# Improving the Interval Ray Tracing of Implicit Surfaces

Jorge Flórez, Mateu Sbert, Miguel A. Sainz, and Josep Vehí

Institut d'Informática i Aplicacions, Universitat de Girona, Campus Montilivi
17071 Girona, Spain
{jeflorez, vehi}@eia.udg.es,
{mateu, sainz}@ima.udg.es

**Abstract.** This paper presents a fast and reliable method to trim non-solution regions in an interval ray tracing process. The "trimming algorithm" uses interval analysis to perform rejection tests in a set of pixels simultaneously, instead of individual pixels at each time. With this approach, the presented algorithm runs faster than the traditional interval ray tracing algorithm. Also, an interval algorithm to remove aliasing in the rendering of implicit surfaces is introduced. This algorithm obtains better visualizations than the traditional point sampling. This algorithm can render thin features that would be impossible to obtain with point sampling algorithms.

## 1 Introduction

Interval Arithmetic is a mathematical theory developed by Ramon Moore [1] that has been used to solve problems of reliability caused by the floating-point arithmetic of computers. Floating-point calculation causes problems of numerical imprecision in geometric modeling and computer graphics [2, 3]. A particular application in which there are problems of reliability is ray tracing of implicit surfaces. These problems arise in the rendering of very special implicit functions with thin features that could be missed in the point sampling process. This paper proposes two improvements for the interval ray tracing algorithm. First, interval analysis is used to evaluate screen regions to perform rejection test over many pixels simultaneously. This implies a reduction of the number of intersection test performed in a traditional ray tracing algorithm. Secondly, interval analysis can also be used as an alternative for point sampling inside a pixel. A ray is infinitely thin, and a pixel covers a finite area. When rays are cast through a pixel, there is the possibility that some rays miss parts of the surface inside the pixel. With the approach presented in this paper, it is possible to evaluate all the area of the surface covered by the pixel instead of considering hits. This principle is used to implement an antialiasing algorithm that improves the traditional interval point sampling.

### 1.1 Interval Ray Tracing

Ray tracing is a process in which rays starting at a point (the camera or eye point) are sent through every pixel of a screen. These rays can intersect objects behind the screen. In that case, the first intersection point is recorded. In the intersection point, the normal of the surface is calculated and used to determine a shade value for the pixel. The ray is represented in a parametric way as:

H.-P. Seidel, T. Nishita, and Q. Peng (Eds.): CGI 2006, LNCS 4035, pp. 655 – 664, 2006.

$$\boldsymbol{p}(t) = c + t(s - c), \quad t \geq 0 . \tag{1}$$

The point c is the camera or eye position. The magnitude s – c indicates the direction of the ray and the parameter t represents a fractional distance from c in the direction of s - c.

An intersection test must be performed between the rays and the implicit surfaces. Given an implicit function defined by:

$$f(x,y,z) = 0 . \tag{2}$$

or in vector form:

$$f(p) = 0 \quad where \quad p = (x,y,z) . \tag{3}$$

The intersection of the ray with the implicit surface is defined as:

$$f(\boldsymbol{p}(t)) = 0, \text{ or, } f(c + t(s - c)) = 0 . \tag{4}$$

To develop a reliable intersection test using interval analysis, the ray parameter is replaced with an interval T that represents a set of values of t. The equation used to find the intersections looks in an interval version as:

$$F(T) = F(c + T(s - c)) . \tag{5}$$

F(T) is known as the inclusion function of the intersection between the implicit surface and the ray. The result of the evaluation of the inclusion function is an interval. If this interval contains zero, a root could exist in the equation for the values of T. In the other case, the interval T can be rejected. This means that F(T) gives a reliable tool to perform rejection tests.

### 1.2 Previous Work

The requirement of a guaranteed ray tracing process to prevent that the intersection test miss thin features of the surface, has been pointed out by authors like Kalra [4], Capriani [5] and Mitchell [6]. They argue that point sampling is not a feasible algorithm to perform intersection test with surfaces that have thin features.

