ROBUST HIERARCHICAL VOXEL MODELS FOR REPRESENTATION AND INTERACTIVE VISUALIZATION OF IMPLICIT SURFACES IN SPHERICAL COORDINATES

Nilo Stolte† and Arie Kaufman‡
{stolte|ari}@cs.sunysb.edu

†School of Computer Science & Eletronic Systems
Kingston University
Penrhyn Road, Kingston upon Thames
Surrey KT1 2EE England

‡Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400 U.S.A.

ABSTRACT

In this article we present a new method of voxelizing implicit surfaces in spherical coordinates. The approach has several advantages: (1) The voxelization algorithm is robust and has low complexity; (2) allowing high resolution voxelizations, including normal calculations, in reasonable time and (3) its visualization using GL points with normals generates Ray-Casting quality images at interactive speeds.

The approach is robust because no part of the surface will be missed in the voxel model and because no floating point errors will occur. The robustness emerges from the use of interval arithmetic. Voxelization is done by subdividing the rectangular space in a recursive way until the voxel level is reached. The rectangular intervals are then converted to spherical intervals and interval arithmetic assists in identifying whether the surface does not cross a subdivided space at a very early stage. This algorithm has a dual advantage: voxelization is guaranteed to envelop the true surface, and its experimental running time increases only by a factor of 4, instead of 8, every time resolution is doubled for every coordinate axis.

The resulting voxelization is stored in an octree defining a new concept called here *robust hierarchical voxel model* which guarantees that every level of the tree defines a set of voxels which always envelop the continuous surface. Each voxel stores the normal vector, calculated by the gradient of the function during the voxelization.

The visualization method used to render the hierarchical voxel model in this article uses GL/OpenGL point primitives and no special hardware is required besides the Z-Buffer.

Keywords: Implicit Surfaces, Voxelization, Spherical Coordinates, Cylindrical Coordinates.

1. INTRODUCTION

Because of axial symmetries, surfaces of CAD objects (and others) are sometimes defined in spherical or cylindrical coordinates. In this paper, we present a method for converting such a representation to a rectangular-voxel representation of the surface.

Given the spherical coordinates: r, θ and ϕ (or the cylindrical coordinates: r and θ); the focus of attention in this paper are the surfaces defined implicitly in the form $\mathcal{F}(r, \theta, \phi) = 0$ or $\mathcal{F}(r, \theta) = 0$, in the cylindrical case. For simplicity, however, the examples shown in this text are in the form $\mathcal{F}(\theta, \phi) - r = 0$ or $\mathcal{F}(\theta) - r = 0$, in the cylindrical case.

One direct application of these techniques involves blending of spherical or cylindrical surfaces with rectangular implicit surfaces in the same voxel space, thus profiting from easy blending with other implicit forms and convenient modeling using spherical/cylindrical coordinates.

Voxels are very widely used in accelerating ray-tracing and radiosity. In this domain, voxelization algorithms that can guarantee that the voxels completely envelop the surface (*i.e.* that no part of the surface will ever be missed), such as the method proposed in this paper, can be considerably beneficial.

The method uses a spatial recursive subdivision and tests the potential spatial occupation of the sur-

face in each subdivided rectangular region by using interval arithmetic. If a certain region does not contain a part of the surface it is not further subdivided. Otherwise the subdivision goes on until the desired resolution is found and the result is stored into an octree. A similar effect can be obtained using Lipschitz conditions instead of interval arithmetic [5, 1]. Taubin [10] also proposed a method with a comparable effect, but restricted to polynomial implicit surfaces. Octrees have also been used for storage purpose in [5], but as an auxiliary data structure for accelerating ray-tracing. Effectively, Lipschitz conditions were used for generating voxels and compared with the interval arithmetic approach in [9]. The result of this comparison was in favor of interval arithmetic as far as performance was concerned. This was the main reason interval arithmetic was adopted in our method. Besides always guaranteeing to envelop the surface (also true when using Lipschitz constants or Taubin's method), the use of interval arithmetic has the additional advantage of avoiding the rounding errors, provided that the proper rounding modes are used [4].

