Reliable algorithms for ray intersection in computer graphics based on interval arithmetic

J.F. Sanjuan-Estrada, L.G. Casado and I. García
Department of Computer Architecture and Electronics, University of Almería, Spain.
jsanjuan@ual.es, leo, inma@ace.ual.es

Abstract

In this work we study the reliability and performance of interval arithmetic for ray tracing implicit surfaces. We analyze when and how to use interval arithmetic as an alternative to the methods used in POV-Ray for ray intersection. Interval methods are applied as robust approaches for solving the ray-surface intersection problem; i.e. to find the minimal root in a set of analytic functions. POV-Ray is able to solve this problem efficiently for relatively simple objects (objects that can be bounded in a box). Interval based algorithms can speed up the rendering process for scenes with some infinite implicit surfaces, including non-differentiable ones, where the automatic bounding box cannot be applied. Interval methods have the advantage of not needing to provide the interval guess of the maximum gradient, as Recursive subdivision equipotential methods (used by POV-Ray) does. Experimental results were obtained from the evaluation of our interval algorithms and the methods used in POV-Ray in scenes with single objects and in complex scenes with several objects.

1. Introduction

An important topic in the computer graphics and visualization field is the concept of implicit surfaces which facilitates the production of synthetic scenarios. In this context, ray tracing is a simple and powerful approach to realistic image visualization. The concept of the ray tracing method is illustrated in Figure 1, where a two-dimensional image is generated from a scene which contains a set of objects and an illumination model. One of the fundamental operations in ray tracing consists of the calculation of intersections between rays and objects [1, 3, 11, 14]. In the case that each object is defined by an implicit surface, intersections can be seen as the roots of a parametric function obtained from the equations which describe the ray and the object. So, given an implicit surface $S$ defined as:

$$S = \{ s = (x, y, z) | f(x, y, z) = 0 \}$$

and a ray defined as the set of points:

$$R = \{ s = (a_x + tV_x, a_y + tV_y, a_z + tV_z) ; \ t \geq 0 \}$$

where $(a_x, a_y, a_z)$ is the viewpoint of the scene, and $(V_x, V_y, V_z)$ is the unit vector defining the direction of the ray. Intersections between $S$ and $R$, is given by the set of points in $S \cap R$; i.e. the points $s(t) : t \geq 0$, such that:

$$f(x, y, z) = f(s(t)) = f(a_x + tV_x, a_y + tV_y, a_z + tV_z) = 0$$

If $f(s(t)) \neq 0 \ \forall t \geq 0$ then $S \cap R = \emptyset$ and there is no intersection between the surface and the ray. In the case where there exist one or more values of $t (t_0, t_1, \ldots ; t_1 \geq 0)$ for which $f(s(t_i)) = 0$, then the point to visualize on the screen corresponds to the smallest value of $t_i$. This problem can also be posed as a minimization problem, where it is necessary to determine:

$$t^* = \min \{ t \ : \ f(s(t)) = 0, \ t \in R \} \quad (1)$$

Most of the algorithms previously proposed were not specifically designed for finding the first root but all the roots. Nevertheless, in the field of computer graphics only the minimal root contains valuable information, so it is advisable to devise specific algorithms for finding only the first root (see [1, 3, 5, 11, 14]).

Several approaches were proposed for solving this problem. The simplest approach is based on the use of grid techniques which produce a dense mesh starting from the left margin of the interval and progressing by $\epsilon$, until two consecutive values of the function $f(t)$ and $f(t + \epsilon)$ have different signs [16]. For small values of $\epsilon$, this approach is very (but not totally) reliable and computationally very expensive.

A second approach consists of using standard local techniques which achieve a rapid convergence to a root. The
main drawback of these methods is that convergence is not assured when \( f(t) \) is a multi-extremal function [16]. Moreover, if the objective function \( f(t) \) has several roots, different choices of the initial conditions can lead to different solutions for the equation \( f(t) = 0 \). A standard and fast local search technique is the well known Newton method, which frequently provides a quadratic convergence. Nevertheless, Newton method may also fail, depending on the initial point. For instance, it does not converge for the function \( f(x) = x^3 - x \), if the initial point is \( x = \frac{1}{\sqrt{3}} \).

