Accurate volume rendering of unstructured hexahedral meshes

Fábio Markus Miranda Waldemar Celes
Tecgraf/PUC-Rio - Computer Science Department
Pontifical Catholic University of Rio de Janeiro, Brazil
{fmiranda, celes}@tecgraf.puc-rio.br

***Abstract*—Important engineering applications use unstructured hexahedral meshes for numerical simulations. Hexahedral cells, when compared to tetrahedral ones, tend to be more numerically stable and to require less mesh refinement. However, volume visualization of unstructured hexahedral meshes is challenging due to the trilinear variation of scalar fields inside the cells. The conventional solution consists in subdividing each hexahedral cell into five or six tetrahedra, approximating a trilinear variation by a piecewise linear function. This results in inaccurate images and increases the memory consumption. In this paper, we present an accurate ray-casting volume rendering algorithm for unstructured hexahedral meshes. In order to capture the trilinear variation along the ray, we propose the use of quadrature integration. A set of computational experiments demonstrates that our proposal produces accurate results, with reduced memory footprint. The entire algorithm is implemented on graphics cards, ensuring competitive performance.**

***Keywords*-volume rendering, hexahedral mesh, unstructured mesh, ray integral**

I. Introduction

Volume rendering is a popular way to visualize scalar fields from a number of different data, from medical MRI to results of scientific simulations. By computing the light intensity along rays as they traverse the volume, it is possible to calculate the color and opacity for the pixels on the screen. How to accurately calculate the interaction between light and volume, while maintaining an acceptable rendering time, is one of the main difficulties of volume visualization algorithms.

One approach to render volumetric data is ray-casting [1]. By tracing a ray for each pixel on the screen, we can traverse the volume and calculate the contribution of a set of discrete volume cells to the final pixel color, using an emission-absorption optical model [2]. The volume dataset is usually decomposed into tetrahedral and hexahedral cells. As the ray traverses the volume, its color and opacity are calculated taking into consideration the variation of the scalar field inside each cell. As the number of nodes per cell increases, so does the order of the scalar function inside the cell.

For obvious reasons, a linear variation of the scalar field translates into a much simpler interaction between the ray and the volume. That is why the common approach to render hexahedral meshes is to split each cell into five or six tetrahedra, approximating the trilinear variation of the scalar function by a piecewise linear function. This results in inaccurate image and increases the memory consumption. In this paper we present an high-accurate ray-casting volume rendering algorithm for unstructured hexahedral meshes that considers the trilinear variation of the scalar field inside the cells, and uses a quadrature integration scheme to calculate the interaction between light and volume. We evaluate our solution considering rendering time, image quality, and memory efficiency, and compare it against a tetrahedral subdivision solution.

The main contribution of this paper is to propose and evaluate an integration scheme that considers the trilinear scalar function of an hexahedral cell, implemented on graphics hardware. As far as we know, no other work has proposed direct volume rendering of unstructured hexahedral meshes considering the trilinear variation of the scalar field inside the cells.

This paper is organized as follows: Section II reviews previous works on volume rendering related to our proposal. Section III details our method. In Section IV, we evaluate the achieved results, and Section V concludes the paper.

II. Related Work

A. Ray integration

The emission-absorption optical model, proposed by Williams and Max [2], computes the interaction between the light and the volume, within each cell, using the following equation:

$$\begin{aligned} I(t_b) &= I(t_f)e^{-(\int_{t_f}^{t_b} \rho(t)dt)} \\ &+ \int_{t_f}^{t_b} e^{-(\int_{t}^{t_b} \rho(u)du)}\kappa(t)\rho(t)dt \end{aligned} \quad (1)$$

where t_f and t_b are the ray length from the eye to the entry and exit points of a cell, respectively; $f(t)$ is the scalar function inside the cell, along the ray; $\rho(t)$ is the light attenuation factor, and $\kappa(t)$ is the light intensity, both given by a transfer function.

Evaluating such integral accurately and efficiently is one of the main difficulties faced by volume rendering algorithms. Williams et al. [3] first proposed to simplify the transfer function as a piecewise linear function. They introduced the concept of *control points*, which represent points where the transfer function (TF) is non-linear (the TF in Figure 1b presents, for example, 5 control points). A later work by Röttger et al. [4] proposed to use pre-integration, storing the parameterized result in a texture, accessed by the entry scalar value, exit scalar value, and ray length. Their proposal works for any transfer function, but any change on the transfer function requires the pre-integration to be recomputed. Röttger [5] later proposed to utilize the GPU to accelerate the precomputation of the 3D table. Moreland et al. [6] re-parameterized the pre-integration result, turning it independent of the transfer function, but under the assumption that the transfer function was piecewise linear. The pre-integration result was then stored in a 2D texture, accessed via the normalized values of s_f and s_b (the scalar value at the entry and exit points of the cell). However, the pre-integration results are computed by assuming a linear scalar field variation inside the cell.

Novins and Arvo [7] presented a study about the accuracy of different numerical integration schemes of the ray integral, considering a maximum precision error bound.

B. Hexahedral mesh

A ray-casting algorithm for unstructured meshes was first presented by Garrity et. al [1]. Weiler et. al [8] later proposed a GPU solution using shaders. The main idea is to cast a ray for each screen pixel, and then to traverse the intersecting cells of the mesh until the ray exits the volume. In order to properly traverse the mesh, we have to store an adjacency data structure in textures; the algorithm will then fetch these textures and determine to which cell it has to step to. At each traversal step, the contribution of the cell to the final pixel color is given by evaluating the ray integral (Equation (1)).

