Approximating implicit curves on triangulations with affine arithmetic

Afonso Paiva Filipe de Carvalho Nascimento
ICMC, USP, São Carlos

Luiz Henrique de Figueiredo
IMPA, Rio de Janeiro

Jorge Stolfi
IC, Unicamp, Campinas

***Abstract*—We present an adaptive method for computing a robust polygonal approximation of an implicit curve in the plane that uses affine arithmetic to identify regions where the curve lies inside a thin strip. Unlike other interval methods, even those based on affine arithmetic, our method works on triangulations, not only on rectangular quadtrees.**

***Keywords*-implicit curves; polygonal approximation; interval methods;**

I. Introduction

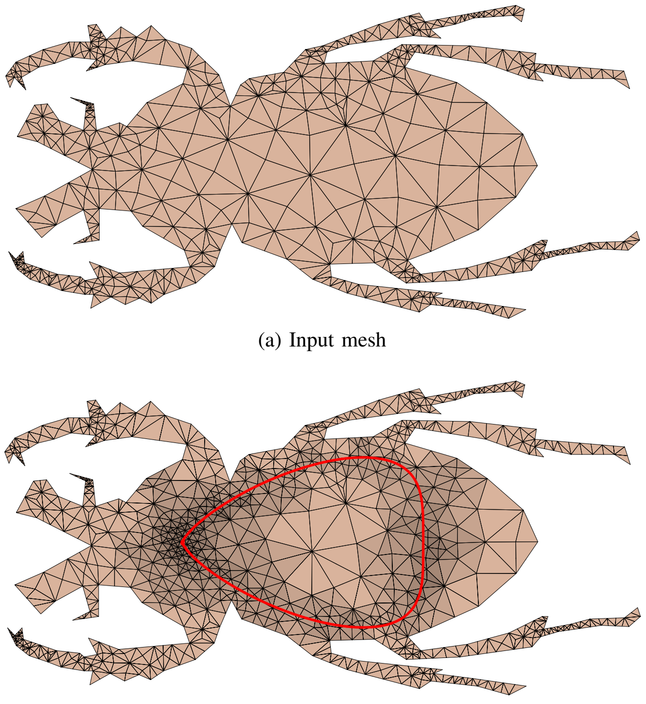
Several interval methods have been proposed for robustly approximating an implicit curve given on the plane by an equation $f(x,y)=0$. These methods typically use a rectangular quadtree to explore a region of interest recursively and adaptively, using interval estimates for the values of f (and sometimes of its gradient) on a rectangular cell as a quadtree subdivision criterion. Some methods have used affine arithmetic (AA) [1] to improve the convergence of the quadtree but none has exploited the additional geometric information provided by AA and none has worked on triangulations. While all interval methods can compute interval estimates on rectangular cells, classical interval arithmetic cannot handle triangles naturally.

In this paper, we describe an interval method for adaptively approximating an implicit curve on a refinable triangular decomposition of the region of interest using the geometric information provided by AA as a flatness criterion to stop the recursion. Fig. 1 shows an example of our method in action. Note how the decomposition is more refined around regions of higher curvature in the implicit curve.

After briefly reviewing some of the related work in Section II, we recall the main concepts in AA in Section III and explain how AA can work on triangular domains and how to extract geometric estimates for the location of the curve in the form of a strip in Section IV. This is the basis of an adaptive method that can be used on triangulations, which we present in Section V. We discuss some examples in Section VI and we report our conclusions and suggest directions for future work in Section VII.

II. Related work

Dobkin et al. [2] described in detail a continuation method for polygonal approximation of implicit curves in regular triangular grids generated by reflections. Since the grid is regular, their approximation is not adaptive. The selection of the grid resolution is left to the user. Persiano et al. [3] presented a general scheme for adaptive triangulation refinements which they applied to polygonal approximation of implicit curves in triangular grids.


(a) Input mesh

(b) Adaptive mesh

Fig. 1. Our method in action for the *pear* curve given by $4y^4=(x+1)^3(1-x)$ on an unstructured triangle mesh: (a) input coarse mesh and (b) adaptively refined mesh and polygonal approximation (red) computed by our method.

Suffern and Fackerell [4] were probably the first to apply interval methods for plotting implicit curves using quadtrees. Mitchell [5] revisited their work and helped to spread the word on interval methods for computer graphics. Lopes et al. [6] presented an interval method for polygonal approximation of implicit curves that uses interval estimates of the gradient for finding approximations that are both spatially and geometrically adaptive, in the sense that it uses larger cells when the curve is approximately flat. Their method is in the same spirit as ours, except that they did not use or exploit AA and worked only with rectangular quadtrees.

Comba and Stolfi [7] introduced AA and showed an example of how it can perform better than classical interval arithmetic when plotting implicit curves. Martin et al. [8] compared the performance of several interval methods for plotting algebraic curves using quadtrees, including methods based on AA. Neither paper exploited the geometric information given by AA. This has been done for ray tracing implicit surfaces by Cusatis et al. [9] and for approximating parametric curves by Figueiredo et al. [10], but as far as we know has not yet been done for polygonal approximation of implicit curves.

Bühler [11], [12] proposed a reliable cell pruning method based on a linearization of implicit objects derived from AA. In addition to reducing the number of enclosure cells, this method provides a tight piecewise linear covering adapted to the topology of the object instead of a covering using overestimated axis-aligned bounding boxes. This approach is in the same spirit as our own, but it uses rectangular cells only and can generate approximations with cracks across cells.

III. Affine arithmetic

In this section, we briefly recall the main concepts in AA. For details, see [1].

In AA, a real quantity z is represented by an *affine form*:

$$\widehat{z} = z_0 + z_1\varepsilon_1 + z_2\varepsilon_2 + \cdots + z_n\varepsilon_n$$

where z_i are real numbers and ε_i are *noise symbols* which vary in $[-1,1]$ and represent independent sources of uncertainty. From this representation, one deduces an *interval estimate* for the value of z:

$$z \in [\widehat{z}] := [z_0 - \delta, z_0 + \delta]$$

where $\delta = |z_1| + \cdots + |z_n|$. More importantly, by design affine forms can share noise symbols and thus may be not completely independent. Quadratic convergence and the explicit representation of first-order partial correlations are the main features of AA, which are absent in classical interval arithmetic [13]. These features allow more efficient methods in several cases, especially when the geometry of AA approximations is exploited [9], [10].

