Fast High Definition Discrete Ray Tracing Implicit Surfaces

Nilo STOLTE

René CAUBET

Institut de Recherche en Informatique de Toulouse
118, Route de Narbonne
31062 – Toulouse – France
tel. (33) 61 55 67 65 / fax. 61 55 62 58
stolte@irit.fr

Abstract: *This article suggests a new approach to visualize implicit surfaces by using discrete Ray-Tracing. A preprocessing phase to rasterize the implicit surface is necessary to make use of this kind of Ray-Tracing. Several methods were proposed to subdivide an implicit surface. We extend their ideas to visualize this kind of surface directly into the voxel space.*

Up to date discrete Ray-Tracing, was only used in quite low resolutions (256^3). At these resolutions the quality of the generated images were not satisfying. One way to solve this problem is increasing voxel space resolution. Nevertheless using high definition 3D raster grids would consume too much memory. Using an octree and assuming that the majority of space will be empty, as such is the case in most scenes, an important memory saving is achieved. Conversely the discrete ray traversal (three-dimensional DDA[1]) performance degrades significantly for high resolutions. We solved this problem by dividing the process in two steps where optimal times of three-dimensional DDA are achieved.

1. Introduction

The main advantages of using implicit surfaces defined by implicit analytic functions are:

- Their generality, since most mathematical objects can be represented implicitly. Planes, quadrics, parametric and more exotic surfaces described even by irreducible equations are easily expressed under the implicit form.
- Compactness, since a very complex object can be defined by just one equation.
- Formal exactness, since it represents mathematical functions, allowing its application in several scientific domains.

Implicit surfaces are very easily rendered by using Ray-Tracing. Algebraic surfaces can be displayed using this algorithm applying Collins theorem [Hanrahan, 1983]. Nevertheless more general implicit surfaces like the one suggested by Blinn [Blinn, 1982,

[1]DDA is an abbreviation for *Digital Differential Analyzer*)

Muraki, 1991, Bidasaria, 1992] are not algebraic and cannot be easily rendered using Ray-Tracing. Fujimoto [Fujimoto *et al.*, 1986] with ARTS, uses this kind of surfaces but loosing the mathematical exactness, since he adopts the simplifications suggested by Blinn, that give very nice images but which are not exact.

More exact Ray-Tracing methods to render this generalized implicit functions appeared later. Kalra and Barr [Kalra and Barr, 1989] have proposed a method using Lipschitz constants. Their method is guaranteed to find the exact intersections between rays and surfaces totally automatically. Duff [Duff, 1992] has suggested using interval arithmetics to ray trace these kind of functions. He also guarantees intersection exactness but his method is simpler and more robust, since even precision errors are discarded.

Both methods have the same characteristics, and have also the same disadvantages. The whole space has to be repetitively subdivided for all rays to test if each one cross the surface in recursively subdivided regions.

A faster Ray-Tracing method was proposed in [Yagel *et al.*, 1992]. It is called Discrete or Raster Ray-Tracing, because it works entirely into the 3D voxel space, the discrete space. A preprocessing phase to rasterize the scene into the voxel space is necessary to make use of this kind of Ray-Tracing. The rasterization process assigns the objects' outline geometry to a 3D voxel grid. In this process only the part of the geometry that pierces the voxel is assigned to it. This voxel grid is normally stored in the 3D array format, but it can also be stored in an octree ([Fujimoto *et al.*, 1986], [Sung, 1991],[Stolte and Caubet, 1992], [Gargantini, 1993]). Normally the data stored into the voxel are simply: normal vector, object's color and sometimes other object's rendering parameters in this region. The voxel is generally with pixel dimensions. After rasterizing the scene all the rays cross this discrete space by a totally discrete traversal algorithm, the three-dimensional DDA. When a non empty voxel is reached an intersection is assumed and the illumination can be calculated using the data stored into the voxel.

To rasterize an implicit surface we extended the subdivision method proposed by Kalra and Barr [Kalra and Barr, 1989] to visualize this kind of surface directly into the voxel space by using discrete Ray-Tracing. Instead of recursively subdividing the surface for each ray, we subdivide the whole space recursively just once, during the rasterization phase. Duff's subdivision method [Duff, 1992] could be used instead. Currently we have been testing this later method comparing the results obtained with Kalra and Barr's method.

