Volume xx (200y), Number z, pp. 1–9

Approximating Parametric Curves with Strip Trees using Affine Arithmetic

Luiz Henrique de Figueiredo,[1] Jorge Stolfi,[2] and Luiz Velho[1]

[1]IMPA–Instituto de Matemática Pura e Aplicada
Estrada Dona Castorina 110, 22461-320 Rio de Janeiro, RJ, Brazil
`lhf@impa.br` `lvelho@impa.br`

[2]Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)
Caixa Postal 6176, 13083-970 Campinas, SP, Brazil
`stolfi@ic.unicamp.br`

Abstract
We show how to use affine arithmetic to represent a parametric curve with a strip tree. The required bounding rectangles for pieces of the curve are computed by exploiting the linear correlation information given by affine arithmetic. As an application, we show how to compute approximate distance fields for parametric curves.

Keywords: multi-resolution, distance fields, interval arithmetic, geometric modeling

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Curve, surface, solid, and object representations; G.1.2 [Numerical Analysis]: Approximation of surfaces and contours; G.1.0 [Numerical Analysis]: Interval arithmetic

1. Introduction

Strip trees were introduced by Ballard[1] as a multi-resolution data structure for representing polygonal curves. The main concept in strip trees is to represent each piece of the curve by a *bounding rectangle* that contains the piece. When this is done in a hierarchical fashion — starting from the whole curve, subdividing the curve at suitable points, and going down to individual edges — we get a tree of rectangles, each rectangle containing a piece of the curve.

Ballard[1] described how the multi-resolution representation provided by strip trees can be used to solve several practical problems efficiently, including displaying a curve at a given resolution, intersecting two curves, computing the length of a curve at a given resolution, testing the proximity of a point to a curve, and testing whether a point is inside a region bounded by a curve.

The heart of the strip tree representation algorithm is the computation of a rectangle bounding a piece $\mathcal{P}$ of the polygonal curve. Ballard used the smallest rectangle with a side parallel to the line segment joining the two endpoints of $\mathcal{P}$. This rectangle is not necessarily the smallest rectangle containing $\mathcal{P}$, but it is easy to compute and the resulting strip tree has good performance in practice.

In this paper, we show how to compute a strip tree representation for a general parametric curve, using affine arithmetic[2] to find good bounding rectangles. (Figure 1 shows an example of a rectangle covering of a parametric curve computed with our algorithm.) In Section 2 we review the details of how polygonal curves are represented by strip trees as described by Ballard[1]. In Section 3 we discuss what primitives are needed to extend strip trees to general parametric curves. In Section 4 we briefly review affine arithmetic and explain how the information it provides can be used to compute bounding rectangles for pieces of parametric curves; as mentioned earlier, this is the heart of the strip tree representation. Section 5 contains some examples of strip trees computed with this algorithm.

Guéziec[3] recently proposed an efficient algorithm for computing the distance of points in the plane to a polygonal curve. In Section 6 we show that his algorithm can be extended to use the strip trees computed in Section 4 and also

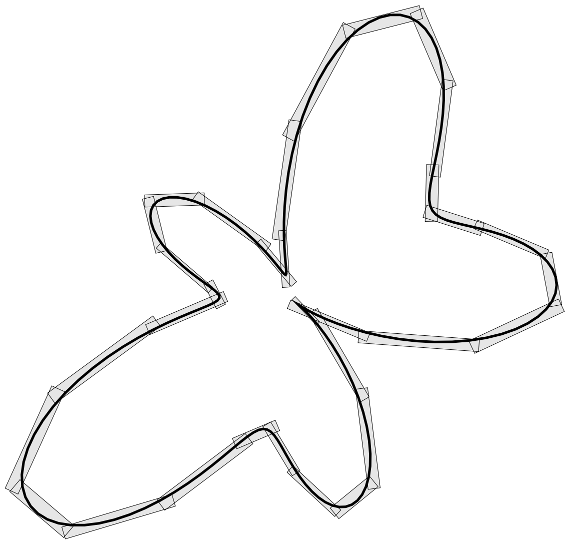

Figure 1: *Approximating a curve with rectangles.*

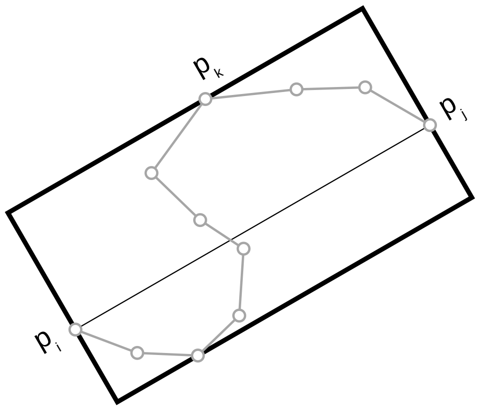

Figure 2: *Geometry of bounding rectangles for strip trees.*

how this extension can be used to provide an implicit approximation to parametric curves using distance fields.

Section 7 contains our conclusions and discusses how this work can be extended to surfaces.

2. Strip trees

Let $\mathcal{C} = p_1 \dots p_n$ be a polygonal curve. As explained by Ballard[1] (see also chapter 4 of Samet's book[4]), a *strip tree* for $\mathcal{C}$ is a binary tree whose nodes represent pieces of the curve by bounding rectangles. (For best performance, these rectangles are *not* aligned to the coordinate axes.) The root of the strip tree represents the whole curve. The children of a node represent two halves of the piece of the curve represented by the node. Leaf nodes correspond to individual edges $p_i p_{i+1}$.