There are simple formulas for operating with affine forms. The formulas for affine operations (addition, subtraction, scalar multiplication, and scalar translation) are immediate, because affine forms represent these operations exactly (except for rounding errors in floating-point arithmetic). The formulas for non-affine operations (multiplication, integer powers, square root, and other elementary functions) rely on a good affine approximation with an explicit error term, as explained in detail elsewhere [1], [14]. By combining the formulas for these basic operations, one can evaluate any complicated formula on affine forms. As in other extended arithmetics, this is especially convenient to implement automatically using operator overloading.

IV. Bounding implicit curves with strips

Affine forms have a rich geometry. We shall now describe how to use AA to compute a strip of parallel lines that contains the piece of the curve given implicitly by $f(x,y) = 0$ in axis-aligned rectangles, arbitrary parallelograms, and triangles. This computation is the basis of our adaptive approximation method, which we shall present in Section V.

A. On rectangles

In the simplest setting, we have a rectangular domain $\Omega = [a,b] \times [c,d]$. Assuming that f is given by a mathematical expression in x and y, all we need to do to evaluate f in AA is to represent x and y with appropriate affine forms:

$$\widehat{x} = x_0 + x_1\varepsilon_1, \qquad x_0 = \frac{a+b}{2}, \quad x_1 = \frac{b-a}{2}$$

$$\widehat{y} = y_0 + y_2\varepsilon_2, \qquad y_0 = \frac{c+d}{2}, \quad y_2 = \frac{d-c}{2}$$

Note that x and y use different noise symbols because they vary independently in Ω.

The result of evaluating f on Ω using AA is an affine form

$$\widehat{f} = f_0 + f_1\varepsilon_1 + f_2\varepsilon_2 + \cdots + f_n\varepsilon_n$$

where $\varepsilon_3, \ldots, \varepsilon_n$ are noise symbols created during the evaluation of non-affine operations that occur in the expression of f, including rounding in floating-point arithmetic. A first-order approximation to the value of f on Ω is given by the *principal terms* $f_0 + f_1\varepsilon_1 + f_2\varepsilon_2$, which directly relate $f(x,y)$ with the input variables x and y. The other terms are second-order terms and can be condensed into a single term $f_3\varepsilon_3$, where $f_3 = |f_3| + \cdots + |f_n|$. (For simplicity, we have reused ε_3 and f_3 here.) In summary, the value of f on Ω is represented by an affine form with three noise symbols:

$$\widehat{f} = f_0 + f_1\varepsilon_1 + f_2\varepsilon_2 + f_3\varepsilon_3$$

In particular, for each $(x,y) \in \Omega$, the value $f(x,y)$ is in the interval $[\widehat{f}]$. If this interval does not contain 0, then the curve does not pass through Ω. This test, called an *absence oracle* by Lopes et al. [6], is the basis for all previous interval methods for approximating an implicit curve using a rectangular quadtree.

The basis of our approach is that important geometric bounds can be extracted from the affine form $\widehat{f}$. Indeed, the affine approximation $\widehat{f}$ says that the graph of $z = f(x,y)$ over Ω is sandwiched between the two parallel planes

$$z = f_0 + f_1\varepsilon_1 + f_2\varepsilon_2 \pm f_3$$

which can be written in cartesian coordinates as

$$z = f_0 + \frac{f_1}{x_1}(x - x_0) + \frac{f_2}{y_2}(y - y_0) \pm f_3$$

by writing

$$\varepsilon_1 = \frac{x - x_0}{x_1}, \qquad \varepsilon_2 = \frac{y - y_0}{y_2}$$

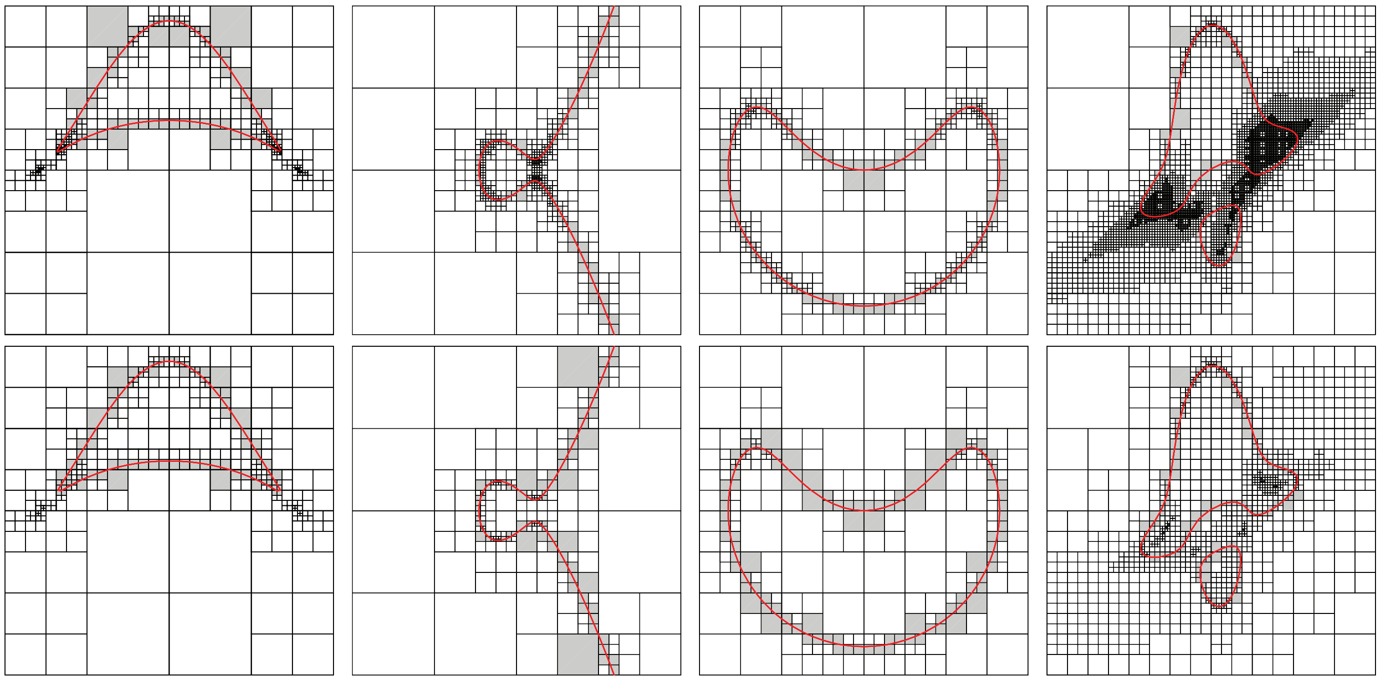
The region where f is zero in Ω is thus contained in the strip defined by the two parallel lines

