Comparison between Different Rasterization Methods for Implicit Surfaces

Nilo Stolte and René Caubet

Institut de Recherche en Informatique de Toulouse
118, Route de Narbonne
31062 – Toulouse – France
stolte@irit.fr

Abstract

Rasterizing implicit surfaces has been an important research problem, since it serves as a base to modelize and visualize these kind of functions. Hence, rasterizing algorithms are useful in many scientific domains: mathematical visualization, medical visualization (modelling tumors, organs, prosthesis, etc.), physical simulations, volume visualization, modelling, Ray-Tracing, Discrete Ray-Tracing, etc.

In this paper we generelise two algorithms previous known as implicit surfaces "subdivision methods" to apply them to the rasterization of these surfaces into a 3D discrete space. We also propose the interactive visualisation of these surfaces directly into the voxel format avoiding convertions to other representations. We finally compare the different rasterization methods in terms of performance, quality, robustness and generality, trying to locate possible pitfalls among them. This comparison is very useful to whom is wanting to implement such rasterization methods. Nevertheless it was virtually impossible until now to estimate which method was the best, since to our knowledge no comparisons between them were ever done. The comparative results we present in this paper will allow much better estimations.

Key Words: Implicit Surfaces, 3D Rasterization, Subdivision, Octree, Voxel, Visualization.

1 Introduction

Implicit surfaces can be defined by functions of the kind:

$$F(x,y,z)=0$$

An interesting property of this kind of function is its ability to determine if a point is inside (when $F(x,y,z)<0$), outside (when $F(x,y,z)>0$), or on (when $F(x,y,z)=0$) the surface. The sphere ($x^2+y^2+z^2-r^2=0$, where r is the ray) and the plane ($ax+by+cz+d=0$, where a, b, and c are the normal vector components and d is an arbitrary constant) equations are the simplest examples of implicit functions. All implicit functions that can be completely defined analytically are called *analytical implicit functions.* These are the functions that are considered in this article. There are still other kinds of implicit surfaces that cannot be expressed analytically, which are called *procedural implicit functions*, since they are defined procedurally [Bloomenthal and Wyvill, 1990]. This kind of surface must count on other representations to estimate the normal vector, since they don't allow derivative calculations [Norton, 1982]. Conversely in analytic implicit functions the normal vector can be calculated by deriving the function equation in relationship to each axis, applying the point coordinates to the derivatives expressions and normalizing the vector. In other words, each component of the normal vector firstly has the same components as the gradient in the point and is normalized afterwards.

The analytic implicit functions can be subdivided in two main groups: algebraic implicit functions and non algebraic implicit functions. The algebraic implicit functions [Sederberg, 1990] can be reduced to polynomials by algebraic manipulation, that is, it contains only arithmetic operations and integer powers.

Non algebraic surfaces cannot be reduced to polynomials. One example particularly useful is the "blobby model" [Blinn, 1982, Muraki, 1991, Bidasaria, 1992] where negative algebraic expressions are placed as exponents of exponential functions. These exponential functions are added (and/or subtracted) together and made equal to a constant (C, below):

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

When the objects are considerably near one to the other, the resulting surface is a fusion between the several algebraic surfaces in the exponents. This fusion is controlled by the parameters a_i and b_i which change the exponential form. The exponential is used as a *"blending function"* (Fig. 1) which gives the amount of mixing in relationship to the distance from functions' ($f_i(x,y,z)$) origins. The main interest of this kind of surfaces is the modelling and animation facilities. Other advantages are the exact simulation of certain physic phenomena: electron clouds, molecules, isopotential fields, etc.

