Discrete Implicit Surface Models using Interval Arithmetics

Nilo STOLTE and Arie KAUFMAN

Visualization Laboratory
Computer Science Department
State University of New York at Stony Brook
Stony Brook
USA

stolte@cs.sunysb.edu, ari@cs.sunysb.edu

Abstract

This article presents a new method for generating discrete implicit surface models using interval arithmetics. This approach is unique to our knowledge and has the advantage of profiting of voxel representation for both modelling and visualization purposes. We've adopted high discrete resolutions to obtain better surface represention and realistic visualisation. Interval arithmetics guarantees a good rasterization quality and calculation precision.

In the surface visualization using high resolution discrete space, each voxel in this space contains the surface normal vector in the middle of the voxel for shading calculation. The advantage of using high resolution discrete spaces is that we can obtain good quality images and at the same time visualize the scene interactively.

Keywords :

- Interval Arithmetics, Implicit surfaces, Voxel, Set-theoretic Operations, Robustness.

1. Introduction

Set-theoretic operations are the basis of the Constructive Solid Geometry (CSG). Set-theoretic operations in the CSG context are used to construct complex objects from basic primitives such as cubes, spheres, cylinders, and cones. These primitives are usually represented by polygons in current CSG modelers. Implicit surfaces **[Sourin96, Pasko96**] generalize this idea by representing objects by their implicit equation and by defining set-theoretic operators as part of the function. This not only allows a wider variety of objects, but also allows a more detailed representation. Nevertheless ray-casting algorithm is often needed to visualise this kind of surfaces. Unfortunately interactive visualisation is not possible since ray-casting is still a very slow rendering algorithm. We propose to transform the surfaces into a high resolution discrete space and visualize them interactively allowing easy inspection of the produced surface.

Duff **[Duff 92]** has introduced interval arithmetics **[Moore 66, 79]** in CSG of implicit surfaces. Algebraic Implicit functions **[Sederberg 90]** were stored in the leaves of a binary tree containing CSG operators. Function evaluation was done by interpreting this binary tree. This approach is a mixture between analytic functions and prodedural evaluation. Another particularity of this method is that interval arithmetic was used to subdivide the implicit surfaces in image space using a quadtree subdivision on the projection plane. This was part of the visualisation process, where a ray-casting was activated when subdivision arrived at the pixel level. Perspective projection is achieved by applying the transformations to the function's equations in order to maintain the viewing volume in rectangular shape. A strong drawback in this method is that it is dependent on the point of view. If the view point changes all the transformations have to be recalculated and the whole subdivision has to be repeated. Since the rendering algorithm is mixed with the subdivision method this approach could hardly profit from recent graphics cards hardwired facilities to accelerate display. Therefore this method could not be used to visualize implicit surfaces in an interactive way.

We propose a totally different approach where interval arithmetics is used to subdivide object space **[Stolte 95a]** to convert the surface into voxels. We use octree recursive subdivision to construct a voxel based model to represent the surface. Since we use high discrete space resolutions we need a very good performance not possible using binary tree interpretation. Implicit functions offer a better representation for function set-theoretic operations. Particularly they offer better performance and higher abstraction level, since set-theoretic operators can be directly integrated into the implicit function expression which defines the surface. This has several advantages: no need of constructing neither interpreting functions set-theoretic operators binary trees, better performance and symbolic set-theoretic operations definition. This later advantage allows us to describe a model in a symbolic way. This means that the models can be defined in a formal way and entered into the machine in this format.

We show that the problem of soft transitions betwen different functions in set-theoretic operations can be solved using blending functions as proposed in **[Stolte 96]**. It is possible, for exemple, to create infinitely derivable transitions by using gaussians as blending functions. Soft transitions avoid all aliasing problems found in **[Duff 92]** when several surfaces share the same region into the space. This is a consequence of the fact that blending functions help to integrate several different surfaces equations in just one surface equation. Thus only one surface is defined in any region into the space, eliminating the problem of deciding which

surface is really visible inside a certain region.

Another difference between our approach and Duff's **[Duff 92**] is that we convert the surface into voxels to visualize it (as proposed in **[Stolte 95a]**, **[Stolte 96]**) while Duff uses ray-casting. The advantage of using voxels is that an interactive visualization becomes possible. In addition, using high resolution 3D grids of voxels (**[Stolte 95a, 95b, 96]**) allows us a very good quality surface rendering almost as using ray-casting.

1. Implicit Surfaces

1.1. Object definition

Implicit surfaces can be used for representation of solid objects on the following way:

f(x,y,z) = 0, define a surface;
f(x,y,z) ≤ 0, define the solid objetc including the surface.

If f(x,y,z) > 0, then the point (x,y,z) is outside the surface. If f(x,y,z) = 0, the point (x,y,z) is at the surface. If f(x,y,z) < 0, the point is inside the solid. This notation is the opposite to the one presented in **[Sourin 96, Pasko 96]** but agrees with common implicit surfaces notation.