In contrast, some voxelization methods [3] simply evaluate the function on eight corners of every voxel. If there is no sign variation in the eight calculated values, the voxel is considered empty; otherwise it is considered full. This method can miss voxels having no corner intersection yet containing the surface or part of it. Also, its running time increases by a factor of 8, every time three-dimensional resolution is doubled in each axis. Unfortunately these two features are contradictory: the effect of missing parts of the surfaces is reduced in high resolution grids, but high resolution voxelizations using this method are very computationally expensive.

This article presents a novel method for voxelizing spherical/cylindrical implicit surfaces simultaneously offering robustness and low complexity. Robustness emerges with the use of interval arithmetic, while the low complexity results from recursive subdivision of the space associated with interval arithmetic (see Section 2).

The voxelization is done, as pointed out before, by subdividing the rectangular space in a recursive way, producing eight equal sized cubes at each interaction. Each of these cubes represents three intervals in interval arithmetic (see section 2), which are converted to spherical intervals (see sections 2.1 and 3) and then applied to the implicit spherical inclusion function for a containment test. This algorithm is shown to display running times asymptotically approaching an increasing factor of 4 every time resolution is doubled in every coordinate axis (see section 4). This provides a considerable advantage over algorithms which exhibit running times increasing by a factor of 8, every time resolution is doubled in every coordinate axis. Quite high resolutions voxelizations (including normal calculations in each voxel) can be obtained in reasonable time.

Hierarchical voxel models allow the definition and the storage of high-resolution voxel spaces, that is an alternative solution for displaying curved surfaces and other complex objects [8, 9, 6]. In this article, voxels are approximated by a point because they are sufficiently small and seen from a reasonable distance, thus being displayed at most by one pixel. The simplicity and the quality gained are the main advantages of this concept. Images rendered this way with standard hardwired Z-buffer, such as those shown in this article, exhibit ray-casting quality at interactive speeds. This shows that hierarchical voxel models are also appropriate for direct visualization.

2. INTERVAL ARITHMETIC

Snyder [7] and Duff [2] have simultaneously introduced Interval Arithmetic to solve problems in Computer Graphics.

Duff [2] has proposed using interval arithmetic to subdivide implicit functions in screen space, in order to ray-cast this kind of surface. The main drawback of this technique is its point of view dependence, since the subdivision is part of the visualization process.

In [9], Duff's method has been generalized to subdivide implicit functions in object space for voxelization purposes. This voxelization method is the basis for the voxelization method described in this article. This new approach isolates the subdivision in a preprocessing stage, storing the resulting voxels into a model, namely the *robust hierarchical voxel model*. This model can then be displayed using GL points with normals. This method allows interactive visualization with image quality similar to Ray-Casting (see Fig. 7 and Fig. 6).

Interval arithmetic guarantees that the exact result of any arithmetic operation is between two values, called *interval bounds*. Any real interval is represented by two interval bounds. For example, the coordinates x, y, and z are represented in interval arithmetic by, respectively:

$$\begin{aligned} \mathbf{X} &= [x_0, x_1] \\ \mathbf{Y} &= [y_0, y_1] \\ \mathbf{Z} &= [z_0, z_1] \end{aligned}$$

The interval bounds in our case are the coordinates of the boundaries of an arbitrary cubic region in space. Substituting in the implicit function equation each regular variable by the corresponding interval and each regular operation by the respective interval operation, produces an interval version of the function, which Snyder [7] calls an *inclusion function*. We can verify if the surface does not pass through the cubic region by simply testing whether the resulting

interval does not *include* zero, that is, when the inclusion function resulting interval does not *include* a solution for the regular function $F(x, y, z) = 0$. Then, if the resulting interval does not include zero, the function certainly does not have a zero in the cubic region, and hence the surface does not pass through the cubic region.

