Fast Reliable Interrogation of Procedurally Defined Implicit Surfaces Using Extended Revised Affine Arithmetic

Oleg Fryazinov, Alexander Pasko, Peter Comninos

The National Centre for Computer Animation, Bournemouth University, UK

Abstract

Techniques based on Interval and Affine Arithmetic and their modifications are shown to provide reliable function range evaluation for the purposes of surface interrogation. In this paper we present a technique for the reliable interrogation of implicit surfaces using a modification of Affine Arithmetic called Revised Affine Arithmetic. We extend the range of functions presented in Revised Affine Arithmetic by introducing affine operations for arbitrary functions such as set-theoretic operations with R-functions, blending and conditional operators. The obtained affine forms of arbitrary functions provide faster and tighter function range evaluation. Several case studies for operations using affine forms are presented. The proposed techniques for surface interrogation are tested using ray-surface intersection for ray-tracing and spatial cell enumeration for polygonization. These applications with our extensions provide fast and reliable rendering of a wide range of arbitrary procedurally defined implicit surfaces (including polynomial surfaces, constructive solids, pseudo-random objects, procedurally defined microstructures, and others). We compare the function range evaluation technique based on Extended Revised Affine Arithmetic with other reliable techniques based on Interval and Affine Arithmetic to show that our technique provides the fastest and tightest function range evaluation for fast and reliable interrogation of procedurally defined implicit surfaces.

Keywords: Ray Tracing, Spatial Enumeration, Implicit Surfaces, Function Representation, Revised Affine Arithmetic, Affine Forms

1. Introduction

In recent years, implicit surfaces (isosurfaces of trivariate real functions) have proved to be a powerful and simple solution to some complex problems in the area of modelling and animation. For example, implicit surfaces provide solutions for surface reconstruction from scattered points and for fluid simulation. Several operations, such as sweeping, metamorphosis and offsetting can be implemented quite easily with implicit models - unlike traditional boundary-representation models. However, modelling with the whole range of implicit surfaces is still a complex task because interactive rendering of arbitrary implicit surfaces is still an open problem, which has recently become of growing interest to researchers in the field [1][2][3]. Currently, there are two ways to render an implicit model: by the generation of a polygonal mesh and by direct rendering using ray-tracing or beam-tracing. Polygonization is a widely used technique, but in many cases, when the model has sharp or thin features, large numbers of small-sized disjoint elements or internal microstructures, the generation of an appropriate polygonal mesh takes a long time and requires a large amount of memory. Direct rendering of implicit surfaces using ray-casting and ray-tracing is a more promising technique for high-quality rendering and for rendering animated implicit surfaces. Additionally, instead of ray-tracing beam-tracing [4] can be used for rendering implicit surfaces with features smaller than a pixel in size. However, due to the large number of ray-surface intersection calls the main disadvantage of direct rendering is its slow speed. With recent developments in hardware the problem of speed becomes less critical, but not all together insignificant. Thus, techniques for speeding up the rendering process remain of great interest.

The other key point in high-quality rendering of implicit surfaces is reliability, i.e.: it is essential that in the case of ray tracing, no roots are missed in the ray-implicit-surface intersections and in the case of polygonization, no surface features are missed. Many techniques for rendering implicit surfaces have been developed. However, the majority of these techniques have disadvantages, either because they work with a small range of implicit surfaces (for instance those defined only by polynomials) or because they are unreliable. For example, classical approximate techniques, such as ray marching [5] for ray-tracing and marching cubes [6] for polygonization, are easy to implement and fast, but can miss sharp features and small components of the surfaces. Techniques based on interval analysis and other reliable numerical computations have recently been applied to the ray-tracing and polygonization of implicit surfaces. However, methods based on classical Interval Arithmetic are slow because of the interval overestimation.

The problem considered in this paper is that of finding a technique for the reliable interrogation of general implicit surfaces in ray-surface intersection in ray-tracing and for spatial cell enumeration in the polygonization process. This technique should have the following properties:

1. Its procedure should be reliable with no roots missed in

ray-tracing and no volumes including a surface missed in spatial enumeration.
2. A wide range of implicit models should be supported – meaning that the algorithm should be able to work with procedurally defined as well as with algebraic models.
3. The procedure should be fast and suitable for a GPU implementation for interactive rendering.

In this paper we propose to use Revised Affine Arithmetic (RevAA) as a fast and reliable technique for calculating the range of a function for a given interval and hence for the core of the interrogation procedure. Also we extend Affine Arithmetic by affine forms related to the procedural definition of the model in order to decrease the number of computations while evaluating the interval for the function and hence to decrease the rendering time. The main contributions of this paper are as follows.

1. The widening of the scope of reliable ray-tracing and spatial enumeration algorithms for surfaces ranging from algebraic surfaces (defined by polynomials) to general implicit surfaces (defined by function evaluation procedures involving both affine and non-affine operations based on Revised Affine Arithmetic).
2. The introduction of a technique for representing procedural models using special affine forms (illustrated by case studies of affine forms for set-theoretic operations in the form of R-functions, blending operations and conditional operations).
3. The detailed derivation of special affine forms for arbitrary operators.

2. Related Work

The rendering of implicit surfaces is a well-researched area. A good sample of such techniques is presented in [7] and more recently in [8]. Most of these techniques however are approximate and can miss small surface features, but on the other hand, they are suitable for all types of implicit surfaces. Additionally, several techniques were introduced for particular types of implicit surfaces producing improvements not only in speed but also in reliability. Thus, in [9] a distance property is needed for the ray-tracing procedure, while in [10] blobs, metaballs and convolution surfaces are the only types of implicit surfaces that can be rendered fast.

Other ways of increasing the speed of ray-tracing include the use of specialised hardware, for example graphical processors (GPUs). Ray-tracing of general implicit surfaces on the GPU, using approximate methods, was performed in [11] and in [1]. In [2] the GPU was used for fast polygonization of implicit surfaces.

Reliable computational techniques based on Interval Arithmetic have been known for a long time. However, most of the literature relates to fields such as global optimisation rather than computer graphics. Applications of Interval Arithmetic in computer graphics were discussed in [12] and [13], and Affine Arithmetic was used for ray-tracing of implicit surfaces in [14]. A good comparison of different interval techniques can be found in [15], however in this paper the type of the implicit models examined is limited to those given in the polynomial form. Some of the interval techniques can be applied for general procedurally defined implicit models, such as Interval Arithmetic in the Centred form [16], however, in this case the general form should be used, such as the mean-value form that is less optimal than the one used for polynomial implicit models. In [17] Reduced Affine Arithmetic was introduced for the purposes of rendering stochastic implicit models. However, the applicability of Reduced Affine Arithmetic to general implicit models cannot currently be verified, as the authors of the aforementioned paper did not provide sufficient information on operations other than those of multiplication and affine operations. Interval Arithmetic and Reduced Affine Arithmetic are used for fast rendering of implicit surfaces on the GPU in [3]. A more detailed comparison of these techniques with the one proposed here can be found in the "Results" section of this paper. Interval Arithmetic and Affine Arithmetic were also used for polygonization purposes. Thus, in [18] Interval Arithmetic is used for implicit surface meshing, and in [19] spatial enumeration using Affine Arithmetic is discussed. Also, in [20] a technique derived from Affine Arithmetic was used for faster spatial enumeration rather than classical Affine Arithmetic.

In this paper, we use Revised Affine Arithmetic (RevAA) [21], which was introduced recently for the purposes of constraint propagation.

3. Background

3.1. Procedurally defined implicit surfaces

A zero level set or an isosurface of a trivariate real function f of a point with coordinates (x, y, z) is traditionally called an implicit surface and is defined as $f(x, y, z) = 0$. An isosurface can also be considered to be the boundary of a solid (three-dimensional manifold) defined by the inequality $f(x, y, z) \geq 0$. There are many different ways to specify the function $f(x, y, z)$. The simplest form is that of an algebraic implicit surface defined by a polynomial function. Most of the extant work on reliable ray-tracing concentrates solely on algebraic surfaces. More complex forms involve exponential, square root, trigonometric and other non-linear functions. Here we deal with the most general form of procedurally defined implicit surfaces, where the function f is evaluated by some procedure involving all kinds of non-linear functions as well as loops and conditional operations. This allows us to cover skeleton-based implicit surfaces [7], Constructive Solid Geometry (CSG) objects defined by nested R-functions [22], solid noise [23][24] and other complex objects.

