EUROGRAPHICS 2001 / A. Chalmers and T.-M. Rhyne
(Guest Editors)

Volume 20 (2001), Number 3

Visualization of Isosurfaces with Parametric Cubes

B. Mora, J.P. Jessel, R. Caubet

Institut de Recherche en Informatique de Toulouse (IRIT), Université Paul Sabatier, Toulouse, France

Abstract

To render images from volume datasets, an interpolation method also called reconstruction is needed. The level of details of the resultant image closely depends on the filter used for reconstruction. We propose here a new filter producing C^1 continue surfaces. The provided image quality is better than current high-quality algorithms, like splatting or trilinear raycasting, where tiny details are often eliminated. In contrast with other studied high quality filters that are practically unusable, our algorithm has been implemented interactively on a modest platform thanks to an efficient implementation using parametric cubes. We also demonstrate the interest of a min-max octree in the visualization of isosurfaces interactively thresholded.

Keywords: Isosurface, Reconstruction filter, Octree, Quantization.

1. Introduction

Visualization of isosurfaces is often needed for the rendering of scientific volumetric data. One of the first algorithms was based on the cuberille model[3]. In this method, sampled points of the volume were considered as parallelepipeds. Thus projecting the visible faces into the image plane performs the rendering. However, the low resolutions of the normals and the step effects have clearly limited the rendering quality. This method can be considered as a zero-order reconstruction method, because there is no interpolation.

Lorensen and Cline have proposed two great improvements for the visualization of isosurfaces by using a cube formed of eight neighbouring voxels. The first one is the well-known marching-cube[4] algorithm where a triangular mesh is deduced from the vertex configuration. This algorithm provides a C^0 continuity of the isosurface between the cells. However, the normals of the triangle vertexes can be trilinearly interpolated from the cube vertexes in order to look like a C^1 continue surface for fine visualizations. Because the isosurface is roughly approximated, tiny details are lost. A better approach is the dividing cube[6] algorithm where the isosurface is approximated with trilinear interpolation, but the same way to compute the normals also limits the rendering quality. These two algorithms use a one-order reconstruction scheme (linear interpolation).

High quality direct volume rendering algorithms can also be used for isosurface visualization[2,5,9,10,11,22]. Raycasting is probably the most used method. This one-order method also uses trilinear interpolation that can be associated with fuzzy classification in order to improve renderings. Although the rendering times are slow, interactive implementations exist[19,21]. It is also true for splatting[11,20,24] which is the other solution in vogue for producing high quality images. Splatting makes an approximate convolution by projecting fuzzy voxels (gaussian kernels) on the image plane. The order of this method depends on the gaussian radius. However, the final result is not a real product of convolution, and the quality is not really better than raycasting.

Until now, all the described algorithms have been thoroughly studied and implemented. However other kinds of methods[7,14,16,17,23,26], generally based on BC-spline or windowed sinus cardinal filters, have also been studied but they seem to be too complex to be implemented in a convivial rendering software. In the best case, they can be used for volume resampling[9]. The signal processing theory states that a signal sampled under the Nyquist frequency, which is often the case with filtered medical datasets, can be reconstructed by a convolution with the ideal sinus

© The Eurographics Association and Blackwell Publishers 2001. Published by Blackwell Publishers, 108 Cowley Road, Oxford OX4 1JF, UK and 350 Main Street, Malden, MA 02148, USA.

cardinal filter. Unfortunately, it is not possible to implement this filter since its size is infinite, so filters of limited order have been studied. Nevertheless we can consider that increasing the filter size improves the signal reconstruction in general. As an important result of those works[14,23], it appears that the trilinear filter is far from the optimum convolution filter.

Thus high quality and interactivity seem to be irreconcilable. The works presented here try to give a solution to this difficult problem with a new two-order filter. Its complexity is less than classical high-order filters, like BC-splines (three order), and it allows an interactive visualization of isosurfaces thanks to an efficient implementation using both parametric cubes and a min-max octree. However, due to the amount of extra-computations needed for direct volume rendering, its use in this case will not be discussed here.

Section 2 will describe the 1D filter in order to have a better understanding, and section 3 will extend this principle to a 3D reconstruction of the volume. Section 4 will describe our interactive implementation and some results will be discussed in section 5.

2. 1D Filter

The filter presented here is a two-order filter, so it takes 3 neighbouring samples (P_{i-1}, P_i, P_{i+1}) to reconstruct the 1D signal (fig. 1). First, the P_{i-} and P_{i+} points must be defined in order to reconstruct the interpolated curve:

$$P_{i-} = \frac{P_{i-1} + P_i}{2} \qquad P_{i+} = \frac{P_i + P_{i+1}}{2}$$

A B-spline curve is now defined from the control points (P_{i-}, P_i, P_{i+}) with t=0 at i-0.5 and t=1 at i+0.5:

$$f(t) = (1-t)\cdot\left((1-t)\cdot P_{i-} + t\cdot P_i\right) + t\cdot\left((1-t)\cdot P_i + t\cdot P_{i+}\right)$$

Finally we have:

$$f_{P_{i-1},P_i,P_{i+1}}(t) = \left(\frac{P_{i-1} + P_{i+1}}{2} - P_i\right)\cdot t^2 + \left(P_i - P_{i-1}\right)\cdot t + \frac{P_{i-1} + P_i}{2}$$

Properties

The first property is that the curve may not pass through the sampled points. Thus the curve never reaches local maximums, which can be useful when noise is present.