A few authors have proposed interval arithmetic as a solution to this problem, using different strategies to create reliable intersection test based on interval analysis [5, 6, 7, 8].

Mitchell [6] was the first author to propose an interval algorithm for the ray intersection test. He proposed two steps: root isolation and root refinement. Mitchell does not use interval arithmetic in the second step because he considered that an interval approach for root refinement was inefficient. Capriani et al [5] demonstrated that interval arithmetic could be used in both steps of Mitchell algorithm without loss of efficiency. They also propose other algorithms as the Newton interval method or Alefeld-Hansen method in the ray tracing process.

Sanjuan-Estrada [7] used a branch-and-bound strategy to make the intersection test without root isolation. They applied a rejection test using an interval inclusion function based on interval arithmetic. De Cusatis [8] suggest the use of affine arithmetic instead of interval arithmetic to solve the intersection test.

The algorithms previously mentioned consider only one interval variable (the ray parameter T). Those algorithms work sending one or more rays through every pixel in the screen.

Kalra and Barr [4] proposed a method for the intersection test based on Lipschitz constants. This method is used to prevent that the rays sampled in a pixel miss thin features of the surfaces.

## 2 Elimination of Screen Space Non-solution Regions

In this section, an algorithm for a fast trimming of screen regions that do not contain intersections with the implicit surface is presented. That is, the algorithm has to identify regions with pixels that should be shaded with the background color.

The rays are traced pixel by pixel, which makes ray tracing a slow algorithm. Interval arithmetic provides a way to evaluate many pixels simultaneously to accelerate the ray intersection process. Instead of a point in the screen, it is possible to take intervals for $x$ and $y$ coordinates to cover a set of pixels. Because the origin is still a point, the figure obtained looks like a pyramid instead of a ray (see figure 1). This process is similar to the beam tracing process introduced by Heckbert and Hanrahan [9] for polygonal objects.

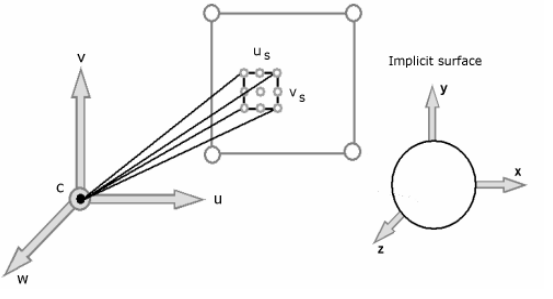


**Fig. 1.** A pyramid could be traced instead of a single ray to cover many pixels

The screen regions not trimmed must be ray traced pixel by pixel, because it is necessary to know the color for every single pixel. The objective of the trimming algorithm is to save time, that is, the algorithm presented is faster than an algorithm in which one or more rays are sent for all the pixels in the screen.

### 2.1 Mathematical Preliminaries

Equation (5) shows the inclusion function used to determine if the interval T contains roots. Figure 1 shows the new approach, in which $s$ is a box instead of a point. The camera position can be in any place in the scene, for that reason, the rays are given in an arbitrary coordinate system $uvw$. The transformation of $uvw$ coordinates to $xyz$ coordinates must be an interval arithmetic operation.

Let $U_s$ and $V_s$ be intervals representing a set of pixels of the screen in $uvw$ coordinates. Let $w_s$ be the distance from the screen to the origin point $c$. The result of the transformation from $uvw$ coordinates to $xyz$ coordinates is represented as

$$S = c + U_s \mathbf{u} + V_s \mathbf{v} + W_s \mathbf{v} . \quad (6)$$

where S is an interval. A more detailed explanation of the transformation process (without intervals) is given by Shirley [10].

The inclusion function in our case is presented in the following equation:

$$F(\mathrm{T},\mathrm{S}) = F(c + \mathrm{T}(\mathrm{S} - c)) \,. \tag{7}$$

If this condition is not accomplished, the evaluated interval S is rejected. This is used to reject a set of pixels that do not have intersections with the implicit function. This case occurs when $0 \notin F(\mathrm{T},\mathrm{S})$, which means there are no roots for the current values of T and S.