Similar effects can be obtained with algebraic surfaces. Addition between two exponential functions in the "blobby model" corresponds to multiplying

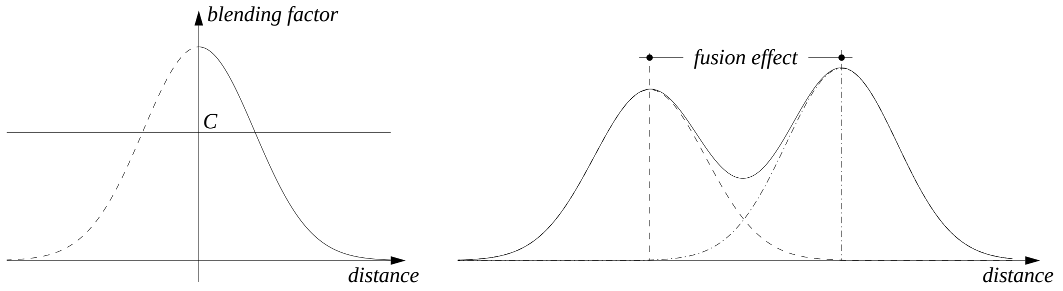

Figure 1: *"Blobby model" blending function and its effects*

the two algebraic functions that were in the exponents of the exponentials in the "blobby model" [Sederberg, 1990]. The disadvantage of using this kind of algebraic surfaces is that the modelling and animation facilities are reduced. Nevertheless polynomial blending functions [Desbrun and Gascuel, 1995, Bittar *et al.*, 1995] (Fig. 2) allowed even more flexibility in animation and modelling than exponential blending functions. If the functions to be blended are algebraic and polynomial blending functions are used, the obtained implicit function is algebraic. However this kind of algebraic surfaces is very flexible and even simpler to manipulate than "blobby models". Nevertheless blending non algebraic functions allows a wider variety of forms, but the resulting surface is non algebraic.

Algebraic surfaces are more easily rendered than non algebraic surfaces. Ray Tracing algebraic surfaces of arbitrary order is straightforward using Collins theorem [Hanrahan, 1983]. Numerical techniques for ray tracing non algebraic surfaces are generally instable. An elegant method to ray trace this kind of surface was proposed by Kalra and Barr [Kalra and Barr, 1989]. They calculate Lipschitz constants to subdivide the surface and to ray trace it. The method always converges and works for algebraic and non algebraic surfaces.

Duff [Duff, 1992] proposed another method to ray trace CSG trees of algebraic surfaces but the subdivision method works also for non algebraic implicit surfaces. The basic principle is the same as in [Kalra and Barr, 1989]: subdivide the surface until a certain level and ray trace the surfaces contained in the sub-regions crossed over by a ray. To subdivide the implicit functions he uses interval arithmetic. In fact Kalra and Barr's method "is a sort of interval arithmetic without intervals" [Duff, 1992].

Taubin [Taubin, 1994a, Taubin, 1994b] presented a method to rasterize implicit algebraic curves which can also rasterize implicit algebraic surfaces [Taubin, 1994b]. Although showing the surfaces directly in the "voxel" format he suggests converting the voxels into polygons using a technique known as "marching cubes" [Lorensen and Cline, 1987] and afterwards use conventional methods to render the polygons. We propose the direct visualization of the voxel volume by keeping the normal vector in the middle of the voxels and using a high subdivision level. To avoid the high memory consumption we store the voxels into an octree. This approach allows us to render the surfaces by using our fast discrete ray tracing

[Stolte and Caubet, 1995a, Stolte and Caubet, 1995d, Stolte and Caubet, 1995b, Stolte and Caubet, 1995c], or a Z-Buffer algorithm considering each voxel a point of the surface. This later visualization method enhances significantly the interactivity with no loose in image quality, but with less realism as ray tracing generated images. The image quality is generally better than using polygons. In the case when all projected polygons are smaller than a pixel the obtained quality is equivalent. Nevertheless we still have the advantage that it does not require polygonization.