The gradient properties are more interesting. The first derivative of the curve is given by:

$$f'_{P_{i+1},P_i,P_{i+1}}(t) = (1-t)\cdot\left(P_i - P_{i-1}\right) + t\cdot\left(P_{i+1} - P_i\right)$$
$$= (1-t)\cdot G_{i-} + t\cdot G_{i+}$$

With: $G_{i-} = P_i - P_{i-1}$ $\qquad$ $G_{i+} = P_{i+1} - P_i$

We see the derivative is a linear interpolation of two middle gradients. Hereby, the C^1 continuity can be easily proved, so we do not need to demonstrate it. We take this way to compute the gradient for more precise than the classical central difference gradient. In order to argue in this way, we analyse those two gradients at both i and i+1/2 abscissas:

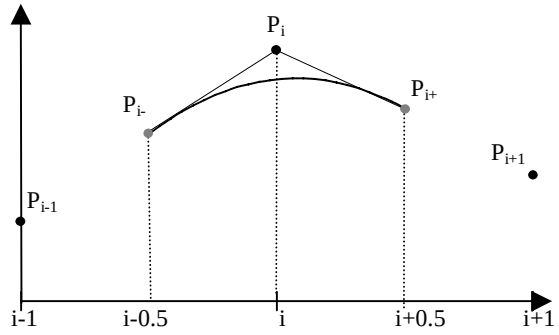


Figure 1: 1D Filter

$$G^{CDIF}(i) = \frac{P_{i-1} + P_{i+1}}{2} = \frac{G_{i-} + G_{i+}}{2} = G^{MDIF}(i)$$

The central difference gradient and the middle difference gradient are the same here. In fact the central gradient can be reconstructed from the middle difference gradient however the other way around is not true. Thus the gradient information is degraded with this latter. At the i+1/2 abscissa we have:

$$G^{CDIF}(i+1/2) = 1/2\cdot G^{CDIF}(i) + 1/2\cdot G^{CDIF}(i+1)$$
$$= \frac{P_{i+1} + P_{i+2} - P_i - P_{i-1}}{4}$$
$$G^{MDIF}(i+1/2) = G_{i+} = P_{i+1} - P_i$$

The two gradients are the most divergent here. The central gradient is averaged with four points while the middle gradient is more compact. So normals computed with the first one should be smoother. On the other hand, tiny details might be lost. We will verify it in section 5.

3. Extension to Volumes

The extension to 3D volumes is easy, and the given properties in section 2 are also true in 3D. However, we will not explicitly develop the equation, because it would be too long. Twenty-seven neighbouring voxels are now needed for the reconstruction within a cube that has the voxel size. We will call this cube (or voxel) the parametric cube because of the parametric way to reconstruct the signal. The previous 1D filter is going to be applied three times on the 27 voxels (referenced P_{ijk}). This h interpolation at the location (x, y, z) can be written as:

$$h_{P000..P222}(x,y,z) = f_{R_0R_1R_2}(z) \qquad (1a)$$

Where R_i is given by:

$$R_i = f_{Q_{0i}Q_{1i}Q_{2i}}(y) \quad 0 \le i \le 2$$

© The Eurographics Association and Blackwell Publishers 2001.

With

$$Q_{ij} = f_{P_{0ij}P_{1ij}P_{2ij}}(x) \quad 0 \le i, j \le 2$$

Nevertheless, this way to compute the reconstruction is too expensive to be processed. Thus, the h function can be rewritten by both developing and substituting the terms of the previous equation (1a):

$$h_{P000..P222}(x, y, z) = \sum_{i,j,k=0}^{2} S_{ijk} \cdot x^i y^j z^k \qquad (1b)$$

The S_{ijk} coefficients are computed from the P_{ijk} values, which requires 316 additions and 80 multiplications per cube. Then the interpolation computation (1b) requires 54 multiplications and 26 additions.

