Adaptive enumeration of implicit surfaces with affine arithmetic

LUIZ HENRIQUE DE FIGUEIREDO[1]   JORGE STOLFI[2]

[1]Computer Graphics Technology Group (TeCGraf), Computer Science Department, PUC–Rio
Rua Marquês de São Vicente 225, 22453–900 Rio de Janeiro, RJ, Brasil
lhf@icad.puc-rio.br

[2]Computer Science Department (DCC/IMECC), Univ. Estadual de Campinas (UNICAMP)
Caixa Postal 6065, 13081–970 Campinas, SP, Brasil
stolfi@dcc.unicamp.br

Abstract. We discuss adaptive enumeration and rendering methods for implicit surfaces, using octrees computed with affine arithmetic, a new tool for range analysis. Affine arithmetic is similar to standard interval arithmetic, but takes into account correlations between operands and sub-formulas, generally providing much tighter bounds for the computed quantities. The resulting octrees are accordingly much smaller, and the rendering faster. We also describe applications of affine arithmetic to intersection and ray tracing of implicit surfaces.
KEYWORDS: cellular models, interval analysis, rendering, implicit surfaces.

1 Introduction

Implicit surfaces have recently become popular in computer graphics and solid modeling. In order to exploit existing hardware and algorithms, it is often necessary to approximate such surfaces by models with simpler geometry, such as polygonal meshes or voxel arrays.

Let S be a surface defined implicitly by the equation $h(x, y, z) = 0$. A simple and general technique for computing an approximation of S in a region Ω is:

- decompose Ω into small cells;
- identify which cells intersect S;
- approximate S within each intersecting cell.

The enumeration of the intersecting cells is usually the most expensive step in this method. In the simplest schema, the cells that intersect S are identified by sampling. The function h is evaluated at the vertices of each cell; if the signs of those values are not identical, then the cell necessarily intersects S.

Obviously, the converse does not hold: if the values have all the same sign, we cannot conclude that the cell does not intersect S. Therefore, the cell size must be carefully chosen to avoid missing features due to undersampling.

Typically, the cellular decomposition of Ω is a regular grid of cubes. If there are n cubes along each main direction, then there are n^3 cubes to be scanned in a full enumeration, but only $O(n^2)$ cubes will intersect S. Thus, choosing a smaller cell size will greatly increase the number of cells to be scanned, and also increase the fraction of "useless" tests.

The remainder of the paper is organized as follows. In section 2, we review some general methods that try to find the cells that intersect S without enumerating all cells in Ω. The most reliable of those seems to be recursive subdivision of space based on range analysis, i.e., on estimates for the range of values taken by h on subsets of Ω. Interval arithmetic is the natural technique for range analysis. However, as we point out in section 3, the excessive conservatism of interval arithmetic may greatly reduce the efficiency of the search. In section 4, we describe *affine arithmetic*, a computation technique that generally provides much tighter bounds than interval arithmetic. In section 5, we describe some experiments that exploit the properties of affine arithmetic to efficiently enumerate implicit surfaces. In sections 6 and 7, we discuss other applications of affine arithmetic for computing implicit objects. Finally, section 8 contains some conclusions and outlines directions for future work.

2 Efficient enumeration methods

Several methods exist for finding the intersecting cells without enumerating all cells in a region. The most popular methods are based on sampling. However, hierarchical decomposition methods are more efficient, more robust, and also suitable for parallel implementation.

Sampling-based methods

Continuation methods for implicit surface enumeration attempt to reduce the number of cells scanned from $O(n^3)$ to $O(n^2)$ by sampling the surface only in the immediate neighborhood of known intersecting cells [Bloomenthal (1988), Chandler (1988), Allgower–Georg (1990), Dobkin–Levy–Thurston (1990)]. These methods only work for cellular subdivisions of Ω whose topology and geometry is well-known, so that it is simple to advance from one intersecting cell to an adjacent one.

Adaptive refinement techniques try to achieve the same goal by starting with a coarse cellular subdivision, and further subdividing only those cells that intersect S. Such techniques frequently use estimates of the curvature of S in a cell as an additional subdivision criterion, thus building a cellular subdivision that is adapted not only to the position of S inside Ω, but also to the intrinsic geometry of S [Bloomenthal (1988), Hall–Warren (1990), Velho (1990)].

All these methods based on sampling have one fundamental difficulty: finding a set of initial "seed" cells intersecting each connected component of S in Ω. Obviously, if S is a general implicit surface, and we are restricted to evaluating h at individual points, then this problem has no satisfactory solution. Even if we assume that S has no extended features thinner than the minimum cell size (so that there are no aliasing problems), we still must sample the whole region Ω uniformly at that resolution; otherwise, we may miss some component of S.