A strip tree for the polygonal curve $\mathcal{C}$ can be built in a top-down fashion by starting with the whole curve $\mathcal{C} = p_1 \dots p_n$, finding a bounding rectangle for it, choosing a *splitting point* p_k, and then recursively building strip trees for the two halves $p_1 \dots p_k$ and $p_k \dots p_n$. (Ballard[1] also discusses a non-recursive bottom-up algorithm, but concludes that the top-down algorithm generates tighter approximations.)

Ballard[1] computed a bounding rectangle for a piece $\mathcal{P} = p_i \dots p_j$ by choosing the smallest rectangle with a side parallel to the line segment $L = p_i p_j$ that joins the two extremes of $\mathcal{P}$. To compute this rectangle, one has to find the points p_k with $i < k < j$ that are farthest from L on both sides. The point at maximum distance to L is chosen as the splitting point. See Figure 2.

The bounding rectangle computed by Ballard is not necessarily the smallest rectangle containing the piece $\mathcal{P}$, but it is easy to compute. (The smallest rectangle containing a polygonal line $\mathcal{P}$ can be computed in linear time by first computing the convex hull of $\mathcal{P}$[5] and then computing its minimal bounding rectangle[6], but the total time is likely to be larger than using Ballard's heuristics. On the other hand, smaller rectangles may mean better overall performance of the strip tree. As far as we know, the impact of using the best rectangles in strip trees has not yet been studied[7].) Moreover, in practice the splitting point is usually near the middle of $\mathcal{P}$, and so the whole strip tree is usually computed in time $O(n \log n)$. (However, in the worst case it can take quadratic time[8].)

3. Strip trees for parametric curves

The notion of strip trees can be carried over to a general parametric curve $\mathcal{C}$ given by $\gamma\colon I \subseteq \mathbf{R} \to \mathbf{R}^2$ as long as we know how to compute bounding rectangles for pieces of $\mathcal{C}$ and how to choose splitting points. Assuming these two basic operations, here is a skeleton algorithm for computing a strip tree for a curve $\mathcal{C} = \gamma(I)$; the strip tree is the result of strip-tree(I):

```
strip-tree($T$):
    $B \leftarrow$ bounding rectangle for $\mathcal{P} = \gamma(T)$
    if good-enough($T,B$) then
        return $\langle T, B, \text{nil}, \text{nil} \rangle$
    else
        split $T$ into $T_1$ and $T_2$
        return $\langle T, B, \text{strip-tree}(T_1), \text{strip-tree}(T_2) \rangle$
```

Here, good-enough(T,B) is an application-dependent predicate that decides when to stop recursion. A suitable predicate for pure geometric approximation is simply

```
good-enough($T,B$):
    return width($B$) $< \varepsilon$
```

where ε is a user-selected tolerance.

Each node in the tree computed by strip-tree has four fields $\langle T, B, L, R\rangle$, where $T \subseteq I$ is the parametric interval corresponding to the piece $\mathcal{P} = \gamma(T)$ represented by the node, B is a rectangle containing $\mathcal{P}$, and L and R are the children of the node. The efficiency of the strip tree representation is closely related to how well B approximates $\mathcal{P}$.

A bounding rectangle B for $\mathcal{P} = \gamma(T)$ could be computed by sampling the interval T and finding the minimal rectangle containing the convex hull of the corresponding sample points on $\mathcal{P}$[6]. However, it would be hard to decide how fine the sampling should be to be faithful to the curve; although heuristics are available[9], they do not guarantee the correctness of the bounding box. In Section 4 we show how to use affine arithmetic to compute bounding rectangles that are guaranteed to contain $\mathcal{P}$. Like the bounding rectangles used by Ballard for polygonal curves, the rectangles computed with affine arithmetic are not always the best possible, but they converge rapidly to the curve.

Choosing splitting points as Ballard did would require finding the parameter $t \in T$ corresponding to the point at maximum distance to line segment joining the two extremes of $\mathcal{P}$. This is a global optimization problem that is best avoided. Another good candidate for the splitting point is the midpoint of $\mathcal{P}$ with respect to arc length. This is the solution adopted by Günther and Wong[10] in their *arc tree*. The midpoint of $\mathcal{P}$ may be difficult to compute; a much simpler choice for the splitting point is the point in $\mathcal{P}$ corresponding to the midpoint of T. We shall adopt this choice in the sequel.

4. Affine arithmetic

Affine arithmetic (AA) was introduced in SIBGRAPI'93[2] as a tool for validated numerics[11]. Since then, AA has been applied to the robust solution of several graphics problems[12, 13, 14, 15, 16], where it has successfully replaced interval arithmetic[17].

In AA, a quantity x is represented as an *affine form*,

$$\hat{x} = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n,$$

which is a polynomial of degree 1 in *noise symbols* ε_i, whose values are unknown but assumed to lie in the interval $[-1,+1]$. From this representation, we conclude that the quantity x lies in the interval $[x_0 - r_x, x_0 + r_x]$, where $r_x = |x_1| + \cdots + |x_n|$. Thus, quantities in AA also naturally represent *intervals*, and so AA can replace interval arithmetic[17].