The diagram illustrates a ray tracing diagram. A robust ray intersection algorithm was proposed by Mitchell [14]. It is based on an interval algorithm introduced by Moore [15] and distinguishes two stages: (i) the root isolation stage consists of finding the set of subintervals which are known to contain one and only one root, and (ii) the root refinement stage which uses a standard and fast local search technique, such as the Newton method. Caprani et. al. [1] improved Mitchell’s algorithm by using interval methods in both stages (isolation and refinement). They demonstrated that using interval analysis in both stages does not increase the execution time, as was stated by Mitchell. One of the advantages of interval methods is that they can provide information on the local uniqueness of the roots at the solution intervals [1, 10, 15]. Interval methods guarantee that an interval is rejected only when it does not contain a root.

The third class of approaches, as that proposed by Kalra and Barr [11], works on the framework of Global Optimization techniques based on the computation of Lipschitzian derivatives, in the visualization and computer graphics field.

The fourth approach to solve (1) consists of using existing Nonlinear Global Optimization algorithms based on the Branch-and-Bound strategy and Interval Arithmetic, as those proposed in [9, 10, 12].

The paper of Cusatis et al. [3] also studies the performance of affine arithmetic as a replacement of interval arithmetic in interval methods for ray tracing implicit surfaces.

In this work we analyze and evaluate the algorithms developed by Casado et al. [2] in the framework of algorithms for ray tracing implicit surfaces. The computational model of the algorithms is based on interval arithmetic computations [14, 15].

In Casado et al. [2], two algorithms were specifically designed for finding the minimal root in one-dimensional functions. These algorithms do not need to determine Lipschitz constants nor to evaluate derivatives. A short description of these algorithms called MRF (Minimal Root Finder) and MRFor (Minimal Root Finder reorder) will be given in Section 2 and a detailed one can be found in Casado et al. [2].

The third algorithm considered in this paper provides a method for solving a generalization of the problem (1). This generalization consists of finding the minimal root in a set of \( p \) functions instead of a single function; i.e. to find the minimum of the minimal root of several functions [2]. This is a typical problem arising in the computer graphic field, where the scene to visualize is composed of a set of objects defined by their implicit surfaces without automatic bounding box. Calculation of intersections between rays and primitive solids or surfaces is one of the fundamental operations to carry out. Visualizing a scene implies that for every ray only one of the intersections between that ray and all the intersected objects is needed (the closest to the observation point) [11, 14]. Mathematically, the problem can be posed as follows:

Given a ray, represented parametrically by a starting point \( a=(a_x, a_y, a_z) \), a direction vector \( V \) and a set of \( p \) implicit surfaces, \( f_i(x, y, z) = f_i(a_x + tV_x, a_y + tV_y, a_z + tV_z) = f_i(t) \); i.e. \( f_i : t \in \mathbb{R} \rightarrow \mathbb{R}; i = 1, \ldots, p \), to determine:

\[
 t^* = \min \{ t : f_i(t) = 0; \quad \forall i, i = 1, \ldots, p \} \tag{2}
\]

Casado et al. [2] have proposed algorithms to deal with the problem described by (2) as extensions of MRF (MRFeX) and MRFor (MRForEx). Both algorithms were extensively evaluated for a wide set of one-dimensional real functions.

In this work we apply versions of MRF, MRFor and MRForEx algorithms to the field of computer graphics, and for scenes defined by differentiable implicit functions the Newton algorithm is also evaluated [1, 8]. They are analyzed and evaluated in the context of visualizing synthetically created scenes using the framework of POV-Ray (Persistence of Vision Ray-Tracer) and the library C-XSC for interval functions evaluation (C++ class library for Extended Scientific Computing) [13].

