$

$$\tilde{\tilde{\mathbb{J}}} := \mathbb{g}_1(\tilde{I}_u) \cap \mathbb{g}_2(\tilde{I}_v) \cap \tilde{\mathbb{J}}$$

The LIEs do not intersect if at least one of the parameter domains $\tilde{\mathbb{J}}, \tilde{\mathbb{I}}$ or $\tilde{\tilde{\mathbb{J}}}$ is empty.

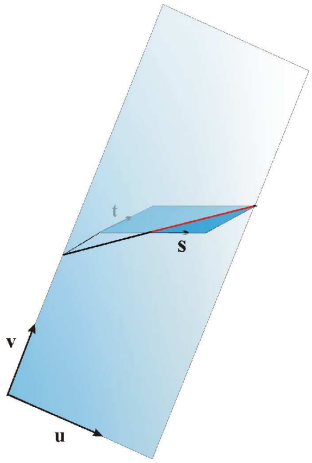

Fig. 13. The parallelograms of Figure 12 after the first reduction step,...

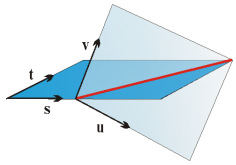

Fig. 14. ... after the second reduction,...

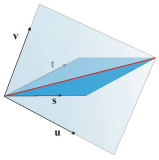

Fig. 15. ... and after the third reduction.

Notice that the computed intervals are very good enclosures of the solution but might still be slightly overestimated due to the computation of intermediate axis-aligned enclosures. Figures 13 – 15 illustrate the three pruning steps.

Special cases occur if two or more of the determinants α, β, γ, and δ disappear (zeros appear always in pairs), which is equivalent to the LIEs having parallel edges. In the proposed algorithm special cases are handled with two different approaches depending on their type:

1. if two determinants are zero, for $\alpha = 0$ set $\boldsymbol{g}_1(u) := \mathbb{J}$, for $\gamma = 0$ set $\boldsymbol{g}_2(v) := \mathbb{J}$, for $\beta = 0$ set $\boldsymbol{h}_1(s) := \mathbb{I}$, and for $\delta = 0$ set $\boldsymbol{h}_2(t) := \mathbb{I}$,
2. if more than two determinants are zero, compute an axis-aligned bounding box.

The complete algorithm is described in Figure 16.

Application of the algorithm The algorithm has been implemented in C++ using the Profil/BIAS package [23] and a modification of the affine arithmetic package by van Iwaarden. The algorithm is part of the new subdivision algorithm for surface patches described in [7]. (The reader is referred to the publication for details.) The results can be summarized as follows: The use of LIEs allows the subdivision algorithm to be optimized in almost all steps (effective bounding volumes, easy and fast intersection, parameter domain pruning, adaptive subdivision, termination criterion, and so forth). The number of subdivisions and intersection tests, as well as computation time have been reduced dramatically

Algorithm: IntersectionTest
Input: A pair of surfaces AB with corresponding LIEs LIE(A) and LIE(B).
Output: Explicit: TRUE, if A and B intersect; FALSE, if not. Implicit: If the test is positive, the algorithm also returns the pruned parameter spaces of A and B. Value of the intersection flag AB.inters

```
Boolean IntersectionTest(Surface pair AB, LIE(A), LIE(B)){
    Compute α, β, γ, δ, K, L, M, N
    If (α = β = γ = δ = 0)                    // special case 2: LIEs parallel or equal
        If axis parallel bounding boxes intersect
            AB.inters = true; Return true;          // Surfaces might intersect.
        else
            AB.inters = false; Return false;        // Surfaces do not intersect.
    Compute J̃;                                // see theorem 4 and special cases 1
    If (J̃ = ∅)
        AB.inters = false; Return false;        // Surfaces do not intersect.
    Compute Ĩ;                                // see theorem 4 and special cases 1
    If (Ĩ = ∅)
        AB.inters = false; Return false;        // Surfaces do not intersect.
    Compute J̃̃;                               // see theorem 4 and special cases 1
    If (J̃̃ = ∅)
        AB.inters = false; Return false;        // Surfaces do not intersect.
    A.domain = Ĩ              //Ĩ defines the reduced parameter space of patch A
    B.domain = J̃̃             //J̃̃ defines the reduced parameter space of patch B
    Return true;                                //Surfaces might intersect.
}
```