Volume rendering of unstructured hexahedral meshes was explored by Shirley et. al [9] and Max et. al [10], where they proposed to subdivide each hexahedron into five or six tetrahedra in order to properly render the volumetric data, approximating the trilinear scalar variation by a piecewise linear function. This not only increases the memory consumption but also decreases the rendering quality. Carr et. al [11] focused on regular grids and discuss schemes for dividing a hexahedral mesh into a tetrahedral one, comparing rendering quality for isosurface and volume rendering.

One of the first proposals to consider something more elaborated than a simple subdivision scheme was made by Williams et al. [3]; the authors, however, focused on cell projection of tetrahedral meshes, only making small notes about how the algorithm could handle hexahedral cells, but did not discuss the results. Recently, Marmitt et al. [12] proposed an hexahedral mesh ray-casting, focusing on the traversal between the elements, but neglecting to mention how they integrated the ray over the trilinear scalar function of a hexahedron.

Marchesin et. al [13] and El Hajjar et. al [14] proposed solutions to structured hexahedral meshes, focusing on how the ray can be integrated over the trilinear scalar function of a regular hexahedron. The first one proposed to approximate the trilinear function by a bilinear one. They then stored a pre-integration table in a 3D texture, where each value was accessed by the scalar values at the ray enter, middle, and exit points. To avoid the use of a 4D texture, they consider a constant ray step size. El Hajjar et. al [14] approximated the trilinear scalar function by a linear one and used the same pre-integration table proposed by [4], accessed via the scalar values at enter and exit points, and the ray length.

These papers made the assumption that the scalar function is either linear or bilinear to calculate an integral that could be stored in a texture with feasible dimensions. Also, their proposals do not support interactive modifications of the transfer function, because the pre-integration table must be recalculated for each TF change. In this paper, we avoid the use of pre-integration and propose the use of a quadrature approach to integrate the ray, supporting interactive modifications of the TF. We consider the actual trilinear scalar function and thus achieve accurate results.

III. Hexahedron Ray-Casting

In this section, we describe how our ray-casting algorithm handles hexahedral meshes. In Section III-A, we detail our integration scheme. Section III-B describes how we find the integration intervals. Section III-C presents our data structure, and Section III-D describes how we traverse the hexahedral mesh. Section III-E details how we also render isosurfaces, and, finally, Section III-F presents the overall ray-casting algorithm.

A. Ray integration

The trilinear scalar function inside a hexahedral cell can be described with the following equation:

$$\begin{aligned} f(x,y,z) &= c_0 + c_1x + c_2y + c_3z \\ &+ c_4xy + c_5yz + c_6xz + c_7xyz \end{aligned} \quad (2)$$

where $c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7$ are the cell coeficients. They are calculated solving the following linear system:

$$\begin{pmatrix} 1 & x_0 & y_0 & \cdots & x_0y_0z_0 \\ 1 & x_1 & y_1 & \cdots & x_1y_1z_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_7 & y_7 & \cdots & x_7y_7z_7 \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c7 \end{pmatrix} = \begin{pmatrix} s_0 \\ \vdots \\ s_7 \end{pmatrix}$$

where $\{x_i, y_i, z_i\}$, with $i = \{0, \ldots, 7\}$, are the hexahedron cell vertex positions, and s_i are the scalar values at each one of the vertices. The system can be solved using singular value decomposition (SVD) [15].

The position of a point inside the cell, along the ray, can be described as:

$$\mathbf{p} = \mathbf{e} + t\vec{d} \quad (3)$$

where $\mathbf{e}$ is the eye position, and $\vec{d}$ is the ray direction.

We then parameterize the hexahedral scalar function (Equation (2)) by the ray length inside the cell, denoted by t:

$$f(t) = w_3t^3 + w_2t^2 + w_1t + w_0, t \in [t_{back}, t_{front}] \quad (4)$$

with:

$$\begin{aligned}
w_0 &= c_0 + c_1 e_x + c_2 e_y + c_4 e_x e_y + c_3 e_z \\
&+ c_6 e_x e_z + c_5 e_y e_z + c_7 e_x e_y e_z \\
&+ c_7 d_x d_y e_z \\
w_1 &= c_1 d_x + c_2 d_y + c_3 d_z + c_4 d_y e_x + c_6 d_z e_x \\
&+ c_4 d_x e_y + c_5 d_z e_y + c_7 d_z e_x e_y + c_6 d_x e_z + c_5 d_y e_z \\
&+ c_7 d_y e_x e_z + c_7 d_x e_y e_z \\
w_2 &= c_4 d_x d_y + c_6 d_x d_z + c_5 d_y d_z + c_7 d_y d_z e_x + c_7 d_x d_z e_y \\
w_3 &= c_7 d_x d_y d_z \qquad (5)
\end{aligned}$$

Considering now the ray integral from Equation (1) and considering the transfer function as a piecewise linear function, we can express:

$$\kappa(f(t)) = \frac{(\kappa_b - \kappa_{front}) * (f(t) - f(t_{front}))}{f(t_{back}) - f(t_{front})} + \kappa_{front} \quad (6)$$

$$\rho(f(t)) = \frac{(\rho_b - \rho_{front}) * (f(t) - f(t_{front}))}{f(t_{back}) - f(t_{front})} + \rho_{front} \quad (7)$$

