Approximating implicit curves on plane and surface triangulations with affine arithmetic

Filipe de Carvalho Nascimento[a], Afonso Paiva[a], Luiz Henrique de Figueiredo[b], Jorge Stolfi[c]

[a]*Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, Brazil*
[b]*Instituto Nacional de Matemática Pura e Aplicada, Rio de Janeiro, Brazil*
[c]*Instituto de Computação, Universidade Estadual de Campinas, Campinas, Brazil*

Abstract

We present a spatially and geometrically adaptive method for computing a robust polygonal approximation of an implicit curve defined on a planar region or on a triangulated surface. Our method 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 both rectangular and triangular decompositions and can use any refinement scheme that the decomposition offers.

Keywords: implicit curves, polygonal approximation, interval methods.

1. Introduction

The numerical solution of systems of non-linear equations in several variables is a key tool in geometric modeling and computer-aided geometric design [1]. In many applications, such as surface intersection and offset computation, the solution is not a set of isolated points but rather a curve or a surface. The simplest case is the solution of an equation $f(x,y) = 0$, which gives an implicit curve on the plane.

Computing a polygonal approximation of an implicit curve is a challenging problem because it is difficult to find points on the curve and also because the curve may have several connected components. Therefore, robust approximation algorithms must explore the whole region of interest to avoid missing any components of the curve. One approach for achieving robustness is to use interval methods [2, 3], which are able to probe the behavior of a function over whole regions instead of relying on point sampling. Interval methods lead naturally to spatially adaptive solutions that concentrate efforts near the curve.

Several interval methods have been proposed for robustly approximating an implicit curve on the plane (see §2). These methods explore a rectangular region of interest by decomposing it recursively and adaptively with a quadtree and using interval estimates for the values of f (and sometimes of its gradient) on a cell as a subdivision criterion.

Affine arithmetic (AA) [4] is a generalization of classical interval arithmetic that explicitly represents first-order partial correlations, which can improve the convergence of interval estimates. Some methods have used AA for approximating implicit curves, successfully exhibiting improved convergence, but none has exploited the additional geometric information provided by AA and none has worked on triangulations. Indeed, while all interval methods can compute interval estimates on rectangular cells, classical interval arithmetic cannot handle triangles naturally, except by enclosing them in axis-aligned rectangles. Thus, existing interval methods are restricted to rectangular regions. Moreover, to handle implicit curves on triangulated surfaces, these methods would have to use a 3d axis-aligned box containing each triangle, which is wasteful.

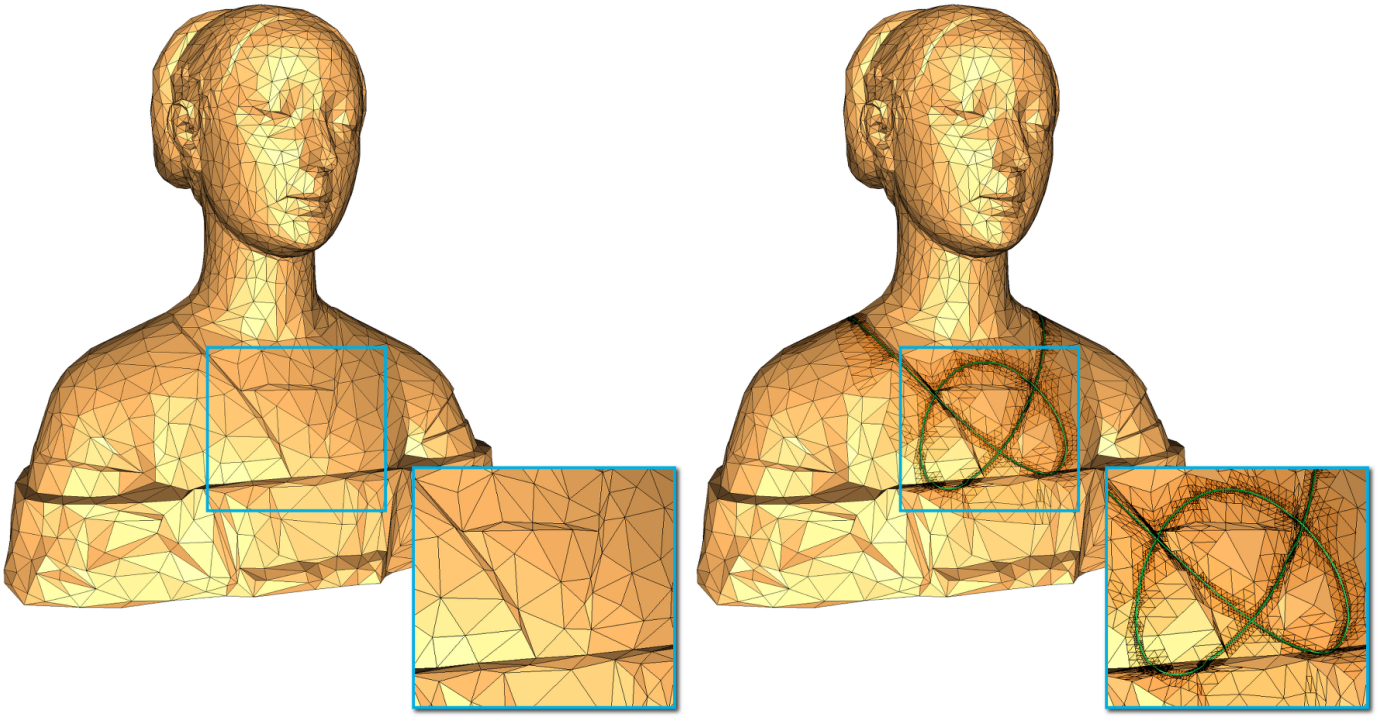
In this paper, we describe an interval method for adaptively approximating an implicit curve on a refinable triangular decomposition of the region of interest. Our method uses the geometric information provided by AA as a flatness criterion to stop the recursion and is thus both spatially and geometrically adaptive in the sense of Lopes et al. [5]. Our method can handle implicit curves given by algebraic or transcendental formulas, works on triangulated plane regions and surfaces of arbitrary genus, and can use any mesh refinement scheme. Fig. 1 shows an example of our method in action on a triangulated surface. Note how the mesh is refined near the implicit curve.

After briefly reviewing some of the related work in §2 and the main concepts of AA in §3, we explain in detail in §4 how to use AA to extract geometric estimates in the form of strips for the location of the curve in a triangle. This is the basis of an interval method that can be used on triangulations, both on the plane and on surfaces, which we present in §5. We discuss some examples of our method in action in §6 and we report our conclusions and suggest directions for future work in §7.

A previous version of this paper [6] focused on plane curves only. Here, we focus on curves on surfaces. We also discuss plane curves for motivation, simplicity of exposition, and completeness. In addition to the material on surfaces presented in §4.4 and §6, we include a performance comparison of the strategies for handling triangles with AA in §4.3 and an expanded and detailed explanation of how our method works in §5.

2. Related work

Dobkin et al. [7] 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. [8] presented a general scheme for adaptive triangulation refinement which they



(a) Input mesh (b) Adaptive mesh

Fig. 1: Our method in action for the *knot* curve given implicitly by $y^2(3+2y)-(x^2-1)^2=0$ on an unstructured triangle mesh: (a) input coarse mesh and (b) adaptively refined mesh and polygonal approximation (green) computed by our method.

applied to the polygonal approximation of implicit curves in triangular grids. These two methods work well for fine grids but they cannot claim robustness since they rely on point sampling.

Suffern and Fackerell [9] were probably the first to apply interval methods for plotting implicit curves adaptively using quadtrees. Mitchell [10] revisited their work and helped to spread the word on interval methods for computer graphics.

Snyder [11, 12] described a complete modeling system based on interval methods which included an adaptive algorithm for approximating implicit curves. His algorithm uses interval estimates for the gradient of the function defining the curve to incorporate global parametrizability in the subdivision criteria. The leaf cells in the resulting decomposition can vary in size, even though the approximation is not explicitly adapted to the curvature of the curve.

Lopes et al. [5] presented an interval method for polygonal approximation of implicit curves that uses interval estimates of the gradient for finding an approximation that is 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 it works only with rectangular quadtrees on the plane and relies on automatic differentiation, which can be avoided by using AA, as we shall show.

Comba and Stolfi [13] introduced AA and showed an example of how it can perform better than classical interval arithmetic when plotting implicit curves. Further examples were given by Figueiredo and Stolfi [14]. Martin et al. [15] compared the performance of several interval methods for plotting algebraic curves using quadtrees, including methods based on AA and variants. None of these papers exploited the additional geometric information provided by AA. This has been done for ray tracing implicit surfaces by Cusatis et al. [16] and for approximating parametric curves by Figueiredo et al. [17], but as far as we know has not yet been done for polygonal approximation of implicit curves, either on the plane or on surfaces.

Bühler [18, 19] proposed a cell pruning method based on a linearization of implicit objects derived from AA. In addition to reducing the number of enclosure cells, her 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.

3. Affine arithmetic

We now briefly review the main concepts of AA. For details, see [20] and [4].

Like classical interval arithmetic [2, 3], affine arithmetic is an *extended arithmetic*: it represents real quantities with more than just one floating-point number; it provides replacements for the standard arithmetic operations and elementary functions that work on such extended representations; and it is able to extract information on the range of computed quantities from the extended representation. The computations in both interval

and affine arithmetic take into account all rounding errors in floating-point arithmetic and so provide reliable results.