Bloomenthal and Wyvil [Bloomenthal and Wyvill, 1990] have presented several techniques for modelling implicit surfaces. We don't claim interactive modelling in our method but an acceptable level of interactivity in the visualization process. Nevertheless a good interactive modelling level can be obtained limiting the voxel space resolution which allows faster prototyping as proposed in [Bloomenthal and Wyvill, 1990]. Bloomenthal and Wyvil [Bloomenthal and Wyvill, 1990] recommend octree display, as proposed here, for a coarse representation of the surface or for volume rendering hardware. Nevertheless the advances in graphics hardware and the memory lowering prices in these later years are drastically changing this situation. Even advances in software have been also reversing this situation. An example is the adaptative subdivision method proposed by Duff [Duff, 1992]. Instead of calculating curvatures as observed in [Bloomenthal and Wyvill, 1990] this subdivision method uses simple interval arithmetic, which is computationally inexpensive in most of today's machines.

On the other hand near to real time interactive modelling could be achieved using our method with special blending functions. "Blobby" models use exponentials as blending functions (Fig. 1) with negative exponents. An inconvenient of using these blending functions is that their values are never zero. This implies that a full function evaluation is needed to calculate its value. If a huge amount of functions is considered, like in biological molecular models, its rasterization time is very time consuming. In these cases most of the times approximations have to be done [Blinn, 1982, Fujimoto *et al.*, 1986], which are not often desired. To avoid these problems other kinds of blending functions can be used [Desbrun and Gascuel, 1995, Bittar *et al.*, 1995]. We can design blending functions that go to zero for a relatively short distance. We show in Fig. 2 a polynomial blending function given by a Bezier curve. The curve appearance is similar to exponentials but with the advantage of having local influence. Only distances between zero and r need to be considered. In this case rasterizations of huge biological molecular models are feasible in quite low time. Even interactive modelling can be envisaged using this approach assuming that most changes are local and can be rasterized in almost real time. The octree display time can be significantly enhanced by storing the voxels into a linear octree. Since the implicit surface subdivision methods subdivide the space in the same order an octree is traversed, the generated voxels are already sorted into the linear octree. In the linear octree the coordinates of each voxel should be explicitly stored. To save space, memory could be allocated in eight voxels groups, each group representing one octant as done in many linear octree implementations [Glassner, 1984, Sung, 1991]. The linear octree would be a chained list of octants. Each octant would contain a pointer to the next octant, the coordinates of the octant (the last voxel coordinate bits are given by voxels' positions into the octant as in standard octrees), and eight pointers each one pointing to the corresponding voxel (or a null pointer if the corresponding voxel is empty). To display the octree the chained list would be linearly traversed and all the voxels would be very efficiently obtained. The voxel search in this octree can be done with the help of a hash table, as done in many linear octree implementations [Glassner, 1984, Sung, 1991].

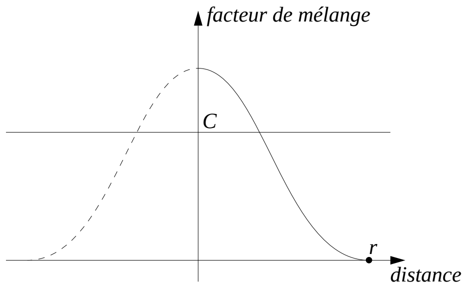

Figure 2: *polynomial blending function*

The voxel based visualization methods have been neglected until recently. Nevertheless its importance is remarkable. Mathematicians could finally analyze implicit functions interactively. Implicit surfaces could be easily ported to medical imagery to simulate tumors, organs, prosthesis, etc. The research of new forms using implicit surfaces could count on well known mathematic expressions where the derivatives are easily calculated, and many other mathematic properties can be easily derived from its equation. Physical simulations using implicit surfaces could be easily obtained. All these applications are very difficult to get using procedural implicit surfaces. Since they have no equation they need to be converted into polygons [Bloomenthal and Wyvill, 1990] or voxels [Norton, 1982] in order to estimate the normal vectors, for example.

These applications are then ready to profit from the benefits of voxel representation of implicit surfaces. Consequently the performance of implicit surfaces rasterization algorithms is very important to the increase of these benefits. The rasterization quality is also very important, since a better rasterization implies a better representation of the surface. We propose in this article the comparison between different rasterisation methods for the two most used implicit surfaces: algebraic implicit functions and exponential implicit functions.