1.1. Blending functions

The idea behind blending functions is that several primitives given by Implicit Functions f_i : $\Re^3 \rightarrow \Re$ are mixed together through the functions g_i : $\Re \rightarrow \Re$ as shown into the following expression:

$$\sum_{i=0}^{n} a_i \cdot g_i (b_i \cdot f_i (x,y,z)) = C$$

One of the most popular blending function is the gaussian. It is the blending function used into the "Blobby Model" **[Blinn 82, Muraki 91, Bidasaria 92]**:

$$\sum_{i=0}^{n} a_i \cdot e^{-(b_i \cdot f_i (x,y,z))} = C$$

If the sign of a_i is positive, function f_i (x,y,z) is added to the rest of the model as in a union operator, but with a soft transition controled by the blending function. If the sign of a_i is negative the primitive is subtracted from the model as in a difference operator, but with soft transitions.

The exponential function can be substituted by a polynomial blending function as in **[Wyvill 86, BW 90]** ($r = b_i \cdot f_i (x,y,z)$) :

$g(r) = -(4/9)r^6/R^6 + (17/9)r^4/R^4 - (22/9)r^2/R^2$ if $r <= R$
$g(r) = 0$ otherwise

or as in **[Stolte 96]**:

$g(r) = (r^2-R^2)^4 /R^8$ if $r <= R$
$g(r) = 0$ otherwise

The advantage of using polynomial blending functions over gaussians is not only its better evaluation times but also its local action. Gaussians have global action, that is, each primitive influences all the others. This is not very convenient in very huge models. With polynomial blending functions blending action is local thanks to the roots in r=R. Every distance superior than R is not considered into the blending action. That's why R is called the "radius of influence" of the blending function.

Stolte's blending function **[Stolte 96]** is particularly better than Wyvill's due to better continuity transition between different primitives.

2. Interval Arithmetic

2.1. Inclusion Functions

Duff and Snyder **[Duff 92, Snyder 92**] have simultaneously but independently introduced interval arithmetic to solve Computer Graphics problems. Duff concentrated in 3D implicit surfaces subdivision and Snyder in more general problems like silhouette edge detection, surface polygonization, minimum distance determination, etc.

Interval arithmetic guarantees that the exact result of any arithmetic operation is between two values, called *interval bounds*. Any real number is represented by two interval bounds. For example, the coordinates, ***X***, ***Y*** and ***Z*** are represented in interval arithmetic as:

$\boldsymbol{X} = [x, X]$, where $x \in \Re$ and $X \in \Re$ and $x \leq X$
$\boldsymbol{Y} = [y, Y]$, where $y \in \Re$ and $Y \in \Re$ and $y \leq Y$
$\boldsymbol{Z} = [z, Z]$, where $z \in \Re$ and $Z \in \Re$ and $z \leq Z$

These interval bounds in our case are the coordinates of the octant's boundaries. In other words $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) \in \Re^3$ represents a 3D brick volume in $\Re^3$. Substituting each regular variable by the correspondent interval and each regular operation by the respective interval operation in the implicit surface equation, produces an interval version of the function, which Snyder **[Snyder 92**] calls an *inclusion function*. We can verify if the surface doesn't pass through the octant simply testing if the resulting interval doesn't *include* zero, that is, when the inclusion function resulting interval doesn't *include* a solution for the regular function F(x,y,z)=0. Then if the resulting interval doesn't include zero, the function certainly doesn't have a zero into the octant. Therefore the surface doesn't pass through the octant.

The interval arithmetic operators are:

$\boldsymbol{X} + \boldsymbol{Y} = [x+y, X+Y]$
$\boldsymbol{X} - \boldsymbol{Y} = [x-Y, X-y]$
$\boldsymbol{X} * \boldsymbol{Y} = [\min(x*y, x*Y, X*y, X*Y), \max(x*y, x*Y, X*y, X*Y)]$
$\boldsymbol{X} / \boldsymbol{Y} = [\min(x/y, x/Y, X/y, X/Y), \max(x/y, x/Y, X/y, X/Y)]$ if $0 \notin \boldsymbol{Y}$

These operators are not enough for the functions used in practice. To include any algebraic expression we need:

$$\boldsymbol{X}^{\boldsymbol{n}} = \begin{cases} [x^n, X^n] & \text{if n odd or } x \geq 0 \\ [X^n, x^n] & \text{if n even and } X \leq 0 \\ [0, \max(x, X)^n] & \text{if n even and } 0 \in \boldsymbol{X} \end{cases}$$

To include gaussians, which are useful as blending functions in "blobby" models, we have:

$$e^{-\boldsymbol{x}} = [\, e^{-X}, \; e^{-x} \,]$$

Any other function can be converted to interval arithmetic similarly by breaking the function in their monotonic intervals**[Duff 92]**.