Interval arithmetic uses two floating-point numbers to represents intervals containing quantities. Affine arithmetic represents a quantity q with an *affine form*:

$$\widehat{q} = q_0 + q_1\varepsilon_1 + q_2\varepsilon_2 + \cdots + q_n\varepsilon_n$$

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

$$q \in [\widehat{q}] := [q_0 - \delta, q_0 + \delta]$$

where $\delta = |q_1| + \cdots + |q_n|$. Thus, AA generalizes interval arithmetic. More importantly, by design affine forms can share noise symbols and thus may be not completely independent. The explicit representation of first-order partial correlations and the quadratic convergence of estimates are the main features of AA that are absent in classical interval arithmetic. Despite the increased computational cost in AA, these features yield more efficient methods in several cases, especially when the geometry of AA approximations is exploited [16, 17], as in this paper.

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 [20, 4]. By combining the formulas for these basic operations, one can evaluate any complicated algebraic or transcendental formula on affine forms. As in other extended arithmetics, this is especially convenient to implement automatically using operator overloading, which is readily available in several programming languages. We use `libaffa`, a C++ library for AA [21].

4. Bounding implicit curves with strips

Affine forms have a rich geometry, which our method tries to exploit. We now describe how to use AA to compute a strip of parallel lines that contains the piece of the plane curve given implicitly by $f(x,y) = 0$ in axis-aligned rectangles, arbitrary parallelograms, and triangles. We then explain how to extend this computation to handle a curve given implicitly by $f(x,y,z) = 0$ on a triangulated surface. This is the basis of our adaptive approximation method, which we present in §5.

4.1. On rectangles

In the simplest setting, we want to evaluate $f(x,y)$ using AA on a rectangular domain on the plane, $\Omega = [a,b] \times [c,d]$. Assuming that f is given by a mathematical expression, we just need to represent $(x,y) \in \Omega$ 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 ε_3, …, ε_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 $f(x,y)$ lies in the interval $[\widehat{f}]$ centered at f_0 with radius $|f_1| + |f_2| + |f_3|$. 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. [5], is the basis for all previous interval methods of approximating an implicit curve using a rectangular quadtree.

The basis of our approach is that useful geometric bounds can be extracted from the affine form $\widehat{f}$. Indeed, the affine approximation $\widehat{f} = f_0 + f_1\varepsilon_1 + f_2\varepsilon_2 + f_3\varepsilon_3$ tells us that the graph of $z = f(x,y)$ over Ω is sandwiched between the two parallel planes given by

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

These equations 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. The method is discussed in detail in §5.

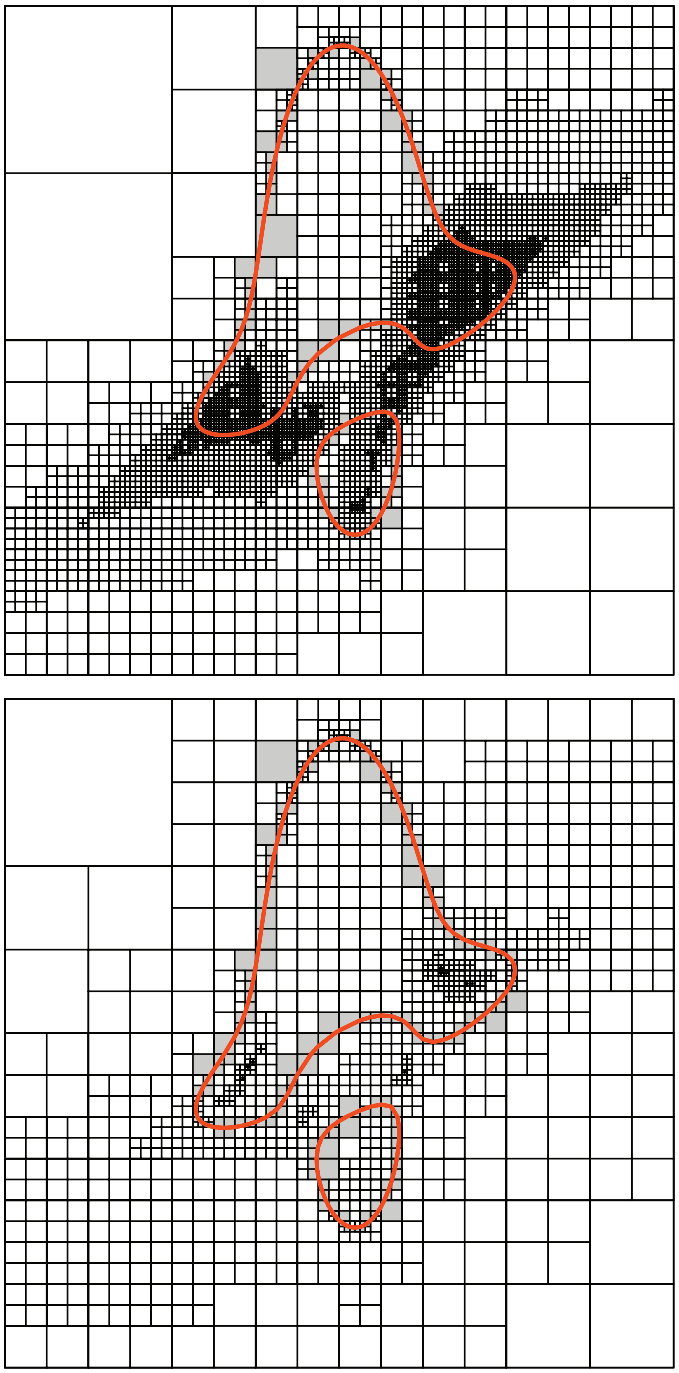


Fig. 2: Approximating implicit curves on a rectangular quadtree using the method of Lopes et al. [5] (top) and our method (bottom) for the quartic curve from Taubin's paper [22]: $0.004 + 0.110x - 0.177y - 0.174x^2 + 0.224xy - 0.303y^2 - 0.168x^3 + 0.327x^2y - 0.087xy^2 - 0.013y^3 + 0.235x^4 - 0.667x^3y + 0.745x^2y^2 - 0.029xy^3 + 0.072y^4 = 0$. The region of interest is $\Omega = [-2.19, 2.19]^2$. The quadtrees go to maximum depth 9. The top decomposition visited 6937 cells, has 341 leaves, and took 97 milliseconds. The bottom decomposition visited 1697 cells, has 221 leaves, and took 43 milliseconds. Our method is faster and generates a more efficient decomposition.

Fig. 2 shows an example of our method in action using a rectangular quadtree to locate and approximate an implicit curve. Note how it compares favorably with the method of Lopes et al. [5] without computing derivatives because AA provides second-order approximations and handles first-order correlations, thus yielding tighter interval estimates for long arithmetic expressions. Our previous paper [6] contains some other results on this comparison.

4.2. On parallelograms

We have seen how to represent an axis-aligned rectangle in AA: it is the joint range of two mutually independent affine forms on two noise symbols. In general, the joint range of m affine forms on n noise symbols is the image of the hypercube $[-1,1]^n$ under an affine transformation $\mathbf{R}^n \to \mathbf{R}^m$. Therefore, 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 in the plane. Indeed, the two affine forms

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

represent the parallelogram centered at (x_0, y_0) with sides parallel to the vectors $v_1 = (x_1, y_1)$ 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 $y_1 = 0$ — in this case, the parallelogram is a rectangle with sides parallel to the coordinate axes, as in §4.1.

Having represented a parallelogram P with two affine forms $\hat{x}$ and $\hat{y}$, we can compute an affine form $\hat{f}$ representing f in P by using AA on $\hat{x}$ and $\hat{y}$. From $\hat{f}$ we can find Cartesian equations for a strip containing the curve $f(x,y) = 0$ for $(x,y) \in P$. In the rectangular case discussed in §4.1, these equations were easy to find by writing ε_1 and ε_2 in terms of x and y. We can do exactly the same thing in the parallelogram case, but now we need to invert a 2×2 matrix, converting

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

to

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

and proceeding as before. The matrix above is invertible if and only if the vectors $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ are linearly independently, that is, when the parallelogram is not degenerate.

4.3. On triangles

We wish to evaluate a function f with AA on a triangle, but triangles are not zonotopes and so cannot be directly represented in AA. The easiest solution is to enclose the triangle in a parallelogram and then evaluate AA there, as explained in §4.2. As illustrated in Fig. 3, there are a few ways to do this enclosure: reflect the triangle with respect to one side, find the smallest rectangle containing the triangle, or use the triangle's axis-aligned bounding box.

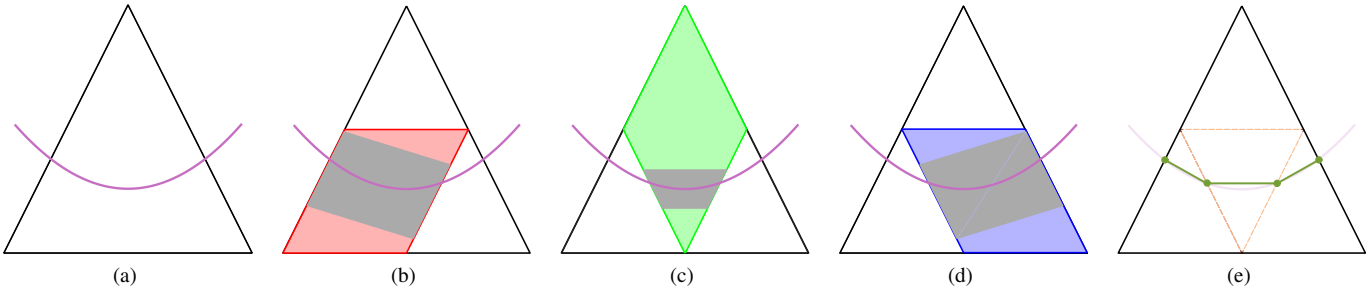


Fig. 4: Polygonal approximation of the parabola $y = 3x^2$ using our method. (a) Input triangle. (b), (c), and (d) evaluating $f(x,y) = 3x^2 - y$ with AA in each parallelogram and its associated strips (gray regions). (e) computing the polygonal approximation (green) using the sub-triangles of the input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

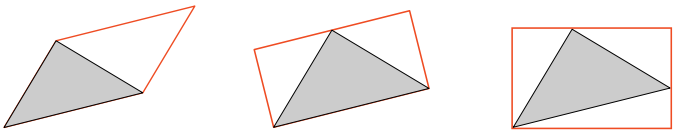


Fig. 3: Enclosing a triangle in a parallelogram.