2 Implicit Surfaces Rasterizing Methods

2.1 Kalra and Barr's Subdivision Method

The subdivision method proposed by Kalra and Barr [Kalra and Barr, 1989] can be used to rasterize implicit surfaces. It consists in a recursive subdivision where certain regions in space are divided into eight subspaces, called octants. This is the same construction logic used to build an octree. In fact we do use an octree in our discrete ray-tracing system [Stolte and Caubet, 1995a, Stolte and Caubet, 1995d, Stolte and Caubet, 1995b, Stolte and Caubet, 1995c] and our interactive visualization software. Nevertheless direct memory allocation for the octree using this method is not recommended, since the method only guarantees discarding octants where the surface doesn't pass through but doesn't guarantees if there is really a piece of surface inside an octant. We subdivide the surface until a maximum level and allocate the memory only when the subdivision arrives to this level. Using a linear octree as commented previously would enhance the performance of this process, since no intermediate octree levels exist and several optimisations can be done.

The octant rejection condition is when the norm of the maximization of the partial derivatives over the octant multiplied by the half of the octant diagonal is less than the absolute value of the function value at the middle of the octant. The influence of this test can be seen as a sphere centered in the middle of the octant with ray greater than octant's diagonal. Therefore it rejects the octants where the surface has no intersection with this sphere and when the surface is not totally inside it. Since this sphere is greater than the octant, the test can catch pieces of the surface that pass close to the octant, but does not necessarily pierce the octant. Hence the "tightness" of the rasterization depends directly on the partial derivatives maximization. This means that if this maximization is too overestimated the rasterization will take more time, since more subdivisions are necessary to correctly rasterize it, or if we limit the number of levels the quality of the rasterization is going to be poor. On the other side, underestimating the maximization can be translated as errors in the rasterization. To accelerate the process, Kalra and Barr proposed to test in advance whether there is variation in sign in the function values at the eight octant vertexes. This would clearly indicate an intersection between the surface and the octant, since negative function values indicate that the vertex are inside the surface and positive function values indicate that the vertex is outside the surface.

2.2 Duff's Subdivision Method

Duff [Duff, 1992] has proposed another method to rasterize algebraic and non algebraic implicit functions. He uses interval arithmetic for calculating function values. Snyder [Snyder, 1992] has extended the idea to other Computer Graphics problems. Interval arithmetic generalizes traditional arithmetic guaranteeing result exactness inside an interval. A certain value in interval arithmetic is given by two values, the lower and the higher bounds of the interval that contains the real value. All arithmetic operations are then redefined to work in this interval giving as result another interval defined by the resulting lower and higher bounds. 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. Interval arithmetic can be generalized to other mathematical operations as integer powers and transcendental functions.

The rasterization is done by subdividing the space in an octree-like way as seen in the precedent method. 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 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. We can notice at this point that this method clearly has a much more efficient octant elimination heuristic. In the precedent method if we calculate the value of the function in the eight vertex, and if all the values are negative or positive, we cannot eliminate the octant. With the interval arithmetic we can. Hence we can expect that this method is faster than the precedent. Nevertheless this must be verified experimentally.

2.2.1 Interval Arithmetic

Duff and Snyder [Duff, 1992, Snyder, 1992] have simultaneously but independently introduced interval arithmetic to solve Computer Graphics problems. Duff concentrated in 3D implicit functions 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:

$$\begin{aligned}\mathbf{X} &= [\mathsf{x}, \mathsf{X}]\\ \mathbf{Y} &= [\mathsf{y}, \mathsf{Y}]\\ \mathbf{Z} &= [\mathsf{z}, \mathsf{Z}]\end{aligned}$$