This paper has been organized in the following sections: Section 2 briefly describes versions of the algorithms MRF,
2. Robust ray-surface intersection algorithms

In this work the methods used for solving the problem (1) falls into the general framework of Branch-and-Bound algorithms (B&B). The basic idea in B&B methods consists of a recursive decomposition of the original problem into smaller disjoint subproblems until the solution is found. The search avoids visiting those subproblems which are known not to contain a solution. B&B methods can be characterized by four rules: Branching, Selection, Bounding, and Elimination. For problems where the solution is determined when a desired accuracy is reached, a Termination rule has to be incorporated.

Algorithms MRF, MRFro and MRFroEx have the following common rules: the Branching rule consists of bisecting the current interval; the Selection rule examines the set of non-rejected intervals and chooses the closest to the viewpoint; the Bounding rule is based on interval arithmetic; the Elimination rules basically consist of verifying that $0 \notin F(X)$; the Termination rule is determined by the width of the interval.

In this work extended interval arithmetic has been used [12]. Other inclusion functions can be obtained by centered forms [12] or affine arithmetic [3]. These inclusion functions often obtain a narrower interval than extended interval arithmetic.

Algorithms MRF and MRFro differ mainly in the order in which these rules are applied. Here we will provide a brief description of these algorithms; details about their implementation can be found in Casado et al. [2].

Given an analytic function $f(x) : x \in [a, b] \in \mathbb{R}$, it is feasible to obtain an inclusion function $\bar{F}$ defined on the subset $X \subseteq [a, b] \subset \mathbb{R}$ such that $f(X) = \{ f(x) : x \in X \} \subseteq F(X)$. $F(X) = [\bar{F}(X), \overline{F}(X)]$ is an interval where the complete set of values of $f$ on $X$ falls in [10, 12, 15]. Specifically, if $0 \notin \bar{F}(X)$ then $0 \notin f(x); \forall x \in X$, so the evaluation of the inclusion function can be used to eliminate intervals where the function $f$ does not touch the zero line at all. Algorithm MRF can be described by the following steps:

**MRF Algorithm**

1. Evaluate $F([a, b])$.
2. If the resulting interval does not contain zero ($0 \notin F([a, b])$), then $[a, b]$ is rejected and this interval is not evaluated anymore.
3. If $(b - a) \leq \epsilon$ then the algorithm finishes and returns $c = \frac{b + a}{2}$ as the minimal root.
4. Divide the interval $[a, b]$ at its midpoint $([a, c], [c, b])$. If $\bar{F}([a, c]) \cdot \overline{F}([c, b]) \leq 0$ then the interval $[c, b]$ is rejected.
5. This algorithm is applied to all the generated but not rejected intervals until the algorithm does find out a solution or all the intervals have been rejected. The algorithm always works on the left most interval.

The MRF algorithm focuses on finding out the minimal root of a function in an interval. It is carried out by removing those intervals that are known not to contain the minimal root. MRF first applies the traditional rejection rule ($0 \notin \bar{F}(a, b)$), and if $[a, b]$ is not rejected then applies the Sign Change rejection test ($\bar{F}([a, c]) \cdot \overline{F}([c, b]) \leq 0$).

The second algorithm studied in this work (MRFro) applies the same rejection rules than MRF, but MRFro takes into account that $\bar{F}(a, b) \leq 0$ assures that $0 \in F([a, b])$, so the evaluation of $F([a, b])$ is not necessary. MRFro first evaluates $F([a, a]) \cdot F([b, b])$, and will compute $F([a, b])$ only for those intervals where $\bar{F}([a, a]) \cdot \overline{F}([b, b]) > 0$.

For scenes with objects that are described by differentiable functions, the interval Newton method was applied. This third algorithm give us an insight of how automatic differentiation can increase the performance of the rendering process.