Strictly speaking, this solution is wrong because we would be evaluating f outside its domain. In practice, it works well for functions defined on the whole plane or at least on a neighborhood of the region of interest. This solution uses a less precise absence oracle due to increased interval overestimation. As a consequence, the adaptive method discussed in §5 will probably subdivide more cells and compute more interval estimates than necessary.

One possibility for representing a triangle with vertices ABC without enlarging it is to use a bilinear parametrization: (u,v) in the unit square goes to $(1-u)A+u(1-v)B+uvC$ in the triangle. However, even if we write this as $A+u(B-A)+uv(C-B)$ to minimize repetitions, it is still a quadratic map and will induce even higher interval overestimation than the one due to enlarging the triangle to a parallelogram. Moreover, the strip computed by AA lies in the uv parametric space, not in xyz Cartesian space, and so is not directly useful for our method.

The solution we have adopted for representing a triangle without enlarging it is to decompose the triangle into three overlapping parallelograms by joining the midpoints of the edges, as in Fig. 5. We can then evaluate f with AA on each of these parallelograms, as in §4.2. If the curve lies within a thin strip in each of these parallelograms, 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 approximation which is the basis of our method, explained in §5.

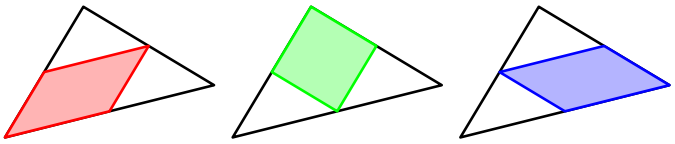


Fig. 5: Decomposing a triangle into parallelograms.

This strategy does not enlarge the triangle and so avoids evaluating f outside its domain. Moreover, since the parallelograms are parametrized linearly, there is no increased overestimation. While this strategy requires up to three AA evaluations of the function for each triangle that is visited, it can yield a better polygonal approximation of the implicit curve inside the triangle with three segments instead of just one, as shown in Fig. 4.

Table 1 and Fig. 6 show a comparison of these four strategies for handling triangles with parallelograms for the curve in Fig. 2. We use midpoint subdivision for refinement in our adaptive method (see §5 for details). As expected, the strategy that decomposes triangles into parallelograms visits fewer cells, produces a smaller mesh and a better polygonal approximation, but takes a bit longer. Producing smaller final meshes is a desirable feature in modeling applications. If necessary, applications using our method can choose one of the other strategies or even mix them dynamically to balance the trade-offs between precision and cost.

strategy	time	output	visited	leaves	AA	seg
decomposition	33	**1445**	**1805**	**250**	4604	502
reflection	**25**	2909	3878	298	3878	**298**
smallest rectangle	28	3392	4522	318	4522	318
bounding box	**25**	2882	3842	316	**3842**	316

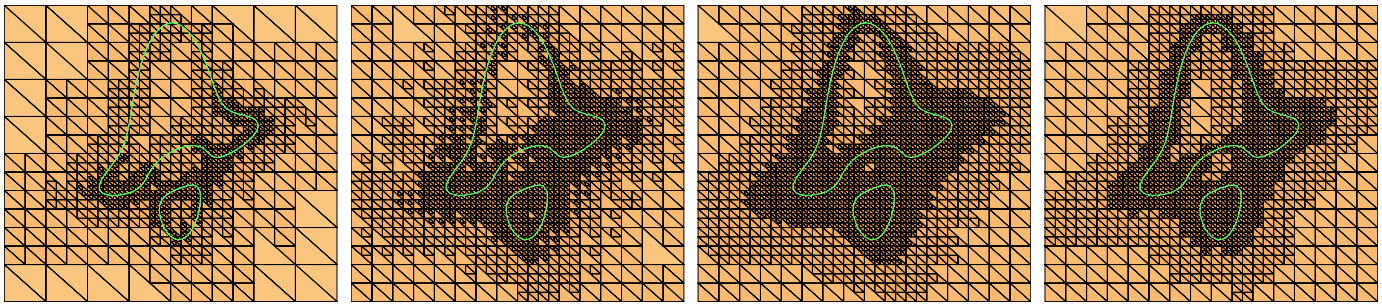
Table 1: Performance of different strategies for handling triangles with parallelograms to approximate the curve in Fig. 2: times in milliseconds and numbers of triangles output in the final mesh, visited triangles, leaf triangles, AA evaluations, and segments in the polygonal approximation.

4.4. *On triangulated surfaces*

An implicit curve on a triangulated surface is given by an equation $f(x,y,z) = 0$. It is thus the intersection of an implicit surface with the triangulated surface. We want to approximate the implicit curve on the triangulated surface without first computing a polygonal approximation for the implicit surface and then intersecting it with the triangulated surface — a wasteful, costly, and potentially numerically delicate task.

To extend the method in §4.3 to handle triangulated surfaces, it suffices to extend the method in §4.2 to handle parallelograms in 3D space. We represent such a parallelogram P with AA using three affine forms in two noise symbols:

$$\widehat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2,\ \widehat{y} = y_0 + y_1\varepsilon_1 + y_2\varepsilon_2,\ \widehat{z} = z_0 + z_1\varepsilon_1 + z_2\varepsilon_2$$



(a) decomposition (b) reflection (c) smallest rectangle (d) bounding box

Fig. 6: Effect of different strategies for handling triangles with parallelograms to approximate the curve of Fig. 2.

As before, evaluating $f(x,y,z)$ using AA on P yields that the implicit curve $f(x,y,z)=0$ in P lies sandwiched between the two parallel planes given by

$$0 = f_0 + f_1\varepsilon_1 + f_2\varepsilon_2 \pm f_3$$

Again, we translate these equations into Cartesian coordinates by writing ε_1 and ε_2 in terms of x, y, z. More precisely, from

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \\ z_1 & z_2 \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \end{bmatrix}$$

we get

$$\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \\ z_1 & z_2 \end{bmatrix}^{+} \begin{bmatrix} x-x_0 \\ y-y_0 \\ z-z_0 \end{bmatrix}$$

where $\mathbf{B}^{+} = (\mathbf{B}^{\top}\mathbf{B})^{-1}\mathbf{B}^{\top}$ is the pseudoinverse of a matrix $\mathbf{B}$. This expression for the pseudoinverse is valid exactly when the matrix has full rank, that is, when its two columns are linearly independent. In geometric terms, this happens exactly when the parallelogram P is not degenerate.

5. 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 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.

Our method is summarized in Fig. 7, in two versions: one for parallelograms, including rectangles, and one for triangles. The method starts with procedure *Adaptive* which explores each cell in an initial mesh decomposition of the region of interest or surface. For rectangular quadtrees in the plane, the initial mesh typically contains a single cell. For triangular decompositions of rectangular regions in the plane, the initial mesh typically contains two cells dividing the rectangle diagonally. Nevertheless, the method works for arbitrary triangulated regions and surfaces.

The core of the method is the *Explore* procedure which recursively tests whether the curve crosses the cell, subdividing

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

procedure Explore(C)                    (parallelograms)
  f̂ ← f(C) with AA
  if 0 ∈ [f̂] then
    w ← width of f̂ in C
    if w ≤ ε then
      Approximate(C)
    else
      divide C into subcells C_i
      for each i, Explore(C_i)
end