Getting back to Equation (1), we use a Gauss-Legendre Quadrature method to integrate the color and opacity along the ray:

$$\begin{aligned}
I(t_b) &= I(t_{front}) e^{-z_{t_{front}, t_{back}}} \\
&+ \int_{t_{front}}^{t_b} e^{-z_{t,tb}} \kappa(t) \rho(t) dt \qquad (8)
\end{aligned}$$

where

$$z_{a,b} = \int_a^b \rho(r) dr$$

$$\begin{aligned}
z_{a,b} = \rho_{front}(b-a) &+ \frac{(\rho_b - \rho_{front})}{12(s_{back} - s_{front})} \\
* &[12(a s_{front} - b s_{front}) + 12 w_0 (b-a) + 6 w_1 (b^2 - a^2) \\
&+ 4 w_2 (b^3 - a^3) + 3 w_3 (b^4 - a^4)] \qquad (9)
\end{aligned}$$

going back to Equation 8 have:

$$\begin{aligned}
I(t_b) &= I(t_{front}) e^{-z_{t_{front}, t_{back}}} \\
&+ \sum_{i=0}^{3} D * GaussPoint_i * e^{-z_{t_g, tb}} \kappa(t_g) \rho(t_g) \quad (10)
\end{aligned}$$

where $D = (t_{back} - t_{front})$ and $t_g = t_{front} + GaussWeight_i * D$. $GaussPoint_i$ and $GaussWeight_i$ are the pre-computed points and weights for the Gauss-Legendre Quadrature.

The Gaussian quadrature integration method gives exact solutions for functions that are well approximated by polynomials up to degree 5, considering a 3 point quadrature. Since we split our integration into several intervals, as we shall explain in Section III-B, we make sure that our ray integral is accurately evaluated.

B. Integration intervals

The scalar field variation along a ray in a hexahedral cell can be illustratively represented by the function in Figure 1a. As a cubic polynomial function (according to Equation (4)), $f(t)$ has two extrema, which can be calculated from its derivative $f'(t)$, a quadratic polynomial. Considering t_{min} and t_{max} the values of minima and maxima, we calculate t_{near} and t_{far}, such that $t_{near} = min(t_{min}, t_{max})$ and $t_{far} = max(t_{min}, t_{max})$. If t_{front} denotes the point the ray enters the cell and t_{back} the point it exits the cell, the function in the intervals $[t_{front}, t_{near}]$, $[t_{near}, t_{far}]$, and $[t_{far}, t_{back}]$ is monotonic, so each one of these intervals has, at most, one root of Equation (4).

To find if there is an isovalue inside an interval, we use a 2D texture first proposed by Röttger et. al [4] for his tetrahedral cell-projection algorithm. Given s_0 and s_1 (scalars at the limit points) as parameters, this texture returns the value of the first control point s_{cp}, if one exists, such that $s_0 < s_{cp} < s_1$ or $s_1 < s_{cp} < s_0$.

With s_{cp}, we can find the value of t_{cp}, which is the ray length from the eye to the isosurface that crosses the hexahedral. We find it by solving Equation (4) for $f(t_{cp}) = s_{cp}$ using the Newton-Raphson method. One of the main problems with such root finding method is its dependency of an initial guess, but we can use the average ray length between the interval limits as our initial guess.

To exemplify this procedure, let us consider the scalar variation inside an hexahedron given by the function in Figure 1a and the transfer function illustrated in Figure 1b. We have $t_{max} = 0.26$ and $t_{min} = 0.73$; we then calculate $t_{near} = min(t_{min}, t_{max}) = 0.26$ and $t_{far} = max(t_{min}, t_{max}) = 0.73$. In this case, there is no control point in $[t_{front}, t_{near}]$, and so the first integration interval is $[0, 0.26]$. In $[t_{near}, t_{far}]$, there are three control points: 0.4, 0.5, and 0.6. These give us four integration intervals: $[0.26, 0.4]$, $[0.4, 0.5]$, $[0.5, 0.6]$, and $[0.6, 0.73]$. Beyond t_{far}, there is no other control point, giving us only an additional interval to complete this illustrative integration: $[0.73, 1]$.

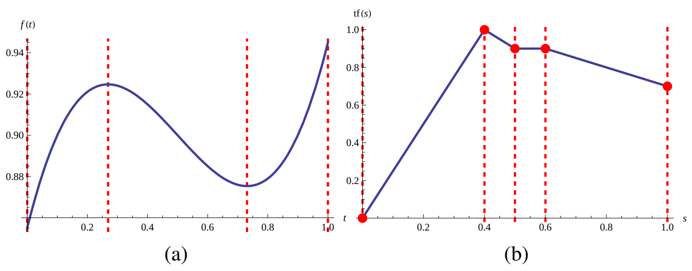


Fig. 1. Example of scalar field variation inside a hexahedral cell: (a) Maxima and minima values of a trilinear function along the ray inside an hexahedron; (b) Transfer function represented by a piecewise linear variation.

C. Data structure

In order to access information such as normals and adjacency, we use a set of 1D textures, presented in Table I.

TABLE I
DATA STRUCTURE FOR ONE HEXAHEDRAL CELL.

Texture	Data			
$Coef_i, i = \{0, ..., 7\}$	c_0	c_1	$\cdots$	c_7
$Adj_i, i = 0, ...5$	adj_0	adj_1	$\ldots$	adj_5
$\vec{p}_{i,j}, i = \{0, ..., 5\}, j = \{0, 1\}$	$vecn_{0,0}$	$vecn_{0,1}$	$\ldots$	$vecn_{5,1}$

As mentioned, we need to store 8 coefficients per cell. For adjacency information, we need more 6 values, each associated to a face of the cell. The third line in the table represents plane equations defined by the cell faces. To compute the intersection of the ray with a hexahedral cell, we use a simple ray-plane intersection test. We then need to split each quadrilateral face of a cell into two triangles and store the corresponding plane equations, totaling 12 planes per cell (48 coefficient values).

