

this process continues until the entire surface is enclosed by the collection of cubes [50, 52]. The surface within each cube is then polygonised, i.e. the surface patch inside each cube is approximated by one or more polygons, as described [49]. As expected, this “marching cubes” method also produces cracks in the surface because of the ambiguities described above. An implementation in C of this method is presented by Bloomenthal [50].

These algorithms combine the principles of both spatial exhaustive enumeration and continuation. As a consequence, the main problem of continuation algorithms—i.e. the computation of at least one seeding point on each component of the surface—may be then solved by applying interval arithmetic to axially aligned rectangular boxes belonging to the complement of the union of surface-straddling cubes inside the bounding box.

7.4 Spatial Subdivision

Subdivision is an *adaptive* space partitioning technique. It is another attempt to solve the ambiguity problems resulting from the use of regular space grids. Before proceeding, let us recall that an *implicit object* is defined as the zero set of a real function $f : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, i.e. it is the solution set of an equation $f(\mathbf{p}) = 0$. For well-behaved functions, this zero set is a $(n - 1)$ -dimensional variety in \mathbb{R}^n ; in particular, such a zero set is an implicit curve in \mathbb{R}^2 or an implicit surface in \mathbb{R}^3 .

7.4.1 Quadtree Subdivision

This section shows how to achieve an *adaptive* polygonal approximation to a curve implicitly defined in \mathbb{R}^2 as follows:

$$\mathcal{C} = \{(x, y) \in \Omega \subseteq \mathbb{R}^2 : f(x, y) = 0\} \quad (7.14)$$

where Ω is the domain given by an axis-aligned bounding box. Following Lopes et al. [245], what we mean by *adaptive* is twofold: first, the subdivision of the bounding box Ω into smaller boxes is more intensive or finer near the curve \mathcal{C} ; second, the polygonal approximation is curvature-adaptive, i.e. the higher the curvature of \mathcal{C} , the finer is the quadtree subdivision (Figure 7.12).

The advantage of the quadtree subdivision over the spatial enumeration is that the size of the boxes is shape-adaptive so that eventual shape ambiguities are resolved by further subdivision. For example, to make sure that the curve depicted in Figure 7.12(b) does not self-intersect on the positive x -axis and near to the origin, the quadtree has been subdivided down to a finer resolution around there.

However, even so, if the resolution of the subdivision—i.e. the minimum size of boxes—is not enough, some small components and isolated points of the curve may remain undetected and are missed. In other words, the topological shape of the curve may be not preserved. The obvious solution for

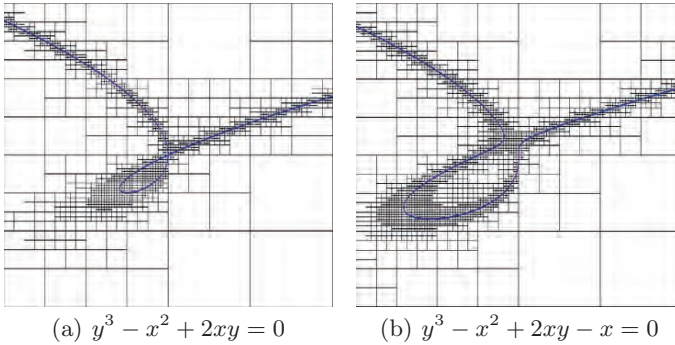


Fig. 7.12. Quadtree subdivision for two implicit curves.

this problem is to use interval arithmetic or affine arithmetic (see Chapter 4 for more details). Doing so, we add robustness to the curve polylineariser, in which the interval arithmetic plays the role of *curve locator* within each box. The curve exists inside a box if f takes on the value 0 over two perpendicular intervals or sides of a square. These boxes are called zero boxes. However, as seen in Chapter 4, there may be false zero boxes, and the results are even worse if floating-point computations are involved.

Note that a curve locator (e.g. interval arithmetic, affine arithmetic, or any of their variants) is not used to sample the curve because that would require to recursively subdivide a box down to a nearly infinitesimal resolution. Instead, we use a *root finder* to compute the curve points that result from the intersection between the curve and the edges of each zero box. Usually, such a root finder builds on some classical numerical method (e.g. bisection method, false position method or Newton's method), but there is no impediment to the usage of a symbolic root finder (e.g. Bézier root finder) based on the Descartes rule. However, most symbolic root finders only apply to polynomials, not to generic real functions.

The quadtree subdivision algorithm for implicit curves is described in Algorithm 24. This algorithm has three subdivision stopping conditions:

- *Inexistence of curve components.* Testing the existence of any curve segment inside a box \square_i is done through interval arithmetic. This criterion appears at the step 4 and discards the boxes of the quadtree that do not contain any curve component or segment. In fact, the box exclusion test $0 \notin \text{Image}(\square_i)$ guarantees that only empty boxes (i.e. boxes without any segment of the curve $f^{-1}(0)$) are immediately discarded. However, it may happen that—as explained in Chapter 4—not only true zero boxes, but also some false zero boxes will be considered for subdivision, i.e. there may be redundant and unnecessary subdivision of some boxes.
- *Maximum resolution.* The maximum resolution is the admissible minimum size of the boxes. When the size of a box falls below a given threshold Δ ,