procedure Explore(C)                    (triangles)
  P_1, P_2, P_3 ← parallelograms of C
  f̂_i ← f(P_i) with AA
  if 0 ∈ [f̂_i] for some i then
    w_i ← width of f̂ in P_i
    if w_i ≤ ε for all i then
      Approximate(C)
    else
      divide C into subcells C_i
      for each i, Explore(C_i)
end
```

Fig. 7: Our adaptive method summarized.

the cell as needed. The recursion stops when the curve does not cross the cell (as proved by the interval estimate $[\widehat{f}]$ not containing zero), when the curve lies within a thin tube inside the cell (of width less than a user-supplied tolerance ε), or when a maximum recursion depth has been reached, for safety.

In the case of triangles, *Explore* is rearranged to test each parallelogram in sequence, avoiding further tests as soon as the curve is not thin inside a parallelogram. The triangle is then immediately subdivided and explored recursively. This change can avoid many unnecessary AA evaluations.

When the curve lies in a thin tube, we perform the *Approximate* procedure, in which we compute a polygonal approxima-

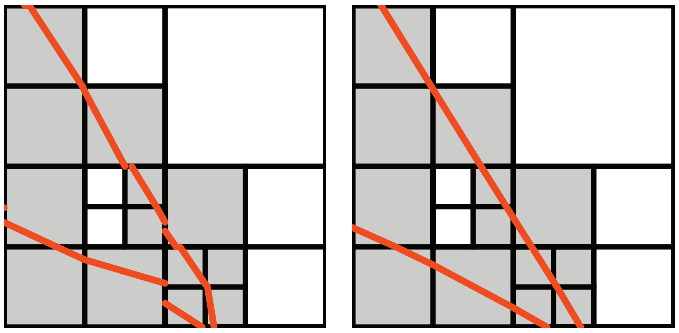


Fig. 8: Linear interpolation (left) produces cracks in the approximation, which are avoided by using the bisection method to full precision (right).

tion for the curve by finding the points where the curve crosses the boundary of the cell. In the case of triangular cells, we find the points where the curve crosses the boundary of each subcell of the midpoint subdivision (regardless of the refinement scheme used by the triangulation), since these are the edges of the parallelograms used for testing the cell using the decomposition strategy (Fig. 4). As in the method of Lopes et al. [5], we use the classic bisection method to full floating-point precision on each edge for finding these crossings to ensure continuity of the polygonal approximation without cracks (Fig. 8).

There are several possibilities for the subdivision step in *Explore*. We get a quadtree decomposition subdividing the cell into four similar subcells by joining the midpoints of the edges of the cell. This *midpoint subdivision* is the easiest subdivision method and can be used for both rectangles and triangles.

Triangulations can offer other subdivision methods for mesh refinement. Our method does not care what subdivision method is used and can use whatever subdivision method is offered by the triangulation or application. In particular, our method works seamlessly with dyadic splits, 4-8 meshes [23], $\sqrt{3}$-subdivision [24], 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 midpoint subdivision, which yields a triangular quadtree inside each triangle in the initial mesh. In this case, we change *Explore* to skip subtriangles contained in parallelograms that have been shown not to contain the curve. This helps to reduce the number of unnecessary subdivisions and redundant interval estimates, as Table 1 and Fig. 6 show. This change makes the strategy that decomposes triangles into parallelograms competitive with the other strategies.

6. 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 topological information, i.e., a triangle mesh with explicit connectivity between the triangles. We used midpoint refinement in the first example and 4-8 mesh refinement in the second one. As in the other pictures, the shading reflects the refinement level.

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. 13 shows that our method approximates nicely both algebraic and non-algebraic curves on complex meshes for regions of arbitrary genus.