3.2. Affine Arithmetic

Affine Arithmetic is a technique for performing computations on uncertain numerical values. The main idea of Affine Arithmetic is the calculation of an uncertain value (function) based on other uncertain values (arguments). Initially this model was introduced for self-validated numerical computations as

an alternative to Interval Analysis – in some literature Affine Arithmetic is still considered as a modification of general Interval Arithmetic – and currently it is used in many different areas of computer science [25]. By keeping track of the errors for each computed quantity, Affine Arithmetic provides generally much tighter bounds for computed quantities compared to classical Interval Arithmetic, especially for complicated expressions. Uncertain values in Affine Arithmetic are represented by affine forms, i.e. polynomials of the form:

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

where x_i are known real coefficients and ε_i are noise symbols, i.e. symbolic variables whose values are assumed to lie in the interval $\varepsilon_i \in [-1, 1]$.

In Affine Arithmetic, formula evaluation is performed by replacing operations on real quantities by their affine forms. Similarly to Interval Arithmetic, the inclusion property is applied to Affine Arithmetic, i.e. for any operation $\otimes$,

$$A \otimes B \supset \{a \otimes b, a \in A, b \in B\}$$

where a and b are real values and A and B are uncertain values in affine form.

All operations on affine forms can be divided into affine (exact) and non-affine (approximate) operations. An affine operation is a function that can be represented by the linear combination of the noise symbols of its arguments. Non-affine operations can not be performed over the linear combination of the noise symbols. In this case an approximate affine function is used and a new noise symbol is added to the affine form to represent the difference between the non-affine function and its approximation. Additional details regarding the construction of both affine and non-affine operations can be found in the literature related to Affine Arithmetic [25] [26].

3.3. Approximation techniques for affine forms

One of the useful properties of Affine Arithmetic is that any non-affine operation can be represented in the same way. In the general case any operation can be represented in an affine form:

$$\hat{x} \otimes \hat{y} = \alpha\hat{x} + \beta\hat{y} + \zeta \pm \delta$$

where the value of the new noise symbol is represented by δ. Here and in subsequent formulations the symbols α, β and ζ represent coefficients used in the literature on Affine Arithmetic to describe the general affine form.

In [25], different approximation techniques are discussed for the affine forms of several functions: Optimal (Chebyshev), Min-range and Interval approximation. All these techniques assume that the function is at least bounded on the argument interval.

3.3.1. Optimal approximation

Optimal approximation of the functions in Affine Arithmetic is based on the Chebyshev approximation theory. Given that a function f is bounded and twice differentiable on some non-empty interval $I = [a, b]$ and given that its second derivative f'' does not change sign inside I, the optimal approximation can be obtained as follows:

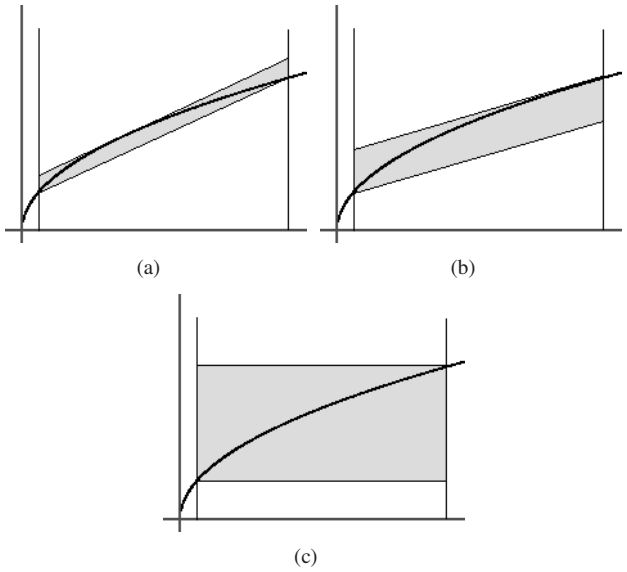


Figure 1: Approximation techniques for affine forms: a) Optimal (Chebyshev) b) Min-range c) Interval

- $\alpha = \frac{f(b)-f(a)}{b-a}$
- ζ is obtained from the equation $\alpha u + \zeta = \frac{f(u)+r(u)}{2}$, where u is a point inside I where $f'(u) = \alpha$ and $r(x)$ is the line passing through the points $(a, f(a))$ and $(b, f(b))$
- $\delta = \frac{|f(u)-r(u)|}{2}$

Thus, the optimal approximation can be found for any function provided that the equation $f'(u) = \alpha$ can be solved for an arbitrary α in I. From the geometric point of view the optimal approximation is a parallelogram with two vertical sides that encloses the graph of the function f in the interval I and has the minimal possible vertical extent (see Fig. 1a).

Unlike unary functions, optimal approximation can not be easily obtained for functions with more than one arguments because of the problems of fitting a plane to boundary points and of searching for interior points using partial derivatives.

3.3.2. Min-range approximation

The affine approximation can be constructed if we choose the coefficients α and ζ in such a way that the joint range P of the forms $\hat{x}$ and $\hat{z} = \alpha\hat{x} + \zeta \pm \delta$ has the same vertical extent as the piece of the graph of $f = f(x)$ on the interval $[a, b]$ (see Fig. 1b). A possible way to obtain such an approximation is the following:

- α is the tangent to the graph in p, where $p = a$ if $f''(x) \geq 0$ on $[a, b]$ or $p = b$ otherwise.
- ζ is obtained from the equation $\alpha u + \zeta = \frac{f(u)+r(u)}{2}$, where $u = b$ in case of $p = a$ or $u = a$ otherwise and $r(x) = \alpha(x - p) + f(p)$

- $\delta = \frac{|f(u)-r(u)|}{2}$.

This approximation has a wider vertical extent than the optimal approximation, however the calculation of this approximation is easier.

3.3.3. Interval approximation

For the case where we can obtain the interval (f_{min}, f_{max}) of function f, the affine form can be constructed by using this interval as follows:

- $\alpha = 0$
- $\zeta = \frac{f_{min}+f_{max}}{2}$
- $\delta = \frac{f_{max}-f_{min}}{2}$

Geometrically speaking, the interval approximation is the rectangle $[a, b] \times [f_{min}, f_{max}]$.

Compared with other affine approximations, the error δ is maximal and therefore the overestimation for the interval approximation is wider (see Fig. 1c). However, in many cases the interval approximation is computationally less expensive. Thus in algorithms where speed is as important as the quality of the approximation, the interval approximation technique can be used.

3.4. Revised Affine Arithmetic

Generally pure AA is both computationally and memory expensive, especially for complex expressions. Thus for algorithms where the reduction of computational complexity is equally important as the quality of the computational result, it is better to use optimised forms of AA. In [27], several reduced affine forms were introduced to reduce the number of computations in Affine Arithmetic by accumulating errors. The Affine Form 1 (AF1) is the simplest one and represents the uncertain quantity as:

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i\varepsilon_i + x_{n+1}\varepsilon_{n+1}$$

The noise symbols $\varepsilon_1, ..., \varepsilon_n$ represent the errors of the initial arguments. The last noise symbol represents all the errors after the non-affine operations.

Revised Affine Arithmetic is an extension of AF1 and was introduced by Vu et al. [21] for the purposes of numerical constraint propagation. As for standard AA, RevAA has an inclusion property. The revised affine form is similar to AF1:

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i\varepsilon_i + e_x[-1, 1], e_x \geq 0$$

The general binary affine operation is defined as:

$f(\hat{x}, \hat{y}) = (\alpha x_0+\beta y_0+\zeta)+\sum_{i=1}^{n} (\alpha x_i + \beta y_i)\varepsilon_i+(|\alpha|e_x+|\beta|e_y)[-1, 1]$

where α, β and ζ can be taken from the affine approximation of the function f. The affine operation with one operand or more than two operands can be defined in the same way.