Hierarchical decomposition methods

In order to discard large portions of Ω quickly and reliably, we need a more powerful test than point sampling. The latter can only prove the *presence* of S in some region of Ω; to reduce the number of cells scanned, we need a test procedure that can also prove the *absence* of S in a region.

The *hierarchical decomposition methods* rely on such a test to explore Ω recursively, starting with Ω itself as the initial cell. If a cell is proved to be empty, it is ignored; otherwise, it is subdivided into smaller cells, which are then explored recursively, until the cells are small enough to approximate S [Suffern (1990), Suffern–Fackerell (1991), Duff (1992), Snyder (1992), Taubin (1994)].

The meaning of "small enough" depends on the application. For rendering, it might mean "smaller than a pixel"; for other applications, such as modeling, it may depend on some other numerical criterion, e.g., how closely S can be approximated by a linear function inside the cell.

Note that the test procedure is not required to be complete, in the sense that it may fail to prove either the presence or the absence of S in a given region. In particular, a cell that is declared "small enough" may still have unknown status. Each application must decide what to do with those "indeterminate" cells: discard them, treat them just like the cells that do intersect S, or handle them in some special way. Point sampling may be helpful here.

The resulting subdivision is not a regular grid, but rather a tree of nested cells. The leaves in this tree form a cellular model for S. There are many variants of this method which differ in the shape of cells and the method of subdivision. In particular, if the cells are cubical and divided into eight equal parts, then the resulting subdivision is called an *octree* [Samet (1990)]. Other choices give *binary space partitions* [Fuchs–Kedem–Naylor (1980)], *3-d trees* [Bentley (1975)], *hierarchical triangulations* [Persiano–Salim–Bueno (1991)], and many more.

3 Interval arithmetic

In order to prove that a given cell C does not intersect S, we must prove that the function h does not vanish inside C. For some classes of surfaces, this fact can usually be established by specific tests. For instance, if h is a polynomial, we can compute its Bézier-Bernstein coefficients for that cell: if they are all positive, or all negative, then the same will be true of h. Unfortunately, this method cannot be used for general (non-polynomial) functions; or even for polynomials of high degree presented in factored form, such as $(x^2 + y^2 + z^4)^{20} - 1$.

The classical technique of *interval arithmetic* (IA), also known as *interval analysis*, provides a natural solution to this problem. In IA, each quantity is represented by an interval of floating-point numbers; these intervals are added, subtracted, multiplied, etc., in such a way that each computed interval is guaranteed to contain the (unknown) value of the quantity it represents [Moore (1966)].

To test whether a cell C intersects the surface S, we can evaluate $h(x, y, z)$ with interval arithmetic, letting x, y and z be the projections of the cell onto the coordinate axes. The interval thus computed will contain all values of h for points inside the cell. If this interval is entirely positive, or entirely negative, then we have proved that C does not intersect S.

Hierarchical decomposition methods based on interval arithmetic have recently been proposed for the enumeration of implicit surfaces in computer graphics applications [Suffern–Fackerell (1991), Duff (1992), Snyder (1992)]. Those methods have become quite popular, due to their ability to handle arbitrarily complex non-polynomial surfaces, and their immunity to round-off errors.

The error explosion problem

The main weakness of IA is that it tends to be too conservative: the computed interval for a quantity may be much wider than the true range of that quantity. The over-conservatism of IA is mainly due to its assumption that the (unknown) values of the arguments may vary *independently* over the given interval

For example, consider evaluating $x(10 - x)$, where x is known to lie in the interval $\bar{x} = [4 .. 6]$. Applying the IA formulas blindly, we get

$$\begin{aligned} 10 - \bar{x} &= [10 .. 10] - [4 .. 6] = [4 .. 6] \\ \bar{x}(10 - \bar{x}) &= [4 .. 6] \cdot [4 .. 6] = [16 .. 36], \end{aligned}$$

which is 20 times wider than the true range of the expression $x(10 - x)$ over $[4 .. 6]$, namely $[24 .. 25]$. The large discrepancy between the two intervals is due to the inverse relation between the quantities x and $10 - x$, which is not known to the interval multiplication algorithm.

This over-conservatism of IA is particularly bad in long computation chains, where the intervals computed by one stage of the chain are the inputs to the following stage. In such cases, one often observes an "error explosion": as the evaluation advances down the chain, the relative accuracy of the computed intervals decreases exponentially, and they soon become too wide to be useful, by many orders of magnitude.