The interval arithmetic operators are:

$$\begin{aligned}
\mathbf{X} + \mathbf{Y} &= [x_0 + y_0, x_1 + y_0] \\
\mathbf{X} - \mathbf{Y} &= [x_0 - y_0, x_1 - y_0] \\
\mathbf{X} \cdot \mathbf{Y} &= [\min(x_0y_0, x_0y_1, x_1y_0, x_1y_1), \\
&\quad \max(x_0y_0, x_0y_1, x_1y_0, x_1y_1)] \\
\mathbf{X} \,/\, \mathbf{Y} &= [x_0/y_1, x_1/y_0] \text{ if } y_0 > 0
\end{aligned}$$

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

$$\mathbf{X^n} = \begin{cases} [x_0^n, x_1^n] \text{ n odd or } x_0 >= 0 \\ [x_1^n, x_0^n] \text{ n even and } x_1 <= 0 \\ [0, \max(-x_0, x_1)^n] \text{ n even and } 0 \in [x_0, x_1] \end{cases}$$

Any other function can be similarly converted to interval arithmetic by breaking the domain of the function to its monotonic intervals [2]. The sine function, for example, can be defined this way [7].

To implement intervals with floating point arithmetic in a computer we should modify the interval in such a way that the real value we want to represent is in a computer-representable floating point interval. To guarantee result exactness we must be sure to round to $-\infty$ in lower bound calculation, and to $+\infty$ in upper bound calculation [4].

2.1. Spherical Intervals

The same way as with rectangular intervals, spherical intervals are interval versions of spherical coordinates. Thus, three intervals are defined, one for each spherical coordinate: r, θ, and ϕ. In spherical coordinates, r is the distance between a certain point and the surface origin. Also, by definition, θ is the angle between the projection of the radius on the XZ plane and the X axis, and ϕ is the angle between the radius and XZ plane.

The three spherical intervals are defined as follows:

$$\begin{aligned}
\mathbf{R} &= [r_0, r_1] \\
\Theta &= [\theta_0, \theta_1] \\
\Phi &= [\phi_0, \phi_1]
\end{aligned}$$

The region defined by these intervals is not cubic, but has the form shown in Fig. 1. These spherical intervals can be inserted in the inclusion function of the spherically-described implicit function. If the resulting interval does not include zero, the region defined by the spherical intervals does not contain any part of the surface.

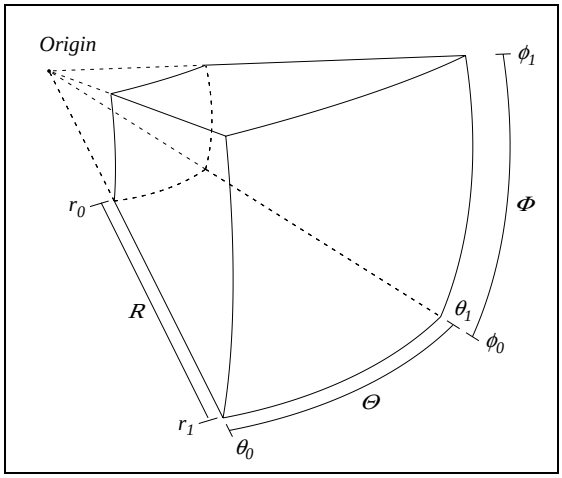


Figure 1: *Intervals* $\mathbf{R}[r_0, r_1]$, $\Theta[\theta_0, \theta_1]$ *and* $\Phi[\phi_0, \phi_1]$

3. THE VOXELIZATION METHOD

The voxelization is accomplished by a recursive three-dimensional spatial subdivision process similar to an octree generation. This process recursively subdivides a given 3D rectangular region into eight equally sized 3D rectangular regions. Each of the eight subdivided rectangular regions is represented by three intervals, one for each variable respectively $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$, where the lower and upper bounds correspond to the rectangular region bounding coordinates.