Because this way to interpolate is also too demanding, it must be optimised. Finding out the isosurface that crosses a parametric cube will be performed by sampling values along every ray that goes through this cube. Let P_0 (x_0, y_0, z_0) and P_1 (x_1, y_1, z_1) be the intersection between the ray and the cube, the position of the sampled point can be written in a parametric way:

$$\begin{aligned} x &= x_0 + t \cdot (x_1 - x_0) \\ y &= y_0 + t \cdot (y_1 - y_0) \\ z &= z_0 + t \cdot (z_1 - z_0) \end{aligned} \qquad (2)$$

By substituting (2) in (1b), the reconstruction along a ray can be written as a six degrees polynomial:

$$h(t) = \sum_{i=0}^{6} C_i \cdot t^i \quad 0 \le t \le 1 \qquad (3)$$

Then the reconstruction requires only 6 additions and 11 multiplications. Furthermore, if the number of steps is known, the t^i values can be pre-computed in order to reduce the number of multiplications to 6. However, before sampling values along a ray, the C_i coefficients must be computed for every ray from P_0, P_1, and the S_{ijk} coefficients. This operation needs around 724 multiplications and 108 additions. But by using quantization (See section 4.2), we can reduce it to 101 additions and 108 multiplications.

Instead of sampling values along a ray, another solution for finding out the intersection with the isosurface would have been to analytically solve the polynomial (3). Unfortunately, this solution is expensive. Nevertheless, in order to have a better accuracy of t when the intersection is detected, the value of t is linearly interpolated from the last value and the value obtained just before using (4).

$$t = t_{n-1} + (t_n - t_{n-1}) \cdot \frac{Threshold - h(t_{n-1})}{h(t_n) - h(t_{n-1})} \qquad (4)$$

With: $h(t_{n-1}) < Threshold \le h(t_n)$

Shading

Once the intersection is detected, shading must be done. A Phong shading with depth-cueing has been chosen for realistic renderings. Thus the normal of the surfaces is needed. Because our 3D filter is C^1 continue, the normal can be analytically solved with (5). This computation requires 51 additions and 54 multiplications, without allowing for the shading implementation, but it is only performed once per ray.

$$\nabla h_{P000..P222}(x, y, z) = \begin{pmatrix} \sum_{i,j,k=0}^{2} S_{ijk} \cdot x^i y^j z^k \Big/ dx \\ \sum_{i,j,k=0}^{2} S_{ijk} \cdot x^i y^j z^k \Big/ dy \\ \sum_{i,j,k=0}^{2} S_{ijk} \cdot x^i y^j z^k \Big/ dz \end{pmatrix} \qquad (5)$$

4. Fast Implementation

As it was mentioned before, our new filter requires a lot of arithmetic instructions, so optimisations must be implemented in order to get interactive renderings.

4.1. The algorithm

We have chosen an object-order algorithm where the voxels will be reviewed in a front-to-back order. The basic algorithm to render a parametric cube can be written as shown below:

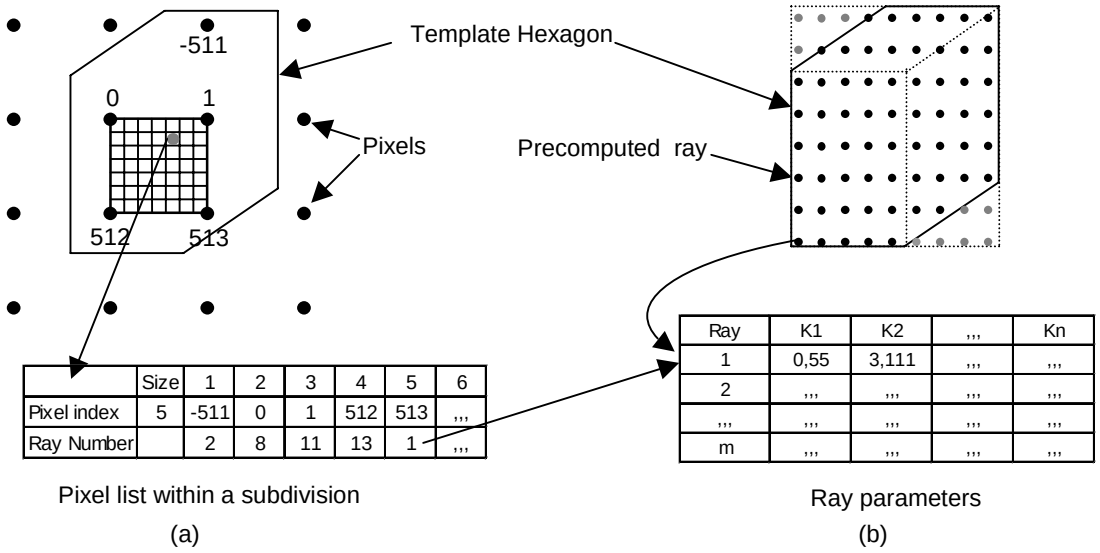
```
BEGIN
COMPUTE Sijk FROM Pijk
FOR every ray that intersects the cube DO
  BEGIN
  COMPUTE Ci FROM P0, P1, Sijk
  FOR n=1 TO NumberOfSteps DO
    IF h(tn)>Threshold THEN
      BEGIN
      COMPUTE t FROM tn, tn-1,, Threshold, h
      COMPUTE x, y, z FROM t
      COMPUTE pixel shading FROM x, y, z, Sijk
      BREAK
      END
  END
END
```

By profiling this code, one can see two very expensive instructions slowing down the algorithm. The first one is that the algorithm must solve all the rays that intersect the parametric cube, which is a complex algorithm, a priori. Next, the algorithm has to perform the expensive instruction of computing the C_i coefficients for each ray. Fortunately, those drawbacks can be partially avoided by a quantization of the image plane.