Fig. 16. Intersection algorithm for two LIEs enclosing surface patches in 3-space

compared to subdivisions that use axis-aligned bounding volumes. For two simple quadric patches in almost rectangular position to one another, both defined on the parameter domain $[-2, 2]^2$, computing the intersection with precision of 0.01 needed 0.01 seconds applying the LIE algorithm, but 2.61 seconds to reach the same precision and reliability with a pure subdivision. The ILIE algorithm described in Section 4.4 needed just 58 subdivisions to enclose the results with interval line segments of a diameter less than 0.01, whereas a reliable enclosure with boxes required almost 100,000 subdivisions. This example demonstrates another important side-effect of the reduction of subdivisions: the amount of data needed to represent the result is much smaller. Final results are enclosed by the interval lines in parameter space and by the corresponding interval surface curves in object space. Tests also show that, applied in a subdivision algorithm for surface-surface intersection, the proposed algorithm is on average about 25% faster and needs 15% fewer subdivisions than the ILSS algorithm included in the Profil/BIAS package. Particularly remarkable was the observation, that, especially for the two simple surface patches described above, the ILSS algorithm was

about 50% slower while requiring the same number of subdivisions. An example is given in Figure 17.

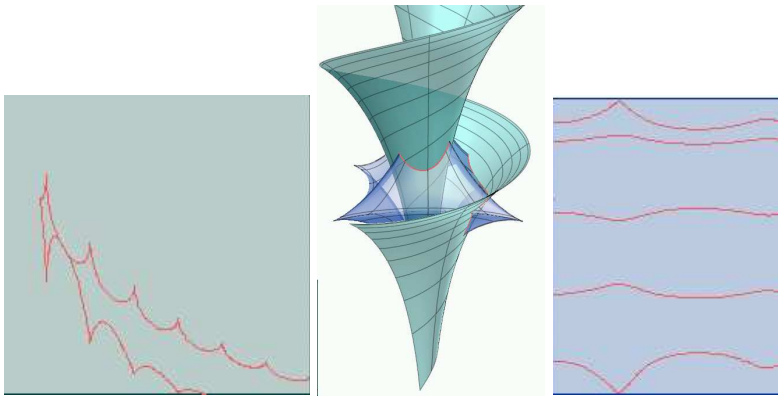

Fig. 17. Intersection of a Dini surface and an astroidal. The boxes left and right show the corresponding parameter domains with the intersection in parameter space.

4.4 Implicit LIEs

Implicit equations are a powerful tool for the representation of curves, surfaces and volumes in computer graphics. Besides the description of mathematical, physical, geological, and other scientific phenomena, implicit surfaces and volumes are mainly used in CSG-Systems to design complex objects by adding, subtracting, and inverting smooth surfaces.

An implicitly defined object in $\mathbb{R}^n$ is produced by an equation of the form $f(\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \mathbb{R}^n$, where f can be either a polynomial or any other real valued function. The implicit representation has the advantage that it allows rapid determination of whether a point lies inside ($f(\boldsymbol{x}) < 0$), outside ($f(\boldsymbol{x}) > 0$) or on ($f(\boldsymbol{x}) = 0$) the object. Despite the many other positive features of implicit descriptions of objects, there is one main drawback: The points building the object are defined as zero-set of f - an equation which is in general not explicitly resolvable. The computation of an approximation of this zero-set for visualization and collision detection has been the topic of many publications over the last two decades; finding solutions that are fast and guaranteed reliable is one of the subjects of recent research [26, 31, 32]. Existing solutions for the 3D case compute a polygonization or voxelization, or determine single points on the surface. Algorithms are based on simple space subdivision, marching cubes, particle systems, ray tracing and stochastic differential equations. Implicit Linear Interval Estimations (ILIEs) can be used to accelerate those algorithms that are based on subdivision and/or incidence tests.