The basic arithmetic operations and elementary functions can be extended to handle affine forms[11]. Affine operations (translation, scale, addition, and subtraction) are straightforward. Non-affine operations, such as multiplication, square root, and trigonometric functions, use a good affine approximation plus an error term (which creates a new noise symbol). So, the result of a function f applied to affine forms is another affine form

$$\hat{f} = f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n + f_k\varepsilon_k,$$

where $|f_k|$ bounds the error committed when replacing f by the affine approximation $f_0 + f_1\varepsilon_1 + \cdots + f_n\varepsilon_n$.

Once the basic operations and functions have been extended to affine forms, one can automatically compute arbitrarily complex functions with AA by decomposing them into a sequence of elementary steps. This can be done very conveniently in languages that support operator overloading, such as C++, but it can also be done easily by hand or with the aid of a precompiler[18]. (We used a precompiler written in Lua[19].)

Exploiting correlations given by affine arithmetic

One key feature of affine arithmetic is that it is able to handle correlations between quantities. We now explain how this is done, as a preparation to showing how to compute a bounding rectangle for a piece $\gamma(T)$ of a parametric curve $\mathcal{C} = \gamma(I)$, where $T = [a,b] \subseteq I$.

Start by writing $\gamma(t) = (x(t), y(t))$. Next, convert $t \in T$ to an affine form

$$\hat{t} = t_0 + t_1\varepsilon_1,$$

where $t_0 = (b+a)/2$ and $t_1 = (b-a)/2$. Note that $\hat{t}$ ranges over T when ε_1 ranges over $[-1,+1]$. Next, compute the coordinate functions x and y at $\hat{t}$ using affine arithmetic, obtaining two affine forms:

$$\hat{x} = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n$$
$$\hat{y} = y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n$$

which, for convenience and without loss of generality, we have written as having the same number of noise symbols.

Now comes the key observation: *The values of x and y are* not *independent* — they have a partial correlation each time their affine forms $\hat{x}$ and $\hat{y}$ share a noise symbol ε_i with non-zero coefficients x_i and y_i.

Taken separately, the equations above say that x is in the interval $X = [x_0 - r_x, x_0 + r_x]$ and y is in the interval $Y = [y_0 - r_y, y_0 + r_y]$, and so the point (x,y) is in the rectangle $R = X \times Y$. However, because of the implicit correlations due to the sharing of noise symbols, the point (x,y) is actually in a smaller region $K \subseteq R$. The narrower this region, the more x and y are correlated. The region K is the image of the hypercube $[-1,+1]^n \ni (\varepsilon_1, \ldots, \varepsilon_n)$ under the affine transformation $\mathbf{R}^n \to \mathbf{R}^2$ given by the matrix

$$\left[\begin{array}{c|ccc} x_0 & x_1 & \cdots & x_n \\ y_0 & y_1 & \cdots & y_n \end{array}\right].$$

So, K is *zonotope*[20], that is, a convex polygon that is centrally symmetric with respect to the point (x_0, y_0), which is the image of the origin $(0, \ldots, 0) \in \mathbf{R}^n$.

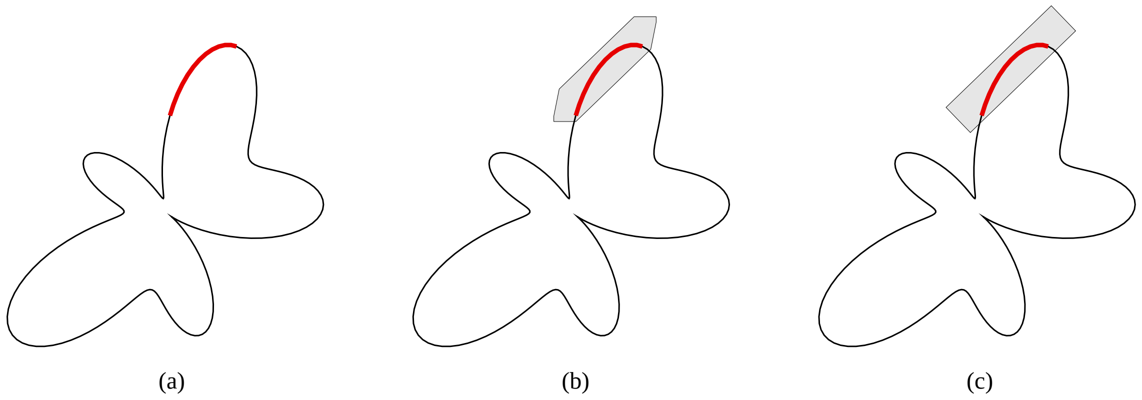

Figure 3: *Steps in approximating a piece of a parametric curve: (a) curve piece $\mathcal{P} = \gamma(T)$; (b) zonotope K given by computing $\mathcal{P}$ with AA; (c) rectangle of minimal width containing K.*

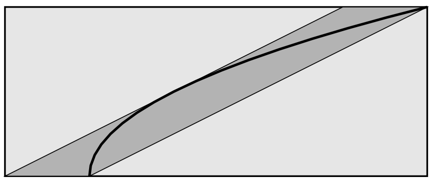

Figure 4: *Zonotope approximation of a parabola segment.*

As an extreme example, take $\mathcal{C}$ to be the straight line segment given by $\gamma(t) = (1,1) + t(4,6)$, for $t \in [0,1]$. Then

$$\hat{t} = 0.5 + 0.5\varepsilon_1$$
$$\hat{x} = 1 + 4\hat{t} = 3 + 2\varepsilon_1$$
$$\hat{y} = 1 + 6\hat{t} = 4 + 3\varepsilon_1$$

These equations say that the point (x,y) is exactly on the line segment, even though, taken separately, they say only that $x \in [1,5]$ and $y \in [1,7]$. This result is exact because AA handles affine operations without truncation errors, and in this case also without rounding errors.