4.2. Quantization

The idea for accelerating the projection in object-order algorithms has simultaneously appeared[20, 24, 25] in some fast methods. The only requirement (but Huang[24]) of the

© The Eurographics Association and Blackwell Publishers 2001.



	Size	1	2	3	4	5	6
Pixel index	5	-511	0	1	512	513	,,,
Ray Number		2	8	11	13	1	,,,

Ray	K1	K2	,,,	Kn
1	0,55	3,111	,,,	,,,
2	,,,	,,,	,,,	,,,
,,,	,,,	,,,	,,,	,,,
m	,,,	,,,	,,,	,,,

Figure 2: Fast determination of the projection (a) and use of precomputed rays (b).

method is the use of orthogonal projections. Huang[24] et al. describe a fast splatting method while Mroz[25] et al. suggest a fast approximation of the cell projection for MIP renderings. However those two methods are a bit flawed on axes-aligned viewings, and only Mora[20] et al. give a solution for solving the exact projection of a voxel. Thus, our algorithm is an extension of this method. We will not describe once again the complete algorithm, but only the main lines. So the reader is invited to take a look at these references.

In orthogonal renderings, a voxel projection is always represented by the same template hexagon. The voxel projections only differ by the translation value. Thus, the projection of the voxel centre is enough to give the translation information. A pixel list depending on the projection of the voxel centre can be used. In order to have a better precision, the pixel have to be subdivided, which involves now a pixel list per subdivision (See Fig. 2a). These precomputed tables ensure to know quickly the rays (i.e. pixels) that go through the parametric cube with a given accuracy.

The second problem is to accelerate the computation of the C_i coefficients. As it was seen before, those coefficients depend on both the S_{ijk} coefficients and the ray. They can be written as follows:

$$C_n = \sum_{i,j,k=0}^{2} S_{ijk} \cdot K_{ijk}^n \qquad (6)$$

Where the K_{ijk} coefficients only depend on the ray. Those coefficients are in fact the main part of the computation. However, by assuming that a limited number of rays is enough to describe all the rays, we can precompute those coefficients for every quantized ray. We will not explicitly describe here the K_{ijk} computations from the ray, so those results will be considered as well established. We are just going to describe how to make a set of precomputed rays (fig. 2b).

For easy precomputations, all the rays are equally spaced (however, unevenly distributed rays could be a potential improvement in future works). This is done by regularly sampling the rectangle surrounding the template hexagon. A point within the template hexagon means that the equivalent ray intersects the cube. Then the algorithm just has to precompute the K_{ijk} coefficients for those rays.

It has been seen earlier that a projection can be described by a pixel list. For every pixel (representing a ray) of this list, the K_{ijk} coefficients must be accessible. Thus, a pointer to the best representative quantized ray is associated to the pixel index (fig. 2a). All these lists can be quickly preprocessed (10 ms) before the rendering step.

Using this method, the computation of the C_i coefficients is reduced to 101 additions and 108 multiplications. Nonetheless this is done for every ray intersecting the cube. So this expensive operation should be avoided as much as possible. We use a min-max octree to determine quickly the voxels of interest.

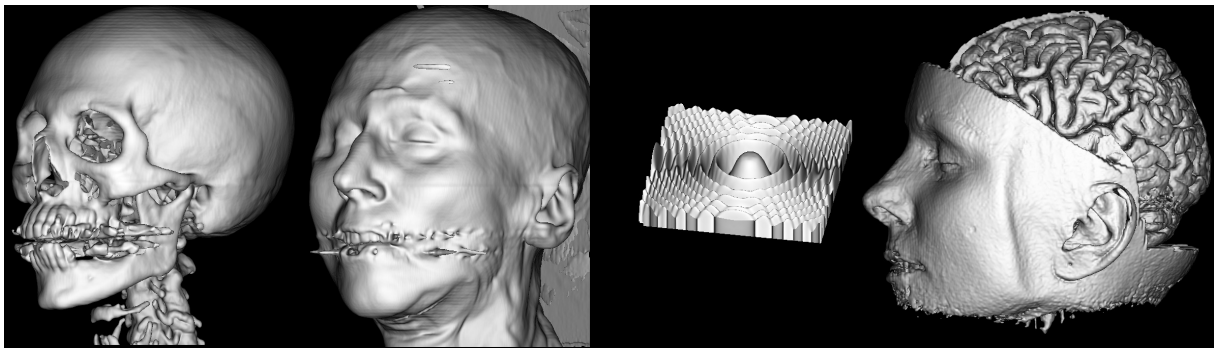
4.3. Min-Max Octree

Octrees have been studied a lot[1,13,15,27], so it is assumed that the reader is familiar with it. Its implementation itself is not an important factor of speed-up. So we will just describe it for more information.

