High-Quality, Semi-Analytical Volume Rendering for AMR Data

Stéphane Marchesin and Guillaume Colin de Verdière

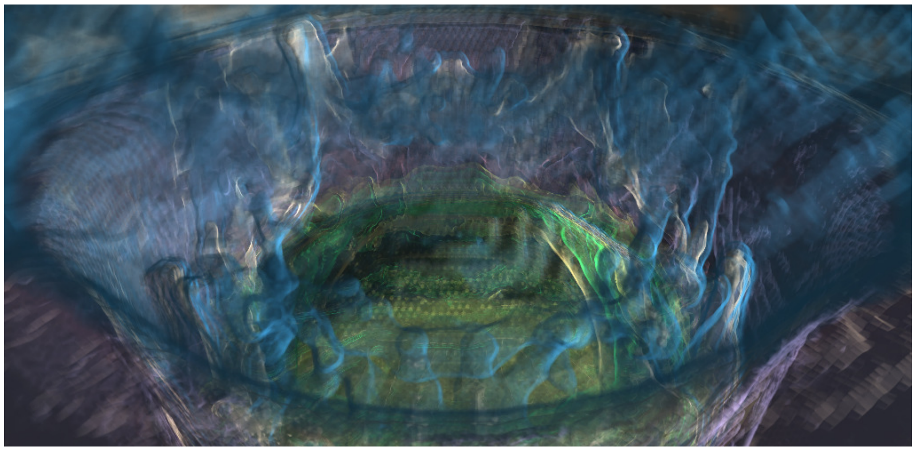

Fig. 1. Close-up of an AMR dataset showing a meteorite falling into the sea rendered using our system.

Abstract—This paper presents a pipeline for high quality volume rendering of adaptive mesh refinement (AMR) datasets. We introduce a new method allowing high quality visualization of hexahedral cells in this context; this method avoids artifacts like discontinuities in the isosurfaces. To achieve this, we choose the number and placement of sampling points over the cast rays according to the analytical properties of the reconstructed signal inside each cell. We extend our method to handle volume shading of such cells. We propose an interpolation scheme that guarantees continuity between adjacent cells of different AMR levels. We introduce an efficient hybrid CPU-GPU mesh traversal technique. We present an implementation of our AMR visualization method on current graphics hardware, and show results demonstrating both the quality and performance of our method.

Index Terms—Volume rendering, AMR data, Volume shading.

1 Introduction and Motivation

Adaptive mesh refinement (AMR) is a mesh refinement strategy aimed at reducing the cost of numerical simulations while maintaining high accuracy results. This design allows both simple programming thanks to identical cell shapes and implicit connectivity, and an efficient use of processing resources thanks to adaptive local refinement. Due to its simplicity and computational efficiency, this scheme is widely used in the numerical simulation field. Figure 2 shows an example of such a mesh in two dimensions. As an AMR mesh can be viewed (locally) as a tree, we define the AMR cell level as the level of cell size in the AMR tree, 0 being the biggest cells at the top of the tree, and higher levels being the smaller cells. We define an AMR patch as a cuboid of homogeneous cells, i.e. where all AMR cells have the same level. In this paper, we focus on the specific case of AMR meshes carrying vertex-centered data (as opposed to cell-centered data). In order to analyze the data resulting from AMR simulations, visualization tools are wanted. In particular, volume rendering is a powerful exploration method for 3D data. As of today, volume rendering of AMR data presents three major challenges:

1. A view-order cell traversal technique is needed. On the CPU, this problem can be solved easily and efficiently with existing data structures. However, direct adaptation of these structures to the GPU is difficult, and can result in a waste of memory or suboptimal performance. As we are interested in interactive visualization, such data structures should allow an efficient implementation. In particular, using additional cells to achieve simpler mesh traversal is not always desirable since it decreases the performance of the system.
2. An interpolation method is required inside the cells to reconstruct a continuous scalar function from the discrete AMR data. This implies defining an interpolant function inside the cells, not only for simple hexahedral cells, but also for cells lying at the border between patches of different levels. In the case of hexahedral cells, the most widespread choice for an interpolation function inside a given cell is the trilinear reconstruction. However, one still needs to restore the continuity at the borders between cells of different levels. This is commonly achieved by splitting cells into tetrahedra or pyramids, though this leads to a more complex mesh, and also needs an additional interpolation scheme per new cell type. This is further complicated by the requirement that different interpolation schemes must be coherent across shared cell faces.
3. High accuracy cell rendering techniques are highly sought after.

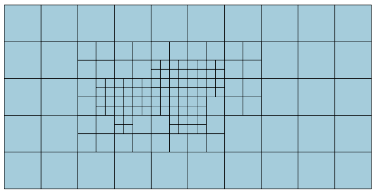

Fig. 2. AMR mesh example with three different cell levels.

• *Stéphane Marchesin and Guillaume Colin de Verdière, CEA, DAM, DIF, F-91297, Arpajon, France. E-mail: marchesi@ocre.cea.fr, guillaume.colin-de-verdiere@cea.fr.*

Manuscript received 31 March 2009; accepted 27 July 2009; posted online 11 October 2009; mailed on 5 October 2009.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org .

By linearly interpolating the signal over each integration interval, techniques like preintegration have historically been of great help in improving the visual quality of volume rendered scenes. However, because of insufficient sampling, these techniques can miss some isovalues. Figure 3 depicts such a situation where a range of isovalues present in in the reconstructed data is missed. This is an important source of artifacts when using preintegrated volume rendering as it leads to holes and cracks in the isosurfaces. Therefore, volume rendering methods that feature a higher rendering accuracy are required.

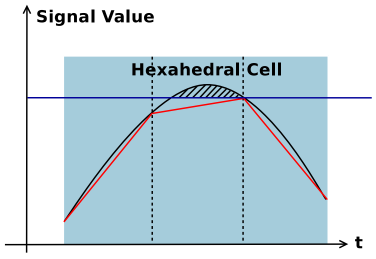

Fig. 3. A scalar signal approximated with 3 piecewise linear functions. The hashed area depicts the range of scalar values that are not covered by preintegration.