$$0 = f_0 + \frac{f_1}{x_1}(x - x_0) + \frac{f_2}{y_2}(y - y_0) \pm f_3$$

whose width is

$$w = \frac{2f_3}{\sqrt{\left(\frac{f_1}{x_1}\right)^2 + \left(\frac{f_2}{y_2}\right)^2}}$$

When w is small, the curve $f(x,y) = 0$ varies little inside Ω. Our method uses this test as a subdivision criterion for an adaptive exploration of Ω: keep subdividing until w is small.



(a) Bicorn curve (b) Cubic curve (c) Clown smile curve (d) Taubin's quartic curve

Fig. 2. Approximating implicit curves (red) on quadtrees using the method of Lopes et al. [6] (top) and our method (bottom): (a) *bicorn*: $y^2(0.75^2-x^2)=(x^2+1.5y-0.75^2)^2$ on $\Omega=[-1.1,1.1]^2$, (b)) *cubic*: $y^2-x^3+x=0.5$ on $\Omega=[-5.21,5.21]^2$, (c) *clown smile*: $(y-x^2+1)^4+(x^2+y^2)^4=1$ on $\Omega=[-1.21,1.21]^2$ and quartic curve from Taubin's paper [15] on $\Omega=[-2.19,2.19]^2$.

TABLE I
PERFORMANCE AND STATISTICS FOR THE CURVES IN FIG. 2 (TIMES IN MILLISECS).

Fig.	**curve**	**error**	**depth**	**IA**				**AA**			
				gradient tol.	#cells visited	#leaves	time	width tol.	#cells visited	#leaves	time
2.a	bicorn	10^{-3}	8	0.80	605	124	11	0.03	461	98	9
2.b	cubic	10^{-2}	8	0.35	805	144	7	0.05	317	100	5
2.c	clown smile	10^{-2}	8	0.50	677	162	15	0.05	373	114	12
2.d	Taubin's quartic	10^{-3}	9	0.99	6937	341	394	0.05	1697	221	139

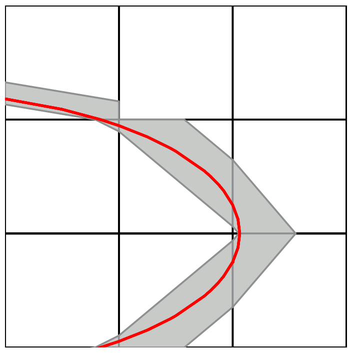


Fig. 3. Approximating a curve with strips computed by AA in rectangles.

Fig. 3 illustrates the approximation of an implicit curve with strips computed by AA in rectangles. Fig. 2 shows an example of the whole method in action. As mentioned in Section II and illustrated in Table I, our method compares favorably with the method of Lopes et al. [6] without having to compute derivatives because AA provides second-order approximations.

B. *On parallelograms*

We have seen how to represent an axis-aligned rectangle in AA. In fact, the objects that AA represents naturally are *zonotopes*: centrally symmetric convex polytopes, or equivalently, Minkowski sums of line segments. In particular, with two noise symbols AA can represent an arbitrary parallelogram. Indeed, the two affine forms

$$\widehat{x}=x_0+x_1\varepsilon_1+x_2\varepsilon_2, \qquad \widehat{y}=y_0+y_1\varepsilon_1+y_2\varepsilon_2$$

represent the parallelogram centered at (x_0,y_0) with sides parallel to $v_1=(x_1,y_1)$ and $v_2=(x_2,y_2)$ of twice their length. Note that now x and y are not completely independent, unless $x_2=0$ and $y_1=0$, when the parallelogram is then a rectangle with sides parallel to the coordinate axes.

Having represented a parallelogram P with two affine forms $\widehat{x}$ and $\widehat{y}$, we can compute an affine form $\widehat{f}$ representing f in P using AA on $\widehat{x}$ and $\widehat{y}$ and from $\widehat{f}$ find cartesian equations for a strip containing the curve $f(x,y)=0$ for $(x,y)\in P$. As seen above, in the rectangular case these equations were easy to find by writing ε_1 and ε_2 in terms of x and y. We can do exactly

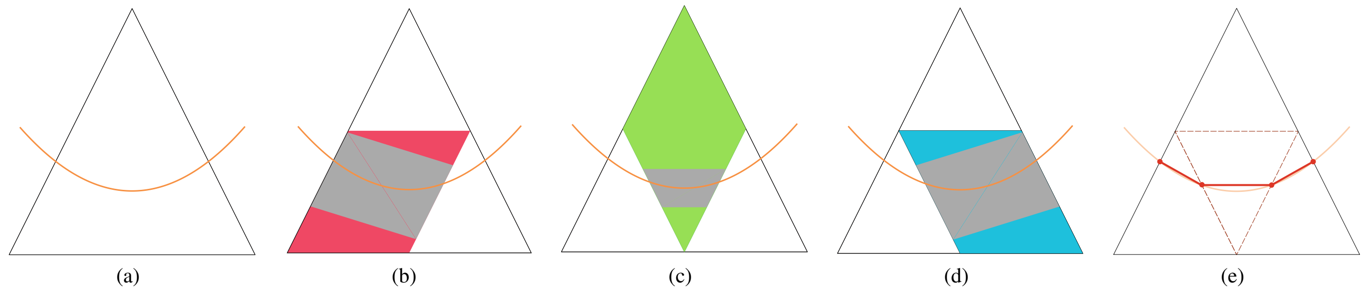


Fig. 4. Polygonal approximation of the parabola $y = 3x^2$ using our method. (a) Input triangle. (b), (c), and (d) evaluating $f(x,y) = 3x^2 - y$ with AA in each parallelogram and its associated strips (gray regions). (e) computing the polygonal approximation (red) using the sub-triangles of the input triangle.

the same thing in the parallelogram case, but now we need to invert a 2×2 matrix, converting

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} \cdot \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \end{bmatrix}$$

to

$$\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix}$$

and proceeding as before. The matrix above is invertible if and only if the vectors v_1 and v_2 are linearly independently, that is, when the parallelogram is not degenerate.