### 2.2 Algorithm Specification

With the equation (7), a set of pixels can be evaluated simultaneously with a unique intersection test, to determine if they intersect the implicit function. The algorithm explained in this section can reject regions without intersections in screen space. In other words, the algorithm offers a fast trimming of non-solution zones. The complete algorithm is presented in figure 2.

```
box.size = screen.size
add box to List_Box
for every box in List_Box
   if width(box)< miminum_box_size
      add box to Final_List
      exit for
   endif
   S = transform (box)
   T = (0, ∞)
   add T to List_T
   for every T in List_T
      if  width(T) < ∈T exit for
      if (0 ∈ F(T,S))
         bisect T into T1,T2
         add T1,T2 to List_T
      endif
      drop T from List_T
   endfor
   if empty List_T
      bisect box into box1, box2
      add box1, box2 to List_Box
   endif
   drop box from List_box
endfor
for every box in Final_List
  for every pixel in box
    ray tracing pixel and shading
  endfor
endfor
```

**Fig. 2.** Algorithm for trimming non-solution boxes in screen space

The algorithm is based in a branch-and-bound strategy. The bisections are performed in both the screen space and in the interval parameter T. The algorithm starts making subdivisions of screen space in $U_s$ and $V_s$ coordinates to generate boxes. For every box, a new branch-and-bound algorithm is started in the interval parameter T. Every section of T is evaluated using the inclusion function (7) and the current value of S. The sections for which $0 \notin F(T,S)$ are rejected. Otherwise, the current interval T is subdivided and the evaluation performed in the new intervals.

The criterion used to stop the subdivision process over the parameter T is:

$$(T.Upper\_bound - T.Lower\_bound) < \in_T . \quad (8)$$

where $\in_T$ is the precision selected to stop the process.

Using a precision of $10^{-6}$, the obtained results are feasible for the example surfaces (section 2.3).

The bisection over the boxes is terminated when a minimum size for the box is achieved. This precision represents the minimum accepted size of the box relative to the size of the screen, that is:

$$(screen\ size) * \in_{box} = minimum\ box\ size . \quad (9)$$

The boxes that achieve this precision are stored to be further ray traced. That is, the ray tracing will be performed only over the non- rejected boxes that achieve the minimum size box. The ray tracing algorithm used in this paper is called MRF, and it was introduced by Sanjuan-Estrada et al [4].

The algorithm presented uses a precision of 0.05 over the size of the screen in pixels. This precision has proved to be enough to obtain efficient results for the tested surfaces (section 2.3).

### 2.3 Experimental Results

The algorithm was tested using four surfaces (see table1 and Figure 5). The surfaces were rendered using an Intel Pentium 4, 2.4 GHz. The resolution used to render the images was 300 x 300 pixels. In the ray tracing process one ray is cast for every pixel and the precision used in the intersection test is $10^{-6}$.

**Table 1.** Tested implicit surfaces

| Surface | Equation |
|---|---|
| Sphere | $x^2 + y^2 + z^2 - 4 = 0$ |
| Blobby | $x^2 + y^2 + z^2 + \sin(4*x) + \sin(4*y) + \sin(4*z) - 1 = 0$ |
| Steiner | $(x^2*y^2 + y^2*z^2 + z^2*x^2)^2 + x*y*z = 0$ |
| Mitchell | $4(x^4 + (y^2 + z^2)^2) + 17x^2(y^2 + z^2) - 20(x^2 + y^2 + z^2) + 17 = 0$ |

The results of the test are presented in table 2. Trimming time column represents the time of the trimming process to reject boxes in which there are no rays intersecting the implicit surface. Next column gives the time of a pixel-by-pixel ray tracing over the non-rejected boxes. The sum of both times is represented in total time column. Finally, the column “only ray tracing” shows the time of a traditional interval ray tracing over all the pixels in the screen for every surface.