The fourth algorithm tested in this paper (MRFroEx) was designed to work with $p$ functions, simultaneously. The basic idea in MRFroEx is that if for one of the function $f_i$ is found that $F_i([a, a]) \cdot F_i([c, c]) \leq 0$ then none of the functions need to be evaluated at the intervals above $a$.

3. Experimental environment

The environment used to evaluate these interval first root finder algorithms in the field of computer graphics was POV-Ray 3.5 (Persistence of Vision Ray-Tracer is available at povray.org) and the interval arithmetic libraries used to develop the interval algorithms was C-XSC [13]. POV-Ray is modified by changing the POV-Ray intersection routine by the new interval root finder algorithms.

The different types of objects supported by POV-Ray can be classified in two groups: finite objects and infinite objects. The main advantage of the finite object is that POV-Ray can do an automatic bounding box to speed up ray-object intersection tests. This system compartmentalizes all finite objects in a scene into invisible rectangular boxes that are arranged in a tree-like hierarchy. Before testing the objects within the bounding boxes only those objects are tested whose bounds are hit by a ray. This can greatly improve
rendering speed. In the experiments presented here each bounding box will contain only one implicit surface.

On the other hand, automatic bounding box cannot be applied to infinite objects. Therefore, the limits of the object are explicitly specified by the user. The user can establish the scene limits as bounding box for each object belonging to a set of objects.

The finite or infinite aspect of an object plays an important role in the root finder algorithms because it allows us to reduce the search space. When interval root finder algorithms are used and the automatic bounding box can be applied, the bounding box will be the search domain of the algorithms.

POV-Ray is capable of defining different types of objects, for instance sphere, poly, superellipsoid and isosurface. Simple objects, such as sphere or torus can be solved by direct methods while poly and superellipsoid are solved by the Sturm Sequences method [6]. Complicated objects can be defined by the primitives poly and/or isosurface. The poly primitive allows us to define a polynomial of any degree, but a degree greater than seven is not suggested. Any object described by a general function can always be defined by isosurfaces. The root finder methods used by POV-Ray to treat the objects defined as isosurfaces consists of sampling points and subdividing the search space recursively. For this kind of objects interval root finder algorithms can improve considerably the accuracy of the rendered volume.

Table 1 summarized how POV-Ray handle the simple objects used in our experiments. Figures 2-4 shows a rendering of such objects.

<table>
<thead>
<tr>
<th>Implicit surface</th>
<th>Definition</th>
<th>Bounding Box</th>
<th>Intersection Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>sphere</td>
<td>Yes</td>
<td>Direct method</td>
</tr>
<tr>
<td>Mitchell</td>
<td>poly</td>
<td>No</td>
<td>Sturm Sequences</td>
</tr>
<tr>
<td>Tangle</td>
<td>poly</td>
<td>No</td>
<td>Sturm Sequences</td>
</tr>
<tr>
<td>Superellip.</td>
<td>superellip.</td>
<td>Yes</td>
<td>Sturm Sequences</td>
</tr>
<tr>
<td>Twister</td>
<td>isosurface</td>
<td>No</td>
<td>Recursive subdiv.</td>
</tr>
</tbody>
</table>

Table 2. Object function definitions.

Table 1. POV-Ray objects used in the experimentation.

It is clear that when the POV-Ray object is defined by a finite primitive (sphere, poly, . . . etc), and we use interval solvers to find the minimal root, the object function definition has to be supplied to the interval algorithm. In this way, interval methods can use the automatic bounding box of POV-Ray. Table 2 shows the function definitions of the objects used for evaluating MRF and MRFo algorithms.

Moreover, in order to evaluate the performance of the interval root finder algorithm extended to a set of functions (MRFoEx), new scenes composed of several of the previously mentioned objects were built. For complex scenes, composed by some infinite objects (like twister), the use of automatic bounding box to speed up the rendering process cannot be applied.