Definition 3. *Let* $\mathcal{F} : f(\boldsymbol{x}) = 0,\ \boldsymbol{x} = (x_1, ..., x_n)^T \in \mathbb{R}^n$ *be the implicit definition of an object in* $\mathbb{R}^n$ *and*

$$L(\boldsymbol{x}) := \sum_{i=1}^{n} a_i x_i + J \tag{10}$$

with $J \in I\mathbb{R}$ *and* $a_i \in \mathbb{R}, i = 1, ..., n$.
The interval hyperplane segment inside the axis-aligned box $\mathbb{I} \in I\mathbb{R}^n$

$$\mathcal{L} := \{\boldsymbol{x} \in \mathbb{I} \mid 0 \in L(\boldsymbol{x})\}$$

is called the Implicit Linear Interval Estimation (ILIE) *of* $\mathcal{F}$ *on* $\mathbb{I}$*, iff for all* $\boldsymbol{x} \in (\mathcal{F} \cap \mathbb{I})$ *holds* $0 \in L(\boldsymbol{x})$.

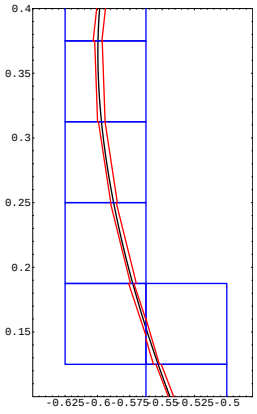

Fig. 18. ILIEs enclosing a curve.

Computation The computation of ILIEs follows roughly the same strategy as in the parametric case. The general recipe for generating ILIEs can be described as follows:

1. Compute any linear approximation $l_f(\boldsymbol{x})$ of $f(\boldsymbol{x})$ on a cell $\mathbb{I}$.
2. Estimate the approximation error with an interval J.
3. Combine both to obtain an ILIE $L_F :\ 0 \in l_f(\boldsymbol{x}) + J$ of F.

Again the linearization can be done using, for example, affine arithmetic or Taylor models. The characteristics of ILIEs are also similar to those of parametric LIEs: They are a kind of thick linearization of the object and the diameter of the interval part can be used as criterion for flatness. Furthermore, if affine arithmetic is used for the computation, low additional computational costs are required compared to a cell/object evaluation, singularities are not a problem and the ILIEs provide a tight enclosure due to implied Tchebycheff approximation.

Application of ILIEs to enumeration algorithms. A classic enumeration algorithm for implicit objects works in the following way:

- Define an initial cell (an axis-aligned box) where the object has to be detected.
- Test whether this box interferes with the object.
- If it does, subdivide and test sub-cells until the termination criterion is fulfilled.
- If it does not, the object does not intersect the cell; stop.

In this algorithm, the cell-object interference test is the most expensive and important part. ILIEs can help to optimize the whole process in the following ways (for a detailed description see [8]):

- The cell-object incidence test and the computation of the corresponding ILIE can be done in (almost) one step.

- The diameter of the interval part of the ILIE can be used as termination criterion.
- ILIEs allow *cell pruning*. An ILIE encloses the object in many cases much more tightly than the corresponding cell. An axis-aligned cell can be easily reduced to those parts containing the ILIE and the enclosed object using interval/affine arithmetic. (Iterated) cell pruning applied on each computed sub-cell reduces the number of necessary subdivisions significantly.
- Unnecessary cell-object tests can be reduced doing a pre-test with the ILIE of the mother cell.
- Cell pruning also allows highly effective adaptive subdivision strategies.

Implementation and experiments. Up to now ILIEs have only been implemented using affine arithmetic based a modification of the affine arithmetic package of van Iwaarden. To prove the usability of ILIEs they have been applied for reliable plotting of implicit curves [6] and the enumeration of implicit surfaces [8] (see also Figure 19). The results can be summarized as follows:

The introduction of ILIEs allowed a complete redefinition of classic enumeration algorithms. In many cases ILIEs provide much better enclosures than axis-aligned cells. The results are much better adapted to the topology of the object. The number of necessary subdivisions decreased substantially. The number of necessary ILIEs to represent a result with certain precision is radically less using ILIEs than using axis-aligned cells. Thus, if the subdivision is used as a basis for polygonization, many fewer polygons are necessary; if it is used as basis for collision detection, many fewer interference tests are necessary; and, if it is used as a basis for ray tracing, it enables the performance of rapid ray/plane tests with unique results. Figure 19 shows two examples where ILIEs have been applied to curve plotting and surface enumeration. For a detailed discussion and more examples the reader is referred to the two papers mentioned above.