C. *On triangles*

Triangles are not zonotopes and so cannot be directly represented in AA. The easiest solution is to include a given triangle into a parallelogram and then evaluate AA there. As shown in Fig. 5, there are a few ways to do this, but conceptually this is wrong because we would be evaluating f outside its domain. Although this would work for functions defined on the whole plane, it does not work for smaller domains or surfaces.

Nevertheless, as shown in Fig. 6, it is easy to decompose a triangle into three overlapping parallelograms by joining the midpoints of the edges. We can then evaluate f with AA on each of these parallelograms. If the curve lies within a thin strip in each of these, then the curve lies within a thin "tube" inside the triangle, even if the curve does not lie within a single thin strip in the triangle. Fig. 4 illustrates this.

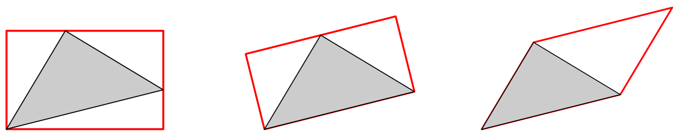


Fig. 5. Including a triangle into a parallelogram.

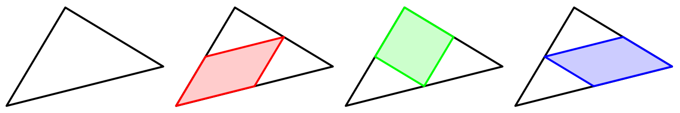


Fig. 6. Decomposing a triangle into parallelograms.

V. Our adaptive method

Once we know how to decide whether an implicit curve can be well approximated by a strip or a series of strips inside a cell (be it a rectangle, a parallelogram, or a triangle), we can use this test as a refinement criterion for adaptive subdivision: if the curve is well approximated inside the cell then we stop the subdivision; otherwise, we decompose the current cell into a number of subcells and recursively explore each subcell. We get a quadtree decomposition by subdividing the cell into four similar subcells by joining the midpoints of the edges of the cell. This is the easiest subdivision method.

Triangulations can offer us other subdivision methods. Our adaptive method does not care what subdivision method is used and can use whatever subdivision method is offered by the triangulation. In particular, our method can work seamlessly with dyadic splits, 4-8 meshes [16], $\sqrt{3}$-subdivision [17], and other subdivision schemes, and with adaptive finite-element meshes, whose subdivision depends on the results of numerical simulations. If the triangulation does not offer its own subdivision method, then we use the standard triangular quadtree subdivision described above.

As usual, the recursion stops when the curve does not cross the cell (as proved by the interval estimate $[\widehat{f}]$ not containing zero) or when the curve lies within a thin tube inside the cell, of width less than a user-supplied tolerance ε. In this case, we compute a polygonal approximation for the curve by finding the points where the curve crosses the boundary of the cell. In the case of triangular cells, we also find the points where the curve crosses the boundary of each subcell of the midpoint subdivision (regardless of the subdivision scheme used by the triangulation), since theses are the edges of the parallelograms used for testing the cell (see Fig. 4). As in the method of Lopes et al. [6], we use the bisection method to full precision on each edge for finding these crossings to ensure continuity of the polygonal approximation (see Fig. 7).

Our method is summarized in Fig. 8, in two versions: one for parallelograms, including rectangles, and one for triangles. The method starts by exploring each cell in an initial mesh decomposition of the region of interest. For rectangular quadtrees, this mesh typically contains a single cell. The *Approximate* procedure, which finds the actual polygonal approximation inside the cell, is the one described above.

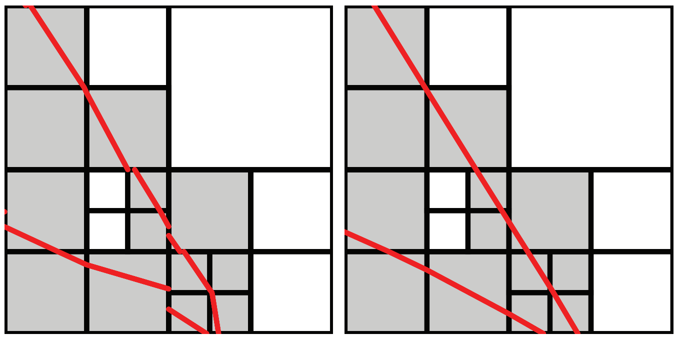


Fig. 7. Linear interpolation (left) produces cracks in the approximation, which are avoided by using the bisection method to full precision (right).

```
procedure Adaptive()
   for all cells $C$ in the base mesh do
      $Explore(C)$
   end
end

procedure $Explore(C)$                    (parallelograms)
   $\widehat{f} \leftarrow f(C)$ with AA
   if $0 \in [\widehat{f}]$ then
      $w \leftarrow$ width of $\widehat{f}$ in $C$
      if $w \leq \varepsilon$ then
         $Approximate(C)$
      else
         divide $C$ into subcells $C_i$
         for each $i$, $Explore(C_i)$
      end
   end
end

procedure $Explore(C)$                    (triangles)
   $P_1, P_2, P_3 \leftarrow$ parallelograms of $C$
   $\widehat{f_i} \leftarrow f(P_i)$ with AA
   if $0 \in [\widehat{f_i}]$ for some $i$ then
      $w_i \leftarrow$ width of $\widehat{f}$ in $P_i$
      if $w_i \leq \varepsilon$ for all $i$ then
         $Approximate(C)$
      else
         divide $C$ into subcells $C_i$
         for each $i$, $Explore(C_i)$
      end
   end
end
```

Fig. 8. Our method summarized.