RevAA uses a special tight form for the multiplication operation:

$\hat{x} * \hat{y} = (x_0y_0 + \frac{1}{2}\sum_{i=1}^{n} x_iy_i) + \sum_{i=1}^{n} (x_0y_i + x_iy_0)\varepsilon_i + e_{xy}[-1, 1]$, where

$e_{xy} = e_xe_y + e_y(|x_0| + u) + e_x(|y_0| + v) + uv - \frac{1}{2}\sum_{i=1}^{n} |x_iy_i|$, $u = \sum_{i=1}^{n} |x_i|$, $v = \sum_{i=1}^{n} |y_i|$.

Unlike pure AA, we do not need to store noise symbols that are not depend on the input values, so they are accumulated in the last noise symbol. Therefore the length of the revised affine form n depends on the number of the uncertain input values or, in other words, on the number of independent input variables. In our technique we use $n = 1$ for ray-tracing purposes, as we have only one uncertain parameter for the ray and $n = 3$ for spatial enumeration in 3-dimensional space, as in this case we have three independent uncertain input values (one for each coordinate).

4. Extending Affine Arithmetic with affine operations for arbitrary functions

The core of the interrogation methods for implicit surfaces is the inclusion test for the surface on the given interval of its arguments. For the inclusion test we use the inclusion property of Revised Affine Arithmetic, in other words we should create a revised affine form for the procedure defining the surface.

The basic revised affine form for the function is obtained from the procedural definition of the function by replacing all the operations on real numbers by operations on the revised affine forms of the coordinate variables: $\hat{x}$, $\hat{y}$ and $\hat{z}$. The affine form derived in this way can be called a natural affine extension. Non-arithmetic operations such as trigonometric, logarithmic and reciprocal operations, can be calculated by using the general affine form applied to RevAA. The formulation for the coefficients of the general affine form and most of the basic non-arithmetic operations can be found in [25].

We noted above that any non-affine operation can be presented by using the general affine form. This leads to the idea that not only mathematical operations can be represented by affine forms, but also combinations of mathematical operations. By testing several different affine forms for procedurally defined surfaces, we found that the affine form leads to better speed and quality characteristics of the interrogation process in the following cases:

- When some complex function is repeated several times in the defining procedure; in other words when we have some repeated operation inside an definition.
- When the defining function is bounded on a given argument interval, but the natural affine extension of the defining function has infinite bound(s) because of the discontinuity of one of the functions included in the defining function.
- When conditional or other operators not supported by Affine Arithmetic are used.

4.1. Overview of the generation of the affine forms for operations

Although in the general case it is not possible to construct the general affine form for an arbitrary operation, it is possible to do so in many particular cases. Note that the performance of the general affine form for each operation is not guaranteed to be better than its natural affine extension and should be considered separately in each case. In this section we present some general observations regarding the construction of general affine forms for operations.

Consider an operation over n arguments as a function of the form $f(x_1, x_2, ..., x_n)$. According to [25] the general affine form on the argument interval I can be constructed in these cases:

1. The function is bounded and continuous in I. In this case at least the interval approximation is possible.
2. The above condition is satisfied, the function is twice differentiable and the sign of its second derivative does not change in I. In this case at least the min-range approximation is possible.
3. The above condition is satisfied and there is a way to easily solve the equation $f'(u) = \alpha$, where α is some known coefficient and u is an unknown point in I. In this case the optimal approximation is possible.

In practice, even the first condition can not be satisfied for complex expressions for an arbitrary I. In this case, the natural affine extension should be used. Moreover, in the case where some approximation is possible to construct for an operation, this approximation can be less effective from the point of view of the performance of the ray-tracing or enumeration subdivision algorithm. Therefore each function should be tested separately and used only in the case where the performance can be improved by using this approximation.

Below we present several case studies of deriving affine forms for non-affine operators. First, we present a case study for operations by deriving the affine form for CSG operations using R-functions. For a function with an argument discontinuity, we present the affine form for the displacement in the blending operation and subsequently, as a case study for the conditional operator, we present a simple conditional function used in one of the test models.

4.2. Case study 1: CSG operations using R-functions

CSG operations are common in implicit surfaces converted from CSG models or created in a constructive way. Here we present the formulation for set-theoretic (CSG) operations using R-functions [22] as follows:

$$R_{uni}(f_1, f_2) = f_1 + f_2 + \sqrt{f_1^2 + f_2^2}$$

$$R_{int}(f_1, f_2) = f_1 + f_2 - \sqrt{f_1^2 + f_2^2}$$

where R_{uni} is the function for the set-theoretic union and R_{int} is the function for the set-theoretic intersection. These functions have C^1 continuity on the whole domain except at the points where both arguments are equal to zero.

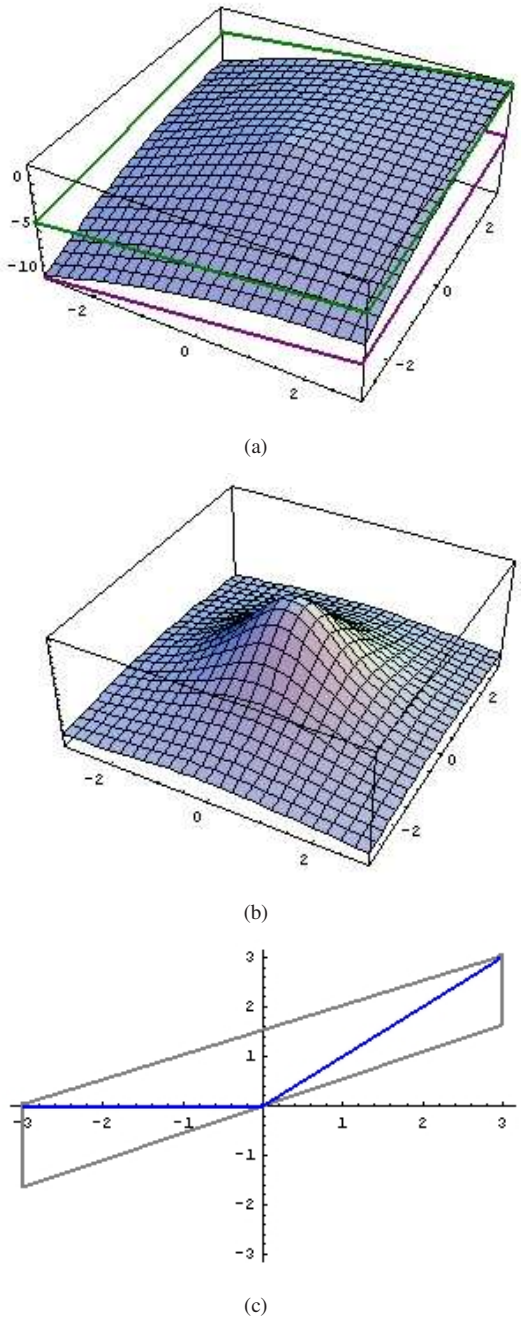


Figure 2: Case studies considered in this paper: a) Construction of an affine form for an intersection operation in a form of R-function, green denotes the tangent plane to the base point, magenta denotes a plane parallel to the tangent plane; b) 3D view of a displacement function in the blending operations using R-functions; c) Construction of an affine approximation for the conditional function;

Optimal and min-range approximations can be obtained for these functions, because all the requirements are satisfied, however the optimal approximation for these particular functions is computationally expensive compared to the natural extension of these functions in RevAA. We propose to use the min-range approximation for these functions. By analogy with the construction of an approximation for the function of one variable discussed above, we construct the approximation for the R-function as follows:

- Construct the surface $z = R(f_1, f_2)$ where R is an appropriate R-function.
- Find the tangent plane for one of the corner points for the patch of the surface on the given argument interval.
- Find the plane parallel to the tangent plane such that the patch lies completely inside these two planes and the distance between these planes is minimal.
- Find the coefficients for the affine form from the equations of these planes.