In this paper, we give solutions to those three core issues pertaining to AMR volume visualization. We present a hybrid CPU and GPU-based AMR mesh representation and traversal technique. We introduce a method suitable for restoring continuity along faces shared by AMR cells of different levels. Basing our work on a trilinear interpolation scheme inside AMR cells, we show how to improve the volume rendering quality with a sampling scheme that suits the discrete data and the reconstruction function in order to avoid artifacts. By ensuring that no isovalue is missed, our method greatly improves the visual quality of our final renderings while retaining interactive performance. Furthermore, we extend our cell rendering technique to handle volume shading.

2 Related works

Unstructured volume rendering is commonly achieved using cell projection techniques. Among these techniques, Shirley *et al.* [18] introduce the projected tetrahedra, which is one of the first methods taking advantage of graphics hardware to accelerate unstructured volume rendering. This algorithm splits the footprint of a tetrahedron in screen space into a number of triangles, and then sends these triangles to the graphics processor for rasterization. Max *et al.* [12] show that for the simple case of a tetrahedron using barycentric interpolation for signal reconstruction inside the cell, the signal variation over rays passing through this tetrahedron is linear. Therefore, using linear interpolation of the scalar values inside the tetrahedron while sampling at the cell faces results in exact rendering, and this is commonly known as preintegrated volume rendering as presented by Roettger *et al.* [17]. Their method first computes the segments for all possible triplets of entry scalar values, exit scalar values and integration lengths and stores them in a 3D table called the preintegration table. Then, the authors adapt the projected tetrahedra algorithm to lookup into this table for each pixel using the current entry and exit scalars, and the integration length. Using floating point preintegration tables and rendering buffers, Kraus *et al.* [8] show that it is possible to achieve better accuracy in the final renderings. Furthermore, the authors use a logarithmic scale for accessing the length component of the preintegration table, resulting in higher accuracy for the smallest lengths. Preintegration has been extended to adaptive sampling in the context of volume rendering by Roettger *et al.* [16] and Ledergerber *et al.* [10], however the authors are not able to guarantee hole-free isosurfaces. Lum *et al.* [11] extend the preintegration techniques to volume shading by linearly interpolating the shading across each integration interval. Since the original projected tetrahedra technique was proposed, different improvements have been introduced to mitigate the computational and storage requirements of using a 3D table. In particular, analytical approximations were proposed by Roettger *et al.* [17, 15], Guthe *el al.* [4] and Moreland *et al.* [13]. However, these techniques remain approximations of the volume rendering integral. In order to speedup the 3D preintegration table computation and therefore make it suitable for real time transfer function changes, one can use the method developed by Lum *et al.* [11]. By precomputing portions of the volume rendering integral, the table computational complexity is reduced, which results in a speedup by two orders of magnitude.

Although the case of tetrahedral cells is well covered in the literature, for more complex cell types exact rendering is not straightforward to achieve. In such cases, the following approximation is commonly used: a continuous function is reconstructed inside the cell using the reconstruction filter, and this function is sampled regularly inside the cell. However, this wastes resources and can still miss some isovalues passing through the cell, which creates artifacts and holes in the isosurfaces.

In the field of isosurface visualisation, it has been shown by Parker *et al.* [14] that analytical techniques can be used to find the exact position of an isosurface inside a trilinearly interpolated hexahedral cell. This technique is more accurate than polygonal reconstruction, and in particular results in better topology for the isosurfaces.

In the field of adaptive mesh refinement (AMR) visualization, Weber *et al.* achieve high quality volume rendering [20] using cell projection and multiple integration steps inside each cell. Kähler *et al.* demonstrate how to exploit 3D graphics hardware to accelerate the volume rendering of AMR data [5]. For this purpose, the authors show a method for packing AMR data into a 3D texture in an efficient fashion. By converting sparse datasets into AMR data, Kähler *et al.* extend their visualization technique to the volume rendering of sparse volume datasets [7]. The authors also discuss packing strategies allowing fitting multiple bricks into a single 3D texture. More recently, Vollrath *et al.* achieved GPU-based volume visualization of AMR data [19] with specific data structures: using a page table and using an octree. AMR visualization techniques were also extended to time-dependent datasets by Kähler *et al.* [6] and Gosink *et al.* [3]. However, all these papers use a rendering stage involving data resampling, which we want to avoid in order to provide the highest possible visual quality. Therefore, the framework we present in this paper is centered around the idea of avoiding data resampling in order to ensure high quality pictures, and in particular crack-free isosurfaces.

3 Cell rendering

We now describe our AMR cell volume rendering technique. First we demonstrate how to achieve high quality, semi-analytical preintegration of a single hexahedral cell, then we show that this technique can be extended to shaded volume rendering. Finally, we show how the first two subsections apply to AMR meshes by handling the case of adjacent cells of different AMR levels.

3.1 Single cell rendering

Our cell rendering technique can be decomposed into three stages: first we reconstruct the analytical function across the current viewing ray. Second, we carefully choose the sampling points over this analytical function according to its properties. Third, we apply the transfer function. We now describe these three stages in more detail.

Signal reconstruction. Trilinear signal reconstruction inside a hexahedral cell is a weighted sum of the eight scalar values s_{ijk} at the eight vertices of the cell. For a given point (x,y,z) inside a hexahedral cell of normalized coordinates in $[0,1]^3$, the trilinear interpolation of the signal $s(x,y,z)$ is defined as follows:

$$\begin{aligned} s(x,y,z) = & s_{000}\cdot(1-x)\cdot(1-y)\cdot(1-z)+s_{100}\cdot x\cdot(1-y)\cdot(1-z)+ \\ & s_{010}\cdot(1-x)\cdot y\cdot(1-z)\cdot+s_{001}\cdot(1-x)\cdot(1-y)\cdot z+ \\ & s_{101}\cdot x\cdot(1-y)\cdot z+s_{011}\cdot(1-x)\cdot y\cdot z+ \\ & s_{110}\cdot x\cdot y\cdot(1-z)+s_{111}\cdot x\cdot y\cdot z \end{aligned} \tag{1}$$

where s_{ijk} is the sample value at the cell corner (i,j,k). Let us consider a single ray parametrized by t traversing a hexahedral cell. We can express the Cartesian coordinates (x,y,z) of a point over this ray as a function of t:

$$\begin{aligned} x &= x_0 + t \cdot v_x \\ y &= y_0 + t \cdot v_y \\ z &= z_0 + t \cdot v_z \end{aligned} \tag{2}$$

where (x_0, y_0, z_0) are the entry coordinates of the ray in the cell, and (v_x, v_y, v_z) are the differences between the exit and entry coordinates. By replacing the x, y and z coordinates with their parametrization over the ray in the trilinear interpolation formula, we can now compute $s(x,y,z)$ over the ray as a function of the parameter t:

$$s(t) = a \cdot t^3 + b \cdot t^2 + c \cdot t + d,\ t \in [0,1] \tag{3}$$

with the a, b, c and d coefficients as follows:

$$\begin{aligned} a = {} & (s_{100} - s_{000} + s_{010} + s_{001} + s_{111} - s_{110} - s_{011} - s_{101}) \\ & \cdot v_x \cdot v_y \cdot v_z \end{aligned} \tag{4}$$

$$\begin{aligned} b = {} & (-x_0 \cdot v_y \cdot v_z - v_x \cdot y_0 \cdot v_z - v_x \cdot v_y \cdot z_0 + v_x \cdot v_z) \cdot s_{101} \\ & + (x_0 \cdot v_y \cdot v_z + v_x \cdot y_0 \cdot v_z + v_x \cdot v_y \cdot z_0) \cdot s_{111} \\ & + (v_y \cdot v_z - x_0 \cdot v_y \cdot v_z - v_x \cdot y_0 \cdot v_z - v_x \cdot v_y \cdot z_0) \cdot s_{011} \\ & + (-v_x \cdot v_z + v_x \cdot v_y \cdot z_0 - v_y \cdot v_z + x_0 \cdot v_y \cdot v_z + v_x \cdot y_0 \cdot v_z) \cdot s_{001} \\ & + (-v_x \cdot v_z + v_x \cdot y_0 \cdot v_z + v_x \cdot v_y \cdot z_0 - v_x \cdot v_y + x_0 \cdot v_y \cdot v_z) \cdot s_{100} \\ & + (-v_x \cdot y_0 \cdot v_z - x_0 \cdot v_y \cdot v_z + v_x \cdot v_y - v_x \cdot v_y \cdot z_0) \cdot s_{110} \\ & + (v_x \cdot v_y + v_y \cdot v_z - v_x \cdot v_y \cdot z_0 + v_x \cdot v_z - x_0 \cdot v_y \cdot v_z \\ & - v_x \cdot y_0 \cdot v_z) \cdot s_{000} + (v_x \cdot v_y \cdot z_0 - v_y \cdot v_z - v_x \cdot v_y + x_0 \cdot v_y \cdot v_z \\ & + v_x \cdot y_0 \cdot v_z) \cdot s_{010} \end{aligned} \tag{5}$$

$$\begin{aligned} c = {} & (-x_0 \cdot y_0 \cdot v_z + x_0 \cdot v_z + v_x \cdot z_0 - v_x \cdot y_0 \cdot z_0 - x_0 \cdot v_y \cdot z_0) \cdot s_{101} \\ & + (v_x \cdot y_0 \cdot z_0 + x_0 \cdot v_y \cdot z_0 + x_0 \cdot y_0 \cdot v_z) \cdot s_{111} \\ & + (y_0 \cdot v_z - x_0 \cdot v_y \cdot z_0 - v_x \cdot y_0 \cdot z_0 + v_y \cdot z_0 - x_0 \cdot y_0 \cdot v_z) \cdot s_{011} \\ & + (x_0 \cdot y_0 \cdot v_z + v_z - y_0 \cdot v_z - v_y \cdot z_0 - x_0 \cdot v_z - v_x \cdot z_0 \\ & + x_0 \cdot v_y \cdot z_0 + v_x \cdot y_0 \cdot z_0) \cdot s_{001} + (v_x \cdot y_0 \cdot z_0 + x_0 \cdot y_0 \cdot v_z \\ & + x_0 \cdot v_y \cdot z_0 - v_x \cdot z_0 - v_x \cdot y_0 - x_0 \cdot v_z - x_0 \cdot v_y + v_x) \cdot s_{100} \\ & + (x_0 \cdot v_y + v_x \cdot y_0 - x_0 \cdot y_0 \cdot v_z - x_0 \cdot v_y \cdot z_0 - v_x \cdot y_0 \cdot z_0) \cdot s_{110} \\ & + (-v_x \cdot y_0 + x_0 \cdot v_y \cdot z_0 - y_0 \cdot v_z - x_0 \cdot v_y + x_0 \cdot y_0 \cdot v_z + v_y \\ & + v_x \cdot y_0 \cdot z_0 - v_y \cdot z_0) \cdot s_{010} + (-x_0 \cdot v_y \cdot z_0 - v_z + v_x \cdot y_0 \\ & - x_0 \cdot y_0 \cdot v_z - v_x \cdot y_0 \cdot z_0 - v_y + y_0 \cdot v_z + v_x \cdot z_0 + v_y \cdot z_0 \\ & + x_0 \cdot v_y - v_x + x_0 \cdot v_z) \cdot s_{000} \end{aligned} \tag{6}$$

$$\begin{aligned} d = {} & (x_0 \cdot z_0 - x_0 \cdot y_0 \cdot z_0) \cdot s_{101} + (y_0 \cdot z_0 - x_0 \cdot y_0 \cdot z_0) \cdot s_{011} \\ & + (-x_0 \cdot z_0 - y_0 \cdot z_0 + x_0 \cdot y_0 \cdot z_0 + z_0) \cdot s_{001} + (-x_0 \cdot z_0 + x_0 \\ & + x_0 \cdot y_0 \cdot z_0 - x_0 \cdot y_0) \cdot s_{100} + (x_0 \cdot y_0 - x_0 \cdot y_0 \cdot z_0) \cdot s_{110} \\ & + (-y_0 \cdot z_0 - x_0 \cdot y_0 + y_0 + x_0 \cdot y_0 \cdot z_0) \cdot s_{010} + (-y_0 - z_0 \\ & - x_0 \cdot y_0 \cdot z_0 + x_0 \cdot z_0 + y_0 \cdot z_0 - x_0 + x_0 \cdot y_0 + 1.0) \cdot s_{000} \\ & + s_{111} \cdot x_0 \cdot y_0 \cdot z_0 \end{aligned} \tag{7}$$

Therefore, when using trilinear interpolation, the reconstructed signal over a ray traversing a hexahedron is a third degree polynomial in t.