These rectangular intervals are then converted to spherical intervals. After, the spherical intervals are applied to the inclusion function of the spherically-described implicit function, giving, as a result, the interval $\mathbf{I}$:

$$\mathbf{I} = [i_0, i_1] = \mathcal{F}(R, \Theta, \Phi)$$

If $0 \notin \mathbf{I}$, the current rectangular region is empty (it is not occupied by the surface) and it is ignored. In this case, the next rectangular region in the list of eight is examined and the process described above is repeated.

If $0 \in \mathbf{I}$, the current rectangular region is further subdivided in eight other rectangular regions and the same process described above is repeated to these eight new regions. Eventually the subdivision arrives to the desired resolution. At this point, the rectangular region is a voxel and it is inserted into the octree. When all eight rectangular regions in a level were visited the process is repeated for the remaining rectangular regions in the upper levels.

The voxelization in a given resolution can be seen as the set of voxels where empty regions could not

be eliminated using interval arithmetic, that is, the set of voxels potentially occupied by the surface at a given resolution.

3.1. Converting Rectangular to Spherical Intervals

As discussed above, rectangular **X**, **Y** and **Z** intervals have to be converted to spherical Θ, Φ and **R** intervals. Although the conversion of rectangular to spherical coordinates is straightforward, the conversion of rectangular to spherical intervals is more involved. To guarantee robustness, the region in the space defined by the resulting spherical intervals (as shown in Fig. 1) must completely contain the region defined by the rectangular intervals. On the other hand, the resulting spherical intervals should be as tight as possible. The method presented here take these requirements in consideration.

3.2. Obtaining Θ bounds

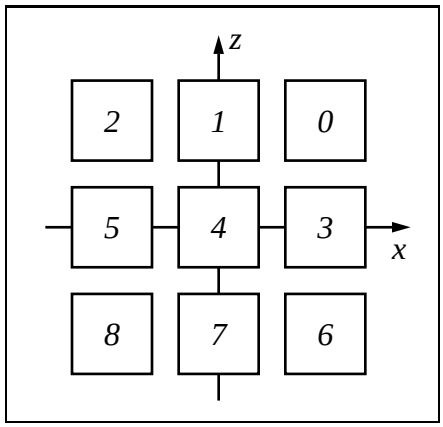


Figure 2: *Nine cases for determining Θ bounds*

To find out the Θ bounds we have defined 9 possible cases (see Fig. 2). These cases cover the whole angular domain ([0, 2π]). In each of these different cases there is a different solution for finding Θ bounds. Each square in Fig. 2 indicates a square defined by the intervals **X** and **Z**, that is, a projection on the XZ plane of the 3D rectangular region defined by **X**, **Y** and **Z**. The numbers in the squares identify the different cases.

Due to space limitations, only one case, case 0, will be discussed. Nevertheless, case 4 is an exception and has no solution, since it would result in an [0, 2π] interval, that is, the whole domain. This problem would not exist if the origin of the surface lies exactly at a vertex of a voxel. However, the origin of the surface can not be imposed by the arrangement of the voxel grid. In this case, if the subdivision is not at the last level (the voxel level), our choice is to assume that the cube contains part of the surface, even if this is not true. Cases different than 4 will eventually show up for subdivided cubes inside this cube, so the calculation would be again possible. At the last level (the voxel level) if this case persists to show up it is ignored, that is, it is assumed that there is no surface in it. This does not constitute a problem because the only voxels of the surface that will be missing are those that intersect the "Y" axis. Hence, these voxels could then be calculated in another way.