4. Experimental results

The experiments were carried out in a AMD Athlon 2,4 GHz with two processors, working under Linux OS. All the numerical results were obtained with the maximum precision given by the processor (termination criterion for the B&B algorithms was: $\epsilon \leq 10^{-10}$) and the resolution of the scenes was 64 x 48. Nevertheless, the images appearing in Figures 2-5 have been rendered with a resolution of 1280 x 960.

For scenes where the set of objects allows POV-Ray to apply direct methods or the Sturm sequence method, the computational burden of the rendering process is extremely light compared to interval methods. This happens for simple objects, like sphere, Mitchell’s surface, tangle and superellipsoid. Nevertheless, this objects can also be defined as isosurfaces, simply providing their function definitions. It should also be noticed that in some cases is easier to define objects as isosurfaces than as poly although the computing time for rendering these scenes is greater. For instance, to describe a poly of degree seven it is necessary to provide 120 coefficients.

Table 3 shows the evaluation of rendering several simple scenes (comprised by a single object inside a room) using the recursive subdivision equipotential method as defined by POV-Ray (see appendix A), and MRF, MRFo and Newton (see appendix B) interval based algorithms. The following objects defined as isosurfaces were used in this trial: sphere, Mitchell’s surface, tangle and superellipsoid ($k = 0.75$). The search domain was in all the cases the scene limits ([0, 10]). For each object, the maximum gradient value used in recursive subdivision equipotential method is shown as Grad. Instead of Grad, the number of interval function evaluations for interval methods are shown (MRF, MRFo and Newton). Table 3 also shows the values of the computing time and the number of ray intersections.
Table 3. Rendering results for simple objects. Objects were implemented as isosurfaces.

(Grad: Grad, Grad: Grad, Grad: Grad, Grad: Grad)

One of the drawback of the recursive subdivision equipotential method is that the gradient is not usually known beforehand. In this case, the user has to supply a trial value and POV-Ray will return a possible better one, after rendering. This process must be repeated until POV-Ray finds a correct value. Due to obvious reasons, this trial time was not included in the values represented in Tables 3 to 5. Interval methods do not suffer of this drawback.

It can be seen that for simple scenes, composed by a single and simple object, the recursive subdivision equipotential method outperforms MRF and MRFr algorithms in all the cases but for Mitchell’s surface, where Newton’s method was faster than MRF, MRFr and the recursive subdivision equipotential methods. In Figure 2 rendering using MRFr algorithm, for the objects evaluated in Table 3, is shown. Visually, differences between the images obtained by the recursive subdivision equipotential method and interval algorithms (MRF, MRFr and Newton’s method) were negligible. Finally, Newton’s method can not render a superellipsoid, because it is a non-derivable function.

Notation used in Tables 4 and 5 is the same than that in Table 3. Experimental results shown in Table 4 refers to scenes containing a single non-derivable object. In this case twister was chosen to evaluate the algorithms as an instance of objects than can only be defined as isosurfaces. Three different values of the parameter which defines the twister shape were used (k = 0.75, k = 0.50 and k = 0.25). Figures 3 and 4 show the renderings of scenes containing only one of these objects. For the scene with the twister with k = 0.75, the recursive subdivision equipotential method was faster than MRF and MRFr algorithms. However, for twister with k = 0.50 and k = 0.25 MRFr was faster than MRF and recursive subdivision equipotential method algorithms. It can be noticed that the minimum value of the gradient necessary for rendering a twister using the recursive subdivision equipotential method increases as the value of k decreases and the greater the value of the Grad. constant is the greater the time spends in the rendering of an object; i.e. the slower the rendering process is. For k = 0.50 and k = 0.25, interval methods produce a better rendering than the recursive subdivision equipotential method. In both cases, the original POV-Ray method recommends to increase the value of the maximum gradient to avoid some holes. However, this increases the computational time of the rendering process. Notice the difference in the number of intersections for the recursive subdivision equipotential and intervals methods. This difference can be visually perceived for the twister with k = 0.25 (see Figure 4) but not for k = 0.50.