These interval bounds in our case are the coordinates of the octant's boundaries. Substituting in the implicit function equation each regular variable by the correspondent interval and each regular operation by the respective interval operation, produces an interval version of the function, which Snyder [Snyder, 1992] 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:

$$\begin{aligned}
\mathbf{X} + \mathbf{Y} &= [\mathsf{x} + \mathsf{y}, \mathsf{X} + \mathsf{Y}] \\
\mathbf{X} - \mathbf{Y} &= [\mathsf{x} - \mathsf{Y}, \mathsf{X} - \mathsf{y}] \\
\mathbf{X} \cdot \mathbf{Y} &= [\min(\mathsf{xy}, \mathsf{xY}, \mathsf{Xy}, \mathsf{XY}), \max(\mathsf{xy}, \mathsf{xY}, \mathsf{Xy}, \mathsf{XY})] \\
\mathbf{X} \;/\; \mathbf{Y} &= [\mathsf{x}/\mathsf{Y}, \mathsf{X}/\mathsf{y}] \text{ if } 0 \notin [\mathsf{y}, \mathsf{Y}]
\end{aligned}$$

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

$$\mathbf{X^n} = \begin{cases} [\mathsf{x}^n, \mathsf{X}^n] \text{ n odd or } \mathsf{x} >= 0 \\ [\mathsf{X}^n, \mathsf{x}^n] \text{ n even and } \mathsf{X} <= 0 \\ [0, \max(-\mathsf{x}, \mathsf{X})^n] \text{ n even and } 0 \in [\mathsf{x}, \mathsf{X}] \end{cases}$$

To include exponential functions with negative exponents, which are useful as blending functions in “blobby” models, we have:

$$e^{-\mathbf{X}} = [e^{-\mathsf{X}}, e^{-\mathsf{x}}]$$

Any other function can be converted to interval arithmetic similarly by breaking the function in their monotonic intervals [Duff, 1992].

2.3 Taubin's Rasterizing Method

Taubin [Taubin, 1994b] has proposed another rasterization method that works for algebraic implicit surfaces only. It also uses space subdivision in octree-like fashion as the precedent methods. The method first translates the origin to the middle point of the octant. Afterwards the algebraic implicit function is simplified and converted to a single variable polynomial. If the evaluation of this new function for the half octant size gives a positive value, then the octant doesn't intersect the surface. The simplification starts by grouping all the terms of the same degree together. Then for each one of these groups the coefficients absolute values are added to form a new coefficient. The three variables are then substituted by just one variable, which is raised to the power correspondent to each group's degree. Finally, the coefficient which doesn't multiply any variable is subtracted from the rest of the polynomial. Notice that the entire polynomial evaluation is not necessary if a partial result is already negative or zero. We suggest then that for each polynomial term we test if the result is negative or zero before the next term is considered. Even with this optimization, we are not sure if this method is really more efficient than the precedent ones. So this must be verified experimentally. Nevertheless we have opted to not examine this algorithm based on the following observations:

- it rasterizes only algebraic functions;
- it recalculates for each octant the polynomial in Taylor series by using Horner's algorithm, which require the function in the polynomial form;
- it does not rasterize well near singular points;

3 Comparative Results

	image 1		*image 2*		*image 3*	
3D Res	*Time*	*Memory*	*Time*	*Memory*	*Time*	*Memory*
512^3	0'34"	6263	1'26"	5730	0'58"	6594
256^3	0'08"	2185	0'22"	2056	0'13"	2350
128^3	< 0'01"	1171	0'05"	1103	0'06	1252

Figure 3: Rasterization times for Kalra and Barr's method

	image 1		*image 2*		*image 3*	
3D Res	*Time*	*Memory*	*Time*	*Memory*	*Time*	*Memory*
512^3	0'21"	8165	0'50"	7802	0'25"	7330
256^3	0'05"	2661	0'09"	2574	0'05"	2456
128^3	<0'01"	1289	<0'02"	1254	<0'01"	1234

Figure 4: Rasterization times for Duff's method

Kalra and Barr's method is not very appropriated for rectangular regions since it rejects regions outside a sphere. This is equivalent to the spheric bounding volume problem. It is not very effective to long narrow objects. Duff's method is more suited to rectangular regions. Therefore it can be used to eliminate long narrow regions more effectively than Kalra and Barr's method.