For a less extreme example, take $\mathcal{C}$ to be the parabolic segment given by $\gamma(t) = (t^2,t)$, for $t \in [0,2]$. Then computing x and y with AA gives

$$\hat{x} = \hat{t}^2 = 1.5 + 2\varepsilon_1 + 0.5\varepsilon_2$$
$$\hat{y} = \hat{t} \quad = 1 + 1\varepsilon_1$$

The new noise symbol ε_2 comes from the (non-affine) squaring operation: the second-order term ε_1^2, whose range is $[0,1]$, is replaced by $0.5 + 0.5\varepsilon_2$, losing its correlation with ε_1. Nevertheless, information on first-order correlation between x and y is preserved because $\hat{x}$ and $\hat{y}$ share ε_1. This information is sufficient to yield a good approximation for the joint range of x and y. Indeed, taken separately, these equations say only that x lies in the interval $X = [-1,4]$ and y lies in $Y = [0,2]$. However, taken jointly, they say that (x,y) lies in the dark parallelogram shown in Figure 4, which is substantially smaller than the rectangle $X \times Y = [-1,4] \times [0,2]$, shown in light grey, which would be produced by standard interval arithmetic.

Computing bounding rectangles with affine arithmetic

We compute a bounding rectangle for $\mathcal{P} = \gamma(T)$ in two steps, as illustrated in Figure 3:

1. compute the zonotope $K \supseteq \gamma(T)$ given by AA;
2. compute a bounding rectangle for K.

For step 1, we have to explain how to find the sides of K from the affine forms $\hat{x}$ and $\hat{y}$ given by AA as a representation of $\gamma(T)$. First, order the $2n$ vectors (x_i,y_i) and $(-x_i,-y_i)$ for $i = 1,\dots,n$ circularly around the origin. Let $v_0, \dots, v_{2n-1}$ be the sorted list. Then, K is the polygon whose $2n$ vertices $p_0, \dots, p_{2n-1}$ are given by

$$p_i = \sum_{k=0}^{n-1} v_{k+i},$$

with indices computed modulo $2n$.

For step 2, we choose to compute the rectangle of minimal width containing K. In general, the rectangle of minimal width containing a convex polygon has one side collinear with some side of the polygon, and can be found in linear time[6]. However, as mentioned earlier, the convex polygons K given by AA are not generic: they are symmetric with respect to their center (x_0,y_0). This fact greatly simplifies finding the rectangle of minimal width containing K: the width is the smallest distance from the center to a side of K. The rectangle of minimal width will have *two* sides overlapping the sides of the polygon closest to the center.

This concludes our explanation of how AA can be used to compute bounding rectangles for pieces of parametric curves. We are now ready to show some examples of the resulting strip trees.

5. Examples of strip-tree approximations

We now present strip approximations for some parametric curves computed with affine arithmetic as described in Section 4. Because it is difficult to show the complete trees and the geometry of the curve, the pictures will show only the nodes at certain levels of the tree. Hopefully, this will show how well the strip trees approximate the curves.

Figure 5 shows levels 0 to 3 of a strip tree for the circle given by $x = \cos(t), y = \sin(t), t \in [0, 2\pi]$. Figure 6 shows levels 5 to 8 of a strip tree for the spiral given in polar coordinates (r,t) by $r = 0.1t, t \in [0, 45]$. Figure 7 shows levels 3 to 6 of a strip tree for the "butterfly" given in polar coordinates by $r = \sin(2t) + \sin(5t) + 2, t \in [0, 2\pi]$. Note how fast the approximations converge to the curve.

6. Distance from a point to a curve

In this section, we show how a strip tree for a parametric curve $\mathcal{C}$ can be used to compute efficiently a guaranteed approximation to the distance $d(p, \mathcal{C})$ of a point $p \in \mathbf{R}^2$ to $\mathcal{C}$:

$$d(p, \mathcal{C}) = \min\{d(p, \gamma(t)) : t \in I\}.$$

The method we describe below is a special case of the branch-and-bound optimization algorithm[21] and is similar to the methods described by Ballard[1] and Guéziec[3]. The main difference is that in our case the endpoints of each arc $\mathcal{P}$ of $\mathcal{C}$ are not known; all we know is that $\mathcal{P}$ is contained in a rectangle B. Therefore, all we have is that $d(p, \mathcal{P}) \in D(p, B)$, where $D(p, B)$ is the *distance interval* from p to B, namely, the range of $d(p, q)$ as q ranges over B:

$$D(p, B) = \{d(p, q) : q \in B\}.$$

Ballard's version of the point-to-curve distance algorithm used a simple recursive enumeration of the strip tree with cutoffs[1]. Like Guéziec[3] and branch-and-bound optimization algorithms, we use a more efficient enumeration based on a priority queue Q. The entries of Q are pairs $\langle N, d_{inf} \rangle$, where N is a node $\langle T, B, L, R \rangle$ from the strip tree (as described in Section 3) and d_{inf} is the lower bound of the distance interval $D(p, B)$. The priority queue Q is ordered such that the entry with minimum d_{inf} is at the front.