To use interval arithmetic into a computer, we should use floating point arithmetic and modify the interval in such a way that the real value we want to represent are in a computer representable floating point interval. To guarantee result exactness we must change the rounding mode to minus infinity in lower bound calculation and to plus infinity in higher bound calculation.

2.2. Subdivision algorithm

We subdivide the space in eight equal sized parts in a recursive way, as in an octree construction **[Stolte 95a]**. Each subdivided octant is represented by three intervals, one for each variable (x,y,z), where the lower and higher bounds correspond to the octant bounding coordinates. The result of applying these intervals in the inclusion function (the function in interval arithmetic) is an interval. If the interval lower bound is greater than zero then the octant is totally outside the surface. If the interval higher bound is less than zero then the octant is totally inside the surface. In both cases the octant is rejected. Otherwise the octant might intersect the surface and deserves being further subdivided. The algorithm below applied in a recursive way summarizes the method:

If zero is contained into the interval calculated by applying the inclusion function to the octant

Then subdivide octant
Else reject octant

The basic difference with Duff's subdivision method is that we subdivide object space and not the viewing volume. This way the subdivision is independent of the view point and doesn't need to be repeated each time the observer changes the position or orientation. Our method is also independent from the rendering algorithm. The same discrete model could be also visualized using discrete ray-tracing **[Stolte 95b]**.

3. Set-theoretic operations

3.1. Ricci's method

Ricci **[Ricci 96]** has proposed the following implementation of set-theoretic operations (considering that $f_1(x,y,z)$ and $f_2(x,y,z)$ are two implicit functions):

$$f_1 \cup f_2 = \min(f_1, f_2)$$
$$f_1 \cap f_2 = \max(f_1, f_2)$$

Subtraction operation can be defined as:

$$f_1 \setminus f_2 = f_1 \cap -f_2$$

These operators have C^1 discontinuity when $f_1 = f_2$ **[Sourin96]**. If this discontinuity is not desired, it can be eliminated using blending functions.

3.2. Blending functions

Set-theoretic operations can be defined using blending functions to obtain soft transitions and function continuity when $f_1 = f_2$. The exemples below use the gaussian as blending function:

$$f_1 \cup f_2 = -e^{-(f_1(x,y,z))^2} - e^{-(f_2(x,y,z))^2} + 1$$
$$f_1 \cap f_2 = e^{-(f_1(x,y,z))^2} + e^{-(f_2(x,y,z))^2} - 1$$
$$f_1 \setminus f_2 = -e^{-(f_1(x,y,z))^2} + e^{-(f_2(x,y,z))^2} + 1$$

Images 1, 2, 3 and 4 have been produced using these operators.

4. Inclusion Functions for Ricci's Set-theoretic operators

Union and intersection operators defined by Ricci **[Ricci 73]** can be defined the following way in interval arithmetics ($\boldsymbol{F_1}=[f_1, F_1]$, $\boldsymbol{F_2}=[f_2, F_2]$) :

$$\boldsymbol{F_1} \cup \boldsymbol{F_2}: \quad \begin{cases} [f_1, F_1] & \text{if } F_1 \le f_2 \\ [f_2, F_2] & \text{if } F_2 \le f_1 \\ [\min(f_1,f_2), \max(F_1,F_2)] & \text{otherwise} \end{cases}$$

$$\boldsymbol{F_1} \cap \boldsymbol{F_2}: \quad \begin{cases} [f_1, F_1] & \text{if } F_2 \le f_1 \\ [f_2, F_2] & \text{if } F_1 \le f_2 \\ [\min(f_1,f_2), \max(F_1,F_2)] & \text{, otherwise} \end{cases}$$

Subtraction can be obtained with:

$$\boldsymbol{F_1} \setminus \boldsymbol{F_2} = \boldsymbol{F_1} \cap -\boldsymbol{F_2}$$

Where $-\boldsymbol{F_2} = [-F_2, -f_2,]$

Images 5, 6 ,7 and 8 were obtained with these operators.