VI. Results

Fig. 11 shows that our method is able to approximate an implicit curve on an unordered collection of triangles (i.e., a triangle soup) as well as on a triangle mesh with topology information, i.e., triangle mesh with connectivity between the triangles. We used midpoint refinement in the first example and 4-8 mesh refinement in the second one.

TABLE II
PERFORMANCE AND STATISTICS FOR OUR METHOD (TIMES IN MILLISECS).

Fig.	**curve**	**mesh**	**depth**	ε	**triangles** in	out	**time**
1	pear	beetle	4	1×10^{-2}	940	1771	280
9.a	Taubin	square	5	1×10^{0}	8	389	109
9.b	Taubin	square	6	1×10^{-1}	8	1466	401
9.c	Taubin	square	6	1×10^{-2}	8	2736	537
11.b	circle	Africa	3	1×10^{-3}	193	922	33
11.c	circle	Africa	5	1×10^{-3}	193	1106	59
12.a	bicorn	octagon	5	5×10^{-3}	126	1168	123
12.b	cubic	aircraft	3	5×10^{-3}	3530	4142	333
12.c	capricorn	tree	22	1×10^{-3}	1015	11678	8016
13	Pisces	circle	6	5×10^{-3}	101	2382	1124
14.a	heart	gear	3	3×10^{-3}	1424	3289	547
14.b	clown	K7	4	5×10^{-3}	1006	2134	391
14.c	irrational	eight	4	1×10^{-3}	1032	3897	454

Fig. 12 shows that our method is able to approximate implicit curves on both convex and non-convex 4-8 meshes. Note how our method tracks the main features of the curves, such as singularities and multiple components. Fig. 14 shows that our method approximates nicely both algebraic and non-algebraic curves on complex meshes.

Fig. 9 illustrates the effect of the geometric criteria in our method. This strategy produces well adapted curves controlled by the tolerance ε (see Table II), the only user-supplied parameter of our method. Moreover, Fig. 9 shows how the mesh concentrates on high-curvature regions as we increase ε.

Fig. 13 shows that our method adapts to the topology of the curve using a small set of triangles. This adaptability cannot be accomplished directly by the marching triangles algorithm. Indeed, the correct topology is reproduced by the marching triangles algorithm only after substantially refining the mesh, which significantly increases the number of triangles.

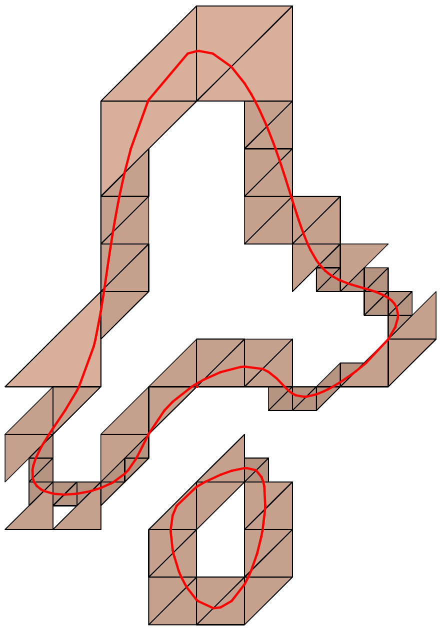
All results were generated on a 1.83GHz Intel Core 2 Duo with 3GB of RAM. Table II shows performance data, timings, and statistics for these computations.

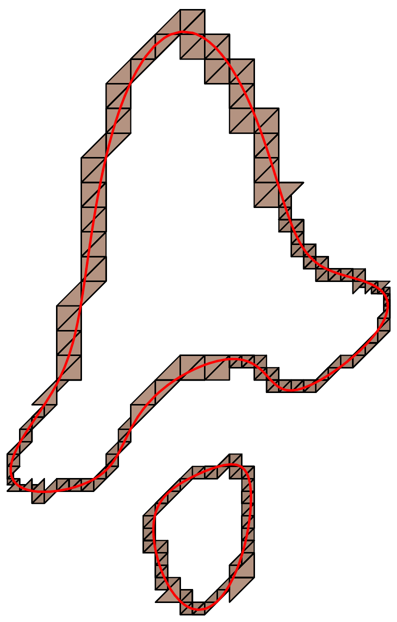
VII. Conclusion

Like the method of Lopes et al. [6], our method computes polygonal approximations of implicit curves that are both spatially and geometrically adaptive. Unlike their method, however, our method does not need to compute derivatives because AA provides second-order approximations. Moreover, our method works for rectangular and triangular decompositions, both structured and unstructured, can use any refinement scheme that the decompositions offer, and can provide its own refinement scheme otherwise.

One limitation of our method is that its results may depend on the quality of the initial decomposition, unless the mesh provides a refinement scheme that avoids bad-quality triangles.

By replacing matrix inverses with pseudo-inverses, we can extend the computation of Section IV to triangles in 3D space and so extend the method to approximating implicit curves on triangulated surfaces. Work along this line is already underway. Fig. 10 shows a preliminary result in this direction.


(a) $\varepsilon = 1$


(b) $\varepsilon = 0.1$

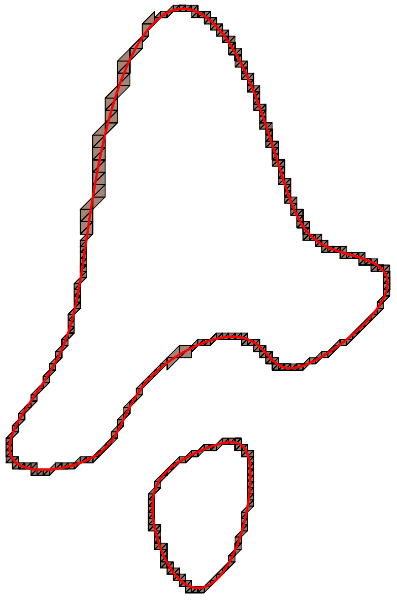

(c) $\varepsilon = 0.01$

Fig. 9. The effect of the geometric criteria on the Taubin's curve in a triangular quadtree. Our method tracks regions with high curvature when ε increases.