The distance algorithm starts by inserting the root N_0 of the strip tree into the queue Q. Then, elements with smallest lower distance bound d_{inf} are repeatedly extracted from Q until a leaf element is reached. Non-leaf elements are split into two sub-elements, which are inserted into the queue. The algorithm also maintains a global bound d_{sup} to the distance $d(p, \mathcal{C})$. Any entry $\langle N, d_{inf} \rangle$ with $d_{inf} > d_{sup}$ could be deleted from Q; for simplicity, this optimization is not shown in the skeleton algorithm below.

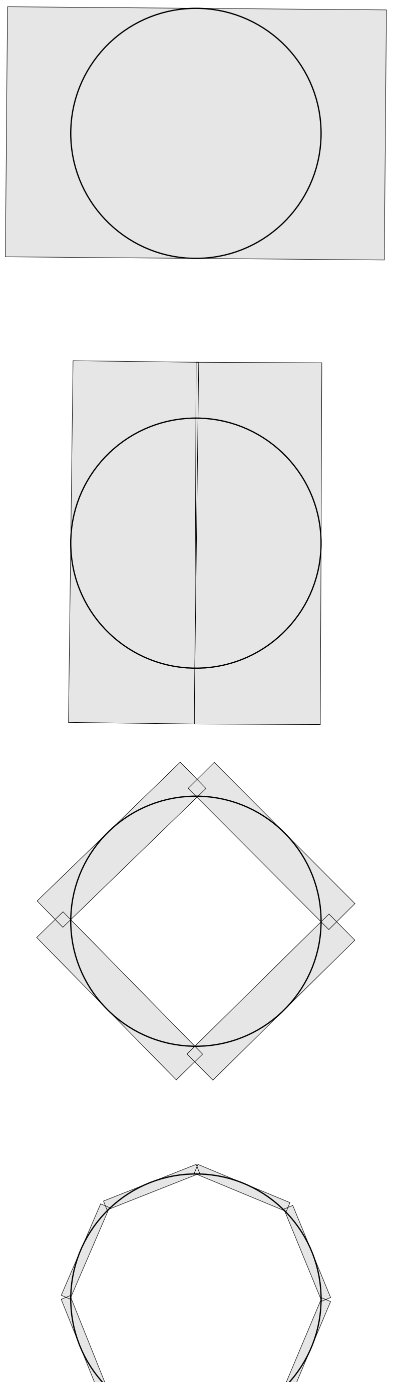
Figure 5: *Strip approximations for a circle.*