Figure 3 shows how to obtain angles θ_0 and θ_1 (Θ bounds) in case 0. In this case:

$$\begin{aligned} \theta_0 &= atan(z_0/x_1) \\ \theta_1 &= atan(z_1/x_0) \end{aligned}$$

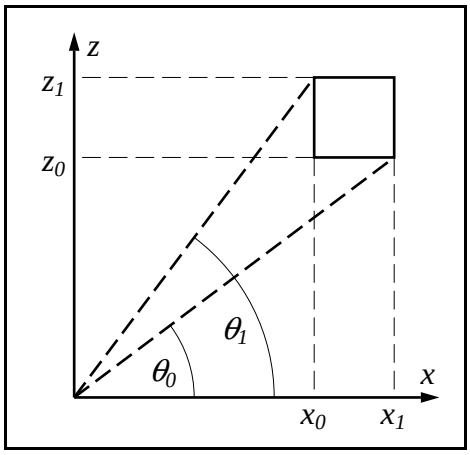


Figure 3: *Determining Θ bounds for case 0*

In the other cases, θ_0 and θ_1 are calculated in a similar fashion. Angles are always defined in such a way that cubes are totally enclosed by them as indicated in Fig. 3. This is done to define a spherical interval which contains the cubic one.

3.3. Obtaining Φ bounds

These bounds are only applicable to spherical coordinates not cylindrical coordinates. To cover the whole spherical space, ϕ_0 and ϕ_1 need to be defined in only half of the angular domain, that is, [-$\frac{\pi}{2}$, $\frac{\pi}{2}$], since θ_0 and θ_1 are already defined in [0, 2π] domain. Since case 4 is eliminated from the analysis, as discussed in the previous section, only three different cases are necessary to fully define Φ bounds.

The three cases are indicated in Fig. 4. Axis r in Fig. 4 is a rotating axis over XZ plane. Suppose that the cube being considered is case 0 (Fig. 2 and 3) and case *a* (Fig. 4), case *0a* for short, ϕ_0 and ϕ_1 are calculated in the following way (r'_1 and r'_0 are not bounds for **R**):

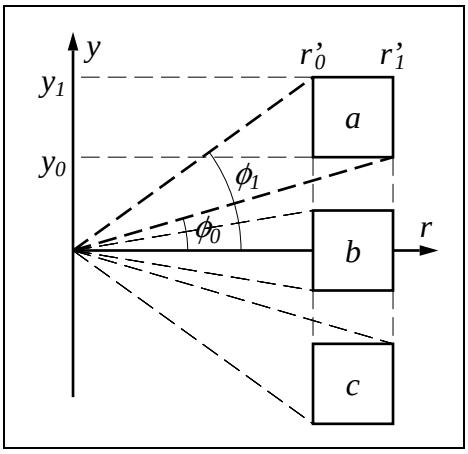


Figure 4: *Three cases for determining Φ bounds*

$$
\begin{array}{lcll}
r'_0 & = & \sqrt{x_0{}^2 + z_0{}^2} & (-\infty\ rounding) \\
r'_1 & = & \sqrt{x_1{}^2 + z_1{}^2} & (+\infty\ rounding) \\
\phi_0 & = & atan(y_0/r'_1) & (-\infty\ rounding) \\
\phi_1 & = & atan(y_1/r'_0) & (+\infty\ rounding)
\end{array}
$$

Notice the suggested rounding modes to guarantee numerical robustness.

3.4. Obtaining R bounds

Interval **R** lower and upper bounds correspond respectively to the minimal and maximal radius value in the 3D region defined by the three rectangular intervals. These maximal and minimal values are the distances between the surface origin and the points of the 3D region defined by the three rectangular intervals which are respectively the nearest and the farthest to this origin.

The process of obtaining these points is straightforward and only requires visualizing the cube in the three-dimensional space with relationship to the origin.

The most trivial calculation is obtaining the lower bounds in cases *1b*, *3b*, *5b* and *7b*. They are respectively: $|z_0|$, $|x_0|$, $|x_1|$ and $|z_1|$. The lower bounds for cases *1a*, *3a*, *5a* and *7a* are respectively: $\sqrt{z_0{}^2 + y_0{}^2}$, $\sqrt{x_0{}^2 + y_0{}^2}$, $\sqrt{x_1{}^2 + y_0{}^2}$, and $\sqrt{z_1{}^2 + y_0{}^2}$. Similarly, the lower bounds for cases *1c*, *3c*, *5c* and *7c* are respectively: $\sqrt{z_0{}^2 + y_1{}^2}$, $\sqrt{x_0{}^2 + y_1{}^2}$, $\sqrt{x_1{}^2 + y_1{}^2}$, and $\sqrt{z_1{}^2 + y_1{}^2}$.