Ray sampling. The next step consists in sampling the ray at the relevant values. In the context of ray casting, the sampling over the ray is usually homogeneous. Doing so is suboptimal when small structures are present in the dataset, as such a sampling could easily miss these structures. A naive solution would consist in analytically integrating the function over the interval, however for high frequency transfer functions this could quickly become inefficient. The alternative of preintegrating over the whole interval is not technically feasible either, as it would require a 5-dimensional preintegration table (4 scalar values which uniquely define the third degree polynomial along with the interval length). Instead, in order to avoid missing isovalues during the sampling stage, we sample at the entry and exit faces of the hexahedron, and also at the local extrema of the reconstructed function as shown on Figure 4. As we know the analytical form of the reconstructed function $s(t)$, we can compute its derivative $s'(t)$ analytically. Since $s'(t)$ is a quadratic function, it has at most two roots, and $s(t)$ has at most two local extrema r_1 and r_2. We compute the extrema of $s(t)$, prune those that lie outside the cell and finally sort them along the ray. At this point we have reconstructed monotonic intervals for $s(t)$ over the ray inside a given cell.

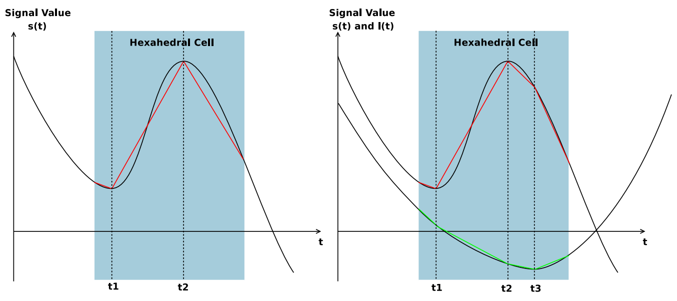

Fig. 4. Sampling schemes for a ray inside a cell. On the left, shading is not used. The cell is shown in blue, the reconstructed scalar values are depicted in black and their piecewise linear approximations are shown in red; the interval is split at t_1 and t_2. On the right, shading is used. The reconstructed quadratic shading function incurs an additional sampling point t_3.

Transfer function application. The last stage is the application of the transfer function. Since the transfer function can be arbitrary, missing a single value in the previous stage can result in missing or discontinuous isosurfaces. We now begin with the standard volume rendering equation where I is the final intensity, $c()$ is the color transfer function and $\tau()$ is the opacity transfer function. If we have two extrema and both those extrema lie within $[0,1]$, we get:

$$\begin{aligned} I = {} & \int_0^L c(s(t)) e^{-\int_0^t \tau(s(u))du} dt = \int_0^{r_1} c(s(t)) e^{-\int_0^t \tau(s(u))du} dt \\ & + \int_{r_1}^{r_2} c(s(t)) e^{-\int_0^t \tau(s(u))du} dt + \int_{r_2}^{L} c(s(t)) e^{-\int_0^t \tau(s(u))du} dt \end{aligned} \tag{8}$$

Over each interval $[t1, t2]$, we replace $s(t)$ with a linear form $(1-t) \cdot s(t1) + t \cdot s(t2)$ and obtain a formulation similar to [2]:

$$\begin{aligned} I \approx {} & \int_0^{r_1} c((1-t) \cdot s(0) + t \cdot s(r_1)) e^{-\int_0^t \tau((1-u) \cdot s(0) + u \cdot s(r_1))du} dt \\ & + \int_{r_1}^{r_2} c((1-t) \cdot s(r_1) + t \cdot s(r_2)) e^{-\int_0^t \tau(s((1-u) \cdot s(r_1) + u \cdot s(r_2)))du} dt \\ & + \int_{r_2}^{L} c((1-t) \cdot s(r_2) + t \cdot s(L)) e^{-\int_0^t \tau((1-u) \cdot s(r_2) + u \cdot s(L))du} dt \\ & \approx C(s(0), s(r_1), r_1) + (1 - \alpha(s(0), s(r_1), r_1)) C(s(r_1), s(r_2), r_2 - r_1) \\ & + (1 - \alpha(s(0), s(r_1), r_1)) \cdot (1 - \alpha(s(r_1), s(r_2), r_2 - r_1)) \\ & \cdot C(s(r_2), s(L), L - r_2) \end{aligned} \tag{9}$$

with

$$\begin{aligned} C(a,b,l) &= \int_0^1 c((1-t) \cdot a + t \cdot b) l e^{-\int_0^t \tau((1-t) \cdot a + t \cdot b) l du} dt \\ \alpha(a,b,l) &= 1 - e^{-\int_0^1 \tau((1-t) \cdot b + t \cdot a) l dt} \end{aligned} \tag{10}$$

Since over each monotonic interval, the ranges of $s(t)$ and of its piecewise linear approximation are identical, we will not miss any isovalue when using piecewise linear approximations over these intervals. We are thereby able to ensure that all the volume features are present and have accurate topology according to the trilinear interpolation. As the topology is consistent from one frame to another, the only remaining kind of artifacts is wobbling of the surfaces and structures inside the cells as the viewpoint changes.

3.2 Extension to volume shading

We now demonstrate how to extend our approach to achieve local Phong shading with directional light sources; this process is described for the case of diffuse shading but straightforwardly generalizes to specular shading by using the half vector instead of the light vector. By analyzing the behaviour of the gradient inside the cell, we reconstruct a shading function over the ray and show how to use it to achieve high quality volume shading.

Shading computation. The gradient $\vec{\nabla}s(x,y,z)$ of the scalar function (as shown in Equation 1) can be obtained analytically by computing the partial derivatives of $s(x,y,z)$. Diffuse shading $d(x,y,z)$ is defined for any point of the cell as the dot product of this gradient with the light vector $\vec{L}(lx,ly,lz)$:

$$\begin{aligned}
d(x,y,z) = \vec{\nabla}s(x,y,z)\cdot\vec{L} &= lx\cdot\frac{\partial s}{\partial x} + ly\cdot\frac{\partial s}{\partial y} + lz\cdot\frac{\partial s}{\partial z}\\
&= lx\cdot(-s_{000} - s_{011}\cdot y\cdot z + s_{101}\cdot z\\
&- s_{101}\cdot z\cdot y - s_{010}\cdot y - s_{100}\cdot z - s_{100}\cdot y - s_{001}\cdot z\\
&+ s_{000}\cdot z + s_{000}\cdot y + s_{111}\cdot y\cdot z - s_{000}\cdot y\cdot z + s_{010}\cdot y\cdot z\\
&+ s_{110}\cdot y - s_{110}\cdot y\cdot z + s_{100} + s_{100}\cdot y\cdot z + s_{001}\cdot z\cdot y)\\
&+ ly\cdot(-s_{000} - s_{011}\cdot z\cdot x - s_{101}\cdot x\cdot z - s_{010}\cdot z - s_{010}\cdot x\\
&- s_{100}\cdot x - s_{001}\cdot z + s_{010} + s_{000}\cdot z + s_{000}\cdot x + s_{111}\cdot x\cdot z\\
&- s_{000}\cdot x\cdot z + s_{010}\cdot x\cdot z + s_{110}\cdot x - s_{110}\cdot x\cdot z + s_{100}\cdot x\cdot z\\
&+ s_{001}\cdot z\cdot x + s_{011}\cdot z)\\
&+ lz\cdot(-s_{000} - s_{011}\cdot y\cdot x + s_{101}\cdot x - s_{101}\cdot x\cdot y - s_{010}\cdot y\\
&- s_{100}\cdot x - s_{001}\cdot y - s_{001}\cdot x + s_{000}\cdot y + s_{000}\cdot x + s_{111}\cdot x\cdot y\\
&- s_{000}\cdot x\cdot y + s_{010}\cdot y\cdot x - s_{110}\cdot x\cdot y + s_{100}\cdot x\cdot y + s_{001}\\
&+ s_{001}\cdot x\cdot y + s_{011}\cdot y)
\end{aligned} \tag{11}$$

We again use the parametrization described in Equation 2 and obtain:

$$d(t) = e\cdot t^2 + f\cdot t + g \tag{12}$$

where e, f and g can be trivially computed as previously done in Subsection 3.1 for a, b, c and d. For a parametrized ray inside a cell, the gradient variation over this ray is thus a quadratic form in t.

Ray sampling. In order to capture all the variations of the diffuse shading function, we sample this function at the cell faces and at the local extrema of $d(t)$. Since $d(t)$ is a quadratic polynomial, there is at most a single local extremum r. Therefore, if we apply shading on top of the previously described preintegration method, we only need at most one additional sampling point per cell. As in the case of scalar value integration, we thereby ensure that all extrema of the shading function are captured. We start from the volume rendering equation with diffuse shading; $c()$ and $\tau()$ are defined as previously, and $d()$ represents the diffuse shading coefficient.

$$\begin{aligned}
I &= \int_0^L c(s(t))d(t)e^{-\int_0^t \tau(s(u))du}dt\\
&= \int_0^r c(s(t))d(t)e^{-\int_0^t \tau(s(u))du}dt + \int_r^L c(s(t))d(t)e^{-\int_0^t \tau(s(u))du}dt
\end{aligned} \tag{13}$$

Over each interval $[t1,t2]$, We replace $s(t)$ with a linear form $(1-t)\cdot s(t1) + t\cdot s(t2)$, similarly to [11]. We obtain:

$$\begin{aligned}
I \approx\ & d(s(0))\cdot C_{front}(s(0),s(r_1),r_1) + d(s(r_1))\cdot C_{back}(s(0),s(r_1),r_1)\\
&+ (1-\alpha(s(0),s(r_1),r_1))(d(s(r_1))\cdot C_{front}(s(r_1),s(L),L-r_1)\\
&+ d(s(L))\cdot C_{back}(s(r_1),s(L),L-r1))
\end{aligned} \tag{14}$$

with

$$\begin{aligned}
C_{back}(a,b,l) &= \int_0^1 c((1-t)\cdot a + t\cdot b)te^{-\int_0^t \tau((1-t)\cdot a+t\cdot b)ldu}dt\\
C_{front}(a,b,l) &= \int_0^1 c((1-t)\cdot a + t\cdot b)(1-t)e^{-\int_0^t \tau((1-t)\cdot a+t\cdot b)ldu}dt\\
\alpha(a,b,l) &= 1 - e^{-\int_0^1 \tau((1-t)\cdot b+t\cdot a)ldt}
\end{aligned} \tag{15}$$

At this point, the ranges for both the scalar and the shading functions are the same as their reconstructed counterparts over each of the integration intervals. The right of Figure 4 depicts this situation: the scalar function integration requires two additional samples, and the shading function integration requires one additional sample. The corresponding piecewise linear approximation for this function is shown in green. In that case, four integration steps over the ray are required to achieve accurate rendering.

Notice that since the reconstructed scalar field is only C^0 across the cell faces, the gradient is not continuous across those faces. Therefore, shading is not continuous across the cell faces either.

3.3 Extension to AMR rendering

After explaining our method for simple hexahedral cell visualization, we now extend it to the case of AMR meshes. One specific issue is the occurrence of hybrid cells, that is, cells that share at least an edge with a cell of a different level. Such cells need to be handled as special cases to achieve continuity along cell faces and therefore avoid artifacts in the final renderings. In order to achieve this, we propagate the cell splits from the lower level cell to the higher level cell as shown on Figure 5. Two types of new points are added to the lower level cell as it is split: source split points (shown in green on Figure 5) which use the value from the higher level cell, and destination split points (shown in red on Figure 5) which use the value bilinearly interpolated inside a face shared with a cell of the same level. In both cases, this results in continuity across cell faces: either by using the neighbour's value (in the case of a source split point) or by using a new value that matches the interpolation scheme of the neighbour cell (in the case of a destination split point). After splitting we obtain cuboids, to which our hexahedral cell rendering technique straightforwardly generalizes.

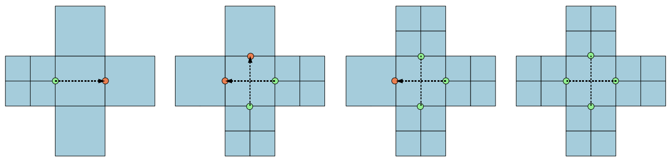

Fig. 5. Propagating cell splits towards higher level cells in different configurations generates cuboids.

4 Mesh Construction and Traversal

Vollrath *et al.* [19] use an homogeneous page table in order to access the AMR data in a regular fashion. However, this wastes memory and decreases performance as it fails to take into account the heterogeneity of AMR datasets at their finest granularity. Notably, this technique adds many additional cells which slows down the rendering stage, especially when high quality cell rendering methods are in use. Therefore, in this section we present a data structure that is hierarchical and does not require additional cells. Because of the difficulty in handling hierarchical structures on the GPU efficiently, we have chosen a hybrid CPU-GPU sorting approach: our hierarchical structure is made of AMR patches (an AMR patch is defined as an axis-aligned box of same-level cells). While this structure is traversed on the CPU, patches can be sorted and rendered completely on the GPU thanks to their regularity. We now present our mesh construction and traversal technique.