**Table 2.** Results for the trimming algorithm (time in seconds)

| Surface | Using trimming strategy | | | Only ray tracing | % Time saved |
|---|---|---|---|---|---|
| | Trimming time | Ray tracing | Total Time | | |
| Sphere | 0.791 | 40.678 | 41.469 | 53.226 | 22.08% |
| Blobby | 1.33 | 62.842 | 64.172 | 107.59 | 40.35% |
| Steiner | 1.59 | 92.84 | 94.43 | 152.63 | 38.13% |
| Mitchell | 3.6 | 378.02 | 381.62 | 544.953 | 29.97% |

The precision used in the pixel-by-pixel ray tracing process was $\in_T = 10^{-6}$ because the results obtained are feasible with that precision. Figure 3 shows the blobby surface visualized using different precisions.

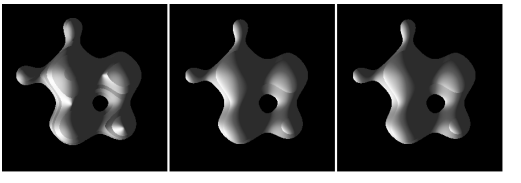

**Fig. 3.** Results for the Blobby surface for different precisions. In the first (*left image*) $\in_T = 10^{-2}$, in the next (*center image*) $\in_T = 10^{-6}$, finally (*right image*) $\in_T = 10^{-10}$. With a precision of $10^{-6}$, the result obtained looks the same as using a higher precision. For a precision of $10^{-2}$ (figure 6a) the result is not acceptable.

In table 2, the total time of the algorithm using a trimming strategy summarizes the time of the interval ray tracing in the non-rejected boxes and the trimming process itself. In all the tested cases, the algorithm using a trimming strategy takes less time than an interval strategy based on a ray casting for all the pixels of the screen.

The intersection test takes more time in pixels corresponding to non-rejected boxes than rejected boxes. This is because the intersection test of rays that intersect the implicit surface must reach a smaller precision to obtain the value for the intersection. Rays that do not intersect the implicit function are rejected before this small precision is reached. Figure 4 shows a color map of the time that the intersection test takes in

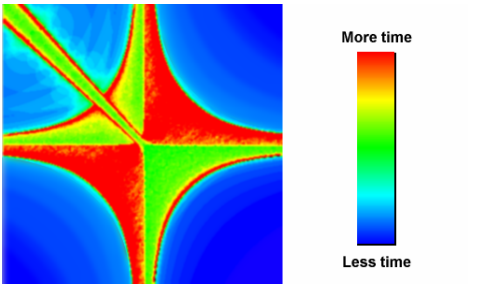


**Fig. 4.** Color map to represent the time that intersection test takes in different regions of the Steiner surface

every pixel in the Steiner surface. The pixels in which the intersection test fails take less time than pixels inside the implicit surface. Also note that in all the borders of the surface, the algorithm takes the maximum time.

Summarizing, the trimming algorithm is faster than the traditional interval ray tracing algorithm in the areas without intersections (the blue zones in figure 4). Table 3 compares the time spent by the trimming algorithm to reject non-solution boxes and the time that a pixel-by-pixel algorithm spends (one ray per pixel) in the same non-solutions boxes. Figure 9 shows the final results of the trimming and ray tracing algorithm for the tested surfaces.

**Table 3.** Comparison of the trimming algorithm and the classical interval ray tracing algorithm over the rejected boxes (time in seconds)

| Surface | Trimming time | Pixel by pixel Time | % Time Saved |
|---|---|---|---|
| Sphere | 0.791 | 12.548 | 93.69 |
| Blobby | 1.33 | 44.748 | 97.027 |
| Steiner | 1.59 | 59.79 | 97.34 |
| Mitchell | 3.6 | 166.933 | 97.84 |

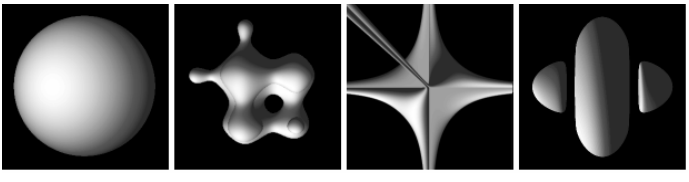

**Fig. 5.** Final result of the algorithm for the tested surfaces. From left to right and top to bottom, Sphere, Blobby, Steiner, Mitchell surfaces.