4 Affine arithmetic

To address this "error explosion" problem, Comba and Stolfi (1993) proposed a new model for numerical computation, called *affine arithmetic* (AA). Like standard interval arithmetic, affine arithmetic keeps track automatically of the round-off and truncation errors affecting each computed quantity. Unlike interval arithmetic, however, affine arithmetic keeps track of *correlations* between those quantities. Thanks to this extra information, AA is able to provide much tighter intervals than IA, especially in long computation chains.

The key feature of AA is an extended encoding of quantities from which one can determine, in addition to their ranges, also certain relationships to other quantities — such as the ones existing between x and $10 - x$ in the example. Specifically, a partially unknown quantity x is represented in AA by an *affine form* $\hat{x}$, which is a first-degree polynomial:

$$\hat{x} = x_0 + x_1\varepsilon_1 + x_2\varepsilon_2 + \cdots + x_n\varepsilon_n.$$

Here, the x_i are known real coefficients (stored as floating-point numbers), and the ε_i are symbolic variables whose values are unknown but assumed to lie in the interval $\mathbf{U} = [-1 \,..\, +1]$. Each ε_i is called a *noise symbol*: it stands for an independent source of error or uncertainty that contributes to the total uncertainty of the quantity x; the corresponding coefficient x_i gives the magnitude of that contribution. The source of error may be external (due to original uncertainty in some input quantity) or internal (due to round-off and truncation errors committed in the computation of $\hat{x}$).

A somewhat similar approach has been proposed by Hansen (1975), in which quantities are represented instead by affine combinations of a *fixed* number of *intervals*.

As one may expect, affine arithmetic is more complex and expensive than ordinary interval arithmetic. However, its higher accuracy is worth the extra cost in many applications, including adaptive enumeration of implicit objects, as we shall see later.

Conversions between IA and AA

If $\hat{x} = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n$ is an affine form for a quantity x, then the value of x is guaranteed to be in the *range* of $\hat{x}$, the interval

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

Note that $[\hat{x}]$ is the smallest interval that contains all possible values of $\hat{x}$, assuming that each ε_i ranges independently over the interval $\mathbf{U} = [-1 \,..\, +1]$.

Conversely, given an interval $\bar{x} = [x.lo \,..\, x.hi]$ representing some quantity x in IA, a valid affine form for the same quantity is given by $\hat{x} = x_0 + x_k\varepsilon_k$, where

$$x_0 = \frac{x.hi + x.lo}{2}, \qquad x_k = \frac{x.hi - x.lo}{2}\,.$$

The noise symbol ε_k symbolizes the uncertainty in the value of x. Since the interval $\bar{x}$ tells us nothing about possible constraints between the value of x and that of other variables, ε_k must be distinct from all other noise symbols previously used in the same computation.

Exploiting correlations in operands

The key feature of AA is that the same noise symbol ε_i may contribute to the uncertainty of two or more quantities (inputs, outputs, or intermediate results) arising in the evaluation of an expression. The sharing of a noise symbol ε_i by two affine forms $\hat{x}, \hat{y}$ indicates some partial dependency between the underlying quantities x, y. The magnitude and sign of the dependency is determined by the corresponding coefficients x_i, y_i.

For example, suppose two quantities x, y are represented by the affine forms

$$\begin{aligned} \hat{x} &= 10 + 2\varepsilon_1 + 1\varepsilon_2 \qquad\quad - 1\varepsilon_4 \\ \hat{y} &= 20 - 3\varepsilon_1 \qquad\quad + 1\varepsilon_3 + 4\varepsilon_4. \end{aligned}$$

From this data, we can tell that x lies in the interval $[6 \,..\, 14]$, and y lies in $[12 \,..\, 28]$; however, since they both include the same noise variables ε_1 and ε_4 with non-zero coefficients, they are not entirely independent of each other. In fact, the pair (x, y) is constrained to lie in the region of $\mathbf{R}^2$ depicted in Figure 1 (dark grey), which is substantially smaller than the rectangle $[6 \,..\, 14] \times [12 \,..\, 28]$ (light grey).

Obviously, this dependency information would be lost if we were to replace $\hat{x}$ and $\hat{y}$ by the ordinary intervals $[\hat{x}]$ and $[\hat{y}]$, even though the latter encode precisely the same ranges of values as the former.

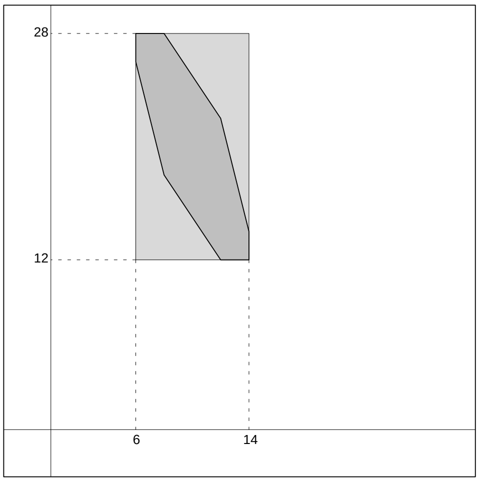

Figure 1: joint range of x and y, for $\hat{x} = 10 + 2\varepsilon_1 + 1\varepsilon_2 - 1\varepsilon_4$ and $\hat{y} = 20 - 3\varepsilon_1 + 1\varepsilon_3 + 4\varepsilon_4$.