We use a linear array for storing the octree. All the nodes but the leaf nodes store the two min-max values and one pointer to the 8 children. The leaf nodes only store the min-max values summarising the 27 voxels involved in the reconstruction within the parametric cube. So the size of the octree is approximately given by:

$$Size = n \times (2 \times S_{\min\max}) + \frac{n}{8} \times (2 \times S_{\min\max} + S_{Pointer}) + \frac{n}{64} \times \ldots$$

© The Eurographics Association and Blackwell Publishers 2001.



(a) (b) (c) (d)

Figure 3: Four renderings from a CT head (a and b), the Marschner & Lobb data sets (c) and a MR brain (d).

$$= n \times \left((2 \times S_{\min\max}) + \frac{1}{7} \times (2 \times S_{\min\max} + S_{Pointer}) \right)$$

Where n is the number of voxels within the volume, S_{minmax} is the size of the min (or max) value and $S_{pointer}$ the size of the pointer. If a voxel value is stored with 16 bits and pointer with 32 bits, then the size of the octree is approximately 2.5 times the size of the volume (without allowing for the compression possibilities of insignificant space region). Experimental results are very close to this formula. Although the standard memory size seems good enough nowadays, it could be a problem. In this case, we suggest not storing the last level of the octree. However, more voxels will be processed.

During the rendering, the octree will be traversed in a front-to-back order. All the nodes where the min and max values do not enclose the surface threshold will be skipped. Thus this method allows the user to interactively change the threshold, which is not possible in some other methods, like the marching-cube algorithm where the triangular mesh must be recomputed.

5. Experimental Results

Experimental results will be analysed here according to two criteria: the speed rendering and the quality rendering. Three data sets will be used for the first criterion: two well-known UNC data sets (a 256×256×225×8 CT head and a 256×256×167×8 MR brain) plus the synthetic Marschner & Lobb data set[14] (41×41×41×8). An additional MR data set (256×256×73×16) will be used in the second part. All the renderings have been performed on a modest platform with a 900 MHz AMD Athlon processor associated with 256 MB. The sampling rate along a ray has been set to 16, the number of subdivision within a pixel and the number of precomputed rays have both been set to 64×64.

Rendering Times

Four renderings have been tested here (See figure 3) with three image sizes. The head data set has been rendered with both a face threshold and a bone threshold. Results (See table 1) show that interactive rendering (around one second) is possible with our method on classical image sizes. We came to a surprising observation: times do not seem to be volume size dependent. Although the Marschner & Lobb times are close to other volumes, its size is two hundred times lower. The rendering time seems to be more affected by the image size, like the raycasting algorithms.

In fact, the time the algorithm needs to run the volume is negligible in comparison with the time the algorithm needs to compute the polynomial coefficients. Furthermore, a voxel that belongs to a small volume will project on more pixels (or rays) than a big volume voxel does. Actually, the total number of projected pixels is approximately constant in both cases, and it only depends on the image size.

Data set	Size	Image	Mean	Min	Max
Head (face)	256^2×225	256^2	0,61	0,51	0,76
Head (face)	256^2×225	512^2	1,6	1,2	2
Head (face)	256^2×225	768^2	2,9	2,2	3,8
Head (Skull)	256^2×225	256^2	0,51	0,42	0,59
Head (Skull)	256^2×225	512^2	1	0,8	1,3
Head (Skull)	256^2×225	768^2	1,9	1,3	2,8
Brain	256^2×167	256^2	0,74	0,49	0,88
Brain	256^2×167	512^2	1,7	1,1	2,1
Brain	256^2×167	768^2	3,9	3,6	4,5
Marschner&Lobb	41^3	256^2	0,22	0,12	0,28
Marschner&Lobb	41^3	512^2	0,8	0,4	1
Marschner&Lobb	41^3	768^2	1,9	1,2	2,7

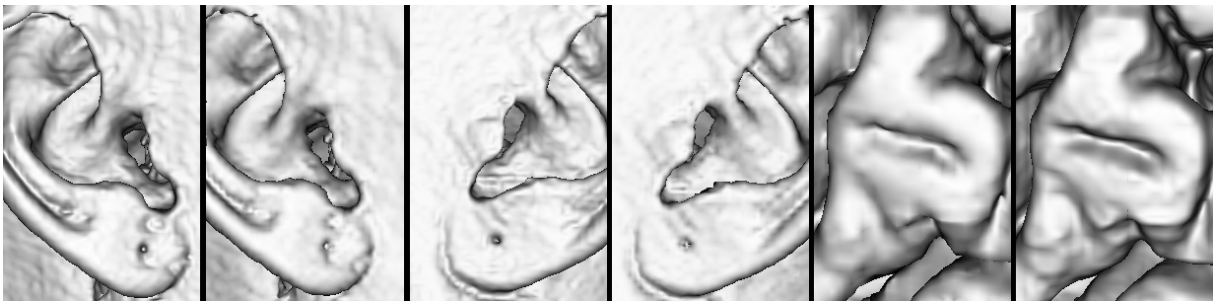
Table 1: Rendering times (in Seconds)