In Fig. 2a we show a surface plot of the R-function for the intersection operation with the tangent planes. It should be apparent that the first partial derivatives for the R-functions have the same sign on the whole domain except at the point (0,0) where a C^1 discontinuity is present. This property means that on the argument interval $[(f_1^{min}, f_2^{min}), (f_1^{max}, f_2^{max})]$ the R-function has a minimum at the point $R(f_1^{min}, f_2^{min})$ and a maximum at $R(f_1^{max}, f_2^{max})$.

Also if we compare the slope of the tangent plane at the minimum and maximum points, it is apparent that the slope of the tangent plane is smaller at the minimum point for the union operation and at the maximum point for the intersection operation. We call this point the base point. Note that the base point can not be the point (0,0), in which case we have to select the opposite point as the base point.

Thus, our approximation in the revised affine form is constructed according to the following rules:

- $\alpha = \frac{\partial R}{\partial f_1}, \beta = \frac{\partial R}{\partial f_2}$, where partial derivatives are taken at the base point. In the general case $\alpha = 1 + \frac{2f_1}{2\sqrt{f_1^2+f_2^2}}, \beta = 1 + \frac{2f_2}{2\sqrt{f_1^2+f_2^2}}$ for the union operation, and $\alpha = 1 - \frac{2f_1}{2\sqrt{f_1^2+f_2^2}}$, $\beta = 1 - \frac{2f_2}{2\sqrt{f_1^2+f_2^2}}$ for the intersection operation.
- We calculate the distance to the other corner points of the surface patch using the equation of the plane $R(f_1, f_2) = \alpha f_1 + \beta f_2 + d$. Note that for the base point $d = 0$, a maximal distance d_{max} between the three points is taken.
- ζ and δ can be obtained from d_{max}. For the union operation $\zeta = \frac{d}{2}$ and for the intersection operation $\zeta = -\frac{d}{2}$ and $\delta = \frac{d}{2}$.

4.3. Case study 2: Blending operations using R-functions

In implicit surface modelling, set-theoretic operations with blending [28] can be defined as follows:

$$B(f_1, f_2) = R(f_1, f_2) + \frac{a_0}{1 + \frac{f_1^2}{a_1^2} + \frac{f_2^2}{a_2^2}}$$

Where, $R(f_1, f_2)$ is an R-function representing some set-theoretic operation. Given that the summation is an affine operation and given that we have an affine approximation for the set-theoretic operations presented above, for blending operations we have to construct an affine form for the following displacement function:

$$d(f_1, f_2) = \frac{a_0}{1 + \frac{f_1^2}{a_1^2} + \frac{f_2^2}{a_2^2}}$$

A surface plot of this function can be seen in Fig. 2b. The natural affine extension of this function is not optimal because of the division operation which can cause an infinite function interval, in the case where the argument interval includes zero. From the definition of the function d it is apparent that the denominator should always be positive, however this does not apply in the general case for Affine Arithmetic, because after the non-affine multiplication, the interval for the denominator can include zero because of the overestimation.

The function d does not satisfy the requirements for optimal and min-range approximations on the whole range, because the signs of the second order partial derivatives are changing. Thus, the only approximation that we can use is the interval approximation. Given that $a_0 > 0$, it should be apparent that the function $d(f_1, f_2)$ has a global maximum at the point [0, 0] and a local maximum at the lines $[f_1, 0]$ and $[0, f_2]$. The affine approximation is constructed according to the following rules:

- $d_{min} = inf(d)$ taken at the four corner points.
- $d_{max} = d = \begin{cases} |a_0|, 0 \in \hat{f}_1 and 0 \in \hat{f}_2 \\ sup_{local}(d), 0 \in \hat{f}_1 or 0 \in \hat{f}_2 \\ sup_{corner}(d), otherwise \end{cases}$
- $\alpha = 0, \zeta = \frac{d_{min}+d_{max}}{2}, \delta = \frac{d_{max}-d_{min}}{2}$

Here sup_{local} denotes the supremum taken at the local maximum point and sup_{corner} denotes the supremum taken at the four corner points.

4.4. Case study 3: Conditional operators

As a simple example of a conditional operator consider the following condition:

$$y = \begin{cases} x, x > 0 \\ 0, x \le 0 \end{cases}$$

Replacing the real variables by the affine forms, we obtain the third line in the condition:

$$\hat{y} = \begin{cases} \hat{x}, x > 0, \forall x \in \hat{x} \\ 0, x \le 0, \forall x \in \hat{x} \\ \alpha\hat{x} + \zeta \pm \delta, otherwise \end{cases}$$

In other words, for affine forms that include zero in their interval, we have to obtain an affine approximation for the conditional function. It should be apparent that the function $y = f(x)$ is bounded on the whole domain and has a special point (0,0). We construct the affine approximation in almost the same way the optimal approximation was constructed (see Fig. 2c):

- $\alpha = \frac{f(x_{max}) - f(x_{min}}{x_{max} - x_{min}}$
- ζ is obtained from the equation $\zeta = \frac{r(u)}{2}$, where $r(x) = f(p) - \alpha p$, p can be either x_{min} or x_{max}. Note that this equation is derived from the equation for the optimal approximation with $u = 0, f(u) = 0$.
- $\delta = \frac{|f(p) - \alpha p|}{2}$

This technique can not be applied to an arbitrary condition because of the complexity of the construction of the affine approximation (in the case of a complex condition). For example, the condition:

$$y = \begin{cases} f_2, f_1 > 0 \\ f_3, f_1 \leq 0 \end{cases}$$

requires us to construct an affine approximation of a ternary function (as long as f_1, f_2 and f_3 are non-constant variables), which is a very complex task. However, in the case of simple conditions, the affine approximation can be very useful for functions with conditional operators.

5. Interrogation of implicit surfaces with Extended Revised Affine Arithmetic

In this section we present algorithms for fast interrogation of implicit surfaces in ray-surface intersection for ray-tracing and for spatial enumeration for polygonization. The main part of these algorithms is the function range computation for the zero roots calculation (in the case of the ray-surface intersection algorithm) and for testing a space block (cell) intersection with the surface (in the case of the spatial enumeration). In both examples we show how our extended RevAA can be used.

5.1. Ray-surface intersection for ray-tracing

Our algorithm is based on the ray-surface intersection technique for implicit surfaces that uses interval analysis, which originally appeared in [12] and later in many papers related to interval-based methods applied to the rendering of implicit curves and surfaces. The ray-surface intersection procedure is shown in Algorithm 1.

The basic idea of the algorithm is quite simple: we calculate the range of the function for the given argument interval using extended RevAA, we reject the interval if the range does not include the zero value, otherwise we subdivide the interval into two subintervals by using dichotomy and we repeat the procedure for both subintervals. An example of the affine form of the function after the dichotomy is shown in Fig. 3. Note that in the case when only the first root is needed (for example, for primary rays), we can exit from the procedure earlier if we have

Algorithm 1 Ray-surface intersection

Procedure: bool intersect(t_{min}, t_{max})
Calculate the affine form F for the function on the interval $[t_{min}, t_{max}]$
Get the range of the function from the affine form
if the range of the function does not include a 0 value **then**
  return FALSE (no roots in this interval);
end if
Calculate the argument estimation from the affine form: t'_{min}, t'_{max}
Find the pruned argument range:
$t_{min} = max(t_{min}, t'_{min})$;
$t_{max} = min(t_{max}, t'_{max})$;
if the length of the argument interval is less than some predefined threshold **then**
  Store the midpoint of the interval as the root;
  return TRUE;
end if
Calculate the midpoint of the argument range:
$t_{mid} = (t_{min} + t_{max})/2$;
Repeat the procedure for the two subintervals:
bool b1 = intersect(t_{min}, t_{mid});
if b1 is TRUE and only the first root is needed **then**
  return TRUE;
end if
bool b2 = intersect(t_{mid}, t_{max});
if b2 is TRUE **then**
  return TRUE;
end if
return FALSE;