Fig. 14 illustrates the effect of the geometric criteria in our method. This strategy produces well-adapted curves controlled by the tolerance ε (see Table 3), the only user-supplied parameter of our method. In order to demonstrate the efficiency of the geometric criteria, the curvature is computed analytically and color coded from low (cyan) to high (magenta) absolute values of the curvature. Fig. 14 also shows how the mesh concentrates on high-curvature regions as we decrease ε.

Fig. 15 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.

Fig. 16 shows that our method is able to approximate implicit curves on a wide variety of triangulated surfaces, from closed surfaces with or without boundary to surfaces with complex topology. Although finding an implicit curve on a triangulated surface S is equivalent to intersecting an implicit surface with S, our method does not need to triangulate the implicit surface and then intersect that mesh with S, a costly and potentially numerically delicate task.

The *nodal domains* (zero sets) of the real spherical harmonics Y_4^k of degree 4 and order k on the unit sphere (see Table 2) are gracefully approximated using our method, as shown in Fig. 17. The heavy refinement seen in the figure is due to the presence of self-intersections in the curves.

Fig. 18 shows the intersection of a cylinder given implicitly with a mesh of the unit sphere. The topology of the intersection curve varies with the position of the cylinder. Fig. 19 shows the intersection of a hyperboloid given implicitly with a Klein bottle surface given parametrically. For comparison, the intersection is found both on a mesh of the surface and directly on the parametric domain. These two examples illustrate how our method would work for finding both intersection curves in space and trimming curves in parametric space in a hybrid CSG modeling system that mixed implicit surfaces, parametric surfaces, and triangle meshes.

All results were generated on a 2.4 GHz Intel Core i7 with 8GB of RAM. Table 3 shows performance data, timings, and statistics for these computations.

7. Conclusion

Like the method of Lopes et al. [5], 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 both rectangular and triangular decompositions, structured and unstructured, on the plane or on surfaces. It can use any

Fig.	curve	mesh	depth	ε	triangles in	triangles out	time
1	knot	Lauranna	4	1×10^{-4}	1852	12604	49
11.b	circle	Africa	3	1×10^{-3}	193	922	9
11.c	circle	Africa	5	1×10^{-3}	193	1106	9
12.a	bicorn	octagon	5	5×10^{-3}	126	1168	13
12.b	pear	beetle	4	1×10^{-2}	940	1771	15
12.c	capricorn	tree	22	1×10^{-3}	1015	11678	93
13.a	heart	gear	3	3×10^{-3}	1424	3289	23
13.b	clown	K7	4	5×10^{-3}	1006	2134	19
13.c	irrational	eight	4	1×10^{-3}	1032	3897	48
14.a	cubic	square	6	8×10^{-1}	4	67	9
14.b	cubic	square	9	4×10^{-1}	4	184	22
14.c	cubic	square	11	1×10^{-1}	4	621	64
15	Pisces	circle	6	5×10^{-3}	101	2382	66
16.a	saddle	horse	4	1×10^{-4}	6484	9184	55
16.b	two circles	mask	3	1×10^{-4}	8474	10524	53
16.c	cubic	teapot	4	1×10^{-4}	1962	20364	62
17.b	$Y_4^0=0$	sphere	4	1×10^{-4}	1536	7386	62
17.c	$i\frac{\sqrt{2}}{2}\left(Y_4^1+Y_4^{-1}\right)=0$	sphere	4	1×10^{-4}	1536	8664	62
17.d	$\frac{\sqrt{2}}{2}\left(Y_4^2+Y_4^{-2}\right)=0$	sphere	6	1×10^{-4}	1536	19624	135
17.e	$i\frac{\sqrt{2}}{2}\left(Y_4^3+Y_4^{-3}\right)=0$	sphere	6	1×10^{-4}	1536	18662	125
17.f	$\frac{\sqrt{2}}{2}\left(Y_4^4+Y_4^{-4}\right)=0$	sphere	7	1×10^{-4}	1536	26854	215
18.a	cylinder	sphere	5	1×10^{-4}	1536	3338	17
18.b	cylinder	sphere	9	1×10^{-4}	1536	11800	70