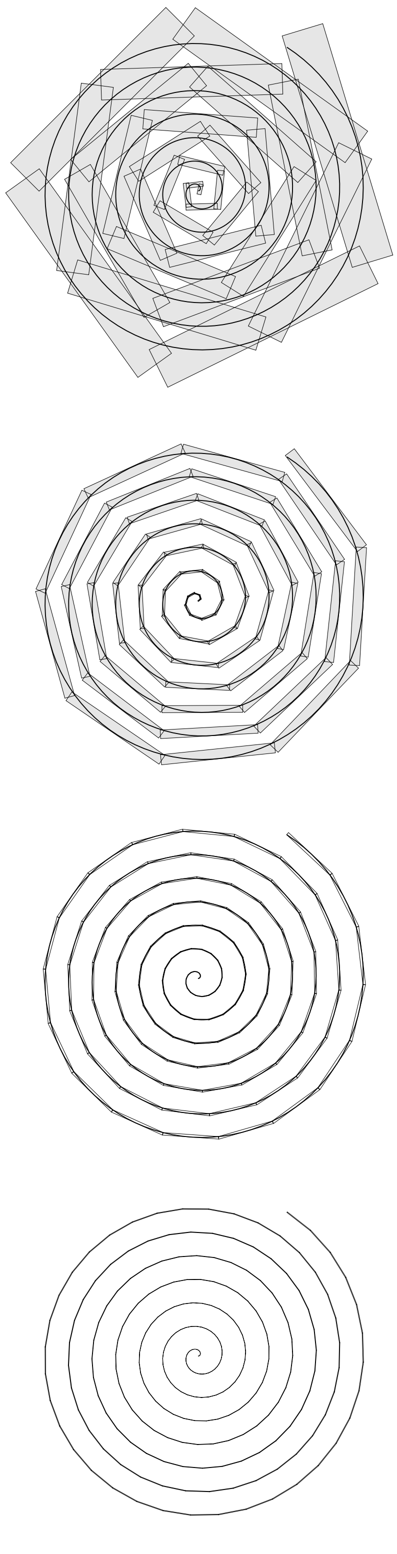

Figure 6: *Strip approximations for a spiral.*

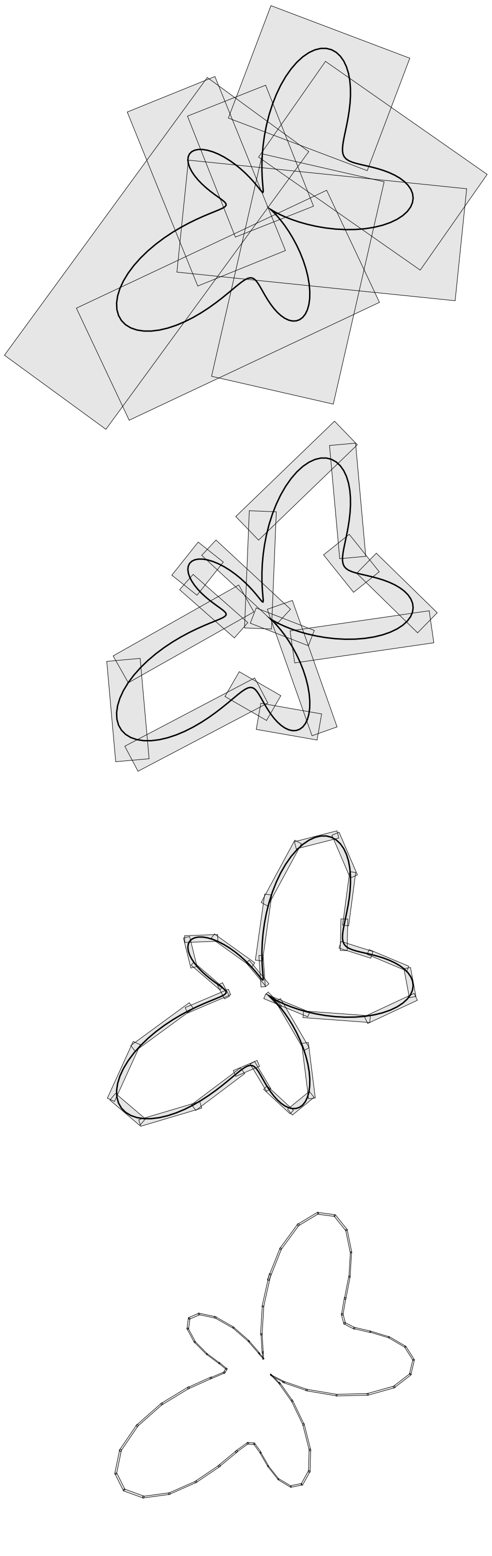

Figure 7: *Strip approximations for a butterfly.*

```
distance-estimate$(p, N_0)$:
    $Q \leftarrow$ empty queue
    $d_{sup} \leftarrow +\infty$
    insert$(Q, \langle N_0, d_{sup}\rangle, p)$
    while $Q \neq \emptyset$ do
        $\langle N, d_{inf}\rangle \leftarrow$ extract$(Q)$
        if is-leaf$(N)$ then
            return $[d_{inf}, d_{sup}]$
        else
            new-entry$(Q, N.L, p)$
            new-entry$(Q, N.R, p)$
```

The auxiliary procedure new-entry adds an entry to the queue, updating d_{sup}:

```
new-entry$(Q, N, p)$:
    compute $[d_{lo}, d_{hi}] \leftarrow D(p, N.B)$
    if $d_{hi} < d_{sup}$ then $d_{sup} \leftarrow d_{hi}$
    if $d_{lo} \leq d_{sup}$ then insert $\langle N, d_{lo}\rangle$ into $Q$
```

Figure 8 shows an approximate distance field for the butterfly used in Section 5; it was computed with the distance-estimate algorithm using a strip tree of uniform depth 7 and d_{inf} as an approximate distance. In this image, the darker the point, the farther it is from the curve. Due to the Mach band effect, the medial axis appears in dark, being the locus of first-order discontinuity of the distance. Figure 8 also shows some approximate offsets computed from this distance field (cf. ref.).

Computing the distance interval

We now show in detail how to compute the distance interval $D(p, B)$ needed in procedure insert. We assume that each rectangle B of the strip tree is represented by its center o and two orthogonal vectors u and v, each parallel to one side of B and whose length is half of the length of that side (see Figure 9). Then the distance interval $D(p, B)$ can be computed with the following algorithm:

```
$D(p, B)$:
    $r \leftarrow p - o$
    $\alpha \leftarrow \langle r, u\rangle / \langle u, u\rangle$
    $\beta \leftarrow \langle r, v\rangle / \langle v, v\rangle$
    $s_{hi} \leftarrow r - \text{sign}(\alpha) \cdot u - \text{sign}(\beta) \cdot v$
    $s_{lo} \leftarrow r + \text{clamp}(\alpha) \cdot u + \text{clamp}(\beta) \cdot v$
    return $[\,|s_{lo}|, |s_{hi}|\,]$
```

where

$$\text{clamp}(x) = \min(1, \max(-1, x)).$$

A quicker approximation, after Guéziec[3], is to extend the rectangle B with two semicircular caps, as shown in Figure 10. Since the resulting capped rectangle B^* contains B, we have $D(p, B) \subseteq D(p, B^*)$. The rectangles in a strip tree are usually very thin compared to their length, and so the difference between $D(p, B)$ and $D(p, B^*)$ is usually very small.