We only used cubic octants which aids Kalra and Barr's method better than Duff's. Anyway Duff's method was always faster than Kalra and Barr's method. Nevertheless Duff's method always used more memory than Kalra and Barr's method. This indicates that Kalra and Barr's method is *"tighter"* than Duff's, at least for cubic regions. On the other hand Kalra and Barr's method is very sensitive to functions like the *heart* (image 3). In this case Kalra and Barr's method took practically twice more time to rasterize it. We can also observe that algebraic functions take less time to be rasterized than an almost equivalent form given by exponential functions in both methods (image 1 and 2). This was expected, since algebraic expressions are faster evaluated than exponential functions. We can also note how Kalra and Barr's method is sensitive to singular points. This sensitivity was noted also in the normal vector calculation near singular points in both methods. The fact that Kalra and Barr's method uses the partial derivatives can explain its sensitivity near singular points, since the normal vector is also calculated using the partial derivatives. At singular points the partial derivatives are zero, which can explain this behavior. Duff's method is insensitive to singular points.

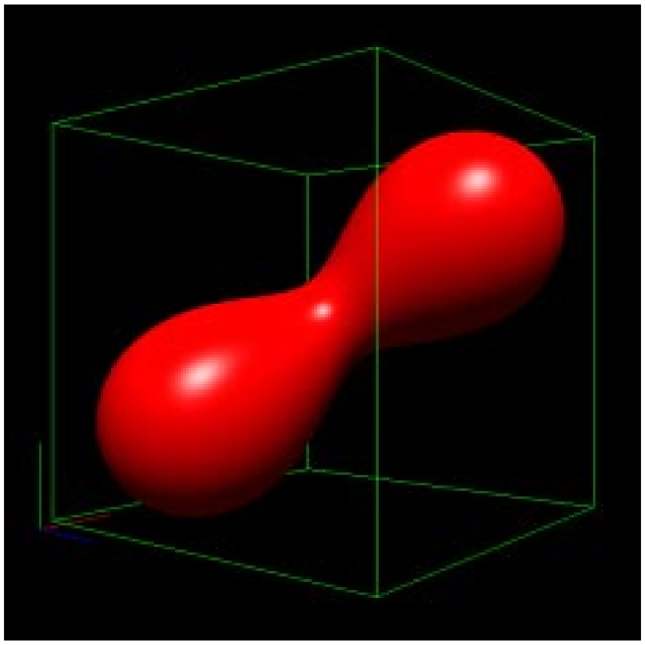

Figure 5: *Image1* $\rightarrow f(x,y,z) \cdot g(x,y,z) - 0.058 = 0$, *where*
$f(x,y,z) = (x-0.78)^2 + (y-0.78)^2 + (z-0.78)^2 + 0.001$
$g(x,y,z) = (x-0.23)^2 + (y-0.23)^2 + (z-0.23)^2 + 0.001$