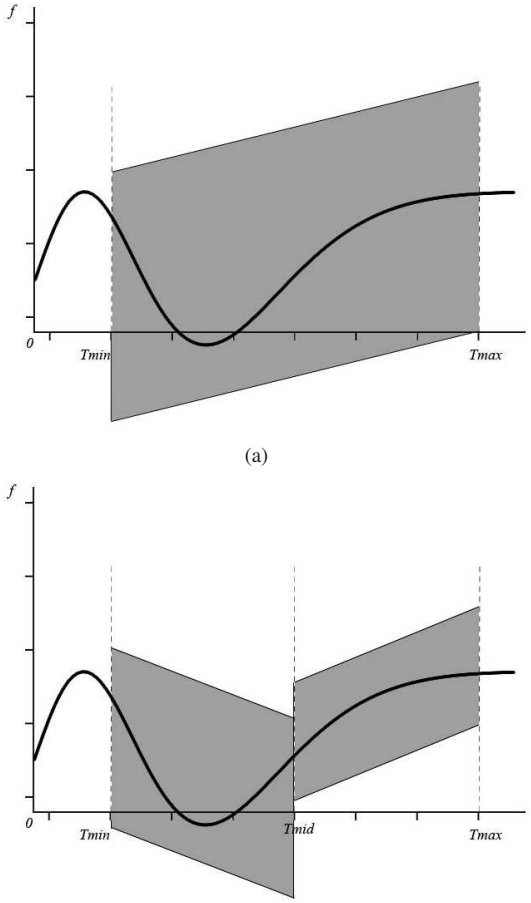


Figure 3: a) The revised affine form of the function on the interval $[t_{min}, t_{max}]$ b) The revised affine form of the function on the two subintervals after the dichotomy.

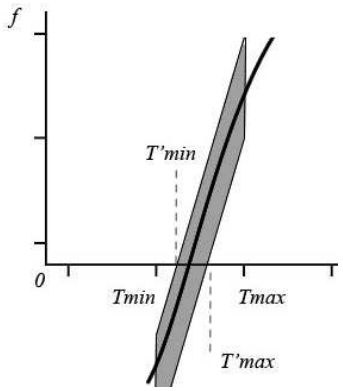


Figure 4: Pruning of the interval $[t_{min}, t_{max}]$ to the interval $[t'_{min}, t'_{max}]$ after the evaluation of the revised affine form for the function.

found a root in the first subinterval after the recursive procedure call.

As noted above, the most convenient way to create the revised affine form applicable to ray-tracing is to use $n = 1$. In this case e_1 denotes the error along the ray. The revised affine form for the function is obtained from the procedural definition of the function by replacing all the operations on real numbers by operations on the revised affine forms of the coordinate variables:

$$\hat{x} = x_0 + \hat{t} * d_x$$

$$\hat{y} = y_0 + \hat{t} * d_y$$

$$\hat{z} = z_0 + \hat{t} * d_z$$

where x_0, y_0, z_0 are the coordinates for the ray origin and d_x, d_y, d_z are the components for the ray direction vector and are constant for each ray, and $\hat{t} = (t_{min}+t_{max})/2+((t_{max}-t_{min})/2)\varepsilon_1$ is an affine form for the argument interval. Non-arithmetic operations, such as trigonometric, logarithmic and reciprocal operations, can be calculated by using the general affine form applied to RevAA. The formulation for the coefficients of the general affine form and most of the basic non-arithmetic operations can be found in [25]. Other non-affine operations used in the procedural definition can be extended by the special affine forms discussed above.

After the affine form computations, we obtain the range of the function from the affine form $\hat{f} = f_0 + f_1\varepsilon_1 \pm e_f$:

$$f_{min} = f_0 - |f_1| - e_f$$

$$f_{max} = f_0 + |f_1| + e_f$$

One of the useful properties of the reduced affine forms, including RevAA, is that of argument pruning (a term taken from the literature of interval slope methods), which means narrowing the argument range in case the root is contained in the interval. In [14] an argument pruning formulation was suggested for Affine Arithmetic that used condensation and in [17]argument pruning was suggested for Reduced Affine Arithmetic. In the latter paper the term interval optimisation was used instead of

argument pruning. As the geometric meaning of the revised affine form is similar to that of the reduced affine form, an analogous formulation can be used as follows. Given the revised affine form for the function $\hat{f} = f_0 + f_1\varepsilon_1 \pm e_f$ for the interval $\hat{t} = t_0 + t_1\varepsilon_1$, provided that $t_1 \neq 0$, $f_1 \neq 0$ and $e_f \neq 0$, the interval can be pruned by the points $t'_{min} = t_0 - \frac{t_1 f_0}{f_1} \pm e_f \frac{t_1}{f_1}$ and $t'_{max} = t_0 - \frac{t_1 f_0}{f_1} \pm e_f \frac{t_1}{f_1}$ if these points lie inside the interval $[t_{min}, t_{max}]$ (see Fig. 4).

5.2. Spatial enumeration

The spatial enumeration procedure allows us to find cells where the surface potentially exist for further processing by known polygonization algorithms such as the marching cubes algorithm [7]. The basic algorithm presented here only finds the cells intersecting with the surface by applying the recursive subdivision to the given bounding box (argument interval) and is based on the classical subdivision method described in [29].

Algorithm 2 Spatial enumeration

Procedure: bool isSurfaceExist($\hat{x}$, $\hat{y}$, $\hat{z}$)
Calculate the affine form F for the function with the arguments [$\hat{x}$, $\hat{y}$, $\hat{z}$]
Get the range of the function from the affine form
if the range of the function does not include a 0 value **then**
  return FALSE (no surface in this space);
end if
if the size of the node defined by the intervals ($\hat{x}$, $\hat{y}$, $\hat{z}$) is smaller than some predefined threshold **then**
  return TRUE (the surface may exists);
end if
Subdivide $\hat{x}$, $\hat{y}$ and $\hat{z}$ using dichotomy into $\hat{x}_1$, $\hat{x}_2$, $\hat{y}_1$, $\hat{y}_2$, $\hat{z}_1$, $\hat{z}_2$.
Construct 8 sub-nodes from all the possible combinations of $\hat{x}_1$, $\hat{x}_2$, $\hat{y}_1$, $\hat{y}_2$, $\hat{z}_1$, $\hat{z}_2$.
Recursively call the procedure for each of the sub-nodes.
if all the recursion calls returns FALSE **then**
  return FALSE;
end if
return TRUE;

The basic idea of the algorithm is that of the classical octree construction - we calculate the range of the function for the given argument interval using the extended RevAA, reject the node if the range does not include the zero value, otherwise subdivide the node into eight sub-nodes and repeat the procedure recursively for each of the sub-nodes while the size of the sub-node is greater than a pre-defined threshold.

To perform dichotomy in RevAA, we can convert to the intervals and back again. However, as the accumulation error for coordinate affine variables is 0, the dichotomy of the coordinate affine variable $\hat{t} = t_0 + t_1 * \varepsilon_1$ can be performed as:

$$\hat{t}_1 = t_0 - t_1 * 0.5 + t_1 * 0.5 * \varepsilon_1$$

$$\hat{t}_1 = t_0 + t_1 * 0.5 + t_1 * 0.5 * \varepsilon_1$$

Unlike in ray-tracing, the simplest Revised Affine Form for this algorithm is the one where $n = 3$, with the first three noise symbols in the affine form representing the errors in x, y and z. As for ray-tracing, the revised affine form for the function is obtained from its procedural definition by replacing all the operations on real numbers by operations on the revised affine forms and by using special affine forms if needed. The function range can be obtained from the affine form in the same way.

6. Implementation

In this section we present several details of the implementation of the interrogation process for procedurally defined implicit surfaces. The affine form representation and the representation of the function in the revised affine form are also described here.