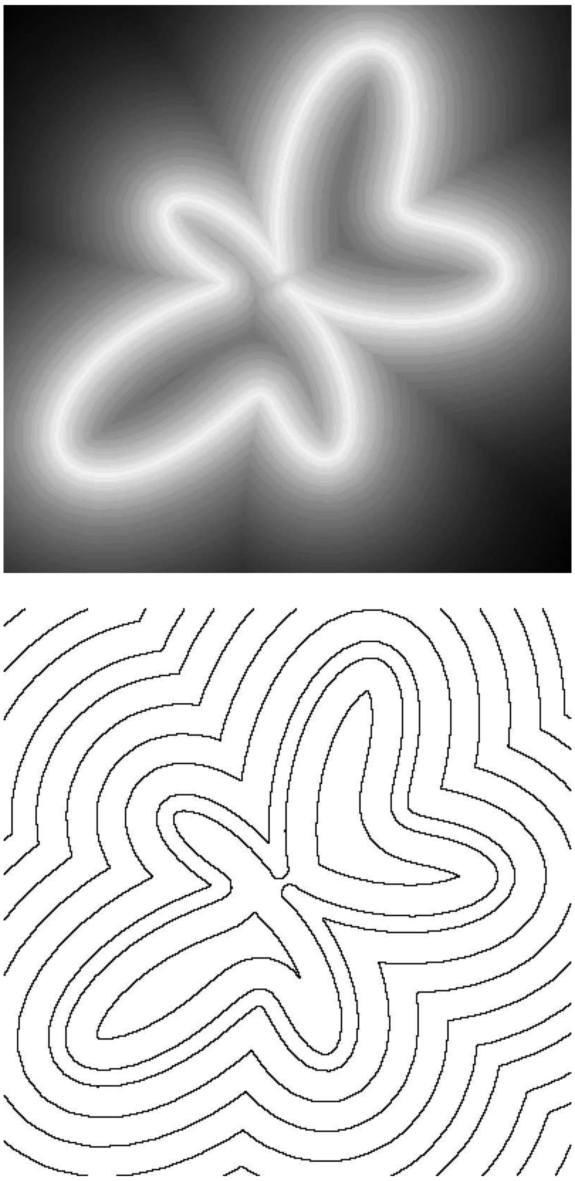

Figure 8: *Approximate distance field and offsets.*

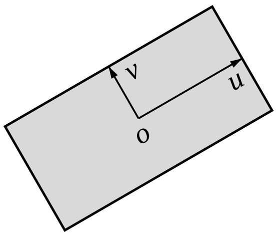

Figure 9: *Geometric parameters defining a rectangle.*

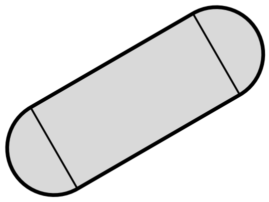

Figure 10: *Capped rectangle.*

We used $D(p,B^*)$ instead of $D(p,B)$ to generate the approximate distance field shown in Figure 8. The distance interval $D(p,B^*)$ can be computed as follows:

```
$D(p,B^*)$:
    $r \leftarrow p - o$
    $\alpha \leftarrow \langle r,u\rangle / \langle u,u\rangle$
    $s_{hi} \leftarrow r - \mathrm{sign}(\alpha)\cdot u$
    $s_{lo} \leftarrow r + \mathrm{clamp}(\alpha)\cdot u$
    $\varepsilon \leftarrow \sqrt{\langle v,v\rangle}$
    return $[\max(0,|s_{lo}|-\varepsilon), |s_{hi}|+\varepsilon]$
```

7. Conclusion

We have shown how affine arithmetic readily provides guaranteed bounding rectangles for pieces of parametric curves. Although these rectangles are not necessarily the smallest ones, they are aligned with the curve and so converge rapidly to the curve. Interval arithmetic can provide guaranteed bounding rectangles, but they are always aligned with the axes, and so converge more slowly to the curve. Compare the two approximations in Figure 11. For a recent approach restricted to splines, see ref. .

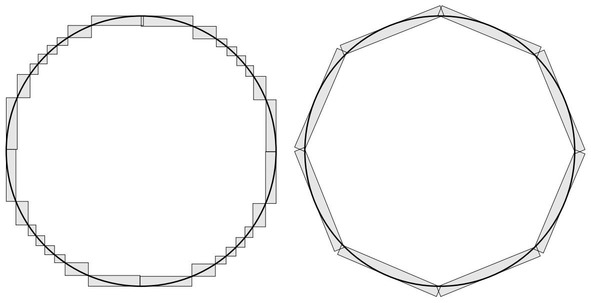

Figure 11: *Rectangle approximation of a circle computed with interval arithmetic (left) and with affine arithmetic (right), using the same tolerance.*

From the bounding rectangles provided by affine arithmetic, we built a multi-resolution representation that generalizes to general parametric curves the strip tree representation introduced by Ballard[1] for polygonal curves. This representation is a key component in solving efficiently several computer graphics problems. As an example, we showed how to compute approximate distance fields for parametric curves by combining the multi-resolution representation computed with affine arithmetic with an algorithm similar to one described recently by Guéziec[3] (see also ref.). Those distance fields provide an implicit approximation to a parametric curve and fit well into sophisticated implicit representations, such as adaptive distance fields[22].

Future work

Further work will be aimed at surfaces. The method for computing bounding zonotopes described in Section 4 is easily extended to handle parametric surfaces: We simply use affine arithmetic to compute a three-dimensional zonotope containing $\sigma(W)$, where $\sigma\colon \Omega \subseteq \mathbf{R}^2 \to \mathbf{R}^3$ defines the surface and $W \subseteq \Omega$. However, several important details are different.