Data Set	Image	Relevant rays	Total rays	Ratio
Head	512^2	158K	728K	4.6
Brain	512^2	123K	901K	7.3
Marschner&Lobb	512^2	123K	972K	7.9

Table 2: Octree efficiency

In order to test the min-max octree, profiling tests have been done. The results are summed up in table 2. The relevant rays' column indicates the number of rays intersecting the volume (i.e. the number of pixels contributing to the final image). The total rays' column is the total number of ray-voxel intersections computed. This is also the number of times the C_i polynomial coefficients

© The Eurographics Association and Blackwell Publishers 2001.



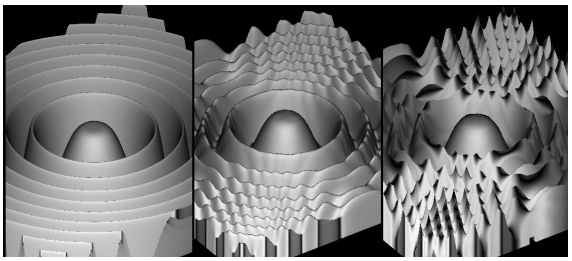
(a) (b) (c) (d) (e) (f)

Figure 4: Zooms of the brain Volume using parametric cubes (a, c, f) and ray-casting (b, d, e)

are computed. The ratio between those two columns can be interpreted as the average number of visited voxels per ray.

Results are really attractive because a ray intersects the isosurface around the seventh voxel (see ratio). If we allow for the total number of pixels instead of the relevant rays, results are even better.

In future works, we will try to improve those results. Here, the program has been implemented with the C language. However we did not have enough time neither to optimise the equation implementation, to perform adaptive sampling, nor to accelerate the implementation with the use of SIMD instructions. We expect our program to be at least twice faster with those optimisations.



(a) (b) (c)

Figure 5: Marschner & Lobb data set: ideal (a), parametric cubes (b) and ray-casting (c)

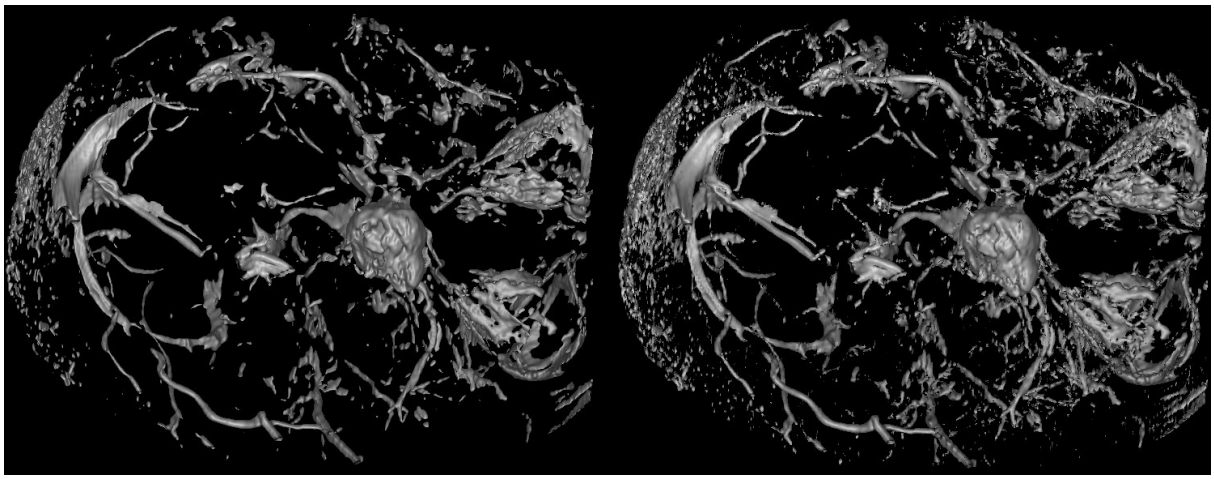
Image Quality

This is probably the most difficult thing to evaluate. The ray-casting algorithm is considered as one of the best methods currently used. Thus, a high-quality ray-casting algorithm has been implemented in order to make a comparison. It uses trilinear interpolation along the ray and normals are also trilinearly interpolated before the shading operation. For a better comparison, the ray-casting has the same parameters, like the shading parameters, the threshold value or the sampling rate along a ray, even if 16 samplings per ray are never used in practice with the standard implementations.