Computing with affine arithmetic

To evaluate a formula in AA, we must replace each of its elementary operations $z \leftarrow f(x, y)$ on real numbers by an equivalent operation $\hat{z} \leftarrow \hat{f}(\hat{x}, \hat{y})$ on affine forms, where $\hat{f}$ is a procedure that computes an affine form for $z = f(x, y)$ that is consistent with $\hat{x}, \hat{y}$.

When f is an affine function of x, y, the value $\hat{z}$ can be expressed exactly as an affine combination of the noise symbols ε_i. Thus, if

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

and $\alpha \in \mathbf{R}$, then

$$\hat{x} \pm \hat{y} = (x_0 \pm y_0) + (x_1 \pm y_1)\varepsilon_1 + \cdots + (x_n \pm y_n)\varepsilon_n$$
$$\alpha\hat{x} = (\alpha x_0) + (\alpha x_1)\varepsilon_1 + \cdots + (\alpha x_n)\varepsilon_n$$
$$\hat{x} \pm \alpha = (x_0 \pm \alpha) + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n.$$

Note that, according to those formulas, the difference $\hat{x} - \hat{x}$ between an affine form and itself is identically zero. In this case, the fact that the two operands share the same noise symbols with the same coefficients reveals that they are actually the same quantity, and not just two quantities that happen to have the same range of possible values. Thanks to this feature, in AA we also have $(\hat{x} + \hat{y}) - \hat{x} = \hat{y}$, $(3\hat{x}) - \hat{x} = 2\hat{x}$, and so on. Such properties are *not* valid in IA, and are one source of error explosion.

When f is not an affine operation, $\hat{z}$ cannot be expressed exactly as an affine combination of the ε_i. In that case, we pick the best affine approximation (best in the Chebyshev sense of minimizing the maximum error), and then append an extra term $z_k\varepsilon_k$ to represent the error introduced by this approximation:

$$\hat{z} = z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n + z_k\varepsilon_k.$$

Here, ε_k must be a brand new noise symbol (distinct from all other noise symbols in the same computation) and z_k must be an upper bound for the approximation error. Thus, unlike Hansen's generalized interval arithmetic, new noise symbols are created during a long AA computation.

Using this approach, the multiplication of two affine forms $\hat{x}$, $\hat{y}$ is given by

$$\hat{x}\hat{y} = x_0y_0 + (x_0y_1 + y_0x_1)\,\varepsilon_1 + \cdots + (x_0y_n + y_0x_n)\,\varepsilon_n + uv\,\varepsilon_k,$$

where

$$u = \sum_{i=1}^{n} |x_i|\,, \qquad v = \sum_{i=1}^{n} |y_i|\,.$$

Similar formulas can be given for the other elementary operations and functions, such as square root.

Example revisited

Consider again evaluating $z = x(10 - x)$, for x in $[4 \mathinner{..} 6]$, but now with AA instead of IA:

$$\begin{aligned} \hat{x} &= \;\; 5 + 1\varepsilon_1 \\ 10 - \hat{x} &= \;\; 5 - 1\varepsilon_1 \\ \hat{z} = \hat{x}(10 - \hat{x}) &= 25 + 0\varepsilon_1 - 1\varepsilon_2. \end{aligned}$$

Observe that the influence of the noise symbol ε_1 in the factors happened to cancel out (to first order) in the product.

The range of z implied by the affine form $\hat{z}$ is

$$[\hat{z}] = [25 - 1 \mathinner{..} 25 + 1] = [24 \mathinner{..} 26],$$

which is much closer to $[24 \mathinner{..} 25]$, the true range of z. Recall that IA gave $[16 \mathinner{..} 36]$ for this expression.

5 Examples of adaptive enumeration with AA

Pictures of three dimensional decompositions are not too easy to understand; so, we shall only give pictures of the enumeration of two-dimensional implicit curves. In all these examples, the goal is to build a model of a curve given implicitly by $h(x, y) = 0$ inside some rectangle Ω. The model consists of a set of cells from some fixed regular grid, whose union is guaranteed to contain all zeros of h in Ω.