We then store a total of 62 values associated to each cell. Considering 4 bytes per value, we store 248 bytes per cell. Even optimized data structures, such as the one described by Weiler et. Al [16], requires at least 380 or 456 bytes per cell (considering a hexahedron subdivided into five or six tetrahedra, respectively). In fact, one great advantage of ray-casting hexahedral cells is its small memory consumption when compared to the subdivision scheme.

D. *Ray traversal*

To begin the ray traversal through the mesh, we follow the work by Weiler et. al [16] and Bernardon et. al [17]. They proposed a ray-casting approach based on depth-peeling that handles models with holes and gaps. The initial step consists in rendering to a texture the external volume boundary, storing the corresponding cell ID for each pixel on the screen. The second step will then fetch the texture and initiate the mesh traversal starting at the stored cell. The algorithm then proceeds by traversing the mesh until the ray exits the volume. In models with holes or gaps, the ray can re-enter the volume. At each peel, the ray re-enters at the volume boundary, and we accumulate the color and opacity from previous peels. The ray-casting algorithm is finished when the last external boundary of the model is reached.

As mentioned, to compute the intersection of the ray with a cell, we subdivide each cell face in two triangles and compute the distance between the eye and each triangle plane. This intersection test is an approximation. In order to improve accuracy, we have implemented both a ray-bilinear patch intersection test and a ray intersection test in Plcker Space [12]. However, neither of these solutions performed well in our implementation. We then decided to stick with a simpler ray-plane intersection algorithm.

E. *Isosurfaces*

We can extend our volume rendering algorithm to also handle isosurface rendering. This is, in fact, very simple, because we already compute all control points along the ray. The surface normal is given by the gradient of the scalar field:

$$\vec{n} = \nabla f(x,y,z) = < \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} >$$

$$\vec{n} = \begin{bmatrix} c_1 + c_4 y + c_6 z + c_7 yz \\ c_2 + c_4 x + c_5 z + c_7 xz \\ c_3 + c_5 x + c_6 y + c_7 xy \end{bmatrix} \tag{11}$$

where (x, y, z) is the intersection between the ray and the isosurface, and is given by Equation (3), considering t_{cp}.

Figures 2 presents an isosurface rendering of the Atom9 Dataset, from [11]. We chose to use the same isovalue (0.12) as the one used by the authors of the paper. As can be noted, if compared to a simple subdivision scheme, our proposal depicts the isosurface shape with significant improved accuracy.

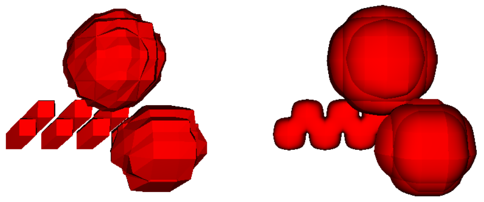

(a) Tetrahedral approach (b) Our proposal

Fig. 2. Isosurface rendering of the Atom9 dataset.

F. *Ray-casting Algorithm*

The algorithm in Table II summarizes our approach. We traverse the mesh accumulating each cell contribution to the pixel color. We first calculate the values of minima and maxima of the ray as it goes through each cell, clamping values outside the cell boundary. t_{near} and t_{far} represent the closest and farthest min/max value to the eye position. We then iterate through t_{front}, t_{near}, t_{far}, t_b, fetching the texture described in Section III-B to find if there are control points in each interval. If there is, the algorithm integrates from the current position t_i to t_{cp}; we then update the value of t_i.

IV. RESULTS

For the evaluation of our proposal, we implemented a ray-casting using CUDA that handles both tetrahedral and hexahedral meshes. We measured the rendering performance with four known volumetric datasets: Blunt-fin, Fuel, Neghip, Oxygen.

We tested the following algorithms for comparison:

1) **Hexa**$_{const}$: Ray-casting for hexahedral meshes that uses a fixed number of 100 steps in each cell to accurately compute the illumination assuming a constant scalar field at each step.
2) **Hexa**$_{ours}$: Our proposal for direct volume rendering of hexahedral meshes.
3) **Tetra**$_{HARC}$: Ray-casting using a pre-integrated table [18], with each hexahedral cell subdivided in six tetrahedra.