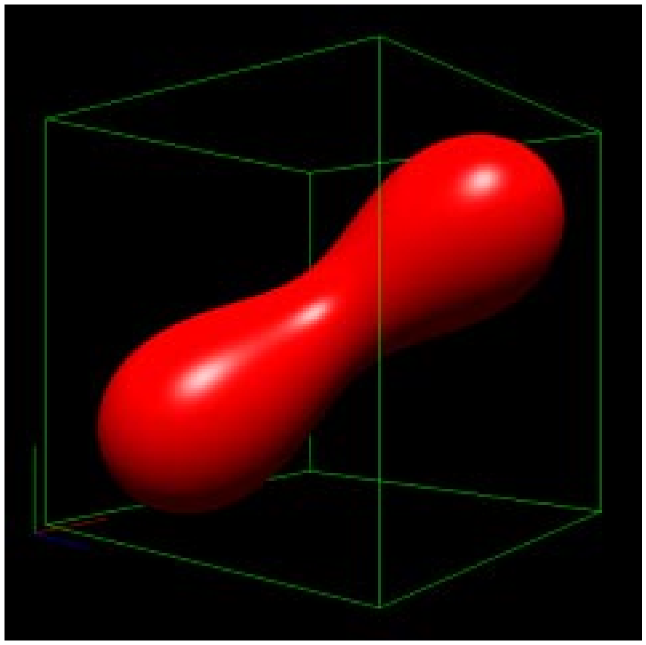

Figure 6: $Image2 \rightarrow e^{-((x-0.78)^2+(y-0.78)^2+(z-0.78)^2)\cdot 3.25)} + e^{-((x-0.23)^2+(y-0.23)^2+(z-0.23)^2\cdot 3.25)} - 0.9 = 0$

All rasterizations were generated into a Crimson SGI workstation with 100MHz R4000/R4010 processors. Memory occupation in figures 3 and 4 are given in 4 Kb blocks. All images were generated into the same machine using our interactive voxel visualization software. This software uses an octree to store the voxels and GL primitives to display each voxel as a 3D point.

4 Conclusion

The importance of implicit surfaces rasterizing methods has been recently emphasized. A number of applications can be devised. The use of implicit representation is a very convenient way to represent 3D objects, since it is very general. Planes, quadratic surfaces, parametric and more exotic surfaces can be expressed under the implicit form. Hence it is a very high level modelling tool. In addition it is very compact, allowing the representation of very complex scenes with less memory consumption.

We have proposed two rasterizing methods based on existing implicit surface subdviding techniques. Unfortunately, as far as we know, no comparison between these methods in terms of quality or performance was given. We have proposed the experimental comparison between these two methods. These results will be very useful to whom is desiring to implement such rasterization methods.

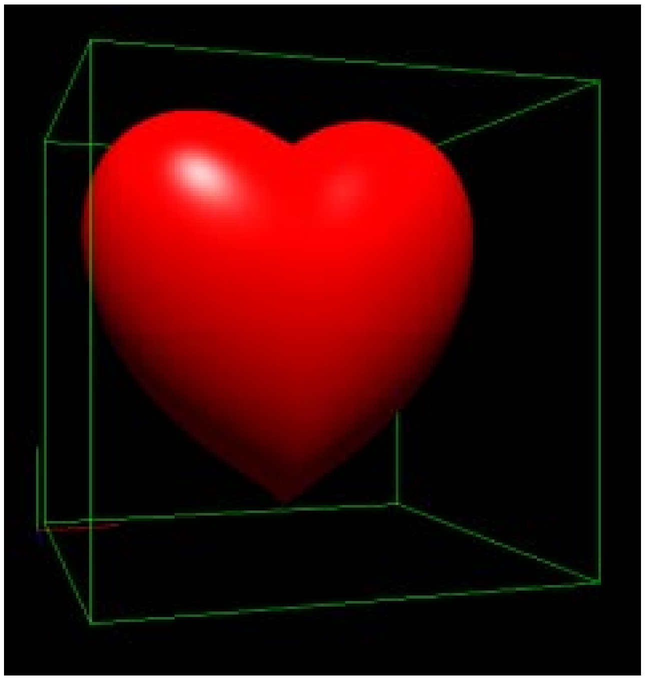

Figure 7: *Image3* $\rightarrow$ $(x^2+y^2+2z^2-1)^3-y^3(x^2+0.1\cdot z^2)=0$

All the three methods overestimate the size of the octant in a way or another. The research of better methods, in which this size is exact or is better approximated, is a constant worry in this research area. Finding out such methods will allow more precise and perhaps faster rasterizations.

We've proposed visualizing implicit functions directly in their voxel format with the help of an octree. It allows a near to real time interaction in quite high resolutions.

References

[Bidasaria, 1992] 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.