18.c	cylinder	sphere	5	1×10^{-4}	1536	4986	27
19.a	hyperboloid	Klein bottle	3	1×10^{-4}	800	4463	29
19.b	hyperboloid	Klein bottle	10	1×10^{-4}	8	4890	37

Table 3: Performance and statistics for our method (times in milliseconds).

spherical harmonic	Cartesian version
Y_4^0	$\frac{3}{16}\sqrt{\frac{1}{\pi}}\left(35z^4-30z^2+3\right)$
$i\frac{\sqrt{2}}{2}\left(Y_4^{-1}+Y_4^1\right)$	$\frac{3}{4}\sqrt{\frac{5}{2\pi}}yz(7z^2-3)$
$\frac{\sqrt{2}}{2}\left(Y_4^{-2}+Y_4^2\right)$	$\frac{3}{8}\sqrt{\frac{5}{\pi}}(x^2-y^2)(7z^2-1)$
$i\frac{\sqrt{2}}{2}\left(Y_4^{-3}+Y_4^3\right)$	$\frac{3}{4}\sqrt{\frac{35}{2\pi}}yz(3x^2-y^2)$
$\frac{\sqrt{2}}{2}\left(Y_4^{-4}+Y_4^4\right)$	$\frac{3}{16}\sqrt{\frac{35}{\pi}}\left(x^4-6x^2y^2+y^4\right)$

Table 2: Real spherical harmonics of degree 4 on the unit sphere.

refinement scheme that the decomposition offers. Otherwise, it provides its own refinement scheme using midpoint subdivision, which is exploited to reduce costs. Our method can also cater for different cost models by using or mixing different strategies for handling triangles with parallelograms.

Limitations. Our method requires that the function defining the curve to be given as an explicit mathematical expression or as an algorithm. This limitation is shared by all interval methods.

The results produced by our method may depend on the quality of the initial decomposition and its subsequent refinements. Ideally, the mesh should provide a refinement scheme that avoids bad-quality triangles.

Finally, like most methods for approximating implicit objects, our method does not attempt to handle singularities. Nevertheless, as shown in Fig. 9, it can detect singularities in the sense that it can flag cells that have reached the maximum depth without meeting any other stopping criteria. The curve most probably cuts these cells but is not flat there, thus indicating a singularity.

Future work. A natural direction for future work is to extend our technique for approximating implicit surfaces in hexahedral and tetrahedral spatial decompositions, another important problem in geometric modeling with several applications [1, 25] and for which there are interval methods [26]. In particular, we would like to apply our method for surface reconstruction from point clouds using Multi-level Partition of Unity (MPU) Implicits [27].

Implicit curves on triangulated surfaces are potentially useful for modeling tasks such as boundary evaluation and Boolean operations in hybrid CSG systems that mix implicit and parametric objects and meshes. Such applications are beyond the scope of this paper and are left as suggestion for future work.

Another direction for future work is to compute the distance field of an implicit curve by adapting the algorithm described in [17] for parametric curves.

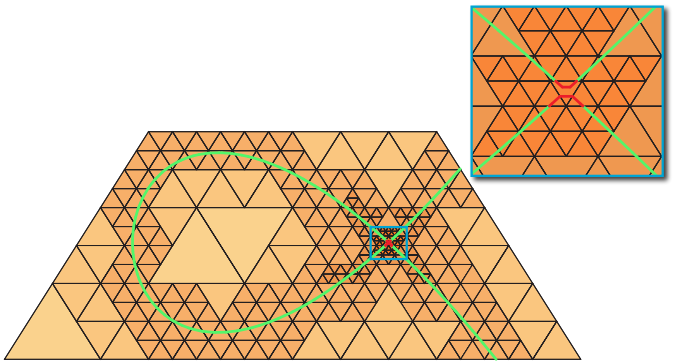


Fig. 9: Our method detects the non-manifold region (red) in *Tschirnhausen cubic* curve $y^2 = x^3 + 3x^2$ on trapezoid, even when the singularity is not recovered (top right zoom).

Our method can be applied to parallelogram decompositions of the plane, including the interesting Penrose tiling of Fig. 10. Penrose tilings with parallelograms admit a global refinement scheme but not a local one, and so cannot be used with our adaptive method. Except for the midpoint subdivision scheme, we do not know any method for decomposing a parallelogram into smaller parallelograms. It would be interesting to find one.

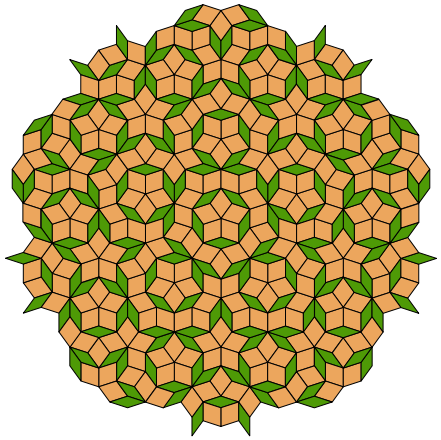
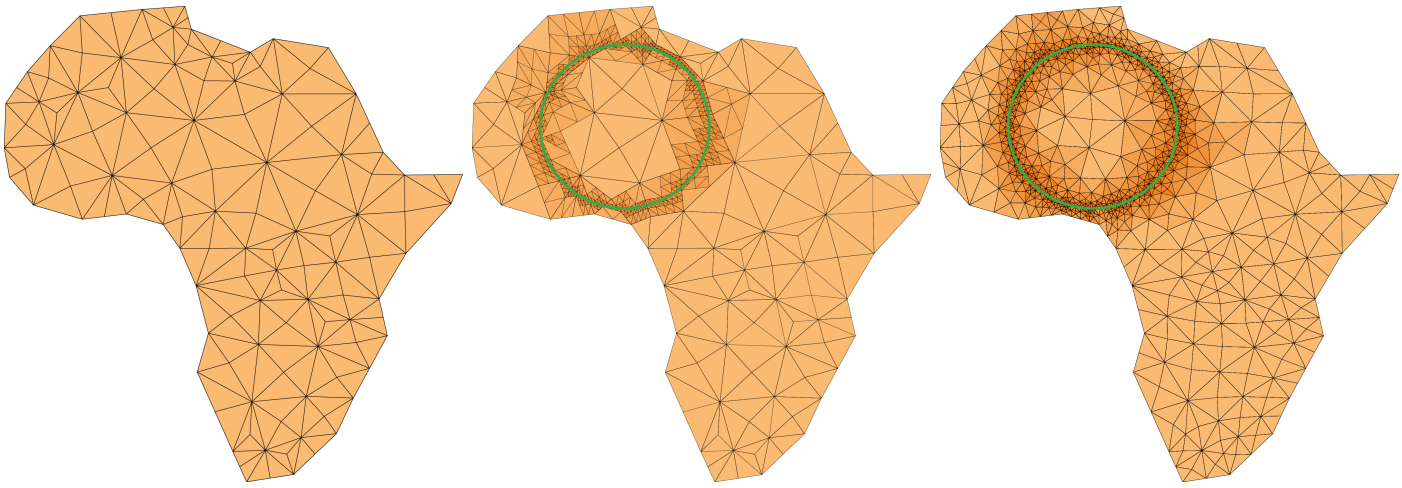


Fig. 10: Penrose tiling with parallelograms (source: Wikipedia).