A quartic

Take the quartic curve defined by

$$h(x, y) = x^2 + y^2 + xy - (xy)^2/2 - 1/4$$

in the square $\Omega = [-2 \mathinner{..} 2] \times [-2 \mathinner{..} 2]$, using a 32×32 grid of cells. For full enumeration, $1024 = 32 \cdot 32$ cells have to be scanned; the curve actually enters only 66 of these cells.

Figure 2 illustrates adaptive enumeration with interval arithmetic and affine arithmetic, using a 2-d tree. With IA, the range of h was evaluated 847 times and 246 cells remained in the model (i.e., could not be shown to be disjoint from the curve). With AA, the range of h was evaluated 451 times and only 70 cells remained in the model.

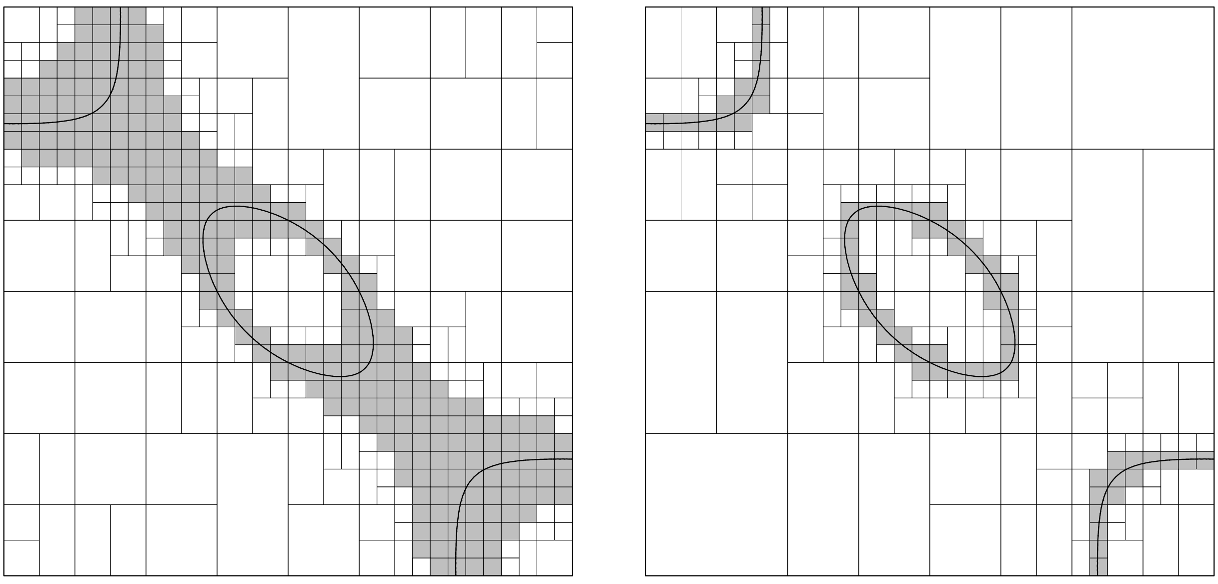

Figure 2: adaptive enumeration of quartic (a) IA (b) AA.

Quadratic electrostatic potential

Consider the "quadratic" electrostatic potential defined by n charges q_i located at points (x_i, y_i):

$$h(x,y) = \sum_{i=1}^{n} \frac{q_i}{(x-x_i)^2 + (y-y_i)^2} \ .$$

(True electrostatic potential depends on distances, not on square of distances). In this case, the interesting curves are the equipotential curves $h(x,y) = h_0$.

Figure 3 illustrates an adaptive quadtree enumeration with interval arithmetic and affine arithmetic of an equipotential curve for the quadratic electrostatic potential defined by four equal charges. The grid is 64×64; full enumeration would have to scan 4096 cells. In both cases, the range of h was evaluated 1449 times (a coincidence). However, 346 cells remained in the model generated by IA, whereas only 299 remained in the model generated by AA.