The lower bounds for cases *0b*, *2b*, *6b* and *8b* are equally simple. They are respectively: $\sqrt{x_0{}^2 + z_0{}^2}$, $\sqrt{x_1{}^2 + z_0{}^2}$, $\sqrt{x_0{}^2 + z_1{}^2}$ and $\sqrt{x_1{}^2 + z_1{}^2}$.

For all the other bounds in these and other cases, the maximal and minimal points are always vertices of the rectangular region defined by the rectangular intervals. In these cases the lower bound is the minimal distance between the origin and one of the vertices; the upper bound is the maximal distance.

The calculation of **R** bounds generally requires a square root per bound. However, the signs of the results of the square roots are ambiguous in certain cases. In spherical coordinates a negative r may have meaning. To simplify the calculation and to save computation time one can sometimes work with the square of r instead. This solution was adopted here when spherical coordinates were used. To use r^2, the function $sin(n\cdot\theta)\cdot sin(m\cdot\phi)-r=0$, for example, can then be evaluated as $[sin(n\cdot\theta)\cdot sin(m\cdot\phi)]^2-r^2=0$.

When r^2 can not be used (when only r shows up in the equation and it can not be isolated) then $-r$ should also be considered if the sign of the radius is important. In this case, two equations are considered instead of only one. An interval can be rejected only if it is rejected in both equations, that is, if the resulting interval of both inclusion functions do not include zero.

4. RESULTS

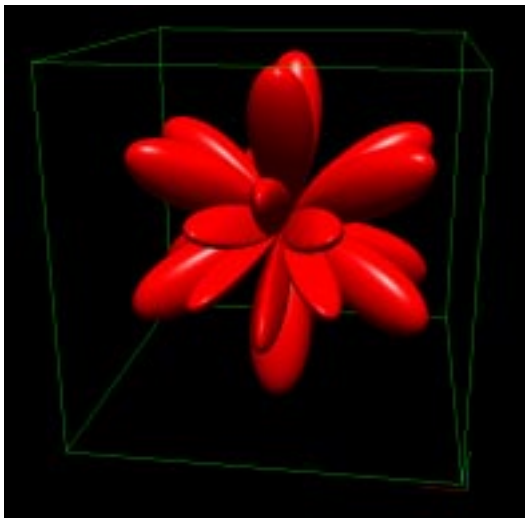
	n=9 m=18		*n=9, m=10*	
Res.	*Time*	*Mem.*	*Time*	*Mem.*
1024^3	653 s	149.363	258 s	113.738
512^3	160 s	39.285	120 s	30.777
256^3	35 s	12.261	28 s	10.308
128^3	6 s	5.738	5 s	5.324

	n=5, m=6		*n=3, m=4*	
Res.	*Time*	*Mem.*	*Time*	*Mem.*
1024^3	307 s	70.472	208 s	48.621
512^3	74 s	20.363	50 s	14.980
256^3	18 s	7.886	12 s	6.597
128^3	3 s	4.816	2 s	4.511

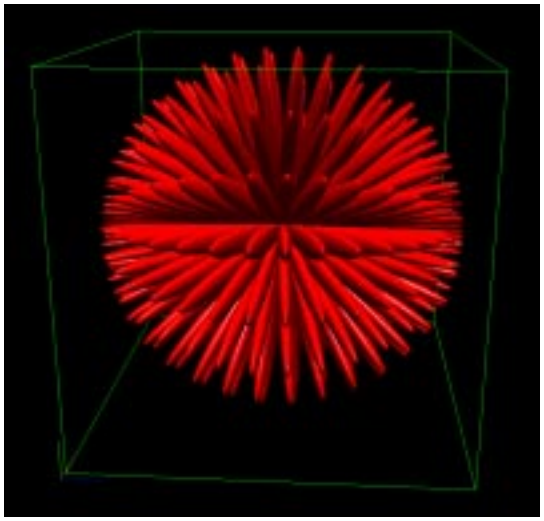
Figure 5: Times for $sin(n\cdot\theta)\cdot sin(m\cdot\phi)-r=0$