The big Discrete Ray Tracing advantage is that no expensive intersection calculation between ray and surfaces is needed, allowing a very fast rendering. The big disadvantage is the high resolution 3D voxel grid necessary to achieve good quality images. This grid is then limited to the machine memory size. The grid resolution in [Yagel *et al.*, 1992] is 256^3 to a 80 Mb machine. The maximum resolution was 320^3 to a 128 Mb machine. This easily denotes the difficulty to run this method in a normal workstation. It is also clear that these resolutions are not enough for getting good quality images.

Considering this problem, we suggested elsewhere [Stolte and Caubet, 1995a, Stolte and Caubet, 1995c, Stolte and Caubet, 1995b] an implementation of the Discrete Ray Tracing using an octree. Using the octree suggested there the utilization of the method will not be dependent on the machine memory, but on the number of occupied voxels. On this way simple scenes can be generated in machines poor in memory. On the other hand, it allows to use higher resolutions necessary to get good quality images, which is impossible nowadays by using directly 3D grids.

Nevertheless as stated in [Yagel *et al.*, 1992] most of method's processing cost falls in the discrete traversal algorithm, namely the Three-dimensional DDA. In [Yagel *et al.*, 1992] a non conventional DDA was created to reduce this time. This solution is not

enough in high resolution 3D grids. It is a known fact that its efficiency drops between the fifth and the sixth levels of the octree ([Fujimoto *et al.*, 1986], [Sung, 1991]). We suggested then a two step process to profit of its optimal performance in both steps. This allows us to visualize quite huge 3D grids in short time. On the other hand, the algorithm precision is very important for the correctness of this approach. The reason is that the transition between the two levels is calculated in floating point arithmetics. Our DDA is similar to the one shown in [Fujimoto *et al.*, 1986] but the mapping of floating point values in fixed point using integer variables, offers, at the same time, efficiency, accuracy and multi-precision flexibility.

2. Rasterizing Implicit Surfaces

To rasterize implicit functions we've implemented the method suggested in [Kalra and Barr, 1989]. Although it is not properly a *"voxelization"* algorithm, its subdivision technique can be used to rasterize a surface. The method guarantees that no part of the object would be lost assuring a good rasterization. The main advantage of the method is fast convergence to the surface which drastically improves the performance. Another advantage of the method is that it subdivides the space the same way an octree does, which permits using almost the same algorithm to both tasks, subdivide the implicit surface and display the resulting octree. An octree approach was implemented in [Wilhelms and Van Gelder, 1992] but for isosurfaces in volumetric data. In this type of data the contents of the voxels are not known *a priori*, which obliges utilizing a setup phase to estimate the minimum and maximum voxels' values of an octree's octant. On the other hand, rasterizing an implicit function avoids any setup phase of the octree since the voxels the function will occupy can be predicted.

In [Bloomenthal and Wyvill, 1990] the octree is used to help polygonizing implicit functions assuming that pure octree is only good for a coarse representation in interactive applications. Although, we've implemented a voxel visualization method using our octree which permits us to visualize interactively implicit functions contained in 512^3 voxel volumes in near real time. The quality obtained is superior than using polygons.

The two functions used in this article are "metaballs" ([Blinn, 1982], [Muraki, 1991], [Bidasaria, 1992]) and their equations are:

a) $$e^{-3.25((x-0.78)^2+(y-0.78)^2+(z-0.78)^2)} + e^{-3.25((x-0.23)^2+(y-0.23)^2+(z-0.23)^2)} - 0.9 = 0$$

b) $$e^{-196((x-0.23)^4+(y-0.23)^4+(z-0.68)^4)} + e^{-196((x-0.68)^4+(y-0.68)^4+(z-0.23)^4)} + e^{-7((x-0.68)^2+(y-0.68)^2+(z-0.68)^2)} + e^{-7((x-0.23)^2+(y-0.23)^2+(z-0.23)^2)} - 0.9 = 0$$

Equation *"a"* corresponds to the surface shown in Figure 5, which was rasterized in 5 minutes and 51 seconds in a Crimson SGI workstation (R4000+R4010,100MHz,128Mb) for a 1024^3 resolution. Equation *"b"* corresponds to the surface shown in Figure 6, which was rasterized in 15 minutes and 30 seconds in the same machine for the same resolution. Several optimizations, already devised but not yet done, are possible and can hopefully reduce these times significantly. Nevertheless there are other rasterization methods that could be faster ([Duff, 1992] and [Taubin, 1994]). We are currently comparing these algorithms to decide which one has the best results in terms of speed, quality and generality.