The MRFr algorithm outperforms MRF in all test scenes. For this reason we compare only MRFr to the other methods in the results that follow.

Table 4. Rendering results for different twister surfaces.

Table 5. Rendering results for objects in a flake.
5. Conclusions and future work

In this paper interval algorithms have been evaluated in the field of ray tracing. It has been shown that interval methods are able to introduce reliability in the rendering process and additionally permit the reduction of the computational time for surfaces that cannot be defined as simple polynomials. It means that for functions which have to be defined as isosurfaces, interval algorithms outperform the methods used by POV-Ray for finding the minimal root. It is worthy to mention that the interval methods can be considerably improved with the use of the information provided by automatic differentiation and the application of the interval Newton method [7, 12]. However, if there is some non-differentiable object in scene, this paper have showed that MRFro and MRFroEx are appropriate in the field of computer graphics.

6. Appendix A - Original POV-Ray method

The equipotential-surface finding in POV-Ray is based on a recursive subdivision method as following:

Recursive subdivision equipotential method

1. Calculates the potential values at the two points \( d_1 \) and \( d_2 \), where \( d_1 < d_2 \): the distance from the initial point of the ray.
2. If there is a possibility (possibility is calculated with a testing function \( T(d_1, d_2, \text{MaxGradient}) \) of existence of the equipotential-surface between \( d_1 \) and \( d_2 \), POV-Ray calculates potential value at another point \( d_3 \) on the ray between the two point \( d_1 \) and \( d_2 \).

\( T(d_1, d_2, \text{MaxGradient}) \)

(a) If \( f(d_1) + f(d_2) - \text{MaxGradient} \cdot |D| \cdot (d_2 - d_1) < 0 \), there is a possibility. Where \( D \) is direction vector of ray.
(b) If \( f(d_1) + f(d_2) - \text{MaxGradient} \cdot |D| \cdot (d_2 - d_1) \geq 0 \), there isn’t a possibility.
3. If there is a possibility between \( d_1 \) and \( d_3 \), POV-Ray calculates another point \( d_4 \) between \( d_1 \) and \( d_3 \).

4. If there is no possibility between \( d_1 \) and \( d_3 \), POV-Ray looks for another point \( d_4 \) between \( d_3 \) and \( d_2 \).
5. These calculation (1-4) will be done recursively until \( (d_n - d_{n+1}) < \text{Accuracy} \).

7. Appendix B - Newton’s interval algorithm

This algorithm is a recursive function for the execution of the extended interval Newton method for the function \( f \).

\( \text{XINewton}(f, [a, b], \epsilon, [\text{Root}]) \)

1. If \( 0 \notin F([a, b]) \) then return;
2. \( c = m([a, b]) \);
3. \( z = c - f(c)/f'(c, b); \)
4. \( V = [a, b] \cap \{z\} \);
5. If \( V \cap \{a, b\} \) then \( V \cap \{a, c\} \land V \cap \{c, b\} \);
6. for \( i = 1 \) to \( 2 \) do
   (a) If \( V \cap \{a, c\} \) then next;
   (b) If \( d_{rel}(V) \leq \epsilon \) then
      i. If \( 0 \in f(V) \) then \( \text{Root} = V \cap \{a, c\} \land V \cap \{c, b\} \);
   (c) Else \( \text{XINewton}(f, V, \epsilon, [\text{Root}]) \);
7. return \( \text{Root} \);

References

Figure 2. From up-left to down-right, Sphere, Mitchell surface, tangle and superellipsoid objects are shown, obtained using MRFro algorithm.


Figure 3. From left to right, twister $k = 0.75$ and twister $k = 0.50$ objects obtained using MRFro algorithm.

Figure 4. Twister $k = 0.25$ using recursive subdivision equipotential method, on the left graph, and MRFro on the right one.

Figure 5. Flake with 10 (left-hand) and 91 (right-hand) objects using MRFroEx.