Fig. 5 shows some voxelization times for the function $sin(n\cdot\theta)\cdot sin(m\cdot\phi)-r=0$ for different values of n and m and different 3D resolutions (see Fig. 6). Normal vector calculation and normalization are included in the voxelization times. The overall complexity increases as the resolution grows, but running time asymptotically approaches an increasing factor of 4 every time resolution doubles in every coordinate axis. At higher resolutions this quadratic behavior is even more pronounced: for n=3 and m=4, at at 2048^3 resolution (i.e., double of 1024^3 in each axis), our estimated time is $2^2\times 208$ s = 832 s; the actual time was 833 s. By contrast, a method that examines all voxels [3] would have the time increase by a factor of 2^3. For a 512^3 resolution our proposed algorithm can be roughly evaluated as being 512 times faster.

The image in Fig. 7-a shows a gear generated using cylindrical coordinates. Gear teeth were generated by the implicit function $r-(0.8+0.05\cdot sin(32\cdot$

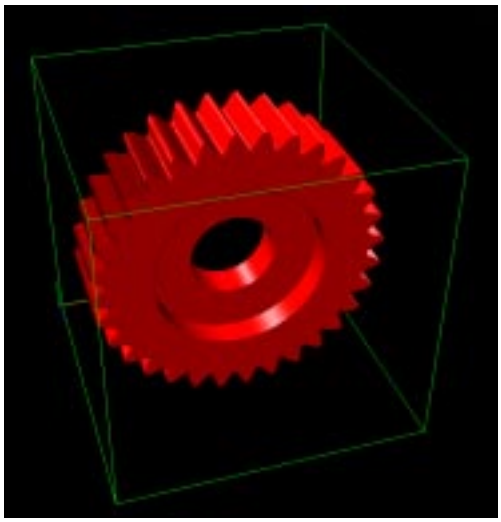


(a)

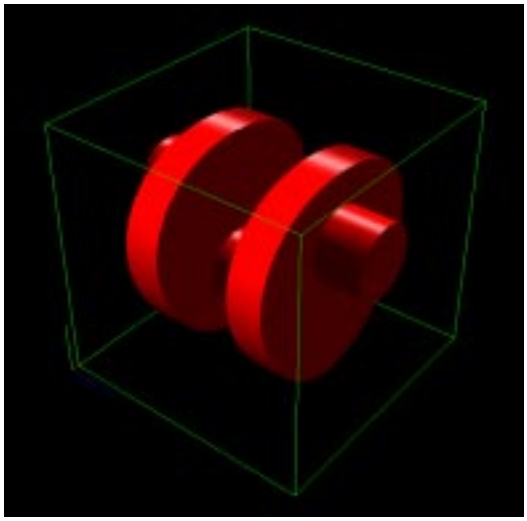


(b)

Figure 6: (a) $sin(3\,\theta)\cdot sin(4\,\phi)-r=0$ and (b) $sin(9\,\theta)\cdot sin(18\,\phi)-r=0$



(a)



(b)

Figure 7: (a) gear and (b) a part of a crank shaft

$\theta))=0$ and voxelized by the method described in this article. All other parts were voxelized using constraints implementing CSG operations of cylinders and planes. The voxelization time for this model at 512^3 resolution was 12 seconds.

The image in Fig. 7-b illustrates another example of practical aplication of our voxelization method using cylindrical coordinates. The contour of the non-cylindrical parts is given by the implicit function $r-(1.2+0.05\cdot sin(3\cdot\theta))=0$, that is, basically the same equation used for the gear. The voxelization time for this model at 512^3 resolution was 14 seconds.