[Bittar *et al.*, 1995] Eric Bittar, Nicolas Tsingos, and Marie-Paule Gascuel. Automatic Reconstruction of Unstrutured 3D Data: Combining a Medial axis and Implicit Surfaces. In *Eurographics 95*, pages 457–468, Maastricht, August 1995. Blackwell.

[Blinn, 1982] James Blinn. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[Bloomenthal and Wyvill, 1990] Jules Bloomenthal and Brian Wyvill. Interactive Techniques for Implicit Modeling. *Computer Graphics*, 24(2):109–116, March 1990.

[Desbrun and Gascuel, 1995] Mathieu Desbrun and Marie-Paule Gascuel. Animating Soft Substances with Implicit surfaces . In *Siggraph 95*, pages 287–290, Los Angeles, August 1995. ACM press.

[Duff, 1992] Tom Duff. Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry. *Computer Graphics*, 26(2):131–138, July 1992.

[Fujimoto *et al.*, 1986] Akira Fujimoto, Takayaki Tanaka, and Kansei Iwata. ARTS: Accelerated Ray Tracing System. *IEEE - CGA*, 6(4):16–26, 1986.

[Glassner, 1984] Andrew S. Glassner. Space Subdivision for Fast Ray Tracing. *IEEE - CGA*, 10(4):15–22, 1984.

[Hanrahan, 1983] Pat Hanrahan. Ray Tracing Algebraic Surfaces. *Computer Graphics*, 17(3):83–90, July 1983.

[Kalra and Barr, 1989] Devendra Kalra and Alan Barr. Guaranteed Ray Intersections with Implicit Surfaces. *Computer Graphics*, 23(3):297–306, July 1989.

[Lorensen and Cline, 1987] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

[Muraki, 1991] Shigeru Muraki. Volumetric Shape Description of Range Data using “Blobby Model”. *Computer Graphics*, 25(4):227–235, July 1991.

[Norton, 1982] Alan Norton. Generation and Display of Geometric Fractals in 3-D. *Computer Graphics*, 16(3):61–67, July 1982.

[Sederberg, 1990] Thomas W. Sederberg. Techniques for Cubic Algebraic Surfaces. *IEEE - CGA*, 10(4):14–25, July 1990.

[Snyder, 1992] John M. Snyder. Interval Analysis For Computer Graphics. *Computer Graphics*, 26(2):121–130, July 1992.

[Stolte and Caubet, 1995a] Nilo Stolte and René Caubet. Discrete Ray-Tracing High Resolution 3D Grids. In *The Winter School of Computer Graphics and Visualization 95*, pages 300–312, Plzen, February 1995. Vaclav Skala.

[Stolte and Caubet, 1995b] Nilo Stolte and René Caubet. Discrete Ray-Tracing of Huge Voxel Spaces. In *Eurographics 95*, pages 383–394, Maastricht, August 1995. Blackwell.

[Stolte and Caubet, 1995c] Nilo Stolte and René Caubet. Fast High Definition Ray Tracing Implicit Surfaces. In *5th DGCI - Discrete Geometry for Computer Imagery*, pages 61–70, Clermont-Ferrand, September 1995. *Université d'Auvergne* - Clermont 1.

[Stolte and Caubet, 1995d] Nilo Stolte and René Caubet. Lancer de Rayons Discret pour des Grilles de Hautes Résolutions. In *Montpellier'95 - L'interface des Mondes Réels et Virtuels*, pages 335–344, Montpellier, June 1995. EC2 & Cie.

[Sung, 1991] Kelvin Sung. A DDA Traversal Algorithm for Ray Tracing. In *Eurographics'91*, pages 73–85, Amsterdam, June 1991. North Holand.

[Taubin, 1994a] Gabriel Taubin. Distance Aproximation for Rasterizing Implicit Curves. *ACM Transactions on Graphics*, 13(1):3–42, January 1994.

[Taubin, 1994b] Gabriel Taubin. Rasterizing Algebraic Curves and Surfaces. *IEEE - CGA*, pages 14–23, Mars 1994.