3. The Octree

3.1 Introduction

The octree used here uses the spatial enumeration in fixed sized blocks, which we call cells, of eight elements each. This approach allows a very fast vertical traverse due to operations' simplicity. We show that memory requirements for this octree are also very interesting.

The octree is defined dividing space recursively in eight regions of equal dimensions in each level until each of these regions has the size of a voxel (Figure 1). Each element of each cell corresponds to a sub-space of the correspondent volume on a given level.

Figure 1 represents the octree implementation with five levels (left) and its visualization into space (right). Five cells are associated to the octree of Figure, where each one is located in successive levels of the octree. Each element on last level corresponds to a voxel into a 3D grid of $2^5 \times 2^5 \times 2^5$ resolution, which can be represented by its coordinates: (X,Y,Z). The eight voxels showed in Figure 1, on last level (k=4), pink-colored, are: $V_0(30,30,30)$ -the only one not seen in 3D diagram-, $V_1(31,30,30)$, $V_2(30,31,30)$, $V_3(31,31,30)$, $V_4(30,30,31)$ -the only explicitly shown in Figure-, $V_5(31,30,31)$, $V_6(30,31,31)$, $V_7(31,31,31)$. They are shown in this order in diagram on the left. This order is the same adopted in [Fujimoto *et al.*, 1986]. Each cell is therefore an array of eight elements where the order of an element is related to its position inside the sub-volume that the cell represents.

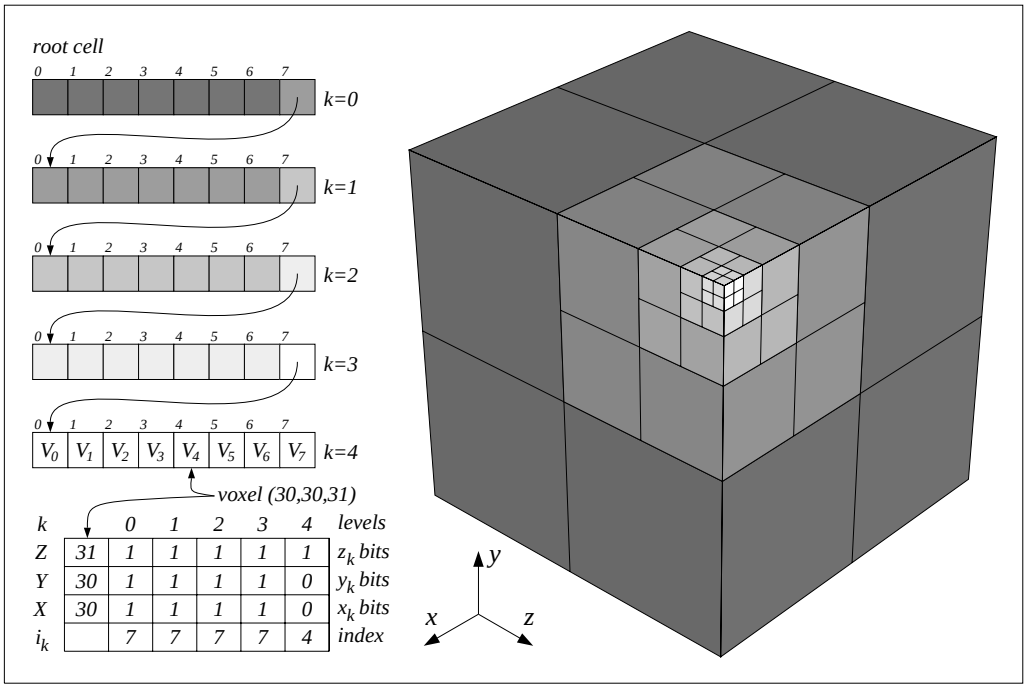

Figure 1: *Block diagram showing the octree*

We define the order of the voxels in cell for octrees with n levels. Each coordinate X, Y, Z of a voxel in an uniformly subdivided space with a resolution $2^n \times 2^n \times 2^n$ is a binary number that is defined as a summation of two powers (see below). The total number of levels of the octree is n. Be k the index of a bit in binary coordinate ($k = n - 1$, the