6.1. Affine form representation

As we have stated above, the revised affine form is a polynomial with three terms. Thus, the affine form in the software implementation can be represented as a three-component vector, where the first component stands for x_0, the second stands for the noise symbol of the error along the ray and the third represents the half-length of the accumulating interval. The calculations in Affine Arithmetic can be performed on these vectors. Almost all of the arithmetic operations have to be overridden as only summation in Revised Affine Arithmetic matches the standard vector summation. For example, the subtraction and multiplication operations can be implemented as follows:

```
vec3 ra_subtraction(vec3 x, vec3 y){
vec3 ret;
ret[0] = x[0] - y[0];
ret[1] = x[1] - y[1];
ret[2] = x[2] + y[2];
return ret;
}

vec3 ra_multiplication(vec3 x, vec3 y){
vec3 ret;
ret[0] = x[0]*y[0]+0.5*x[1]*y[1];
ret[1] = x[0]*y[1]+y[0]*x[1];
ret[2] = x[2]*y[2]+
   y[2]*(fabs(x[0])+fabs(x[1]))+
   x[2]*(fabs(y[0])+fabs(y[1])) +
   0.5*fabs(x[1]*y[1]);
return ret;
}
```

Similarly, non-affine operations can be implemented as operations on the three-component vectors. Note that for non-affine operations we are most likely to use the affine constructor described above. For example, the square root operation can be implemented in this way using the optimal approximation described in [25]:

```
vec3 ra_sqrt(vec3 x){
vec2 i = ra_getinterval(x);
if (i[1] < 0) return 0;
if (i[0] < 0) i[0] = 0;
double sq1 = sqrt(i[0]), sq2 = sqrt(i[1]);
//calculate arguments for the revised affine form
double alpha = 1/(sq1+sq2);
double dzeta = (sq1+sq2)/8.0+0.5*sq1*sq2/(sq1+sq2);
double delta = (sq2-sq1)*(sq2-sq1)/(8.0*(sq1+sq2));
//create the revised affine form
vec3 ret;
ret[0] = alpha*x[0]+dzeta;
```

```
ret[1] = alpha*x[1];
ret[2] = alpha*x[2]+delta;
return ret;
}
```

In the above examples and in our implementation we do not use rounding control as the errors caused by using floating-point arithmetic are generally lower than the overestimation for all the tested models. However, in some cases rounding control is needed, as for example when defining functions with a large number of affine operations and relatively small number of non-affine operations. In these cases, we need to use the versions of non-affine operations adapted to rigorous computations. Note that in the case of rigorous computations the speed of computation decreases because of the larger number of computations. An example for rigorous version of multiplication and the general non-affine operation in Revised Affine Arithmetic can be found in [21].

6.2. Representation of the function

The ray-tracing algorithm works with objects defined by a real-valued functions of real-valued arguments. In the same way this function can be rewritten using the following rules:

- Each variable depending on the input arguments is replaced by a variable of the revised affine type, while each variable not depending on the input arguments and constants is left in the real form.
- If we have affine forms for functions or composition of functions for the given procedural model, we replace the code over the affine variables by these forms.
- We ensure that the implementations of the remaining operations are overridden in RevAA.

The returned value of the rewritten function is the range of the function in the affine form, which is used in the ray-surface intersection procedure described earlier.

Here we would like to note that in some cases the defining function needs to be optimised before constructing the affine form in order to decrease the potential number of computations. Such optimisation can include the usage of additional variables for repeating blocks of code containing a large number of non-affine operations, copy propagation, etc.

7. Results

In our tests we used a modified version of the POV-Ray ray-tracing software for the ray-tracing tests and a stand-alone spatial enumeration piece of software for the polygonization tests. The results were generated on a PC with an Intel Pentium 4 3.20GHz processor and 1Gb of RAM. To compare Revised Affine Arithmetic with other reliable techniques for uncertain computations, we used the following libraries:

- For Interval Arithmetic, we used a slightly modified version of the freely available C library *libia* by Jorge Stolfi.
- For Affine Arithmetic, we used the freely available C++ library *libaffa*.

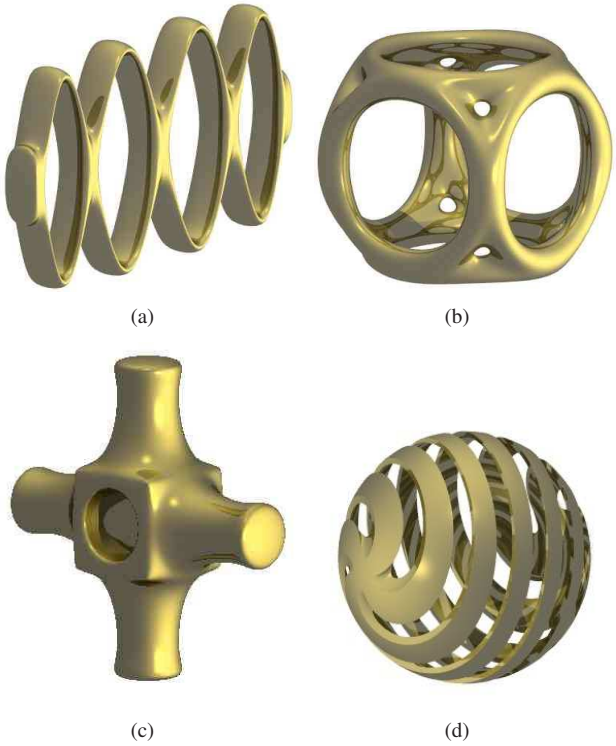


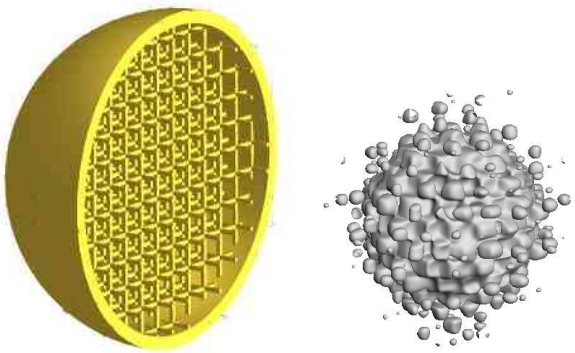
Figure 5: Ray-tracing of algebraic surfaces: a) Bretzel b) Decocube; and non-algebraic implicit surfaces: c) CSG with blending union and blending intersection d) Sphere with trimming

- Other libraries were written ab initio based on the above two libraries.

7.1. Ray-tracing of procedurally defined implicit surfaces

We tested our ray-tracing algorithm on a wide range of procedurally defined implicit models (see Figs. 5, 6). First, we compared our procedure with other reliable techniques based on uncertain computations, the technique based on Interval Arithmetic described in [3], the technique based on Interval Arithmetic in the Centred form is based on that described in [15] (where the mean value form [16] is used), the technique based on pure Affine Arithmetic described in [14] and the technique based on Reduced Affine Arithmetic described in [17]. The results can be found in Table 1.

The results show that other rendering algorithms based on standard Interval and Affine Arithmetic are significantly slower than our algorithm which is based on Revised Affine Arithmetic. The reason for this is that with Interval Arithmetic algorithms there is an overestimation and with Affine Arithmetic algorithms there is an overestimation as well as a large number of terms in the polynomial form and thus there is a large number of arithmetic operations in the affine operation calculations. Reduced Affine Arithmetic, as presented in [17] and implemented in [3], can only be used for algebraic models and proves to be faster than the Interval and standard Affine Arithmetic for algebraic models, however the overestimation of the function in



(a) (b)

(c)

Figure 6: Ray-tracing of procedural implicit surfaces with thin elements or small disjoint components: a) Sphere with microstructure b) Sphere with procedural noise c) Procedural hair

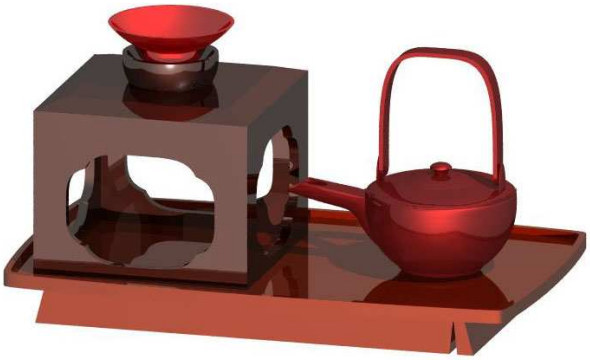
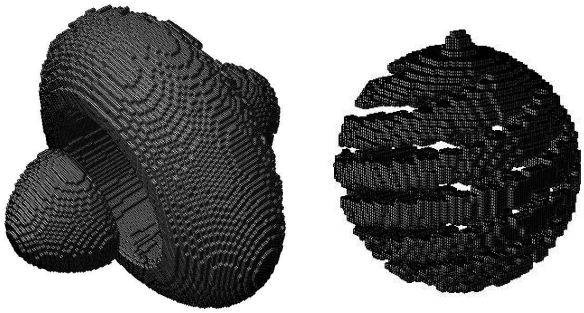


Figure 7: Ray tracing of procedural scenes: Virtual Shikki



(a) (b)

Figure 8: Spatial enumeration of the procedural implicit surfaces, voxels intersecting the surface are shown for: a) Mitchell surface b) Sphere with trimming.