## 3 Antialiasing Using Interval Analysis

As was mentioned in section 2, some parts of the surfaces could be missed when methods based on point sampling are used. Figure 6 illustrates this situation.

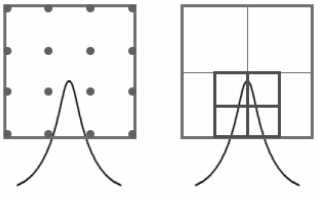

**Fig. 6.** Point sampling (*left*) and Sampled pixel using intervals (*right*). The evaluation in the sample points does not indicate the presence of the surface inside the pixel. No matter how many regularly distributed rays are sent through the pixel, there will be zones without rays traced and some rays could miss the surface. Using intervals, the pixel is subsampled and every region is considered in the evaluation of the function. There is no part of the surface missed in the evaluation.

### 3.1 Evaluation of the Pixel Area

Interval analysis can be used to develop an algorithm to "detect" if an implicit surface crosses any part of the pixel.

```
function detect_surface(area)
   S = transform(area)
   T = (0,∞)
   add T to List_T
   for every T in List_T
      if  width(T) < ∈T exit for
      if (0 ∈ F(T,S))
         bisect T into T1,T2
         add T1,T2 to List_T
      endif
      drop T from List_T
   endfor
   if empty List_T
      return false
   else
      return T
   endif
endfunction
```

**Fig. 7.** Algorithm to detect when a surface crosses a region of the pixel

The technique used is this algorithm is similar to the presented in section 2. The process is based in a branch and bound strategy, in which the pixel area is subdivided and every new area is further evaluated to find out whether it contains part of the surface. In this case, the same inclusion function (7) is used to evaluate a region of the pixel.

When only a ray is considered, the "detection algorithm" will return an intersection value. Using point sampling, if the ray misses the surface, there is no way to know if the surface crosses the pixel.

This algorithm is useful to visualize special cases, for example, the intersection of implicit surfaces. When two implicit surfaces intersect, the result is a curved line without thickness. This line cannot be visualized using point sampling because the rays will always miss the line. Using the presented algorithm, only an intersection test over the pixel is needed (see figure 8).

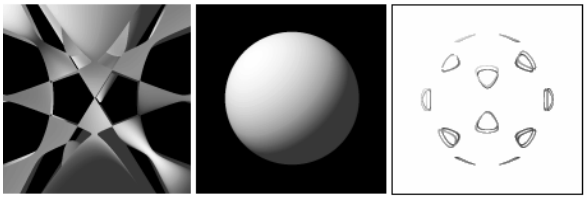

**Fig. 8.** Intersection between implicit surfaces

### 3.2 Antialiasing Strategy

In this section, the detection algorithm is used to "inform" when a pixel must be shaded, that is, if the pixel contains any part of the surface.

If the detection algorithm finds a part of the surface inside a pixel, a branch-and-bound process is started over the pixel to generate subpixels. Every subpixel region is evaluated to determine if it contains part of the surface. The subdivision process continues until a subpixel subdivision level is achieved. In that case, the normal at the surface is calculated and used to determine the shading color. If the evaluated region does not contain any part of the surface, the background color is assigned to the region. At the end, the shading value for every pixel is calculated according to the color and area of every subpixel.

### 3.3 Experimental Results

The described antialiasing algorithm was used to generate a visualization of the Steiner surface. This visualization is compared with the traditional interval ray tracing. The resulting images are shown in figure 9. In the sampling algorithm, 16 rays are sent for every pixel. In the interval antialiasing, the algorithm is stopped when the subdivision takes a size equivalent to 1/16 of the pixel area. The time spent by the interval antialiasing is 765.18 seconds, and the sampling algorithm takes 1504.31 seconds. The time difference is because the interval antialiasing algorithm verifies if a part of the surface is contained in every pixel before the algorithm starts to make subdivisions. In the sampling algorithm, 16 rays are sending for every pixel in the screen.