4.1 Mesh construction

Assuming a function *homogeneity*() that returns the ratio of homogeneity of cell levels (1 if all the cells share the same level, 0 if all the cells have different levels) inside a given box, our hierarchy building algorithm is described in Algorithm 1. The dataset is recursively processed to create a KD-tree [1]. If a node has an homogeneous patch attached, it is made into a leaf node, otherwise it is split recursively again.

Algorithm 1 Mesh construction algorithm

```
Procedure split(box $b$)
best_score = 0
for each axis direction $d$ do
    for each possible cut plane $p$ in direction $d$ do
        current_score = homogeneity($b$ cut by $p$)
        if current_score $>$ best_score then
            best_score = current_score
            best_cut = $p$
        end if
    end for
end for
Return ($b$ split by best_cut)
End Procedure

Procedure build_mesh(box $b$, node $n$)
if homogeneity($b$) = 1 then
    $n$ is a leaf node
    Copy $b$'s data to node $n$
else
    Create two nodes (child_node_1,child_node_2)
    (child_box_1,child_box_2) = split(n)
    $n$.children = (child_node_1,child_node_2)
    build_mesh(child_box_1, child_node_1)
    build_mesh(child_box_2, child_node_2)
end if
End Procedure
```

4.2 Mesh traversal

Once the data structure is built, the list of AMR patches is traversed hierarchically on the CPU in back to front order, and each patch is sent to the GPU for rendering. The GPU is then in charge of traversing the cells inside a given patch. Although this traversal is trivial in the most common case (and can be achieved using a simple back-to-front loop in each dimension), we have to take perspective into account. Figure 6 depicts a situation where the position of the observer O (facing a single AMR patch, orthogonally to the green cell) requires further subdivision of the data into sub-patches. On the left of the figure, the arrows depict the back-to-front rendering order dependency between the cells. On the right of the picture, we show a proposed traversal order that follows these dependencies. By subdividing the patch as shown on this figure, we are thereby able to ensure perspective-correct sorting of the cells in all cases.

5 Implementation

We have implemented our AMR volume rendering method on the GPU using OpenGL. In order to improve the accuracy of the preintegration table, it is stored in a 16 bit per component $256 \times 256 \times 32$ RGBA texture, and we use logarithmic table access for the length of the preintegration interval which allows greater precision at small lengths [8]. For better blending accuracy, a 16 bit floating point offscreen buffer is used [8]; once the whole picture is rendered, it is copied to the front buffer. We now detail the implementation of our hybrid CPU-GPU AMR structure traversal. The whole rendering pipeline is visible on Figure 7. Initially, the mesh is stored as a KD-tree with AMR patches at the leaf nodes. The KD-tree hierarchy is traversed on the CPU in a back-to-front fashion, by choosing the appropriate order at each internal node. Once a leaf node is found, a series of consecutive indices is sent to the vertex shader, where the number of indices matches the number of cells of the current patch.

Each of these indices is then modified inside the vertex shader according to the current observer position to generate proper cell ordering as shown in Subsection 4.2. This new index is used to access a vertex texture holding cell data.

The cell data is then passed to the geometry shader, which checks that the cell is visible according to the current transfer function using a look-up table as introduced in [9]. The look-up table is a $2D$ table built in a way similar to the preintegration tables but holding Boolean values: for each pair of values $s1$ and $s2$, if there exists at least an opacity which is non-zero for scalar values in $[s1, s2]$ the table holds a 1, otherwise the table contains a 0. In order to know whether a cell is visible or not, we compute the minimum and maximum of the 8 cell corner values. Since the variation of the scalar value inside a cell is bounded by these two values, we can use them to determine if the whole cell is visible. Therefore, these two values are used as table lookup indices and a resulting visibility value is fetched. If the cell is deemed visible, the geometry shader then instantiates its 6 faces. The back faces are culled by the standard OpenGL pipeline. The front faces are then rasterized, and each pixel executes a fragment shader.

The fragment shader is depicted as Algorithm 2. First, the ray exit point is computed. Then, the polynomial coefficients for signal and shading reconstruction are calculated as shown in Subsections 3.1 and 3.2. Then the local extrema of the scalar polynomial or of both polynomials are found and added to a list of points. Points outside $[0, 1]$ are pruned and the remaining points are sorted. These sorted points are subsequently used as bounds for preintegration intervals. Over each interval, we apply preintegration and shading interpolation using front- and back-weighted tables as in [11].

6 Results

We now present results obtained with our method. Performance measurements and screen captures were performed on an PC with two Xeon E5345 processors, 4GB of memory and a GeForce 8800 GTX graphics card with 768MB of memory. Although 8 cores were available, only a single core was used for the computations. For these tests we used a time step of the "meteorite" dataset, which is an AMR simulation of a meteorite falling into the sea. This dataset contains 2377878 cells divided as follows: 6300 cells of level 0, 9258 cells of level 1, 112869 cells of level 2 and 2249451 cells of level 3. The pictures produced in this paper visualize the ρ attribute (density) of this dataset.

6.1 Single cell rendering

In order to assess the quality improvement of our cell rendering technique, we now present volume rendering results in the context of a single cell. For these tests, we used a single unshaded hexahedral cell and a transfer function showing multiple transparent isosurfaces. These results are visible on Figure 8. On the top left of the figure, the hexahedral cell is rendered using a single preintegration interval;

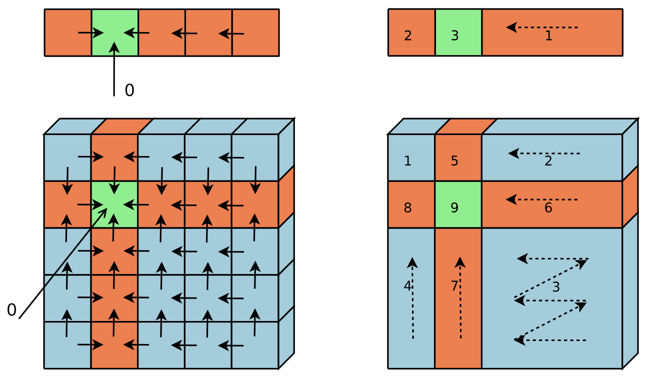

Fig. 6. Traversing a single row of an AMR patch (top) and a single slice (bottom).