TABLE II
RAY-CAST ALGORITHM

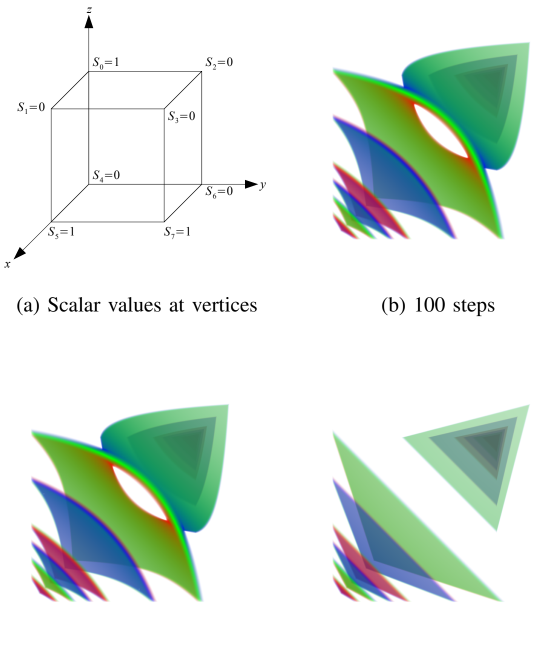
```
$color \leftarrow (0, 0, 0, 0)$
$cell.id \leftarrow VolumeBoundary()$
**while** $color.a < 1$ and ray inside volume **do**
    $t_{back}, cell.nid \leftarrow IntersectRayFaces(t, cell)$
    $t_{min}, t_{max} = Solve(cell.f'(t) = 0)$
    $t_{min} = clamp(t_{min}, t_{front}, t_{back})$
    $t_{max} = clamp(t_{max}, t_{front}, t_{back})$
    $t_{near} = min(t_{min}, t_{max})$
    $t_{far} = max(t_{min}, t_{max})$
    $t_i = [t_{front}, t_{near}, t_{far}, t_{back}]$
    $i = 0$
    **while** $i < 3$ **do**
        {Find control points}
        $s_{cp} = fetch(s_{cp}, s_{i+1})$
        **if** $s_i < s_{cp} < s_{i+1}$ or $s_i > s_{cp} > s_{i+1}$ **then**
            $t_{cp} = Newton(cell.f(t) = s_{cp}, \frac{t_i + t_{i+1}}{2})$
        **else**
            $t_{cp} = t_{i+1}$
        **end if**
        $color \leftarrow Integrate(t_i, s_i, t_{cp}, s_{cp})$
        $t_i = t_{cp}$
        **if** $t_i >= t_{i+1}$ **then**
            $i{+}{+}$
        **end if**
    **end while**
    $cell.id = cell.nid$
    $t = t_{back}$
**end while**
```

Although the **Hexa**$_{const}$ algorithm is far from being an efficient solution, it produces accurate results, and we use it as our rendering quality reference. We ran the experiments on Windows 7 with an Intel Core 2 Duo 2.8 GHz, 4 GB RAM and a GeForce 460 GTX 1 GB RAM. The screen size was set to 800 x 800 pixels in all experiments.

A. *Rendering quality*

We evaluate our ray-casting algorithm regarding rendering quality. We first devised an experiment using synthetic volume data composed by only one hexahedral cell. Figure 3 shows the scalar values set to each vertex and presents the three images achieved by the three algorithms. For these images, we used a transfer function with six thin spikes, isolating six different slabs. Even though such a scalar field variation is rare in actual datasets, this synthetic test aims to show how the subdivision of an hexahedral can lead to unrecognizable results. The test also demonstrates that our proposal is capable of accurately rendering scalar fields with complex variations.

Figure 4 shows and compares the results achieved by the three algorithms for rendering the Fuel model. We can note that our proposal is quite accurate. Figures 4d and 4e shows the difference between the images achieved with our proposal



(a) Scalar values at vertices (b) 100 steps

(c) Our proposal (d) Hexahedron subdivision

Fig. 3. Rendering on a synthetic model composed by on hexahedron.

and the subdivision scheme when compared to our quality reference. In Figures 4f, 4g, and 4h, a zoomed view of the model clearly depicts the difference in quality of our algorithm when compared to the subdivision scheme.

Figure 5 shows the result of a similar experiment but now considering the Bucky model. As can be noted, the results are equivalent: our algorithm produces images with much better quality.

B. *Quantitative error analysis*

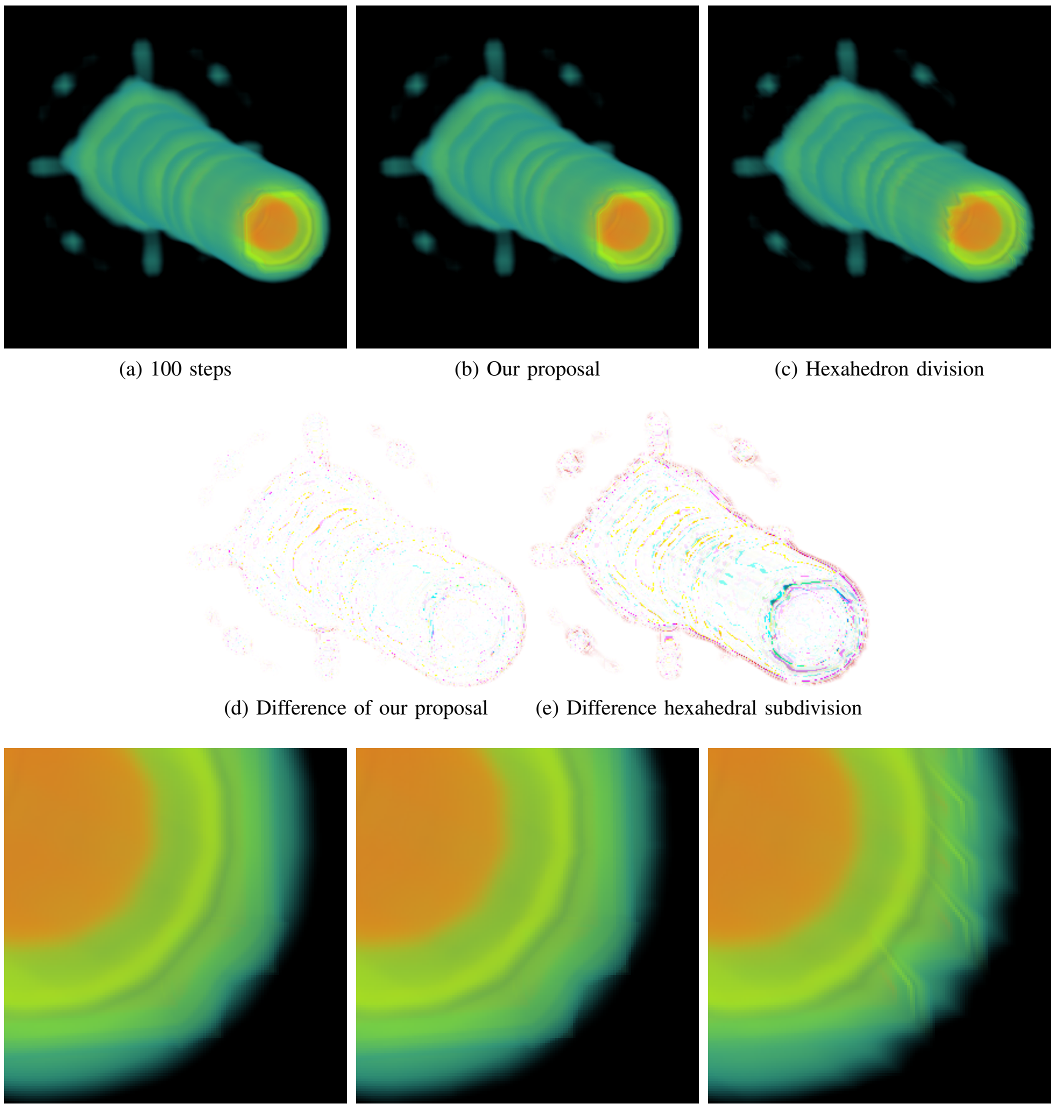
To properly evaluate the rendering quality of our proposal, we compute the peak signal-to-noise ratio (PSNR) and the structural similarity (SSIM) [19] of the rendered images in Figure 4 and Figure 5. The error values are presented in Table III. All values were obtained comparing the rendered images of the subdivision scheme and our proposal to the rendered images using constant steps. We observe that our proposal in fact presents better results.