5 Conclusion

In this paper we have discussed the impact of the exact scalar product, common interval arithmetic and its refinements, for example, affine arithmetic and Taylor models. These techniques provide a complete framework for modeling geometric structures with different levels of detail and accuracy. Octrees are adequate data types when only accurate rough bounds for distances are needed. The exact scalar product is crucial to derive tight a priori error bounds for distance computation between convex polyhedra. Certain classic distance algorithms can be enhanced by introducing standard interval arithmetic. Linear Interval Estimations are a natural and intuitive generalization directly connected to the representation of the enclosed object. LIEs provide tight and reliable bounds for parametric and implicit objects and are easy to compute and to intersect. The proposed methods allow the development of adaptive algorithms and the reduction of the number of necessary subdivision steps in bounding volumes, intersection and enumeration algorithms in robotics, geometric modeling and computer graphics.

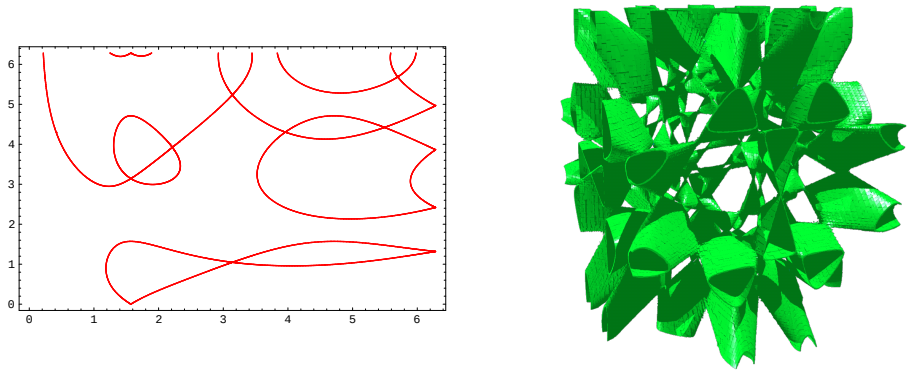

Fig. 19. The figure on the left shows the plot of a trigonometric implicit curve created using ILIEs. The figure on the right shows an enclosure of a Barth Decic built of ILIEs. The surface is algebraic degree 10. Notice, that in both examples singularities are properly enclosed by bounding volumes.

References

1. Alefeld, G., Herzberger, J.: Introduction to Interval Computations. Academic Press Inc, New York (1983)
2. Barnhill, R. E., Kersey, S.N.: A marching method for parametric surface/surface intersection. CAGD 7(1-4) (1990) 257–280
3. Beaumont, O.: Solving interval linear systems with linear programming techniques. Linear Algebra Appl. 281(1-3) (1998) 293–309
4. van den Bergen, G.: A Fast and Robust GJK Implementation for Collision Detection of Convex Objects. Journal of Graphics Tools, Vol. 4, No. 2, (1999) 7–25
5. Berz, B., Hofstätter, G.: Computation and application of Taylor polynomials with interval remainder bounds. Reliable Computing **4** (1998) 83–97
6. Bühler, K.: Fast and reliable plotting of implicit curves. In Proc. of Workshop on Uncertainty in Geom. Comp., Sheffield University, UK, July 2001. Kluwer (2002)
7. Bühler, K.: Linear interval estimations for parametric objects. (Proc. of Eurographics 2001) Computer Graphics Forum 20(3) (2001)
8. Bühler, K.: Implicit linear interval estimations. In Proc. Spring Conference of Computer Graphics, SCCG, Budmerice (SK). April 2002. ACM Siggraph (2002)
9. Chung, Tat Leung.: An Efficient Collision Detection Algorithm for Polytopes in Virtual Environments. M. Phil. Thesis, University of Hong Kong (1996)
10. Cohen, J., Lin, M. C., Manocha, D., Ponamgi, K.: COLLIDE: An interactive and exact collision detection system for large-scale environments. Proc. Symp. of Interactive 3D Graphics (1995) 189–196
11. Comba, J. L. D., Stolfi, J.: Affine Arithmetic and Its Applications to Computer Graphics. Proceedings of the VI Sibgrapi. Recife, Brazil, October (1993)
12. Dyllong, E., Luther, W., Otten, W.: An Accurate Distance-Computation Algorithm for Convex Polyhedra. Reliable Computing, Vol. 5 (1999) 241–253
13. Dyllong, E., Luther, W.: An accurate computation of the distance between a point and a polyhedron. M. Berveiller, A. K. Louis and C. Fressengeas (eds.), ZAMM Zeitschrift für angewandte Mathematik und Mechanik, Vol. 80 of GAMM 99 Annual Meeting, Metz, France, April 12-16, WILEY-VCH, Berlin (2000) S771–S772