5. Visualisation Method

The surfaces are subdivided at a high discrete resolution, where each voxel has its color and a normal vector calculated through the gradient of the function in the middle of the voxel. The normal vector is sensitive to singular points but not the rasterization. To avoid the high memory consumption we store the voxels into an octree **[Stolte 95a, 95b,96]**. This allows to achieve high resolutions with low memory consumption, since the majority of the scenes are almost empty. Once the surface is subdivided we don't need surface's equation neither the need to convert the voxels into polygons to visualize it. We have used hardwired Z-Buffer but discrete Ray Tracing **[Stolte 95b]** could also be used. Here each voxel is displayed as a 3D point with the voxel normal and color. This way the scene can be visualized interactively with high quality shading. Other data can be stored into the voxels as shadows information. Shadows can be precalculated casting discrete rays to the light sources. Radiosity data can also be precalculated and stored into the voxels. The difference in our approach is that we use high resolution to increase image quality and we're able to visualize the scene in near to real time. Notice that we don't need any special architecture, neither a parallel approach. We use conventional machines with hardwired Z-Buffer capable of showing 3D points with normals. Even though we still cannot allow near close-ups this can be solved by using lazy evaluation and storing the extra space in virtual memory. Smooth continuity in the movement can be guaranteed using an efficient cache system. However a powerful machine is advisable to be able to rasterize pieces of the surface in real time.

6. Conclusion

This article presents a new method for creating complex discrete models using combinations of Implicit Surfaces given by set-theoretic operations. Interval arithmetics is used to convert the continuous model to a high resolution discrete space. This high resolution discrete space is stored into an octree to save memory and to accelerate surface display. Since one voxel is sufficiently small it is considered as a 3D point into the Z-buffer. Shading is calculated into the hardwired Z-buffer by using voxel's normal vector previously calculated during subdivision through the gradient of the function in the middle of the voxel. This produces high quality interactive visualisation which allows quick inspection of the modelled surface. Images approach ray-casting quality. Shadows are omitted but can be pre-calculated after the surface

has been subdivided.

Set-theoretic operations are directly integrated into the surface definition. Therefore the whole model can be considered as an Implicit Surface, which allows modelling scenes in a symbolic way. This introduces a higher abstraction programming level and a formal tool for models designing. Models are not restricted to curved surfaces. Even polygonal surfaces can be specified by Implicit Surfaces and set-theoretic operators as indicated in images 5, 6, 7 and 8. The set-theoretic operations inclusion functions defined here allow a high performance convertion of the model into voxels. This is very important to obtain high resolution discrete models in reasonable time.

References

[Bidasaria 92] H.B. BIDASARIA "Defining and Rendering of Textured Objects through the Use of Exponential Functions" Graphical Models and Image Processing, 54 (2): 97-102, March 1992

[Blinn 82] James BLINN "A Generalization of Algebraic Surface Drawing" ACM Transaction on Graphics, 1 (3) : 235-256, July 1982

[BW 90] Jules BLOOMENTHAL & Brian WYVILL "Interactive techniques for implicit modelling" Computer Graphics 24 (2) : 109-116, March 1990

[Duff 92] Tom DUFF "Interval Arithmetic and Recursive Subdivision for Implicit surfaces and Constructive Solid Geometry", Siggraph 92, 26 (2) : 131-138, July 1992

[Moore 66] MOORE R.E. "Interval Analysis", Prentice Hall, Englewood Cliff s, New Jersey, 1966.

[Moore 79] MOORE R.E. "Methods and Applications of Interval Analysis", SIAM, Philadelphia.

[Muraki 91] Shigeru MURAKI "Volumetric Shape Description of Range Data using 'Blobby Model' " Computer Graphics, 25 (4) : 227-235, July 1991

[Pasko 96] A. Pasko, V. Adzhiev, A. Sourin and V. Savchenko, "Function Representation in Geometric Modelling: Concepts, Implementation and Applications", The Visual Computer, 1996

[Ricci 73] A. Ricci, "A constructive Geometry for Computer Graphics" , The Computer Journal 16, 157-160, 1973

[Stolte 95a] Nilo STOLTE & René CAUBET "Comparison between different Rasterization Methods for Implicit Surfaces" Visualization and Modelling, pages 61-70, Leeds, December 1995, University of Leeds

[Stolte 95b] Nilo STOLTE & René CAUBET "Fast High Definition Ray Tracing Implicit Surfaces" 5th DGCI- Discrete Geometry for Computer Imagery, pages 61-70, Clermont-Ferrand, September 1995. Université d'Auvergne - Clermont 1.

[Stolte 96] Nilo STOLTE, "Espaces discrets de hautes résolutions : une nouvelle approche pour la modélisation et la synthèse d'images réalistes", PhD Thesis, Paul Sabatier University, Toulouse, France, 1996.

[Sederberg 90] Thomas W. SEDERBERG "Techniques for Cubic Algebraic Surfaces" IEEE-CGA, pages 14-25, July 1990

[Snyder 92] John M. Snyder "Interval Analysis For Computer Graphics", Siggraph 92, 26(2) : 121-130, 1992

[Sourin 96] A. Sourin, A.Pasko and V. Savchenko "Using Real Functions with application to Hair Modelling ", Computer & Graphics, 20 (1): 11-19, 1996

[Wyvill 86] Geoff WYVILL, Craig McPHEETERS, and Brian WYVILL "Data structure for soft objects" The Visual Computer, pages 227-234, August 1986