TABLE III
QUANTITATIVE ERROR ANALYSIS OF THE SUBDIVISION SCHEME AND OUR PROPOSAL.

Figure	Algorithm	PSNR	SSIM
Bucky	$Tetra_{HARC}$	42.10	97.9%
	Hex_{ours}	58.79	99.9%
Fuel	$Tetra_{HARC}$	42.51	98.8%
	Hex_{ours}	55.65	99.9%

C. *Time results*

Table IV shows a comparison of memory consumption and rendering time. The time reported in the table represents the


(a) 100 steps (b) Our proposal (c) Hexahedron division

(d) Difference of our proposal (e) Difference hexahedral subdivision

(f) 100 steps (g) Our proposal (h) Hexahedral subdivision

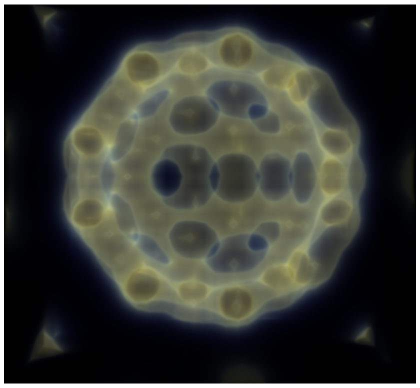
Fig. 4. Achieved images for the Fuel model.

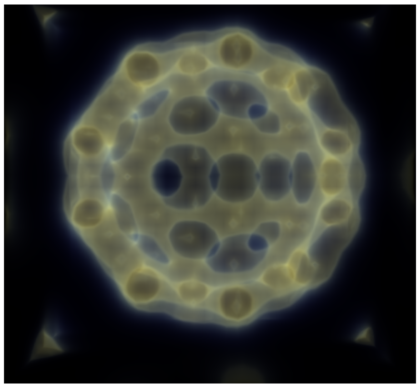
rendering time for one frame. For each entry, we repeated the experiment 5 times, and, for each run, we changed the camera to 64 different positions, averaging the measured time. Since the algorithms are heavily dependent on the transfer function, we tested them with a fixed TF with 23 control points.

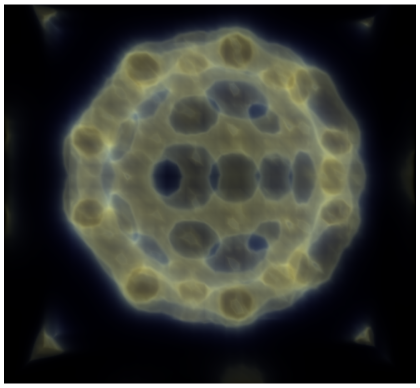
As can be noted, when compared to the subdivision scheme of 6 tetrahedra per hexahedral cell, our proposal reduces memory consumption by a factor of 2.2. Our algorithm also presents competitive performance. The rendering time of our approach is about 15% worse than the algorithm based on the subdivision scheme. To achieve accurate results, we need to perform part of the integration computation in double precision. That is the main reason for losing performance. If we had used single precision, our algorithm would be 15 to 20% faster than the subdivision scheme, but this would bring numerical inaccuracy. Figure 6 presents a comparison of the Bluntfin model rendered with both single and double precision. As can be noted, the use of single precision does result in inaccurate results.