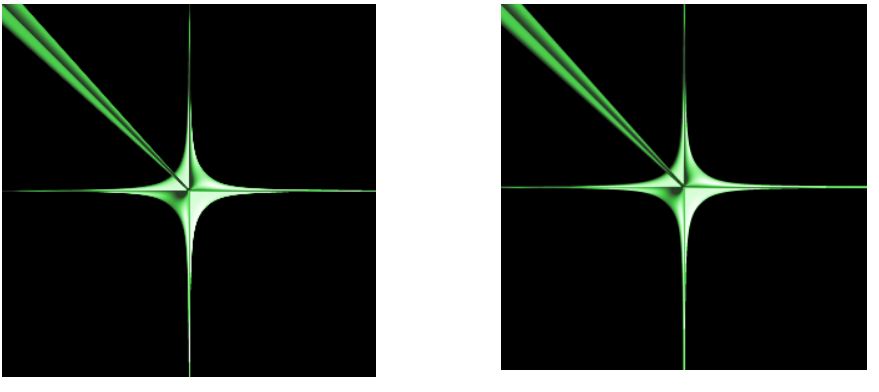

**Fig. 8.** Comparison of a point sampling algorithm and the interval antialiasing algorithm introduced in this paper. Using only point sampling (left) and using the interval antialiasing algorithm (right).

## 4 Conclusions

In this paper we have introduced several improvements to the ray tracing of implicit surfaces, using interval analysis. The first presented algorithm offers an important improvement in efficiency. Using intervals to perform intersection tests over regions

of the screen is faster than an evaluation of individual pixels. The second algorithm improves the quality of the visualization of the surfaces. This is obtained using an interval evaluation of all the area of the pixel instead of sampled points. It is not possible to obtain that kind of improvement using algorithms based on the floating-point arithmetic of the computer.

As future work, we plan to add reflections and refractions to the presented algorithms. The idea is to create a more complete shader, applicable to visualize more realistic images.

## Acknowledgements

This work has been partially funded by the European Union (European Regional Development Fund) and the Spanish Government (Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica, Ministerio de Ciencia y Tecnología) through the co-ordinated research projects DPI2002-04018-C02-02, DPI2003-07146-C02-02, DPI2004-07167-C02-02 and TIN2004-07451-C03-01 and by the government of Catalonia through SGR00296.

## References

1. Ramon Moore. Interval analysis. Prentice-Hall, Englewood, 1966
2. Amitabh Agrawal and Aristides A. G. Requicha. A paradigm for the robust design of algorithms for geometric models. Eurographics'94, Computer Graphics Forum, Vol.13, No.3, pp.33-44, 1994
3. Helmut Ratschek and Jon Rokne. Geometric computations with interval and new robust methods. Horwood Publishing, 2003
4. D. Kalra and A. Barr. Guaranteed ray intersection with implicit surfaces. In Computer Graphics (Siggraph proceedings), volume 23, pages 297-- 306, 1989
5. O. Capriani, L. Hvidegaard, M. Mortensen and T. Schneider. Robust and efficient ray intersection of implicit surfaces. Reliable Computing, volume 6, p. 9 – 21, 2000
6. Don Mitchell. Robust ray intersection with interval analysis. Proceedings on graphics interface, pages 68-74. 1990
7. J.F. Sanjuan-Estrada, L.G. Casado and I. García. Reliable algorithms for ray intersection in computer graphics based on interval arithmetic. XVI Brazilian symposium on computer graphics and Image processing, pages 35-44, 2003
8. A. de Cusatis, L. de Figueiredo and M. Gatas. Interval methods for ray casting implicit surfaces with affine arithmetic. In Proceedings of SIBGRAPI'99, pages 65--71. IEEE Press, October 1999
9. Heckbert and Hanrahan, Beam Tracing Polygonal Objects. Proceedings of the 11th annual conference on computer graphics and interactive techniques, pages 119-127, 1984
10. Fundamentals of computer graphics. Peter Shirley. A K Peters Ltd., 2002