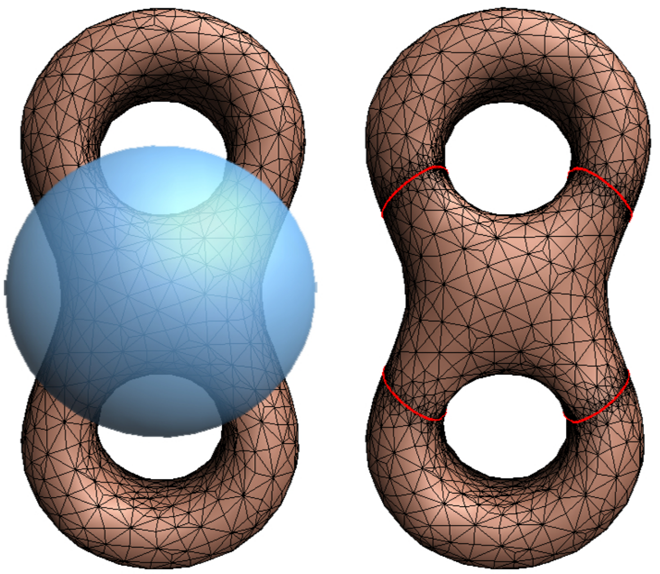


Fig. 10. Approximating an implicit curve on a triangulated surface: input bitorus mesh and the sphere $x^2+y^2+z^2=1$ (left) and the implicit curve (red) given by the restriction of $x^2+y^2+z^2=1$ on bitorus (right).

Another natural direction for future work is to extend our technique for approximating implicit surfaces in tetrahedral spatial decompositions.

Acknowledgments

The authors are partially supported by CNPq and FAPESP. This work was done in the Visgraf laboratory at IMPA, which is sponsored by CNPq, FAPERJ, FINEP, and IBM Brasil.

References

[1] L. H. de Figueiredo and J. Stolfi, "Affine arithmetic: Concepts and applications," *Numerical Algorithms*, vol. 37, pp. 147–158, 2004.
[2] D. P. Dobkin, S. V. F. Levy, W. P. Thurston, and A. R. Wilks, "Contour tracing by piecewise linear approximations," *ACM Transactions on Graphics*, vol. 9, no. 4, pp. 389–423, 1990.
[3] R. C. M. Persiano, J. ao L. D. Comba, and V. Barbalho, "An adaptive triangulation refinement scheme and construction," in *Proceedings of SIBGRAPI'93*, 1993, pp. 259–266.
[4] K. G. Suffern and E. D. Fackerell, "Interval methods in computer graphics," *Computers & Graphics*, vol. 15, no. 3, pp. 331–340, 1991.
[5] D. P. Mitchell, "Three applications of interval analysis in computer graphics," in *Frontiers in Rendering course notes*. SIGGRAPH'91, 1991, pp. 14–1–14–13.
[6] H. Lopes, J. B. Oliveira, and L. H. de Figueiredo, "Robust adaptive polygonal approximation of implicit curves," *Computers & Graphics*, vol. 26, no. 6, pp. 841–852, 2002.
[7] J. L. D. Comba and J. Stolfi, "Affine arithmetic and its applications to computer graphics," in *Proceedings of SIBGRAPI'93*, 1993, pp. 9–18.
[8] R. Martin, H. Shou, I. Voiculescu, A. Bowyer, and G. Wang, "Comparison of interval methods for plotting algebraic curves," *Computer Aided Geometric Design*, vol. 19, no. 7, pp. 553–587, 2002.
[9] 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*. IEEE Press, 1999, pp. 65–71.
[10] L. H. de Figueiredo, J. Stolfi, and L. Velho, "Approximating parametric curves with strip trees using affine arithmetic," *Computer Graphics Forum*, vol. 22, no. 2, pp. 171–179, 2003.
[11] K. Bühler, "Fast and reliable plotting of implicit curves," in *Uncertainty Geometric Computations*. Kluwer Academic, 2002, pp. 15–28.
[12] K. Bühler, "Implicit linear interval estimations," in *Proceedings of SCCG '02*. ACM, 2002, pp. 123–132.
[13] R. E. Moore, *Interval Analysis*. Prentice-Hall, 1966.
[14] J. Stolfi and L. H. de Figueiredo, *Self-Validated Numerical Methods and Applications*. 21st Brazilian Mathematics Colloquium, IMPA, 1997.
[15] G. Taubin, "Rasterizing algebraic curves and surfaces," *IEEE Computer Graphics and Applications*, vol. 14, pp. 14–23, 1994.
[16] L. Velho and D. Zorin, "4-8 subdivision," *Computer-Aided Geometric Design*, vol. 18, no. 5, pp. 397–427, 2001.
[17] L. Kobbelt, "$\sqrt{3}$-subdivision," in *Proceedings of SIGGRAPH '00*. ACM, 2000, pp. 103–112.