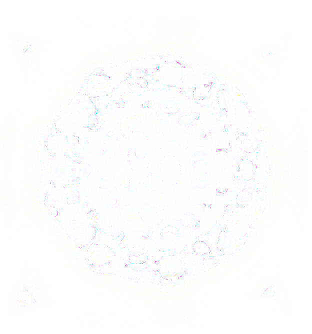
TABLE IV
RENDERING TIMES AND MEMORY FOOTPRINT OF THE SUBDIVISION SCHEME AND OUR PROPOSAL.

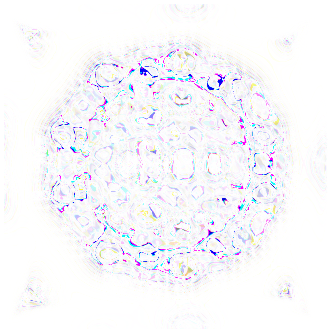
Model	Algorithm	# cells	Memory (Mb)	Time (ms)
Fuel	$Tetra_{HARC}$	1,572,864	150,994,944	159.41
	Hex_{ours}	262,144	65,011,712	177.27
Neghip	$Tetra_{HARC}$	1,572,864	150,994,944	152.39
	Hex_{ours}	262,144	65,011,712	177.34
Oxygen	$Tetra_{HARC}$	658,464	63,212,544	55.24
	Hex_{ours}	109,744	27,216,512	70.07
Blunt-fin	$Tetra_{HARC}$	245,760	23,592,960	48.57
	Hex_{ours}	40,960	10,158,080	56.50

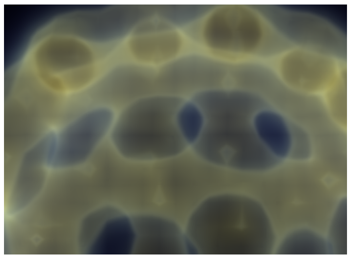

(a) 100 steps

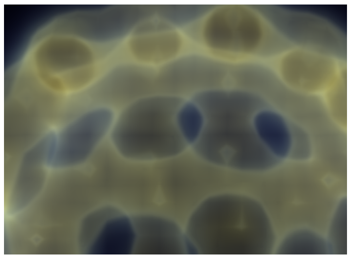

(b) Our proposal

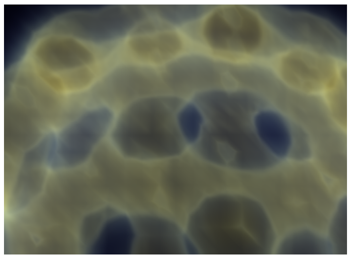

(c) Hexahedron division.

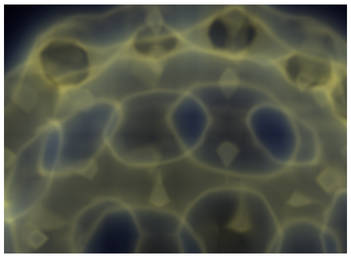

(d) Difference of our proposal

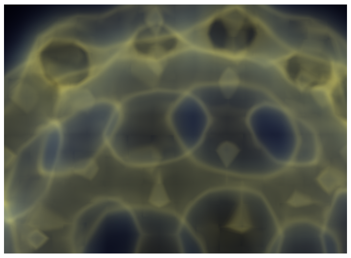

(e) Difference of subdivision


(f) 100 steps


(g) Our proposal


(h) Hexahedral subdivision


(i) 100 steps


(j) Our proposal

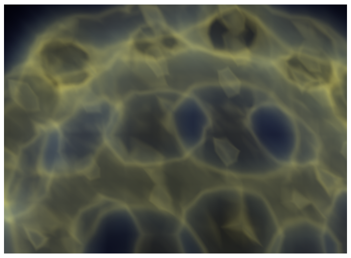
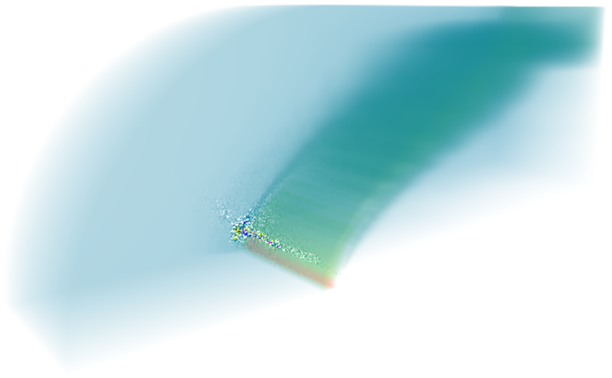

(k) Hexahedral subdivision

Fig. 5. Achieved images for the Bucky model.

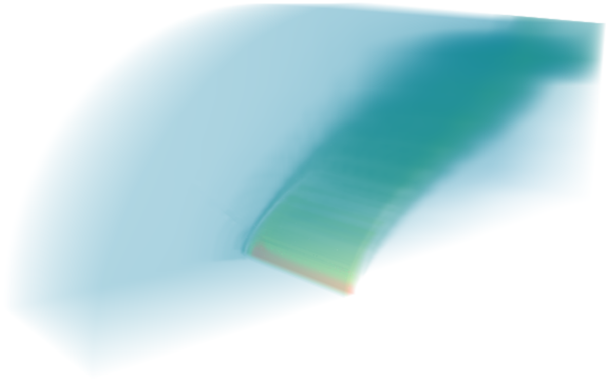
V. Conclusion

We have presented an accurate hexahedral volume rendering suitable for unstructured meshes. Our proposal integrates the trilinear scalar function using a quadrature approach on the GPU. Although our performance was about 15%-16% worse than a tetrahedral algorithm, our proposal produces images with better quality. By the use of a proper integration of the trilinear function inside an hexahedron, our proposal ensures that no isosurface value is missed during integration.