Reduced Affine Arithmetic is slightly wider than the overestimation range for Revised Affine Arithmetic because of the non-optimised multiplication operation. Therefore the range of a function based on Revised Affine Arithmetic is tighter than the range in all other techniques and hence the speed of the calculation of the ray-surface intersections is significantly better, especially for models with a large number of non-affine operations. Also, we used the special affine forms for the set-theoretic operations in the procedurally defined models. The results show that the speed of our algorithm increases dramatically in this case.

The reliability of Revised Affine Arithmetic allows us to test our technique on several procedurally defined implicit models with small features or thin surface features. For example, by using the proposed ray-surface intersection calculation we can reliably render models with internal microstructures (see Fig. 6a), stochastic procedural models with disjointed components (see Fig. 6b) and even procedurally-defined hair (see Fig. 6c).

Our ray-tracing technique can be applied to complex scenes with a number of procedurally defined implicit surfaces. For example, we show how the functionally defined scene ”Virtual Shikki” can be rendered using our technique (see Fig. 7). Note that because of the thin features in the model of this scene approximate ray-marching techniques and polygonization do not work well for this scene.

	Resolution (pixels)	Number of operations All / Non-affine / Multiplications	IA	IAC	AA	RAA	RevAA	RevAA*
Mitchell	1280*1024	19 / 6 / 6	38	26	33	7	6	6
Bretzel	1280*1024	16 / 9 / 9	25	57	86	22	18	18
Decocube	1280*1024	30 / 17 / 17	17	98	226	19	13	13
CSG	640*480	96 / 40 / 32	126	269	129	n/a	18	11
Sphere with trimming	1024*768	142 / 54 / 37	837	2326	2566	n/a	285	83
Sphere with noise	800*600	36 / 11 / 5	17	82	51	n/a	9	9
CSG with blending	640*480	105 / 42 / 32	266	79	72	n/a	31	16
Hair	640*480	88 / 34 / 22	4004	2620	1935	n/a	658	23
Sphere with microstructure	640*480	65 / 33 / 22	1006	3683	1079	n/a	293	12
Virtual Shikki	320*240	822 / 306 / 213	29244	7169	50000+	n/a	390	32

Table 1: Comparison of the ray-tracing procedures for different computational models. IA stands for Interval Arithmetic, IAC for Interval Arithmetic in the Centred form, AA for Affine Arithmetic, RAA for Reduced Affine Arithmetic, RevAA for Revised Affine Arithmetic and RevAA* for Revised Affine Arithmetic extended by special non-affine operations. The timings for ray-tracing all the rays are shown in seconds.

7.2. Spatial enumeration

We tested our spatial enumeration algorithm on the same set of procedurally defined models (see Fig. 8). We compared our procedure with the spatial enumeration technique based on Interval Arithmetic, described in [30], and the technique based of Affine Arithmetic, described in [19]. The results can be found in Table 2. It is apparent that for simple polynomial models spatial enumeration using Revised Affine Arithmetic is comparable in speed to the enumeration using standard Affine Arithmetic. This is because of the longer interval length of the Revised Affine Form compared with that for ray-casting and the nature of polynomial models. This leads us to the conclusion that computations using Revised Affine Arithmetic become similar to the computations using Affine Arithmetic. However for complex expressions, containing a number of non-affine operations, the results show that the spatial enumeration with Revised Affine Arithmetic using special affine forms is faster than other reliable techniques.

8. Conclusion

In this paper we have presented techniques for the interrogation of general procedurally defined implicit models based on RevAA. By using the inclusion property of RevAA, we were able to obtain reliable ray-surface intersections for ray-tracing and a reliable cell-surface intersection test for the spatial enumeration in polygonization. At the same time, RevAA proved to be the fastest compared with other interval techniques.

Also in this paper we have shown how speed can be dramatically improved by using special affine forms for arbitrary operators. We derived the affine forms for the set-theoretic operations based on R-functions and for blending operations, and have shown that the speed and the quality of rendering can be dramatically improved using these.

However, not all functions can be rewritten in the affine form, either because they do not meet the requirements for the affine approximation or because they have a large number of arguments. Moreover, currently there is no general criterion for the derivation of these forms for arbitrary implicit surface models. Further research needs to be conducted in this area. Also currently the set of procedurally defined implicit models does not include models with complex conditional operators. We have shown how the conditional functions can be represented by special affine forms, however this technique can not be applied for general conditions. For instance, it would be hard to represent a conditional operators such as

$$f = \begin{cases} g, condition(y) == true \\ h, condition(y) == false \end{cases}$$

by a special affine form, as doing so requires the construction of an affine function with three arguments which is a quite complex task. Some research on conditional operators was done using Interval Arithmetic [31], however further research using Affine Arithmetic and especially RevAA is also needed.

9. Bibliography

[1] J. M. Singh, P. J. Narayanan, Real-time ray tracing of implicit surfaces on the gpu, IEEE Transactions on Visualization and Computer Graphics 16 (2) (2010) 261–272.

[2] C. Dyken, G. Ziegler, C. Theobalt, H.-P. Seidel, High-speed marching cubes using histopyramids, Computer Graphics Forum 27 (December 2008) 2028–2039(12).

[3] A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. D. Hansen, H. Hagen, Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic, Computer Graphics Forum 28 (1) (2009) 26–40.

[4] M. Gavriliu, Towards more efficient interval analysis: corner forms and a remainder interval newton method, Ph.D. thesis, Pasadena, CA, USA, adviser-Barr, Alan H. (2005).

[5] H. K. Tuy, L. t Tuy, Direct 2-d display of 3-d objects, IEEE Computer Graphics and Applications 4 (10) (1984) 29–33.

[6] W. E. Lorensen, H. E. Cline, Marching cubes: A high resolution 3d surface construction algorithm, SIGGRAPH Comput. Graph. 21 (4) (1987) 163–169.

[7] J. Bloomenthal et al. (Ed.), Introduction to Implicit Surfaces, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[8] A. Gomes, I. Voiculescu, J. Jorge, B. Wyvill, C. Galbraith, Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms, Springer Verlag, 2009.

[9] J. C. Hart, Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces, The Visual Computer 12 (1994) 527–545.

[10] A. Sherstyuk, Fast ray tracing of implicit surfaces, Computer Graphics Forum 18 (2) (1999) 139–147.

	IA	AA	RevAA	RevAA*
Mitchell	13.685	1.674	1.654	1.654
Bretzel	1.325	0.387	0.548	0.548
Decocube	2.923	3.647	1.322	1.322
CSG	140.14	17.650	9.601	5.064
Sphere with trimming	1650.390	82.846	81.083	12.872
Sphere with noise	29.451	17.611	9.740	9.457
CSG with blending	425.047	17.760	13.384	7.696
Sphere with microstructure	1847.248	183.155	342.276	8.765

Table 2: Comparison of the spatial enumeration procedures for different computational models. IA stands for Interval Arithmetic, AA for Affine Arithmetic, RevAA for Revised Affine Arithmetic and RevAA* for Revised Affine Arithmetic extended by special non-affine operations. The grid resolution for all models is 128*128*128. The timings for spatial enumeration are shown in seconds.

[11] O. Fryazinov, A. Pasko, Interactive ray shading of FRep objects, in: WSCG' 2008, Communications Papers proceedings, 2008, pp. 145–152.

[12] D. P. Mitchell, Three applications of interval analysis in computer graphics, in: Frontiers in Rendering course notes, 1991, pp. 1–13.

[13] J. M. Snyder, Interval analysis for computer graphics, in: Computer Graphics, 1992, pp. 121–130.