Like other studies[17, 22, 23] about the rendering quality, we also use the Marschner & Lobb data set[14] for the first test. So the reader is invited to take a look at those publications for a better understanding of this section. The principal property of this data set is that the main part of the frequencies is right below the Nyquist frequency. Three rendering examples are given in figure 5.

This figure clearly shows the superiority of our new filter in the high frequencies domain. The rendering made with parametric cubes is closer to the ideal rendering of the function than the one made with ray-casting.

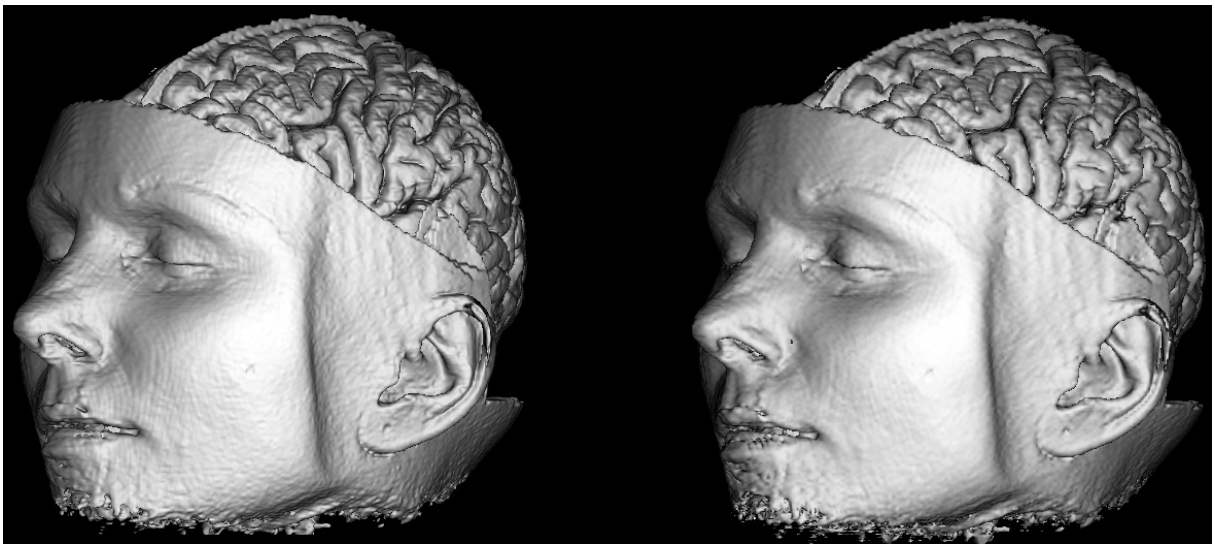
However, high frequencies are only visible in tiny details. In order to see those differences in real situations,



(a) (b)

Figure 6: Two renderings of an MR brain using Parametric Cubes (a) and Ray-casting (b)

© The Eurographics Association and Blackwell Publishers 2001.



(a) (b)

Figure 7: Whole brain data set rendered with both parametric cubes (a) and ray-casting (b)

renderings of the brain data set have been magnified. Results are available in figure 4. Looking for significant tiny details is not easy, since we do not know the real shape of the volume. However it appears that the brain data set has two earring holes (fig. 4a, fig. 4b, fig. 4c, fig. 4d), and so we have magnified those details. One can see that the earring holes rendered using our new filter are closer to the reality than those with the ray-casting algorithm. Two magnifications have also been done on the brain (fig. 4e, fig. 4f) clearly showing that more details are visible with our method.

Another interesting property of the new filter proposed here is its ability to smooth local maximums within the signal, which is helpful when noise is important. We use an additional MR brain data set to show it (fig. 6). Although the two images have been rendered with the same threshold, the image rendered with our filter is less noisy than with a traditional ray-casting. However, larger objects are preserved. This property should be a great improvement for the visualization of noisy MR datasets, like angiography renderings.

The last point of our study is about normal smoothing. Normals computed with the central difference gradient (cf. section 2) are better smoothed than the middle difference gradient (see fig.7). It can be helpful when one wants a nice visualization, or when the volume is insufficiently sampled. In figure 7, the hatching is clearly reduced on the ray-casting generated image. However, it is also a loss of information in general.

6. Conclusion and Future Works

We have shown here a new two-order filter having some remarkable qualities. Its application to the visualization of isosurfaces has been successfully applied with the use of parametric cubes. The image accuracy is better than the one produced with a ray-casting algorithm. We have also demonstrated the potentialities of a min-max octree for isosurface visualization, and especially with parametric cubes. Thus our algorithm reaches an interactive frame rate on standard volumes and standard image sizes with an orthogonal projection, which has never been reported before for this level of quality.

However, our algorithm is currently restricted to isosurface renderings. By both optimising the algorithm and the use of assembled instructions, we will try to adapt it for direct volume renderings. Finally the parametric cubes could be used in the same way to accelerate other methods like trilinear ray-casting, or more complex filters like cubic filters.