Acknowledgments. A previous version of this paper was presented at SIBGRAPI 2012 [6]. 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.

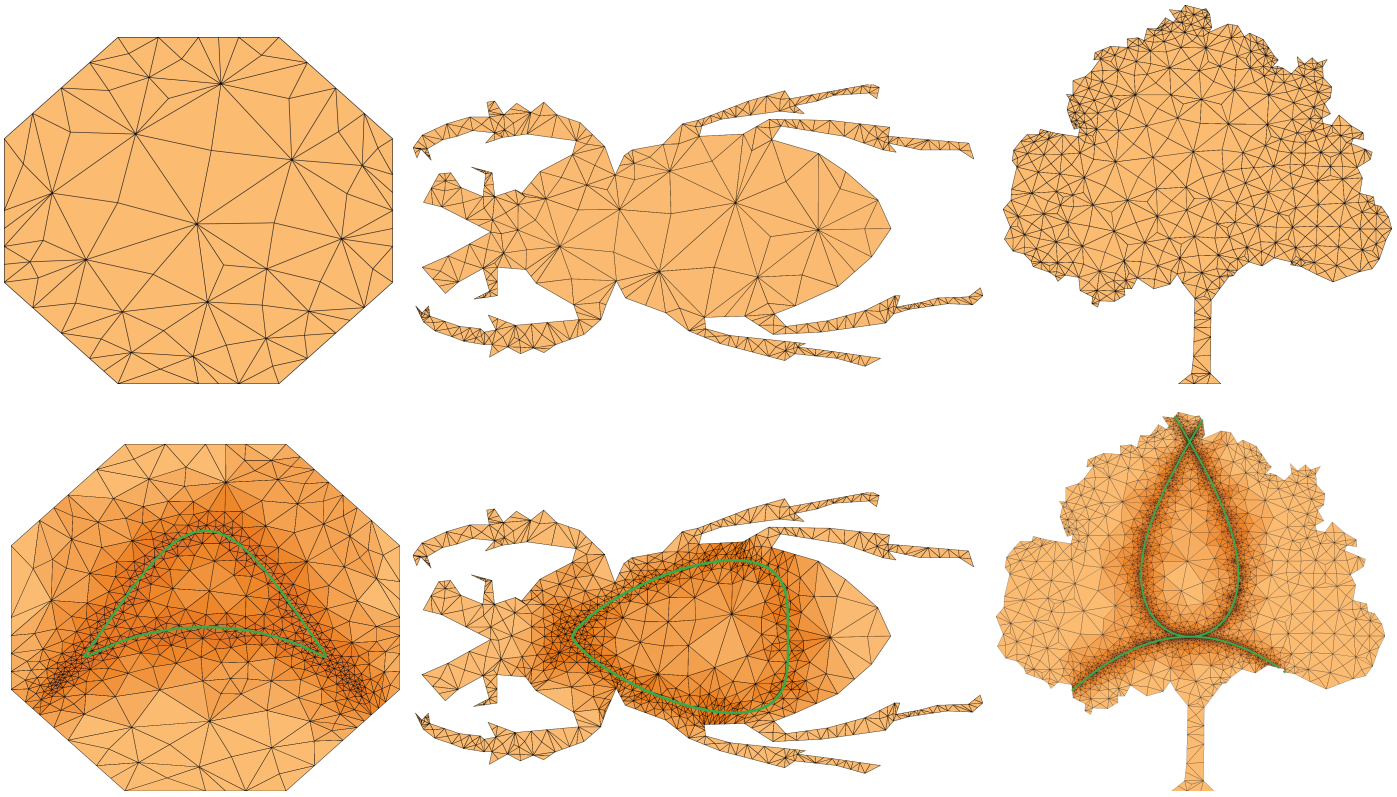
References

[1] Patrikalakis NM, Maekawa T. Shape interrogation for computer aided design and manufacturing. Springer-Verlag; 2002.
[2] Moore RE. Interval Analysis. Prentice-Hall; 1966.
[3] Moore RE, Kearfott RB, Cloud MJ. Introduction to Interval Analysis. SIAM; 2009.
[4] de Figueiredo LH, Stolfi J. Affine arithmetic: Concepts and applications. Numerical Algorithms 2004;37(1):147–58.
[5] Lopes H, Oliveira JB, de Figueiredo LH. Robust adaptive polygonal approximation of implicit curves. Computers & Graphics 2002;26(6):841–52.
[6] Paiva A, de Carvalho Nascimento F, de Figueiredo LH, Stolfi J. Approximating implicit curves on triangulations with affine arithmetic. In: Proceedings of SIBGRAPI 2012. IEEE Press; 2012, p. 94–101.
[7] Dobkin DP, Levy SVF, Thurston WP, Wilks AR. Contour tracing by piecewise linear approximations. ACM Transactions on Graphics 1990;9(4):389–423.
[8] Persiano RCM, Comba JLD, Barbalho V. An adaptive triangulation refinement scheme and construction. In: Proceedings of SIBGRAPI'93. 1993, p. 259–66.
[9] Suffern KG, Fackerell ED. Interval methods in computer graphics. Computers & Graphics 1991;15(3):331–40.
[10] Mitchell DP. Three applications of interval analysis in computer graphics. In: Frontiers in Rendering course notes. SIGGRAPH'91; 1991, p. 14–1–14–13.
[11] Snyder JM. Interval analysis for computer graphics. Computer Graphics 1992;26(2):121–30. (SIGGRAPH'92 Proceedings).
[12] Snyder JM. Generative Modeling for Computer Graphics and CAD. Academic Press; 1992.
[13] Comba JLD, Stolfi J. Affine arithmetic and its applications to computer graphics. In: Proceedings of SIBGRAPI'93. 1993, p. 9–18.
[14] de Figueiredo LH, Stolfi J. Adaptive enumeration of implicit surfaces with affine arithmetic. Computer Graphics Forum 1996;15(5):287–96.
[15] Martin R, Shou H, Voiculescu I, Bowyer A, Wang G. Comparison of interval methods for plotting algebraic curves. Computer Aided Geometric Design 2002;19(7):553–87.
[16] de Cusatis Jr. A, de Figueiredo LH, Gattass M. Interval methods for ray casting implicit surfaces with affine arithmetic. In: Proceedings of SIBGRAPI'99. IEEE Press; 1999, p. 65–71.
[17] de Figueiredo LH, Stolfi J, Velho L. Approximating parametric curves with strip trees using affine arithmetic. Computer Graphics Forum 2003;22(2):171–9.
[18] Bühler K. Fast and reliable plotting of implicit curves. In: Uncertainty Geometric Computations. Kluwer Academic; 2002, p. 15–28.
[19] Bühler K. Implicit linear interval estimations. In: Proceedings of SCCG '02. ACM; 2002, p. 123–32.
[20] Stolfi J, de Figueiredo LH. Self-Validated Numerical Methods and Applications. 21st Brazilian Mathematics Colloquium, IMPA; 1997.
[21] Gay O, Coeurjolly D, Hurstand N. Libaffa, C++ affine arithmetic library. 2006. http://www.nongnu.org/libaffa/.
[22] Taubin G. Rasterizing algebraic curves and surfaces. IEEE Computer Graphics and Applications 1994;14(2):14–23.
[23] Velho L, Zorin D. 4-8 subdivision. Computer-Aided Geometric Design 2001;18(5):397–427.
[24] Kobbelt L. $\sqrt{3}$-subdivision. In: Proceedings of SIGGRAPH '00. ACM; 2000, p. 103–12.
[25] Gomes A, Voiculescu I, Jorge J, Wyvill B, Galbraith C. Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms. Springer; 2009.
[26] Paiva A, Lopes H, Lewiner T, de Figueiredo LH. Robust adaptive meshes for implicit surfaces. In: Proceedings of SIBGRAPI 2006. IEEE Press; 2006, p. 205–12.