All voxelizations were generated on a Challenger SGI workstation using a single 200MHz R10000 processor. Memory occupation in Fig. 5 is given in MBytes. All images were generated on the same machine using our interactive voxel visualization software. This software traverses all the voxels stored in the octree during the voxelization process and uses GL primitives to display each voxel as a 3D point, using the stored normal vector.

5. LIMITATIONS

It is commonplace to express, in spherical coordinates, surfaces on which points with $r<0$ are actually significant. To be more precise, one has a mapping from $\mathcal{R}^3\rightarrow\mathcal{R}^3$ given by $(r,\theta,\phi)\rightarrow(r\cdot cos\ \theta\cdot cos\ \phi,\ r\cdot sin\phi,\ r\cdot sin\ \theta\cdot cos\ \phi)$; this mapping can be applied to an arbitrary subset of the domain $\mathcal{R}^3$ to define a subset of the codomain $\mathcal{R}^3$. Unfortunately, the mapping is not invertible, so we cannot simply map points in the codomain $\mathcal{R}^3$ to the corresponding points in the domain $\mathcal{R}^3$. Thus without ad-hoc methods, our algorithm is limited to computing isosurfaces that lie

within the $(-\infty, \infty) \times [0, 2\pi] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$ interval in the domain space. In certain cases, it is possible to express r in terms of θ and ϕ, and hence get an expression for r^2; in these cases, besides the considerable economy in calculation times, only one version of the equation can used instead of the two suggested above.

6. CONCLUSION

This paper has presented a new algorithm for voxelizing spherical implicit surfaces. The voxelization algorithm uses interval arithmetic together with hierarchical subdivision to identify empty regions in early stages of the recursive subdivision. Thus, the algorithm times have a tendency to increase only by a factor of 4, instead of 8, each time the resolution doubles for every coordinate axis. In addition, interval arithmetic guarantees that the resulting voxelization totally envelops the surface. Therefore, this method is robust and insensitive to rounding errors if proper rounding modes are set for calculating lower and upper interval bounds. The resulting voxelization is stored in an octree defining a new concept called here *robust hierarchical voxel model* which guarantees that every level of the tree defines a set of voxels which always envelop the continuous surface.

An original method for converting cubic intervals into spherical intervals has also been presented. This method allows using cubic recursive subdivision (easily generating voxels) with spherical implicit functions. The problem is divided into different cases, simplifying interval conversion. Implicit functions in spherical coordinates define a scalar field just as rectangular implicit surfaces do. As a consequence, both kinds of surfaces can be blended together. Our approach allows this blending by using voxel models to represent both surfaces and their blending.

Our visualization method, based on high resolution voxel spaces, not only allows us to show the effectiveness of our voxelization method, but also suggests an alternative way to display implicit surfaces, offering at the same time interactive speeds and raycasting quality.

7. REFERENCES

[1] 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.

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

[3] Pascal J. Frey and Houman Borouchaki. Finite Element Meshes by Means of Voxels. In *6th DGCI'96 - Discrete Geometry for Computer Imagery*, pages 165–172, Lyon, France, November 1996. Springer.

[4] David Goldberg. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991.

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

[6] Arie Kaufman, Daniel Cohen, and Rony Yagel. Volume Graphics. *IEEE Computer*, 26(7):51–64, July 1993.

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

[8] Nilo Stolte. *Espaces Discrets de Haute Résolutions: Une Nouvelle Approche pour la Modelisation et le Rendu d'Images Réalistes.* PhD thesis, Université Paul Sabatier - Toulouse - France, April 1996.

[9] Nilo Stolte and René Caubet. Comparison between different Rasterization Methods for Implicit Surfaces. In Rae Earnshaw, John A. Vince and How Jones, editor, *Visualization and Modeling*, chapter 10, pages 191–201. Academic Press, April 1997. ISBN: 0122277384.

[10] Gabriel Taubin. Rasterizing Algebraic Curves and Surfaces. *IEEE - CGA*, 14(2):14–23, March 1994.