[14] A. de Cusatis Jr., L. H. Figueiredo, M. Gattass, Interval methods for ray casting surfaces with affine arithmetic, in: Proceedings of SIBGRAPI'99 - the 12th Brazilian Symposium on Computer Graphics and Image Processing, 1999, pp. 65–71.

[15] R. Martin, H. Shou, I. Voiculescu, G. Wang, A comparison of Bernstein hull and affine arithmetic methods for algebraic curve drawing, in: Proc. Uncertainty in Geometric Computations, Kluwer Academic Publishers, 2001, pp. 143–154.

[16] H. Ratschek, J. Rokne, Computer Methods for the Range of Functions, Halsted Press, Wiley, New York, 1984.

[17] M. N. Gamito, S. C. Maddock, Ray casting implicit fractal surfaces with reduced affine arithmetic, The Visual Computer 23 (3) (2007) 155–165.

[18] S. Plantinga, G. Vegter, Isotopic approximation of implicit curves and surfaces, in: SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, ACM, New York, NY, USA, 2004, pp. 245–254.

[19] L. H. D. Figueiredo, J. Stolfi, Adaptive enumeration of implicit surfaces with affine arithmetic, Computer Graphics Forum 15 (1996) 287–296.

[20] K. Bühler, Implicit linear interval estimations, in: SCCG '02: Proceedings of the 18th spring conference on Computer graphics, ACM, New York, NY, USA, 2002, pp. 123–132.

[21] X.-H. Vu, D. Sam-Haroud, B. Faltings, Combining multiple inclusion representations in numerical constraint propagation, in: ICTAI, 2004, pp. 458–467.

[22] V. Shapiro, Semi-analytic geometry with R-functions, Acta Numerica 16 (2007) 239–303.

[23] K. Perlin, E. M. Hoffert, Hypertexture, SIGGRAPH Comput. Graph. 23 (3) (1989) 253–262.

[24] G. Y. Gardner, Simulation of natural scenes using textured quadric surfaces, SIGGRAPH Comput. Graph. 18 (3) (1984) 11–20.

[25] L. H. de Figueiredo, J. Stolfi, Self-Validated Numerical Methods and Applications, Brazilian Mathematics Colloquium monographs, IMPA/CNPq, Rio de Janeiro, Brazil, 1997.

[26] L. H. D. Figueiredo, J. Stolfi, Affine arithmetic: Concepts and applications, Numerical Algorithms 37 (2004) 147–158.

[27] F. Messine, Extensions of affine arithmetic: Application to unconstrained global optimization, Journal of Universal Computer Science 8 (11) (2002) 992–1015.

[28] A. Pasko, V. Adzhiev, A. Sourin, V. Savchenko, Function representation in geometric modeling: concepts, implementation and applications., The Visual Computer 11 (8) (1995) 429–446.

[29] D. Kalra, A. H. Barr, Guaranteed ray intersections with implicit surfaces, in: SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, 1989, pp. 297–306.

[30] N. Stolte, A. Kaufman, Parallel spatial enumeration of implicit surfaces using interval arithmetic for octree generation and its direct visualisation, in: Implicit Surfaces'98, 1998, pp. 81–88.

[31] J. F. Diaz, Improvements in the ray tracing of implicit surfaces based on interval arithmetic, Ph.D. thesis, Departament d'Electronica, Informatica i Automatica, Universitat de Girona, Girona, Spain (Nov. 2008).

Appendix A. Formulae for surfaces used in the paper

Mitchell:

$f = 20*(x^2+y^2+z^2) - 4*(x^4+(y^2+z^2)^2) - 17x^2*(y^2+z^2) - 17$

Bretzel:

$f = 2 - 60*z^2 - (x^2*(1.21-x^2)^2*(3.8-x^2)^3 - 10*y^2)^2$

Decocube:

$f = 0.02 - ((x^2+y^2-0.82)^2 + (z^2-1)^2)*((y^2+z^2-0.82)^2 + (x^2-1)^2)*((x^2+z^2-0.82)^2 + (y^2-1)^2)$

CSG:

$f = b|(s\&((c_1 \setminus c_2)|(c_3 \setminus c_4)) \setminus c_5$, where $b = (0.36-x^2)\&(0.36-y^2)\&(0.36-z^2)$, $s = 0.7056 - x^2 - y^2 - z^2$, $c_1 = 0.09 - y^2 - z^2$, $c_2 = 0.04 - y^2 - z^2$, $c_3 = 0.09 - x^2 - z^2$, $c_4 = 0.04 - x^2 - z^2$, $c_5 = 0.25 - x^2 - y^2$

Sphere with noise:

$f = 81 - x^2 - y^2 - z^2 + (3.8*sin(1.5*x) + sin(1.111*x + 1.1*sin(1.5*x))*1.624)*(3.8*sin(1.5*y) + sin(1.111*x + 1.1*sin(1.5*x))*1.299)*(3.8*sin(1.5*y) + sin(1.111*x + 1.1*sin(1.5*x))*2.598)$

Sphere with microstructure:

$f = (((1 - x^2 - y^2 - z^2)\&((sin(20*y) - 0.9)\&(sin(20*z) - 0.9))|((sin(20*x) - 0.9)\&(sin(20*z) - 0.9))|((sin(20*x) - 0.9)\&(sin(20*y) - 0.9)))|((1 - x^2 - y^2 - z^2) \setminus (0.75 - x^2 - y^2 - z^2)))\&(-z)$

CSG with blending:

$f = (((c_1 \vee_b c_2) \wedge_b s) \vee_b b) \wedge_b (-c_3)$, where $b = (0.36-x^2)\&(0.36-y^2)\&(0.36-z^2)$, $s = 0.7056 - x^2 - y^2 - z^2$, $c_1 = 0.09 - y^2 - z^2$, $c_2 = 0.09 - x^2 - z^2$, $c_3 = 0.25 - x^2 - y^2$, $\vee_b$ denotes blending intersection: $f_1 \vee_b f_2 = f_1 + f_2 - \sqrt{f_1^2 + f_2^2 + \frac{0.5}{1+f_1^2+f_2^2}}$, $\wedge_b$ denotes blending union: $f_1 \wedge_b f_2 = f_1 + f_2 + \sqrt{f_1^2 + f_2^2 + \frac{0.5}{1+f_1^2+f_2^2}}$

Hair:

$f = o|(((1.8*sin(1.8*x*\frac{9}{\sqrt{x^2+y^2+z^2}}) + s_x)*(1.8*sin(1.8*y*\frac{9}{\sqrt{x^2+y^2+z^2}}) + s_y)*(1.8*sin(1.8*z*\frac{9}{\sqrt{x^2+y^2+z^2}}) + s_z) - 10)\&(o+2)\&y)$, where $o = (1 - \frac{x^2}{16} - \frac{y^2}{36} - \frac{z^2}{16})|(1 - \frac{x^2}{1.6129} - \frac{(y+2.5)^2}{2.25} - \frac{(z-3)^2}{1.6129})$, $s_x = 1.538*sin(1.33*x*\frac{9}{\sqrt{x^2+y^2+z^2}} + 1.4*sin(1.8*x*\frac{9}{\sqrt{x^2+y^2+z^2}}))$,

$$s_y = 1.538 * sin(1.33 * y * \frac{9}{\sqrt{x^2+y^2+z^2}} + 1.4 * sin(1.8 * y * \frac{9}{\sqrt{x^2+y^2+z^2}})),$$
$$s_z = 1.538 * sin(1.33 * z * \frac{9}{\sqrt{x^2+y^2+z^2}} + 1.4 * sin(1.8 * z * \frac{9}{\sqrt{x^2+y^2+z^2}}))$$

The sphere with trimming model is described in the paper "Trimming implicit surfaces":
http://hyperfun.org/wiki/doku.php?id=frep:trimming

The Virtual Shikki description files in HyperFun format can be found at:
http://www.hyperfun.org/App/shi/Shikki.html

In the above formulae & denotes set-theoretic intersection with R-functions, | denotes set-theoretic union with R-functions and \ denotes set-theoretic subtraction with R-functions.