First, computing a rectangular box containing the zonotope is much more complicated in $\mathbf{R}^3$ than in $\mathbf{R}^2$. Second, and more important, is the *form* of W. In the curve case, the domain of γ was an interval, which was naturally decomposed in other intervals. In the surface case, even if the domain Ω is a rectangle, there are several choices for decomposing it into subregions W. The simplest form for W is a rectangle $U \times V$. This is also the best form for computing with affine arithmetic. However, this choice leads to a quadtree decomposition of Ω, which is not always suitable for geometric processing because it is topologically inconsistent[23] and can generate cracks in the surface. Quadtrees can be converted to triangulations[23], but triangles are not suitable for computing with affine arithmetic, which prefers zonotopes. We plan to use 4-8 meshes[24]: they are hierarchical, topologically consistent, and can be seen to contain only rectangles (aligned with the axes or rotated 45 degrees), which are suitable for computing with affine arithmetic.

Acknowledgements. The authors are partially supported by CNPq research grants. L. H. de Figueiredo and L. Velho are members of Visgraf, the computer graphics laboratory at IMPA, which is sponsored by CNPq, FAPERJ, FINEP, and IBM Brasil.

References

1. D. H. Ballard, “Strip trees, a hierarchical representation for curves”, *Communications of the ACM*, **24**(5), pp. 310–321 (1981).
2. J. L. D. Comba and J. Stolfi, “Affine arithmetic and its applications to computer graphics”, in *Proceedings of SIBGRAPI’93*, pp. 9–18, (October 1993).
3. A. Guéziec, ““Meshsweeper”: dynamic point-to-polygonal-mesh distance and applications”, *IEEE Transactions on Visualization and Computer Graphics*, **7**(1), pp. 47–61 (2001).
4. H. Samet, *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, (1990).

5. A. Melkman, “On-line construction of the convex hull of a simple polyline”, *Information Processing Letters*, **25**(1), pp. 11–12 (1987).

6. G. T. Toussaint, “Solving geometric problems with the rotating calipers”, in *Proc. IEEE MELECON ’83*, pp. A10.02/1–4, (1983). Available at `http://cgm.cs.mcgill.ca/~godfried/publications/calipers.ps.gz`.

7. O. Günther and S. Dominguez, “Hierarchical schemes for curve representation”, *IEEE Computer Graphics and Applications*, **13**(3), pp. 55–63 (1993).

8. J. Hershberger and J. Snoeyink, “Speeding up the Douglas-Peucker line simplification algorithm”, in *Proc. 5th Intl. Symp. on Spatial Data Handling*, pp. 134–143, (1992). Also available as TR-92-07, CS Dept., U. of British Columbia, `http://www.cs.ubc.ca/tr/1992/TR-92-07`.

9. L. H. de Figueiredo, “Adaptive sampling of parametric curves”, in *Graphics Gems V* (A. Paeth, ed.), pp. 173–178, Boston: Academic Press, (1995).

10. O. Günther and E. Wong, “The arc tree: an approximation scheme to represent arbitrary curved shapes”, *Computer Vision, Graphics, and Image Processing*, **51**(3), pp. 313–337 (1990).

11. J. Stolfi and L. H. de Figueiredo, *Self-Validated Numerical Methods and Applications*. Rio de Janeiro: Monograph for 21st Brazilian Mathematics Colloquium, IMPA, (1997). Available at `ftp://ftp.tecgraf.puc-rio.br/pub/lhf/doc/cbm97.ps.gz`.

12. L. H. de Figueiredo and J. Stolfi, “Adaptive enumeration of implicit surfaces with affine arithmetic”, *Computer Graphics Forum*, **15**(5), pp. 287–296 (1996).

13. L. H. de Figueiredo, “Surface intersection using affine arithmetic”, in *Proceedings of Graphics Interface’96*, pp. 168–175, (May 1996).

14. A. de Cusatis Jr., L. H. de Figueiredo, and M. Gattass, “Interval methods for ray casting implicit surfaces with affine arithmetic”, in *Proceedings of SIBGRAPI’99*, pp. 65–71, IEEE Press, (October 1999).

15. W. Heidrich, P. Slusallek, and H.-P. Seidel, “Sampling procedural shaders using affine arithmetic”, *ACM Transactions on Graphics*, **17**(3), pp. 158–176 (1998).

16. W. Heidrich and H.-P. Seidel, “Ray-tracing procedural displacement shaders”, in *Graphics Interface ’98*, pp. 8–16, (June 1998).

17. R. E. Moore, *Interval Analysis*. Prentice-Hall, (1966).

18. F. D. Crary, “A versatile precompiler for nonstandard arithmetics”, *ACM Transactions on Mathematical Software*, **5**(2), pp. 204–217 (1979).

19. R. Ierusalimschy, L. H. de Figueiredo, and W. Celes, “Lua: an extensible extension language”, *Software: Practice & Experience*, **26**(6), pp. 635–652 (1996).

20. G. M. Ziegler, *Lectures on polytopes*. Graduate Texts in Mathematics, 152, Springer-Verlag, (1995).

21. C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, (1982).

22. S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones, “Adaptively sampled distance fields: A general representation of shape for computer graphics”, *Proceedings of SIGGRAPH 2000*, pp. 249–254 (2000).

23. B. von Herzen and A. Barr, “Accurate triangulations of deformed, intersecting surfaces”, *Computer Graphics*, **21**(4), pp. 103–110 (1987). (Proceedings of SIGGRAPH ’87).

24. L. Velho and D. Zorin, “4-8 subdivision”, *Computer-Aided Geometric Design*, **18**(5), pp. 397–427 (2001).