rightmost bit; $k = 0$, the leftmost bit). Therefore the index of an element of a cell in a certain level k is given by $i_k = x_k + 2{\cdot}y_k + 4{\cdot}z_k$, where x_k, y_k and z_k are the correspondent bits of the coordinates X, Y and Z indexed by k:

$$X = \sum_{k=0}^{n-1} x_k \cdot 2^{(n-1)-k} \qquad Y = \sum_{k=0}^{n-1} y_k \cdot 2^{(n-1)-k} \qquad Z = \sum_{k=0}^{n-1} z_k \cdot 2^{(n-1)-k}$$

$$0 \leq X \leq 2^n - 1 \qquad 0 \leq Y \leq 2^n - 1 \qquad 0 \leq Z \leq 2^n - 1$$

$$x_k, y_k, z_k \in \{0, 1\}$$

3.2 Vertical Traverse Performance

Because of the property of dividing space in eight portions recursively, the vertical traverse of the octree shown here, has a complexity of $\mathrm{O}(log_8 N)$ for a full octree, where N is the total number of cells.

Although the vertical traverse performance depends also on the processing in each of its levels. This processing is done by the algorithm that implements the lookup of a voxel. This algorithm is given in a notation similar to the C language (Figure 2). X, Y and Z are the coordinates of the voxel to be searched. The $<<$ and $>>$ operators are respectively the binary left and right shift. The array *cell* is a pointer but the notation of array is more convenient for clarity purposes. Hence variable i does not really exist. At algorithm exit variable *cell* will contain the address of the cell where the voxel is (a null address indicates that voxel does not exist) and the value of i is the voxel index in this cell (in the real implementation, the pointer *cell* has the voxel address).

```
mask = 1 << (n − 1)
cell = octree root cell
bool = TRUE
while (bool)
      {  i = 0
         if  (Z and mask) ≠ 0)    i = i + 4
         if  (Y and mask) ≠ 0)    i = i + 2
         if  (X and mask) ≠ 0)    i = i + 1
         if  ((mask and 1) == 0)
           { if  (cell [i] ≠ 0)
                { mask = mask >> 1
                  cell = cell [i]
                }
             else bool = FALSE
           }
         else bool = FALSE
      }
```

Figure 2: *Octree vertical traverse*

In Figure 2 algorithm, variable *mask* has initially the bit $n-1$ set (n is the number of octree levels, as seen before) and all the other reset. The pointer *cell* receives the address of the first octree cell, which is the father of the whole octree. The main processing is done into a loop shown here by a *while* command. The loop is totally controlled by

variable *mask* which is shifted right in each successive cell. When *mask* arrives to one, the loop is interrupted as shown. Otherwise, the index of the element in cell is initialized and calculated. Making a logical "*and*" between *mask* and each coordinate is an easy and fast way to ignore a zero bit, which indicates no contribution to the index. On the other hand, if it is one, its contribution is accumulated in the index.

Therefore, variable *mask* has in reality two different functions in algorithm. These functions are the exit control of loop and filter of the bit to be considered. This is very important because with this scheme the proper bit and the loop control are both updated by just one shift operation. When a whole branch of the octree does not exist (when $cell[i]$ is zero) the process is immediately interrupted. This extra test is necessary to the algorithm correctness and reduces useless processing when voxel does not exist.

Then, in this algorithm, there are the following operations: 4 logical *ands*, 3 additions, 2 comparisons and 1 shift. This makes only 10 integer arithmetic operations for each octree level. But it is even faster for empty regions.

It is easily shown that this implementation is faster than the linear octrees ([Gargantini, 1982]), but this demonstration is out of the scope of this article. This conclusion is particularly interesting since linear octree serves as a base to many octree implementations ([Glassner, 1984], [Sung, 1991], [Gargantini, 1993]).

3.3 Memory Requirements

The octree shown here is formed with cells of eight elements each. Each element can have eight son cells, right from the first cell. Hence, the memory allocation for the full octree obeys a geometric series with step 8:

$$8^0 + 8^1 + 8^2 + 8^3 ... + 8^{n-1} = \frac{1 - 8^n}{1 - 8}$$

For last level, there are 8^{n-1} cells that will correspond to 8^n leaf elements or voxels. Then for indexes, there will be in a full octree the following number of cells:

$$8^0 + 8^1 + 8^2 + 8^3 ... + 8^{n-2} = \frac{1 - 8^{n-1}}{1 - 8} = \frac{1}{7} \cdot 8^{n-1} - \frac{1}{7}$$

A 3D array of voxels of $2^n \times 2^n \times 2^n$, which has 8^n voxels, will contain 8^{n-1} cells. Therefore a full octree has practically $\frac{1}{7} \times 8^{n-1}$ more cells than the equivalent 3D array.