References

1. I. Gargantini, “Linear octtrees for fast processing of three-dimensional objects”, computer graphics and image processing”, vol. 20, 1982, pp. 365-374.
2. J. Kajiya and B. Von Herzen, “Ray tracing volume densities”, SIGGRAPH’84, July 1984, pp. 165-174.
3. L.S. Chen, G.T. Herman, R.A. Reynolds, J.K. Udupa: “Surface shading in the cuberille environment”, IEEE Computer Graphics and Applications 5(12), 1985, pp. 33-43.
4. W. Lorensen and H. Cline, “Marching cubes: A high resolution 3D surface construction algorithm”, SIGGRAPH’87, 1987, pp. 163-169.
5. J. Amanatides and A. Woo, “A fast voxel traversal algorithm for raytracing”, Eurographics’87, 1987, pp. 3-9.

© The Eurographics Association and Blackwell Publishers 2001.

6. H.E. Cline, W.E. Lorensen, S. Ludke, C.R. Crawford and B.C. Teeter, "Two algorithms for the three-dimensional reconstruction of tomograms", Medical physics, may 1988, vol. 15 no. 3, pp.320-327.
7. P. Sabella, "A rendering algorithm for visualizing 3D scalar fields", SIGGRAPH'88, 1988, pp. 51-58.
8. R. A. Drebin, L. Carpenter and P. Hanrahan, "Volume rendering", SIGGRAPH'88, 1988, pp. 65-74.
9. M. Levoy, "Display of surfaces from volume data", IEEE Comp. Graph. & App., Vol. 8, no. 5, 1988, pp. 29-37, 1988.
10. M. Levoy, "Efficient raytracing of volume data", ACM Transactions on graphics, vol. 9, no. 3, 1990, pp. 245-261.
11. L. Westover, "Footprint evaluation for volume rendering", SIGGRAPH'90, 1990, pp. 367-376.
12. R. Yagel and A. Kaufman, "Template-based volume viewing", Proc. Eurographics'92, vol.11, no.3, pp. 153-167.
13. P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation", SIGGRAPH'94, 1994, pp. 451-458.
14. S.R. Marschner and R.J. Lobb, "An evaluation of reconstruction filters for volume rendering", Proceedings of visualization'94, October 1994, pp. 100-107.
15. N. Stolte and R. Caubet, "Discrete ray-tracing of Huge Voxel Spaces", Eurographics'95, vol. 14, no. 3, 1995, pp. 383-394.
16. G. Thurmer and C. A. Wuthrich, "Normal Computation for discrete surfaces in 3D space", Eurographics'97, Vol. 16, no. 3, pp. 15-26.
17. T. Moller, R. Machiraju, K. Mueller and R. Yagel, "Evaluation and design of filters using a Taylor series expansion", IEEE Transaction on visualization and computer graphics, vol.3, no. 2, April 1997, pp. 184-190.
18. K. Mueller, T. Möller and R. Crawfis, "Splatting without the blur", Proc. Visualization'99, 1999, pp. 363-371.
19. M. Wan, A. Kaufman and S. Bryson, "high performance presence-accelerated ray casting", proc. Visualization'99, 1999, pp. 363-371.
20. B. Mora, J.P. Jessel and R. Caubet, "Accelerating volume rendering with quantified voxels", IEEE/ACM SIGGRAPH Volume visualization and graphics symposium 2000, October 2000, pp. 63-70.
21. G. Knittel, "The Ultravis System", IEEE/ACM SIGGRAPH Volume visualization and graphics symposium 2000, October 2000, pp. 71-78.
22. M. Meiβner, J. Huang, D. Bartz, K. Mueller, R. Crawfis, "A practical comparison of popular volume rendering algorithms", IEEE/ACM SIGGRAPH Volume visualization and graphics symposium 2000, October 2000, pp. 81-90.
23. T. Theubl, H. Hauser,E. Gröller, "Mastering windows: Improving reconstruction", IEEE/ACM SIGGRAPH Volume visualization and graphics symposium 2000, October 2000, pp. 101-108.
24. J. Huang, K. Mueller, N. Shareef, R. Crawfis, "Fast splats: optimized splatting on rectilinear grids", IEEE Visualization'00 proceedings, October 2000.
25. L. Mroz, H. Hauser, E. Gröller, "Interactive high quality maximum intensity projection", Eurographics'00, vol. 19, no. 3, 2000.
26. L. Neumann, B. Csébfalvi, A. König and E. Gröller, "Gradient Estimation in Volume Data using 4D Linear Regression", Eurographics'00, vol. 19, no. 3, 2000.
27. F. Dong, M. A. Krokos and G. J. Clapworthy, "Fast Volume Rendering and Data Classification Using Multiresolution Min-Max Octrees", Eurographics'00, vol. 19, no. 3, 2000.

© The Eurographics Association and Blackwell Publishers 2001.