Because of its parallel nature, we believe that ray-casting is better suited for massively parallel environments, such as the GPU. Its main drawback, however, is its memory consumption; our algorithm presents a smaller memory footprint than regular hexahedral subdivision schemes.



(a) Single precision



(b) Double precision

Fig. 6. Volume rendering of the Bluntfin Dataset.

In the future, we plan to revisit the impact of a ray-bilinear patch intersection test, and to analyze how it could improve the image quality and its impact on the performance. We also plan to extend the algorithm to higher-order cells and to investigate its use for rendering models that support adaptive level-of-detail.

Acknowledgment

We thank CAPES (Brazilian National Research and Development Council) and CNPq (Brazilian National Research and Development Council) for the financial support to conduct this research. This work was done in the Tecgraf laboratory at PUC-Rio, which is mainly funded by the Brazilian oil company, Petrobras.

References

[1] M. P. Garrity, "Raytracing irregular volume data," in *Proceedings of the 1990 workshop on Volume visualization*, ser. VVS '90. New York, NY, USA: ACM, 1990, pp. 35–40. [Online]. Available: http://doi.acm.org/10.1145/99307.99316

[2] P. L. Williams and N. Max, "A volume density optical model," in *Proceedings of the 1992 workshop on Volume visualization*, ser. VVS '92. New York, NY, USA: ACM, 1992, pp. 61–68. [Online]. Available: http://doi.acm.org/10.1145/147130.147151

[3] P. L. Williams, N. L. Max, and C. M. Stein, "A high accuracy volume renderer for unstructured data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, pp. 37–54, 1998.

[4] S. Röttger, M. Kraus, and T. Ertl, "Hardware-accelerated volume and isosurface rendering based on cell-projection," in *Proceedings of the conference on Visualization '00*, ser. VIS '00. Los Alamitos, CA, USA: IEEE Computer Society Press, 2000, pp. 109–116. [Online]. Available: http://portal.acm.org/citation.cfm?id=375213.375226

[5] S. Röttger and T. Ertl, "A two-step approach for interactive pre-integrated volume rendering of unstructured grids," in *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, ser. VVS '02. Piscataway, NJ, USA: IEEE Press, 2002, pp. 23–28. [Online]. Available: http://portal.acm.org/citation.cfm?id=584110.584114

[6] K. Moreland and E. Angel, "A fast high accuracy volume renderer for unstructured data," in *Proceedings of the 2004 IEEE Symposium on Volume Visualization and Graphics*, ser. VV '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 9–16. [Online]. Available: http://dx.doi.org/10.1109/VV.2004.2

[7] K. Novins and J. Arvo, "Controlled precision volume integration," in *Proceedings of the 1992 workshop on Volume visualization*, ser. VVS '92. New York, NY, USA: ACM, 1992, pp. 83–89. [Online]. Available: http://doi.acm.org/10.1145/147130.147154

[8] M. Weiler, M. Kraus, M. Merz, and T. Ertl, "Hardware-based ray casting for tetrahedral meshes," in *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, ser. VIS '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 44–. [Online]. Available: http://dx.doi.org/10.1109/VISUAL.2003.1250390

[9] P. Shirley and A. Tuchman, "A polygonal approximation to direct scalar volume rendering," in *Proceedings of the 1990 workshop on Volume visualization*, ser. VVS '90. New York, NY, USA: ACM, 1990, pp. 63–70. [Online]. Available: http://doi.acm.org/10.1145/99307.99322

[10] N. L. Max, P. L. Williams, and C. T. Silva, "Cell projection of meshes with non-planar faces," in *Data Visualization: The State of the Art*, 2003, pp. 157–168.

[11] H. Carr, T. Moller, and J. Snoeyink, "Artifacts caused by simplicial subdivision," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 231–242, March 2006. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2006.22

[12] G. Marmitt and P. Slusallek, "Fast Ray Traversal of Tetrahedral and Hexahedral Meshes for Direct Volume Rendering," in *Proceedings of Eurographics/IEEE-VGTC Symposium on Visualization (EuroVIS)*, Lisbon, Portugal, May 2006.

[13] S. Marchesin and G. de Verdiere, "High-quality, semi-analytical volume rendering for amr data," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 6, pp. 1611 –1618, nov.-dec. 2009.

[14] J. E. Hajjar, S. Marchesin, J. Dischler, and C. Mongenet, "Second order pre-integrated volume rendering," in *IEEE Pacific Visualization Symposium*, March 2008.

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C (2nd ed.): the art of scientific computing*. New York, NY, USA: Cambridge University Press, 1992.

[16] M. Weiler, P. N. Mallon, M. Kraus, and T. Ertl, "Texture-encoded tetrahedral strips," in *Proceedings of the 2004 IEEE Symposium on Volume Visualization and Graphics*, ser. VV '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 71–78. [Online]. Available: http://dx.doi.org/10.1109/VV.2004.13

[17] F. F. Bernadon, C. A. Pagot, J. L. D. Comba, and C. T. Silva, "Gpu-based tiled ray casting using depth peeling," *Journal of Graphics, GPU, and Game Tools*, vol. 11, no. 4, pp. 1–16, 2006.

[18] R. Espinha and W. Celes, "High-quality hardware-based ray-casting volume rendering using partial pre-integration," in *Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 273–. [Online]. Available: http://portal.acm.org/citation.cfm?id=1114697.1115365

[19] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.