Although memory requirements for the full octree will be about 14% greater than a 3D grid, it is highly improbable that this situation would happen without being forced. Anyway, memory requirements for a 3D grid are still prohibitive for high resolutions. Hence the octree as proposed here seems to be much more appealing than the 3D grid in the Discrete Ray Tracing context, where grid resolution is high.

4. Skipping Empty Regions

The algorithm in Figure 3 allows us to skip empty spaces without descending or ascending the octree while Three-dimensional DDA traverses empty regions. Variable *mask* is shown in algorithm of Figure 2. It is used in our Three-dimensional DDA to identify the exact bit that changes the value only when the DDA leaves the empty region. If it has been

changed the DDA's loop is finished and the octree's vertical traverse restarts as shown in algorithm of Figure 3.

Push and *pop* operators of Figure 3 denote an external stack to keep the addresses of the parents cells, as done by many authors ([Fujimoto *et al.*, 1986], [Sung, 1991] et [Gargantini, 1993]). The first part of main loop ascends the octree while no common father is found. The second part descends the octree (basically the same as Figure 2). The third part is the Three-dimensional DDA.

```
/* Initialize XYZ and mask as follows (each square is a bit) */
```

$XYZ \leftarrow$	0	x	x	...	x	0	y	y	...	y	0	z	z	...	z
$mask \leftarrow$	0	1	0	...	0	0	1	0	...	0	0	1	0	...	0

```
cell = root cell address of octree
push(cell)
i_c = 0 /* index of last changed coordinate*/
test_end = mask
XYZ_ant = XYZ
/* repeat while inside octree*/
while (true)
      {  XYZ_ant = XYZ_ant xor XYZ
         if (XYZ_ant and test_end) break
         mask = mask >> 1
         /* ascend octree until common father arrives*/
         while (XYZ_ant and mask)
               { pop
                 mask = mask << 1
               }
         mask = mask << 1
         pop(cell)
         X = extract X from XYZ
         Y = extract Y from XYZ
         Z = extract Z from XYZ
         /* The lines different from fig. 2 are commented */
         bool = TRUE
         while (bool)
               { push(cell) /* push address cell */
                 i = 0
                 if ((Z and mask) ≠ 0) i = i + 4
                 if ((Y and mask) ≠ 0) i = i + 2
                 if ((X and mask) ≠ 0) i = i + 1
                 if ((mask and 1) == 0)
                    { if (cell[i] ≠ 0)
                         { mask = mask >> 1
                           cell = cell[i]
                         }
                      else bool = FALSE
                    }
                 else bool = FALSE
               }
         if (cell[i] ≠ 0)
           { /*second step*/ }
         while (in empty region)
               { /*execute 3DDDA step*/ }
      }
```

Figure 3: *Octree traversal with empty space skip*

5. Results

In the tables of Figure 4, our results for the test image 1 and test image 2 are summarized. Images (Figures 5 and 6) were calculated using a Crimson SGI workstation (R4000+R4010,100MHz,128Mb). These tables indicate the screen resolution (first columns), 3D grid resolution (second columns), and times for the two step ray tracing with different number of levels of the octree in the first step and the second step (third

Results for image 1 (Figure 5)		
Res. 2D	Res. 3D	RT 2 steps
1024^2	1024^3	3'10" (4+6)
		2'21" (5+5)
		2'03" (6+4)
		2'01" (7+3)
512^2	1024^3	0'47" (4+6)
		0'35" (5+5)
		0'31" (6+4)
		0'30" (7+3)
256^2	1024^3	0'12" (4+6)
		0'09" (5+5)
		0'07" (6+4)
		0'07" (7+3)

Results for image 2 (Figure 6)		
Res. 2D	Res. 3D	RT 2 steps
1024^2	1024^3	3'51" (4+6)
		2'49" (5+5)
		2'25" (6+4)
		2'27" (7+3)
512^2	1024^3	0'58" (4+6)
		0'42" (5+5)
		0'36" (6+4)
		0'37" (7+3)
256^2	1024^3	0'14" (4+6)
		0'10" (5+5)
		0'09" (6+4)
		0'09" (7+3)

Figure 4: *Results for image 1 and image 2*

columns). The numbers in parentheses are respectively the number of levels in the first step and in the second step. The "+" separating them indicates that their addition gives the total number of octree levels. All times include the generation of a PostScript file of the image (the code was compiled without optimization options).