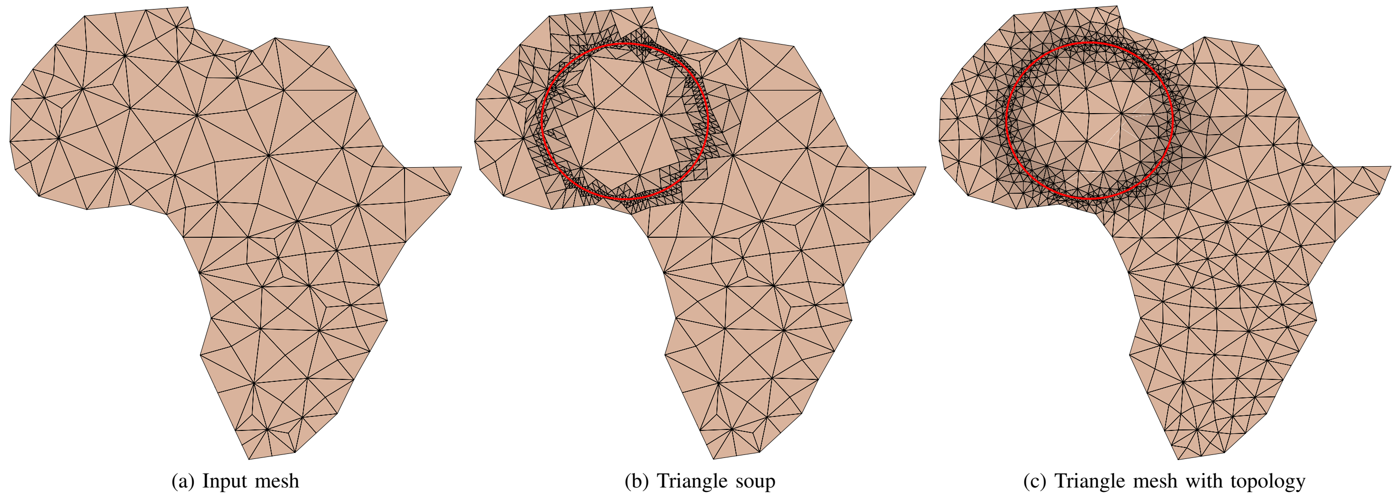


Fig. 11. Approximating the circle (red curve) $x^2 + y^2 = 1$ on adaptive meshes: (a) input coarse mesh, (b) each triangle of the triangle-soup is divided into four sub-triangles, adding new vertices in the midpoints of each edge and (c) using 4-8 mesh refinement.

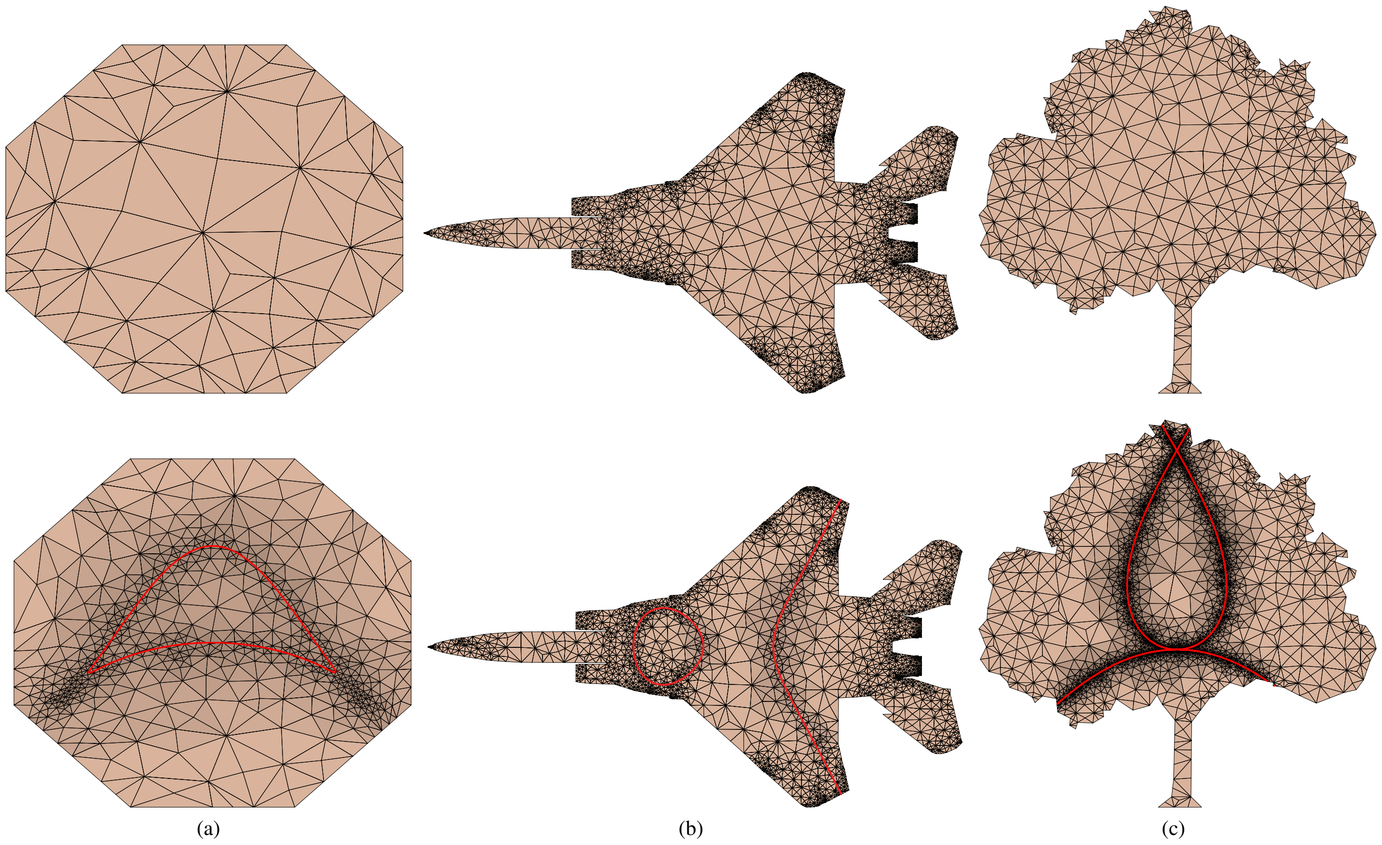


Fig. 12. Approximating implicit curves (red) on adaptive triangle meshes using 4-8 mesh refinement, input coarse mesh (top) and refined mesh (bottom): (a) bicorn curve on octagon model, (b) cubic curve $y^2 = x^3 + x$ on aircraft model and (c) the capricornoid curve given implicitly by $4x^2(x^2+y^2) - (2y - x^2 - y^2)^2 = 0$ on tree model.