Algorithm 24 Quadtree Subdivision Algorithm for Implicit Curves

```

1: procedure QUADTREE-BASEDIMPLICITCURVE( $f, \mathcal{C}, \Omega, \Delta, \tau$ )
2:   Subdivide  $\Omega$  into 4 equally sized boxes  $\square_i$ 
3:   for  $i \leftarrow 0, 3$  do
4:     if  $0 \notin \text{Image}_f(\square_i)$  then ▷ box exclusion test
5:       Discard  $\square_i$ 
6:     else
7:       if  $(\text{size}(\square_i) < \Delta) \vee (\text{curvature}(\mathcal{C}) < \tau)$  then
8:         Find roots of  $f$  along edges of  $\square_i$  ▷ curve points  $\mathcal{C} \cap \text{Fr}(\square_i)$ 
9:         Polylinearise the curve across  $\square_i$ 
10:       else
11:         QUADTREE-BASEDIMPLICITCURVE( $f, \mathcal{C}, \square_i, \Delta, \tau$ )
12:       end if
13:     end if
14:   end for
15: end procedure

```

the recursive subdivision stops and the box becomes a leaf box of the quadtree.

- *Minimum curvature.* The minimum curvature τ of the curve inside a given box works as a threshold below which the subdivision also stops. The idea is to stop subdividing a box when a curve segment inside it is approximately flat.

Note that Algorithm 24 also has the classical structure of a space partitioning algorithm, namely: partitioning (step 2), sampling (step 8), and polylinearisation (step 9). It is a robust algorithm because it uses interval arithmetic as a fast and robust discarder of empty boxes (i.e. boxes that do not contain any curve segment). However, it is not strictly necessary to use the interval arithmetic as a discarder of empty boxes. By evaluating f at the vertices of a given box \square_i , we are able, in principle, to check whether a box is empty or not. In fact, if f does not change sign at the vertices of \square_i , we conclude that \square_i is an empty box. But, this alternative technique for checking the transversality of the curve within a box fails if a small component of the curve lies entirely in a box; hence the use of interval arithmetic.

It seems that Suffern [377] was who first tried to use adaptive enumeration, instead of full enumeration, to approximate implicit curves. Shortly afterwards, Suffern and Fackerell [379] introduced interval arithmetic as a robust support for the enumeration of implicit curves, whose algorithm is essentially the Algorithm 24. Nevertheless, the credit of the first application of interval arithmetic in computer graphics is due to Mudur and Koparkar [287]. In [370, 371], Snyder describes a geometric modelling system based on interval arithmetic, which includes an approximation algorithm for implicit curves, but the corresponding quadtree decomposition is not adapted to the curvature. These pioneering works on interval methods in computer graphics have

given rise to interesting research results, in particular to the development of several variants of interval arithmetic (e.g. affine arithmetic [17, 90, 102]).

7.4.2 Octree Subdivision

Similar to 2D implicit curves, a 3D implicit surface \mathcal{S} is defined as a zero set of some real function f , whose domain is now in \mathbb{R}^3 :

$$\mathcal{S} = \{(x, y, z) \in \Omega \subseteq \mathbb{R}^3 : f(x, y, z) = 0\} \quad (7.15)$$

where Ω is the domain given by an axis-aligned bounding box.

As for quadtrees, the idea behind the octree subdivision is to provide an *adaptive* approximation to implicit surfaces. That is, a cubic box through which the surface passes is subdivided into eight smaller boxes. These smaller cubes are stored into an octree data structure (see Chapter 2 for further details). Therefore, the first stopping criterion for an octree approximation to an implicit surface is the emptiness of a given octree box (i.e. the box exclusion criterion). Also, the curvature of the surface within a box and box resolution (i.e. a box has reached its minimum size) can work as stopping criteria for the octree subdivision (step 7 of Algorithm 25). Other criteria appear listed in [49]. Their importance come from the fact that they reinforce the *adaptivity* of the approximation to the implicit surface.

However, these adaptive criteria are not sufficient to resolve all the topological ambiguities. Note that a leaf box is topologically unambiguous if the surface can be approximated by a single polygon within such a box. Topological unambiguity may work as the fourth stopping criterion of the octree subdivision. But, resolving all ambiguities may not be an easy task. For example, it is not easy to distinguish a surface consisting of two touching spheres

Algorithm 25 Octree Subdivision Algorithm for Implicit Surfaces

```

1: procedure BLOOMENTHAL( $f, \mathcal{S}, \Omega, \Delta, \tau$ )
2:   Subdivide  $\Omega$  into eight equally sized boxes  $\square_i$ 
3:   for  $i \leftarrow 0, 7$  do
4:     if  $0 \notin \text{Image}_f(\square_i)$  then ▷ box exclusion test
5:       Discard  $\square_i$ 
6:     else
7:       if  $(\text{size}(\square_i) < \Delta) \vee (\text{curvature}(\mathcal{S}) < \tau)$  then
8:         Find roots of  $f$  along edges of  $\square_i$  ▷ surface points  $\mathcal{S} \cap \text{Fr}(\square_i)$ 
9:         Polygonise the surface  $\mathcal{S}$  across  $\square_i$ 
10:      else
11:        BLOOMENTHAL( $f, \mathcal{S}, \square_i, \Delta, \tau$ )
12:      end if
13:    end if
14:  end for
15: end procedure

```