As observed, better times are with 6 levels in first step. Although the two figures are clearly of completely different complexities the times are quite similar. A clear advantage of our approach in comparison with classic Discrete Ray Tracing is that performances are higher when the scene is emptier. This situation is ideal with an octree, since it implies better memory savings and better performances.

6. Conclusion

We have presented an implicit surface visualization algorithm based in our discrete Ray-Tracing system [Stolte and Caubet, 1995a, Stolte and Caubet, 1995c, Stolte and Caubet, 1995b]. The method is quite different than previous ones [Kalra and Barr, 1989, Duff, 1992], since instead of recursively subdividing the space for each ray, we recursively subdivide the space just once, storing the voxels into an octree. This allows also recalculating other images under other view points without a new rasterization. We avoid expensive intersection calculations between rays and surfaces by simply assuming that a reached non empty voxel is already the intersection as suggested in [Yagel *et al.*, 1992]. Nevertheless our images' quality is better since we can rasterize the surfaces in high 3D resolution.

As verified by the results our method's rendering performance is similar to the ones obtained previously [Stolte and Caubet, 1995a, Stolte and Caubet, 1995c, Stolte and Caubet, 1995b], showing that good quality high resolution images can be produced in quite low times. The rasterization time can be longer than using polygons depending on the complexity of the function. Considering that this kind of surface is very difficult to be consistently converted into polygons, that several images can be produced under different view points without any further rasterization, and that image quality and mathematical exactness are better than using polygons, this rasterization time is negligible.

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.

[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.

[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.

[Gargantini, 1982] Irene Gargantini. Linear Octtrees for fast Processing of Three- Dimensional Objects. *Computer Graphics and Image processing*, 20(4):365–374, 1982.

[Gargantini, 1993] Irene Gargantini. Ray tracing an Octree: Numerical Evaluation of the First Intersection. *Computer Graphics forum*, 12(4):199–210, 1993.

[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.

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

[Stolte and Caubet, 1992] Nilo Stolte and René Caubet. Some More Enhancements to Ray Tracing. In *Compugraphics'92*, pages 53–60, Lisbon, December 1992. Harold P. Santo.

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

[Wilhelms and Van Gelder, 1992] Jane Wilhelms and A. Van Gelder. Octrees for Faster Isosurface Generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[Yagel *et al.*, 1992] Roni Yagel, Daniel Cohen, and Arie Kaufman. Discrete Ray Tracing. *IEEE - CGA*, 12(5):19–28, 1992.

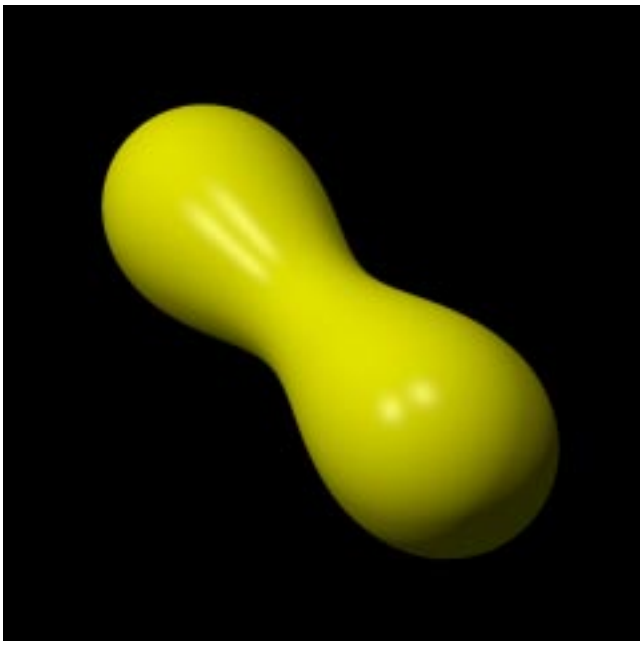

Figure 5: *Image 1*

Figure 6: *Image 2*