14. Dyllong, E., Luther, W.: Distance calculation between a point and a NURBS surface. Curve and Surface Design: Saint-Malo 1999, P.-J. Laurent, P. Sablonnière, L. L. Schumaker (eds.), Vanderbilt University Press, Nashville, TN (2000) 55–62
15. Dyllong, E., Luther, W.: The GJK Distance Algorithm: An Interval Version for Incremental Motions. Submitted to Scan 2002 Proceedings.
16. Eckardt, U.: Digital Lines and Digital Convexity. Hamburger Beiträge zur Angewandten Mathematik 164 (2001)
17. Ehmann, St. A., Lin, Ming C.: Accurate and Fast Proximity Queries between Polyhedra Using Surface Decomposition. (Proc. of Eurographics 2001) Computer Graphics Forum 20(3) (2001)
18. Fausten, D., Luther, W.: Verified solutions of systems of nonlinear polynomial equations. In Walter Krämer, Jürgen Wolff v. Gudenberg (eds.), Scientific Computing, Validated Numerics, Interval Methods. Kluwer (2001) 141–152
19. Gilbert, E. G., Johnson, D. W., Keerthi, S. S.: A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE Journal of Robotics and Automation, Vol. 4 (1988) 193–203
20. Gilbert E. G., Ong, Chong Jin: Fast Versions of the Gilbert-Johnson-Keethi Distance Algorithm: Additional Results and Comparisons. IEEE Trans. Robotics and Automation, Vol. 17, No. 4 (2001) 531–539
21. Guibas, L. J., Salesin, D., Stolfi, J.: Epsilon Geometry: Building Robust Algorithms from Imprecise Computations. Symposium on Computational Geometry (1989) 208–217
22. van Iwaarden, R., Stolfi, J.: Affine arithmetic software (1997)
23. Knüppel, O.: PROFIL/BIAS – A fast interval library. Computing, Vol. 53 (1994) 277–288
24. Lin, M. C., Canny, J. F.: A fast algorithm for incremental distance calculation. Proc. IEEE Int. Conf. on Robotics and Automation, (1991) 1008–1014
25. Lohner, R.: On the Ubiquity of the Wrapping Effect in the Computation of the Error Bounds, in U. Kulisch and R. Lohner and A. Facius (eds.), Perspectives on Enclosure Methods, Springer Wien New York, (2001) 201–217.
26. Martin, R., Shou, H., Voiculescu, I., Bowyer, A., Wang, G.: Comparison of Interval Methods for Plotting Algebraic Curves. CAGD 19(7) (2002) 553–587
27. Mirtich, B.: V-Clip: Fast and robust polyhedral collision detection. ACM Trans. Graphics, Vol. 17 (1998) 177–208
28. Neumaier, A.: Interval Methods for Systems of Equations, Vol. 37 of Encyclopedia of Mathematics and its Applications. Cambridge University Press (1990)
29. Sato, Y., Hirita, M., Maruyama, T., Arita, Y.: Efficient collision detection for convex and nonconvex objects. Proc. IEEE Int. Conf. Robotics and Automation (Minneapolis, MN) (1996) 771–777
30. Sherbrooke E. C., Patrikalakis, N. M.: Computation of the solution of nonlinear polynomial systems. Computer Aided Geometric Design **10** (1993) 379–405
31. Stolte, N., Kaufman, A.: Parallel spatial enumeration of implicit surfaces using interval arithmetic for octree generation and its direct visualization. In Implicit Surfaces'98 (1998) 81–88
32. Voiculescu, I., Berchtold, J., Bowyer, A, Martin, R. and Zhang, Q.: Interval and affine arithmetic for surface location of power- and Bernstein-Form polynomials. In: Cipolla, R., Martin, R. (eds.): The Mathematics of Surfaces, Vol. IX. Springer-Verlag (2000) 410–423
33. Yap, C. K.: Robust geometric computation. In: Goodman, J. E., O'Rourke, J. (eds.): CRC Handbook in Computational Geometry. CRC Press (1997) 653–668