[27] Ohtake Y, Belyaev A, Alexa M, Turk G, Seidel HP. Multi-level partition of unity implicits. ACM Transactions on Graphics 2003;22(3):463–70. (SIGGRAPH'03 Proceedings).



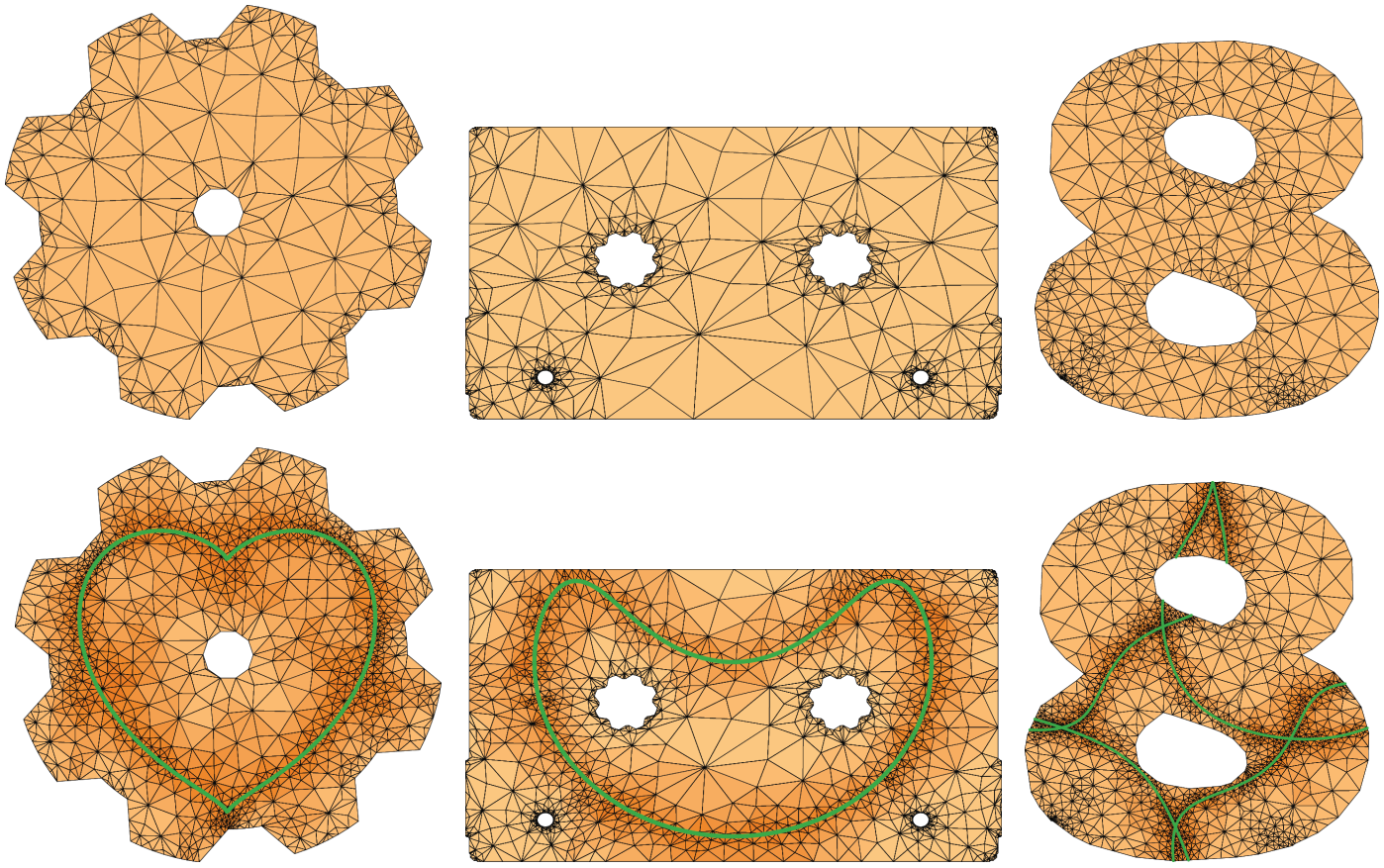
(a) Input mesh (b) Triangle soup (c) Triangle mesh with topology

Fig. 11: Approximating the circle (green curve) $x^2+y^2=1$ on adaptive meshes: (a) input coarse mesh, (b) using midpoint refinement, (c) using 4-8 mesh refinement.



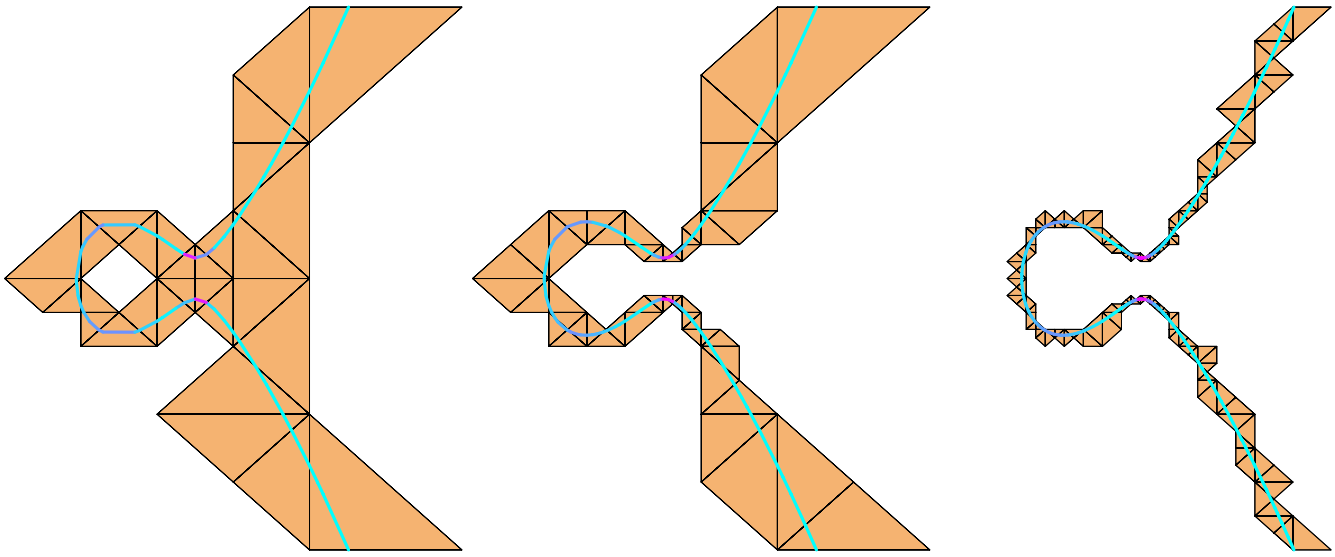
(a) (b) (c)

Fig. 12: Approximating implicit curves (green) on adaptive triangle meshes using 4-8 mesh refinement; input coarse mesh (top) and refined mesh (bottom): (a) *bicorn curve* $y^2(0.75^2-x^2)=(x^2+1.5y-0.75^2)^2$ on octagon model, (b) *pear curve* $4y^4-(x+1)^3(1-x)=0$ on beetle model, (c) *capricornoid curve* given implicitly by $4x^2(x^2+y^2)-(2y-x^2-y^2)^2=0$ on tree model.



(a) (b) (c)

Fig. 13: Approximating implicit curves on complex meshes; input coarse mesh (top) and adaptive mesh (bottom): (a) *heart* curve $(x^2+y^2-1)^3 = x^2y^3$ on gear model, (b) *clown smile* $(y-x^2+1)^4+(x^2+y^2)^4 = 1$ on K7 model, (c) transcendental curve given implicitly by $(xy+\cos(x+y))(xy+\sin(x+y)) = 0$ on eight model.



(a) $\varepsilon = 0.8$ (b) $\varepsilon = 0.4$ (c) $\varepsilon = 0.1$

Fig. 14: The effect of the geometric criteria on a cubic curve $y^2 - x^3 + x = 0.5$ in a triangular quadtree. Our method tracks regions with high curvature (magenta) when ε decreases.

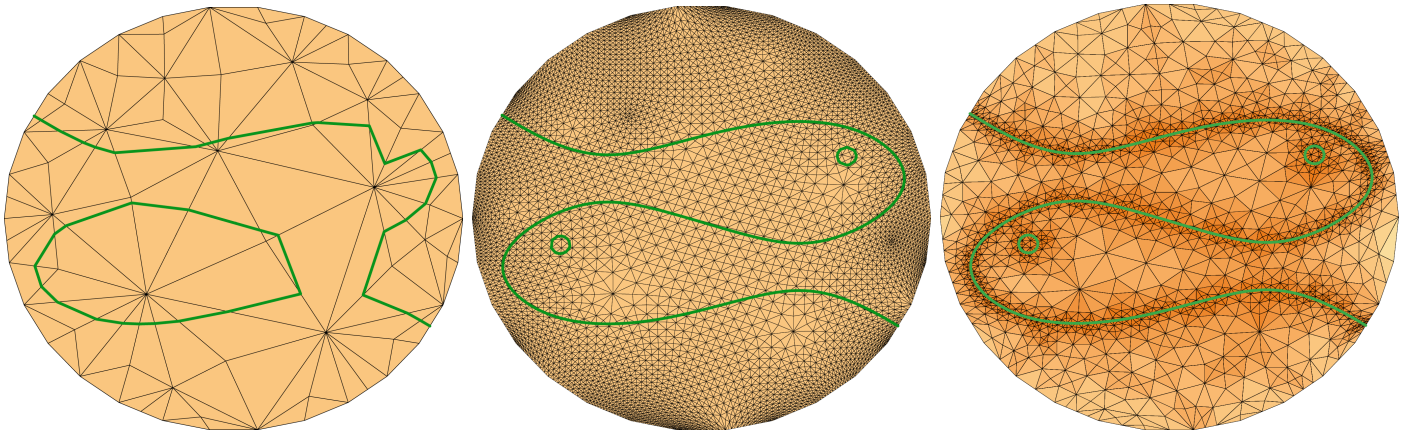
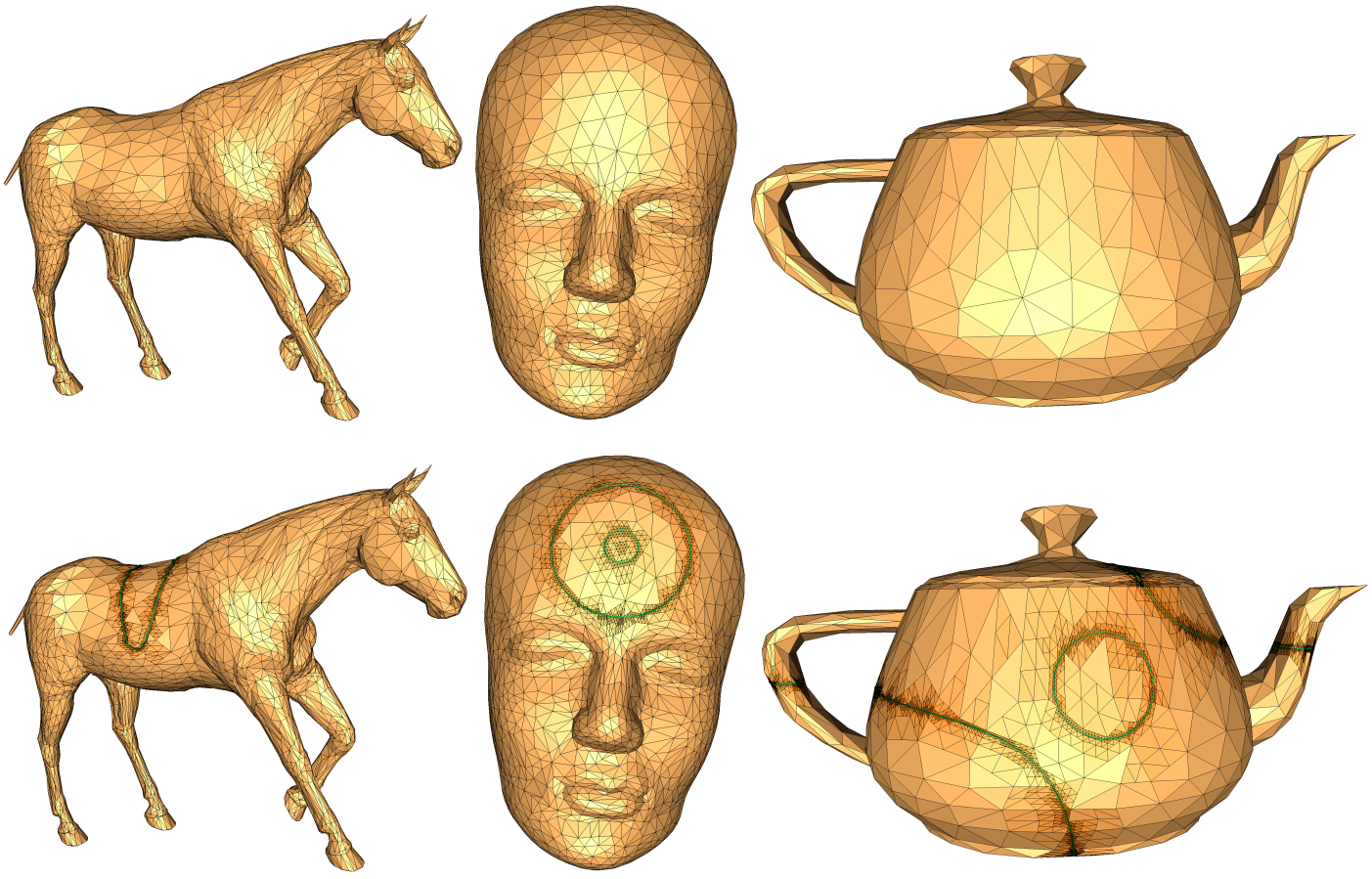
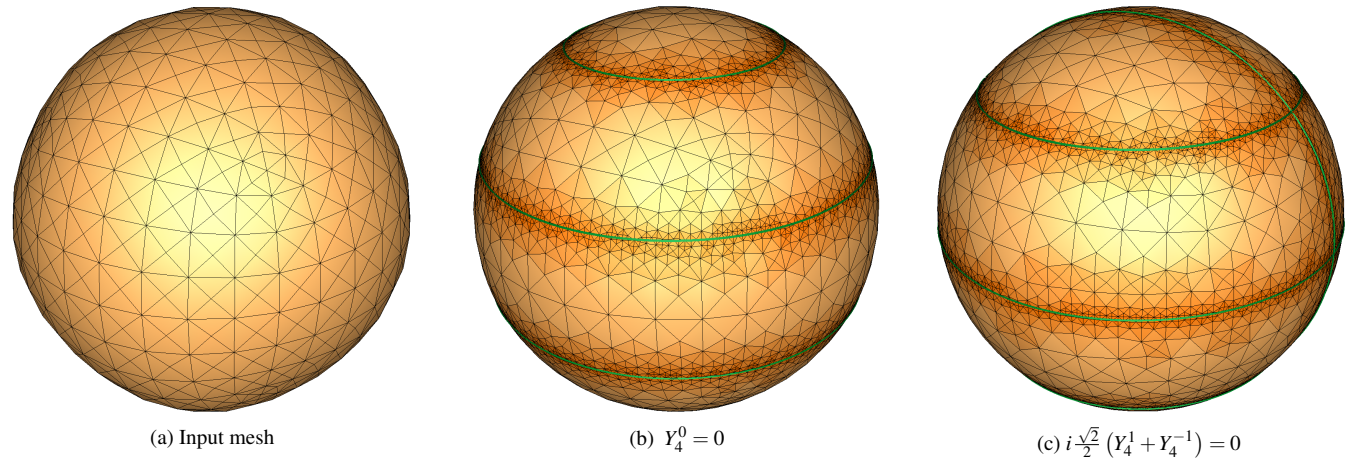


Fig. 15: In order to reproduce the Pisces logo (http://www.geom.uiuc.edu/~fjw/pisces/docs/models/Pisces.html), the marching triangles algorithm alone generates a curve with wrong topology on coarse mesh with 101 triangles (left) and it only provides a satisfactory result on a fine mesh with 12928 triangles (middle). Our method adaptively tracks the geometry and topology of the curve on the mesh with just 2382 triangles.

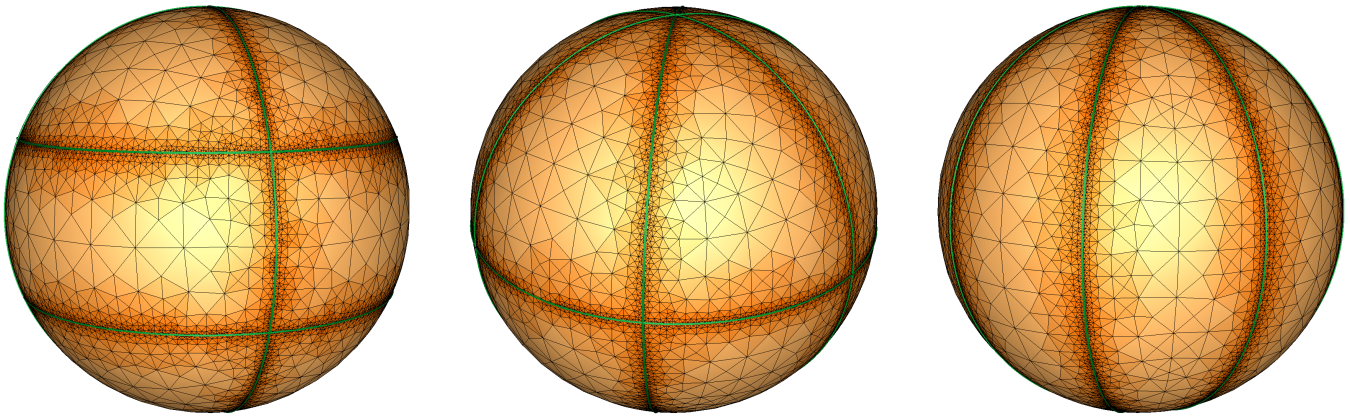


(a) (b) (c)

Fig. 16: Approximating implicit curves on surfaces; input coarse mesh (top) and refined mesh using midpoint mesh refinement (bottom): (a) saddle $3x^2 - 48y^2 = 8z$ on horse model, (b) *two circles* curve $xy^2(1 - \sqrt{xy^2}) = 0.04$ on mask model, (c) cubic curve $(xy-2)(x^2+y^2-1) = 0$ on teapot model.

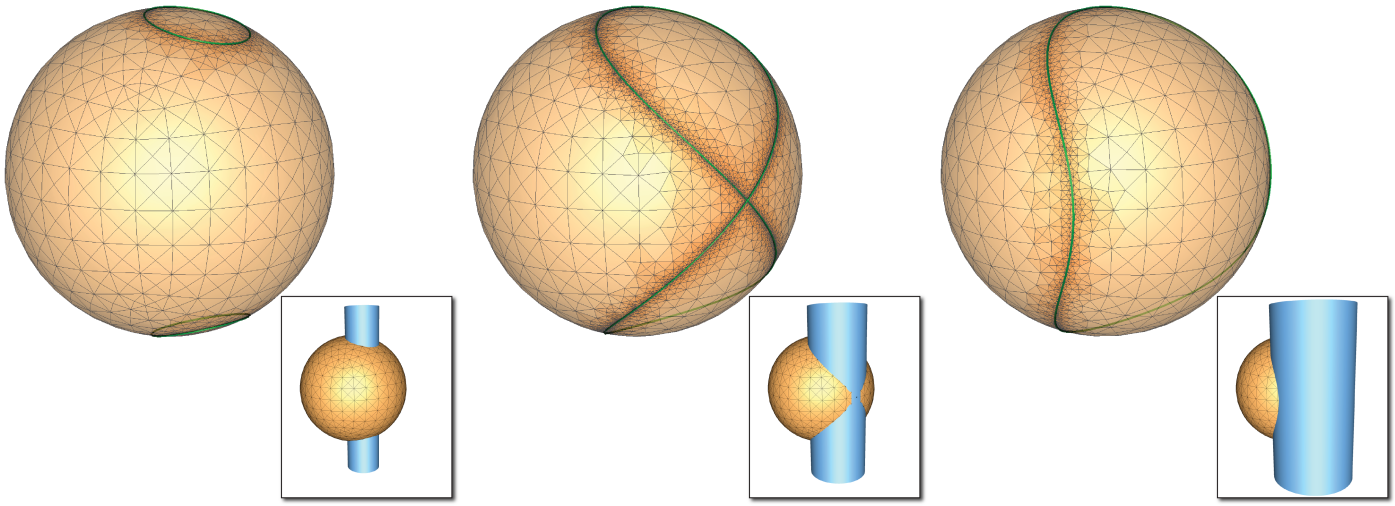


(a) Input mesh (b) $Y_4^0 = 0$ (c) $i\frac{\sqrt{2}}{2}\left(Y_4^1 + Y_4^{-1}\right) = 0$



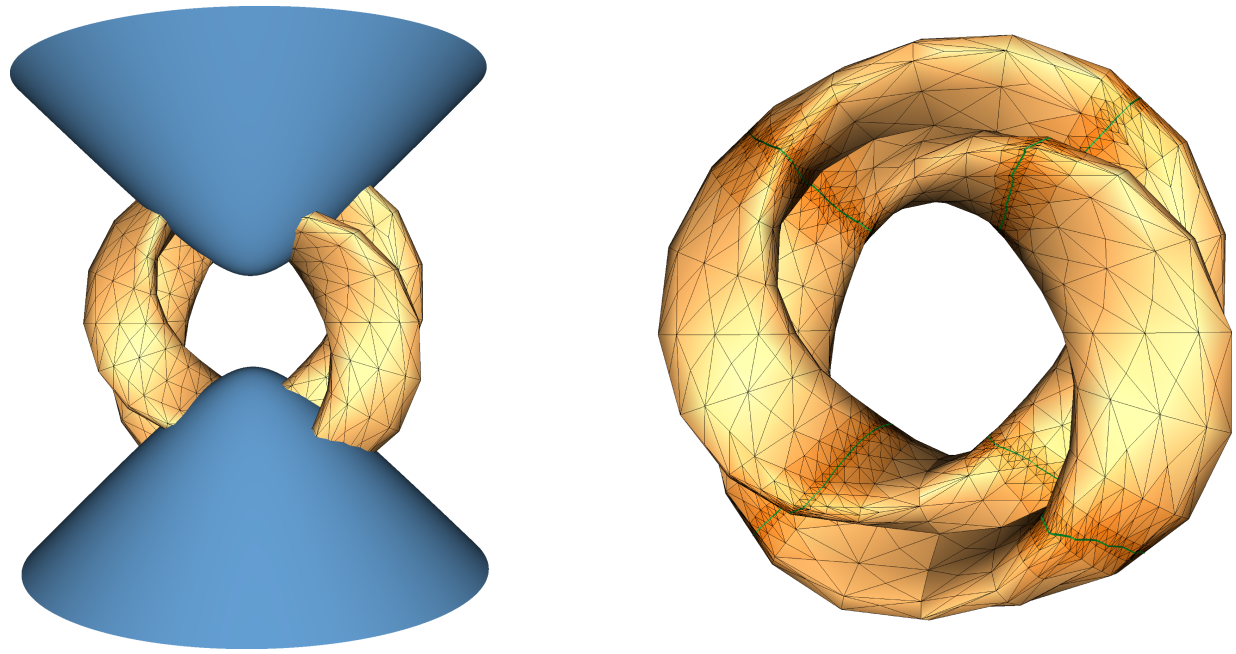
(d) $\frac{\sqrt{2}}{2}\left(Y_4^2 + Y_4^{-2}\right) = 0$ (e) $i\frac{\sqrt{2}}{2}\left(Y_4^3 + Y_4^{-3}\right) = 0$ (f) $\frac{\sqrt{2}}{2}\left(Y_4^4 + Y_4^{-4}\right) = 0$

Fig. 17: Approximating the nodal domains (green lines) of the real spherical harmonics of degree 4 on the unit sphere.

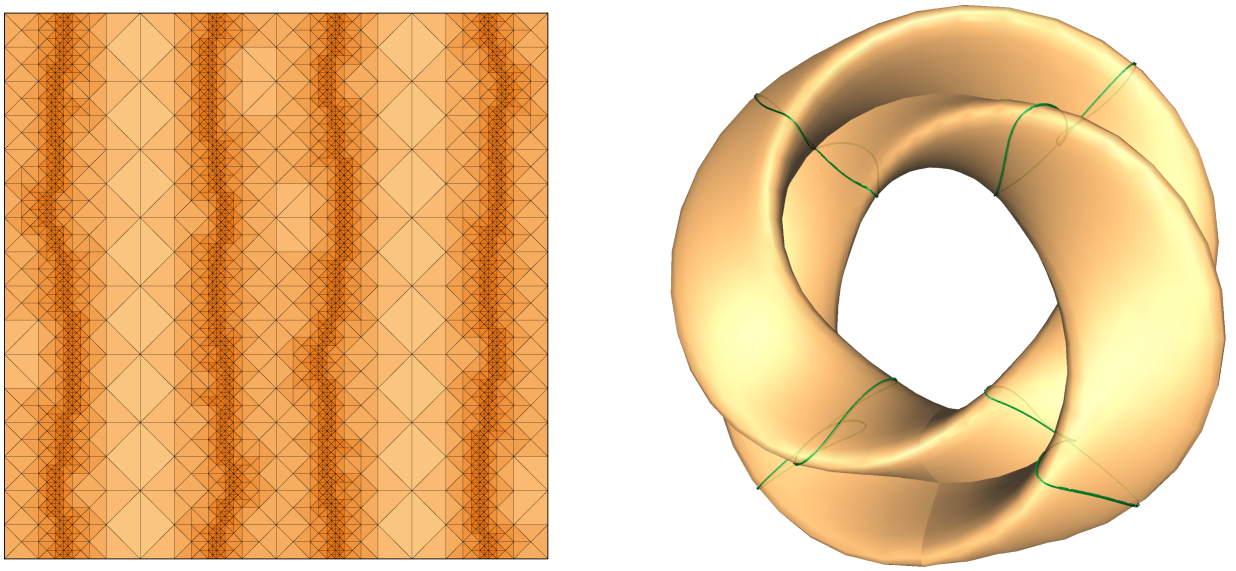


(a) $a = 0.3$ (b) $a = 0.5$ (c) $a = 0.7$

Fig. 18: Approximating implicit curves from the intersection of the cylinder given implicitly by $(x-a)^2 + y^2 = a^2$ on the unit sphere.



(a)



(b)

Fig. 19: Approximating implicit curves (green) from the intersection of the hyperboloid given implicitly by $x^2 - y^2 - z^2 = 1$ on Klein bottle given parametrically by $x(u,v) = (a + \cos(u)\sin(v) - \sin(u)\sin(2v))\cos(u)$, $y(u,v) = (a + \cos(u)\sin(v) - \sin(u)\sin(2v))\sin(u)$, $z(u,v) = \sin(u)\sin(v) + \cos(u)\sin(2v)$, where $a = 2.7$ and $(u,v) \in [0,2\pi) \times [0,2\pi)$: (a) curves computed on a mesh of the surface and (b) directly from the parametric domain.