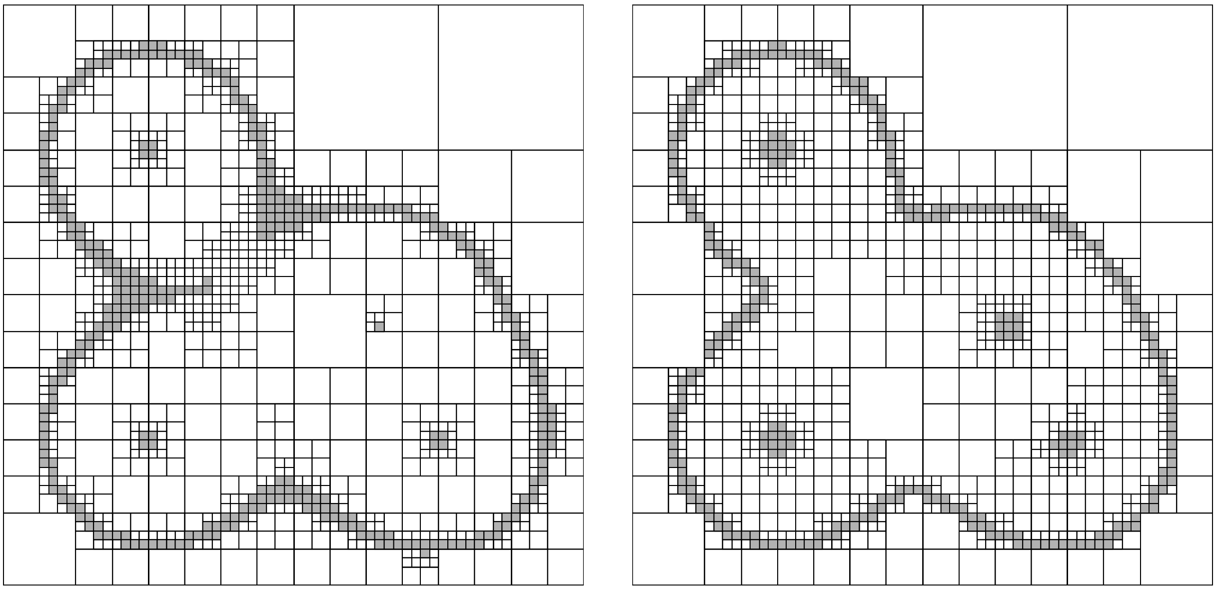

Figure 3: adaptive enumeration of quadratic electrostatic potential (a) IA (b) AA.

The quartic in 3d

Consider an octree enumeration of the quartic surface defined by

$$h(x,y,z) = x^2 + y^2 + xy - (xy)^2/2 - 1/4 - z,$$

in the cube $\Omega = [-2 \,..\, 2] \times [-2 \,..\, 2] \times [-2 \,..\, 2]$. Figure 4a shows how the size of the model (number of leaves in the octree) and the time (in seconds) taken to compute it using IA depend on the resolution of the underlying grid (the resolution is 2^L at level L). Figure 4b shows the same data for AA. We observe that AA evaluates h half as many times as IA, generates a model half the size of the model generated by IA, but is 4.3 times slower than IA (times measured in an otherwise idle SPARCstation 10).

6 Rendering

The octrees built during enumeration can be used to speed up ray tracing algorithms [Glassner (1984), Wyvill–Kunii–Shirai (1986)]. However, a "lower" quality rendering can be done directly, concurrently with the enumeration; there is no need to explicitly build an octree.

By composing the function h with the appropriate projective map, we can take the domain Ω to be a rectangular box, aligned with the coordinate axes, and compute the image of the orthogonal

level	evaluations	leaves	time
1	9	8	0.0
2	73	62	0.0
3	569	334	0.0
4	3241	1276	0.1
5	13449	5122	0.2
6	54425	20580	0.4
7	219065	82502	1.5
8	879081	329746	5.7
9	3517049	1318058	22.5
10	14061513	5275400	89.5

(a) IA

level	evaluations	leaves	time
1	9	8	0.0
2	73	56	0.0
3	521	190	0.1
4	2041	710	0.2
5	7721	2664	0.5
6	29033	10104	1.7
7	109865	39960	6.4
8	429545	158282	24.8
9	1695801	630380	97.8
10	6738841	2516356	388.5

(b) AA

Figure 4: performance of adaptive enumeration of 3d quartic.

projection of the surface S defined by h onto a horizontal plane located above the box Ω. For simplicity, we assume that the resolution of the image is $n \times n$, where $n = 2^L$. The enumeration proceeds down the octree to level L. When we reach a leaf of the octree (i.e., a cell at level L), we compute the color of the corresponding pixel in the image. Each cell has an integer address (x, y, z); all cells with the same x, y project onto the same pixel (x, y). To remove hidden parts, we use a "painter's algorithm", carefully traversing the octree from bottom to top (in the z direction).

In order to compute the color of a pixel in shaded images of S, we need to compute also the mean normal direction to S in the cells corresponding to the pixel. The normal direction is the direction of the gradient of h. Full enumeration and adaptive enumeration with interval arithmetic do not directly provide an estimate for ∇h. For full enumeration, one typically uses an estimate based on central differences [Levoy (1988)]. For adaptive enumeration with interval arithmetic, techniques such as formal differentiation (evaluated in either floating point or IA) or Bauer–Strassen differentiation [Matijasevich (1985)] would be required to estimate ∇h in each visible cell. On the other hand, affine arithmetic directly provides an estimate for ∇h. Recall that, to compute h in a cell represented by the intervals $\bar{x}$, $\bar{y}$, $\bar{z}$, we take the corresponding affine forms $\hat{x} = x_0 + x_1\varepsilon_1$, $\hat{y} = y_0 + y_2\varepsilon_2$, $\hat{z} = z_0 + z_3\varepsilon_3$, and compute $\hat{h} = h_0 + h_1\varepsilon_1 + h_2\varepsilon_2 + h_3\varepsilon_3 + \ldots$ Then we can then take $(h_1/x_1, h_2/y_2, h_3/z_3)$ as an estimate for the mean value of ∇h in this cell.