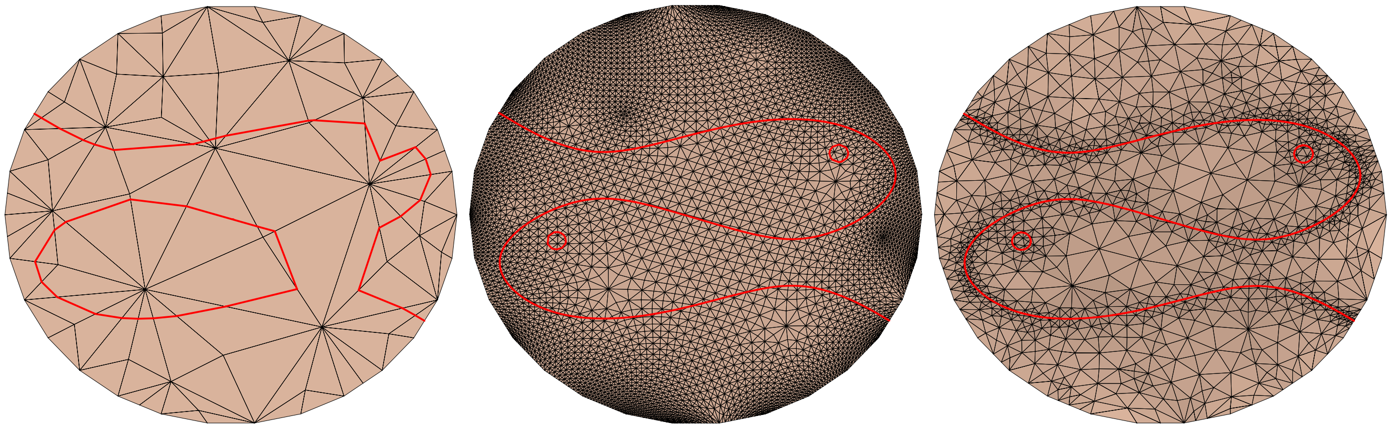


Fig. 13. In order to reproduce the Pisces logo (`http://www.geom.umn.edu/~fjw/pisces/`), the marching triangles algorithm alone generates a curve with wrong topology on coarse mesh (left) with 101 triangles and it just provides a satisfactory result on fine mesh with 12928 triangles after a refinement process (middle). While the adaptivity of our method tracks the geometry and topology of the curve on the mesh with 2382 triangles.

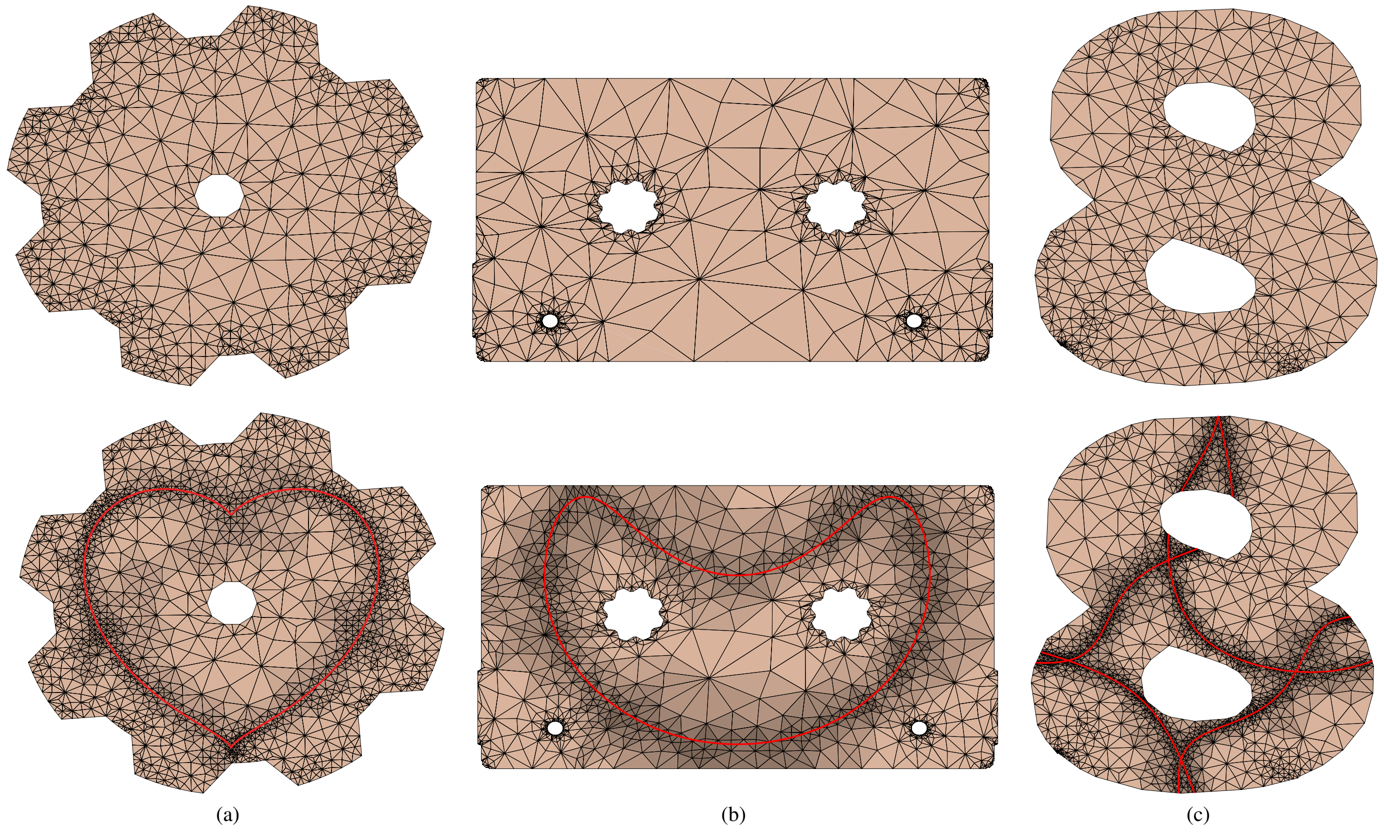


Fig. 14. Approximating implicit curves on complex meshes, input coarse mesh (top) and adaptive mesh (bottom): (a) the *heart* curve $(x^2+y^2-1)^3 = x^2y^3$ on gear model, (b) clown smile on K7 model, and (c) the transcendental curve given implicitly by $(xy+\cos(x+y))(xy+\sin(x+y)) = 0$ on eight model.