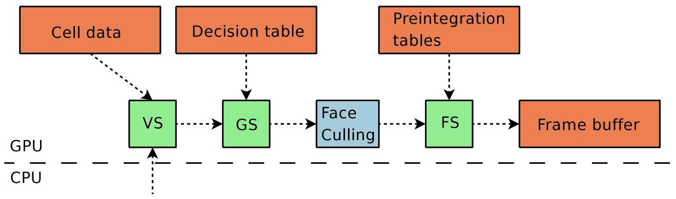

Fig. 7. Our AMR volume rendering pipeline. The buffers are shown in orange, programmable functionality (vertex shaders, geometry shaders, fragment shaders) is shown in green and fixed OpenGL functionality is depicted in blue.

Algorithm 2 Fragment shader computation

```
Find the exit point of the ray
Compute the $a,b,c,d$ cubic coefficients as per Subsection 3.1
Compute the $e,f,g$ quadratic coefficients as per Subsection 3.2
$T = \emptyset$
**if** (a!=0 and b!=0) **then**
    $\Delta = 4*b^2 - 12*a*c$
    $T = T \cup \{\frac{-2*b+\sqrt{\Delta}}{6*a}, \frac{-2*b-\sqrt{\Delta}}{6*a}\}$
**else**
    **if** (a=0 and b!=0) **then**
        $T = T \cup \{\frac{-c}{2*b}\}$
    **end if**
**end if**
**if** (e!=0) **then**
    $T = T \cup \{\frac{-f}{2*e}\}$
**end if**
**for** each point $t$ in $T$ **do**
    **if** ($t \leq 0$ or $t \geq 1$) **then**
        $T = T \setminus \{t\}$
    **end if**
**end for**
$T = T \cup \{0,1\}$
Sort the points in $T$
**for** each two consecutive points $(t1,t2)$ in $T$ **do**
    Reconstruct the scalar values $s1 = d+t1*(c+t1*(b+t1*a))$ and $s2 = d+t2*(c+t2*(b+t2*a))$ at the interval boundaries.
    Reconstruct the shading values $l1 = g+t1*(f+t1*e)$ and $l2 = g+t2*(f+t2*e)$ at the interval boundaries.
    Compute the integration length $l = t2-t1$
    Fetch the front and back textures at $(s1,s2,l)$
    Combine these textures as described in [11]
    Accumulate the value
**end for**
```

2, 4 and 100 integration intervals are used for the top right, middle left and middle right pictures, respectively. Our adaptive technique is shown on the bottom left, and its number of integration intervals is shown on the bottom right (red means one, green means two and blue means three). These pictures show that thanks to our adaptive sampling method, it is possible to reach higher quality levels than with regular oversampling, using less samples. In particular, our technique reconstructs correct topology as can be seen when comparing it with the highly oversampled version that uses 100 intervals. Furthermore, adding more integration steps generates banding artifacts (which can be seen inside the isosurfaces), which are not present with our technique since less steps are required.

Since raycasting methods always line up the viewing ray and the integration segment, and since the piecewise linear interpolation results in the same range over each integration interval as the original function, we know that the only possible error lies along the ray. Using a stochastic maximization process over both the eight cell corner values and the ray trajectory through the cell, we tried maximizing the distortion of isovalue positions along the ray over all possible rays traversing the cell. The result corresponds neither to a singular ray nor to eight

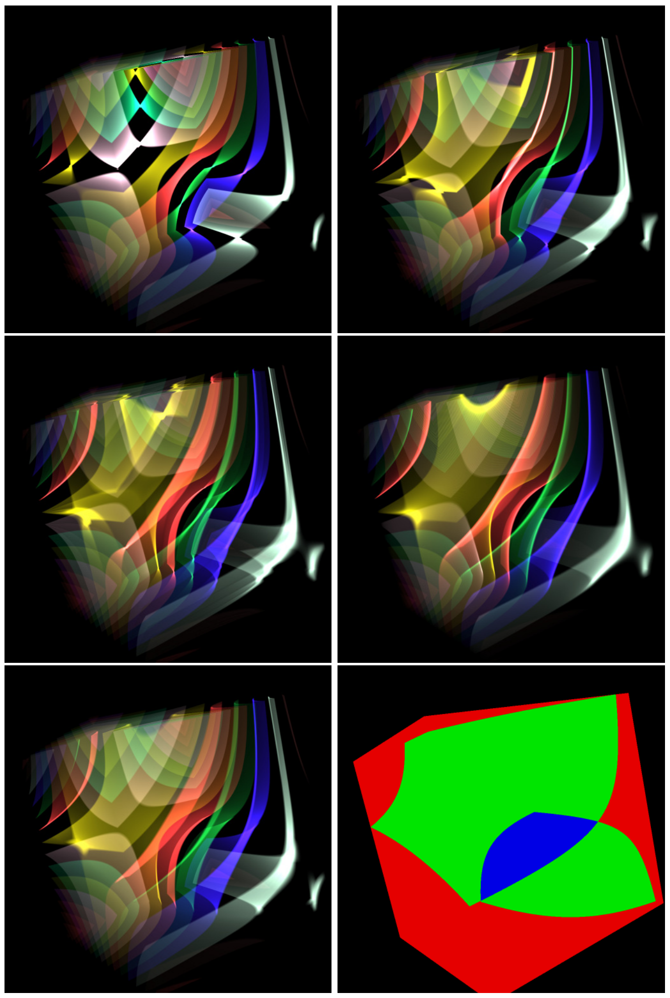

Fig. 8. Comparison of classical, single-step preintegration (top left), oversampled preintegration using 2 (top right), 4 (middle left) and 100 (middle right) regular integration intervals for all pixels and our technique (bottom left). The number of integration steps for the adaptive method is shown on the bottom right (red means 1 step, green means 2 steps and blue means 3 steps). The increase of image quality due to our method can be easily spotted in the middle left portion of all images.

singular cell values. The biggest distortion found is approximately 0.468 ray units, meaning that any isosurface is at most 0.468 times the ray length inside the cell away from its real (trilinearly interpolated) position.