Figure 5 shows a 256×256 image of the union of two spheres rendered using this technique.

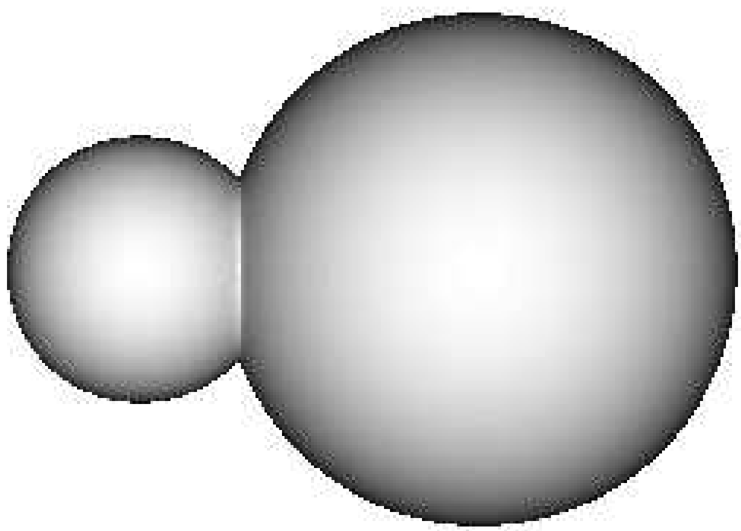

Figure 5: the union of two spheres enumerated and shaded with AA.

7 Other applications

Computing implicit objects with AA is also useful for other applications. We describe two such applications briefly: surface intersection, and rendering with ray tracing.

Intersection of implicit surfaces

The intersection of two implicit surfaces defined by functions f and g can be given implicitly by a single function $h = f^2 + g^2$. This expression is not useful for enumeration based on point sampling because h is always positive (and singular at the intersection); such enumeration depends on changes of sign in h to identify intersecting cells. On the other hand, enumeration using range analysis is perfectly feasible for such functions. Again, affine arithmetic will be able to exploit the correlations present in the expression of h to compute tighter bounds. Alternatively, the enumeration for the intersection can be done by combining the simultaneous enumeration of the two surfaces, rejecting cells that are rejected in at least one enumeration.

Ray tracing

A fundamental step in ray tracing a surface is finding the first intersection of a ray with the surface. If the surface is given implicitly by $h(x, y, z) = 0$, and the ray is the line segment $\overrightarrow{pq}$, then one needs to find the smallest root of the univariate function

$$f(t) = h((1-t)x_p + tx_q,\ (1-t)y_p + ty_q,\ (1-t)z_p + tz_q),$$

in the interval $t \in [0 \mathinner{..} 1]$.

A simple algorithm for finding all roots of f is the following. Evaluate $\bar{u} = \bar{f}(\bar{t})$ using IA, for the whole interval $\bar{t} = [0 \mathinner{..} 1]$. If the resulting interval $\bar{u}$ is strictly positive or strictly negative, then the ray does not intersect the surface. Otherwise, split $\bar{t}$ in two equal parts, and recursively search each half. Stop the recursion when the interval $\bar{t}$ is small enough for the application. A simple variant of this algorithm finds the smallest root: only search the right half if the left half has been shown to contain no roots.

One drawback of this algorithm is that evaluating $\bar{f}(\bar{t})$ with IA is equivalent to evaluating $\bar{h}(\bar{x}, \bar{y}, \bar{z})$ on the intervals $\bar{x} = [x_p \mathinner{..} x_q]$, $\bar{y} = [y_p \mathinner{..} y_q]$, $\bar{z} = [z_p \mathinner{..} z_q]$ — that is, evaluating h on the axis-aligned bounding box of the segment pq, instead of only along the segment itself. Again, the problem arises because IA does not know that the arguments $\bar{x}$, $\bar{y}$, and $\bar{z}$ of $\bar{h}$ are highly correlated. Obviously, the bounding box of the segment pq may intersect the surface even when the segment itself does not.

Replacing standard IA by affine arithmetic will generally improve the performance of this algorithm. Even without any algebraic manipulation, AA will automatically notice that the affine forms $\hat{x}$, $\hat{y}$, and $\hat{z}$ are strongly correlated, and thus will usually produce tighter bounds for $f(\hat{t})$.

Moreover, as we have seen above, AA also automatically provides estimates for the derivative of f, allowing us to replace the binary search used in the IA method by Newton's method, without explicitly computing derivatives.

8 Conclusion

Numerical experiments show that affine arithmetic is indeed more accurate than standard interval arithmetic. On the other hand, AA is more complex and expensive than IA. However, its higher accuracy is worth the extra cost for adaptive enumeration of implicit objects because the models resulting from enumeration with AA are smaller. Although the relative speeds of IA and AA depend on the particular function being evaluated, in our implementation, AA is typically 4–5 times slower than IA. Further work is necessary to improve the performance of AA.

An interesting extension of the techniques described here would be to use the extra information provided by AA to estimate surface curvature, so that the enumeration is adapted not only to the location of the surface in the ambient space, but also to its intrinsic geometry, as adaptive refinement techniques do.

Acknowledgements

We thank João Comba, who helped implement a prototype AA package in Modula-3, and Marcus Vinicius Andrade, who helped debug the C version and wrote an implicit surface ray-tracer based on it. This work was performed on equipment kindly ceded by Paulo Correa de Mello (Chemistry Dept., PUC-Rio), and on equipment acquired within a FAPESP thematic project. The authors are partially supported by research grants from the Brazilian Council for Scientific and Technological Development (CNPq). Some of this research was performed while the second author was at the DEC Systems Research Center (SRC) in Palo Alto.

9 References

E. L. Allgower, K. Georg, *Numerical Continuation Methods: An Introduction*, Springer-Verlag, 1990.

J. L. Bentley, Multidimensional binary search trees used for associative searching, *Communications of the ACM* **18** (1975) 509–517.

J. Bloomenthal, Polygonization of implicit surfaces, *Computer Aided Geometric Design* **5** (1988) 341–355.

R. E. Chandler, A tracking algorithm for implicitly defined curves, *IEEE Computer Graphics & Applications* **8** (1988) 83–89.

J. L. D. Comba, J. Stolfi, Affine arithmetic and its applications to computer graphics, *Proceedings of VI SIBGRAPI* (1993) 9–18 (VI Brazilian Symposium on Computer Graphics and Image Processing, Recife, 1993).

D. P. Dobkin, S. V. F. Levy, W. P. Thurston, A. R. Wilks, Contour tracing by piecewise linear approximations, *ACM Transactions on Graphics* **9** (1990) 389–423.

T. Duff, Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry, *Computer Graphics* **26** (1992) 131–138 (Proceedings of SIGGRAPH '92).

H. Fuchs, Z. M. Kedem, B. F. Naylor, On visible surface generation by a priori tree structures, *Computer Graphics* **14** (1980) 124–133 (Proceedings of SIGGRAPH '80).

A. Glassner, Space subdivision for fast ray tracing, *IEEE Computer Graphics & Applications* **4** (1984) 15–22.

M. Hall, J. Warren, Adaptive polygonization of implicitly defined surfaces, *IEEE Computer Graphics & Applications* **10** (1990) 33–42.

E. R. Hansen, A generalized interval arithmetic, *Lectures Notes in Computer Science* **29** (1975) 7–18.

M. Levoy, Display of surfaces from volume data, *IEEE Computer Graphics & Applications* **8** (1988) 29–37.

Yu. V. Matijasevich, A posteriori interval analysis, *Lectures Notes in Computer Science* **204** (1985) 328–334.

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

R. M. Persiano, M. Salim, L. P. Bueno, Boundary evaluation of CSG solids by simplicial subdivision, *Proceedings of COMPUGRAPHICS '91*, volume II, 220–229.

H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990.

J. M. Snyder, Interval analysis for computer graphics, *Computer Graphics* **26** (1992) 121–130 (Proceedings of SIGGRAPH '92).

K. G. Suffern, Quadtree algorithms for contouring functions of two variables, *The Computer Journal* **33** (1990) 402–407.

K. G. Suffern, E. D. Fackerell, Interval methods in computer graphics, *Computers & Graphics* **15** (1991) 331–340.

G. Taubin, Rasterizing algebraic curves and surfaces, *IEEE Computer Graphics & Applications* **14** (1994) 14–23.

L. Velho, Adaptive polygonization of implicit surfaces using simplicial decomposition and boundary constraints, *Proceedings of Eurographics '90*, 125–136.

G. Wyvill, T. L .Kunii, and Y. Shirai, Space division for ray tracing in CSG, *IEEE Computer Graphics & Applications* **6** (1986) 28–34.