6.2 Quality results

This section introduces results pertaining to image quality. The left column of Figure 10 shows a comparison of the same AMR dataset rendered without (top) and with shading (bottom). It can be seen from these pictures that shading greatly helps understanding the internal structure of the data. Notably, the shape of the structures resulting from the meteorite splashing into the water is easier to infer from our shaded rendering. When viewing the full dataset, we obtain the following framerates: 3.77 fps for unshaded volume rendering, and 3.62 fps for shaded rendering.

The middle and right columns of Figure 10 compare two pairs of renderings with the same viewpoints using 2 integration steps per cell (top) and our adaptive technique (bottom). Both rendering methods run at the same framerate. In the middle column, artifacts are visible in the top row which disappear when using our adaptive sampling technique. In the right column, erroneous holes appear in the thin yellow isosurface which our adaptive sampling technique removes.

Figure 9 compares renderings obtained when taking the local ex-

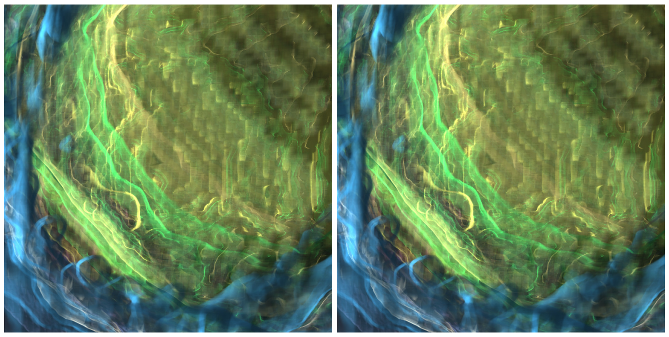

Fig. 9. Volume rendering comparison with the full AMR dataset when using the shading function extrema (left) and when omitting the extrema (right).

trema of the shading function into account (on the left), and when omitting those extrema (on the right). Although these pictures exhibit small differences, these differences are not relevant enough to enable this feature except when very high quality images are to be generated.

Finally, Figure 1 presents a close-up of the meteorite splashing into the water as seen from above.

6.3 Performance results

This subsection introduces performance results obtained when visualizing the full AMR dataset.

Table 1 exemplifies our results from a performance viewpoint, using different optimizations. Without the look-up table, performance is low, at less than a single frame per second. Thanks to the use of the look-up table, lots of cells are culled and therefore the load on the rendering stage decreases. This increases the framerate to approximately 3.48 frames per second. Our last optimization is the introduction of our hybrid CPU-GPU mesh traversal mechanism which further increases the framerate to 3.62 frames per second. For later performance measurements, both the look-up table and the CPU-GPU mesh traversal optimizations are used.

Optimization	Performance
Without look-up table	0.76 fps
With look-up table and CPU mesh traversal	3.48 fps
With look-up table and CPU-GPU mesh traversal	3.62 fps

Table 1. Rendering performance with different optimizations at a 1024×1024 screen resolution

Performance results when changing the target resolution are given on Table 2. These results show that the performance remains interactive at high screen resolutions (including 2048×2048), for two reasons: first, cell-projection based techniques have an advantage over raycasting techniques as they are able to cull cells and also can share computations among multiple pixels of the same cell. Second, as the resolution decreases, our system becomes limited by the throughput of the mesh traversal and geometry generation stages; we expect this problem to be lifted by the next generation of GPUs. Table 3 shows

Resolution	Performance
256×256	3.95 fps
512×512	3.83 fps
1024×1024	3.62 fps
2048×2048	3.50 fps

Table 2. Rendering performance with shading at different screen resolutions.

the respective performance visualizing the complete dataset with different algorithms for cell rendering: preintegration using 1,2,3 and 4

Sampling	Performance
4 integration intervals	3.32 fps
3 integration intervals	3.51 fps
2 integration intervals	3.64 fps
1 integration interval	3.76 fps
Fully adaptive sampling	3.62 fps
Adaptive sampling without shading extrema	3.65 fps

Table 3. Rendering performance with different preintegration schemes at a 1024×1024 screen resolution.

intervals, and our semi-analytic adaptive sampling technique with and without taking the shading extrema into account. The performance difference between the cell rendering techniques, albeit small, exists in these results. Such a small performance variation can be explained by the fact that the AMR mesh traversal and geometry generation phases, which are common to all the rendering methods, take a fair amount of time. These results show that our adaptive sampling technique has approximately the same performance as using two integration intervals per cell, whereas the quality is much higher as shown in Subsections 6.1 and 6.2. Thanks to its adaptivity, our cell rendering method uses fewer integration steps on average for the same quality, and therefore generates less preintegration table accesses during rendering than oversampling-based cell rendering. This in turn reduces the memory bandwidth pressure, which is usually the limiting factor for volume rendering. This more than offsets the computational intensity of calculating the polynomial coefficients and evaluating the extrema with our technique. This explains why, although our method has higher computational requirements than raw oversampling, its final performance is higher.

7 Conclusion and future works

We have shown that very high quality volume rendering was possible in interactive time for moderately-sized AMR datasets using our system. However, a number of issues remain open. First, we intend to mathematically prove the maximal distortion inside a hexahedral cell instead of exhibiting an empirical maximum. Second, as we are interested in bigger datasets, we would like to experiment with techniques allowing further scalability. This could be achieved by adding level-of-detail support to our AMR rendering system, or by parallelizing our system in order to increase the performance without sacrificing the quality. Third, we would like to traverse the whole AMR mesh from within the geometry shader. Although we already had unsuccessful experiments with this (because geometry shaders have performance issues when a high number of vertices is to be output), we hope that future generations of graphics hardware will lift this limitation and finally allow traversing the mesh on the GPU only. This will enable the development of new AMR traversal algorithms on the GPU. Finally, we would like to adapt our accurate cell rendering algorithm to structured datasets. Although such datasets can be seen as AMR datasets with a single patch, and therefore our method would be usable as-is, we think that the nature of structured datasets should be taken into account in order to optimize the traversal algorithms. Therefore, we expect that our semi-analytical cell rendering technique could be generalized to classical volume raycasting methods and achieve the high quality results we demonstrated together with interactive performance.

References

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[2] Klaus Engel and Thomas Ertl. Interactive high-quality volume rendering with flexible consumer graphics hardware. In *Eurographics State of The Art Report*, 2002.

[3] Luke Gosink, John C. Anderson, E. Wes Bethel, and Kenneth I. Joy. Query-driven visualization of time-varying adaptive mesh refinement data. In *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization)*, October 2008.

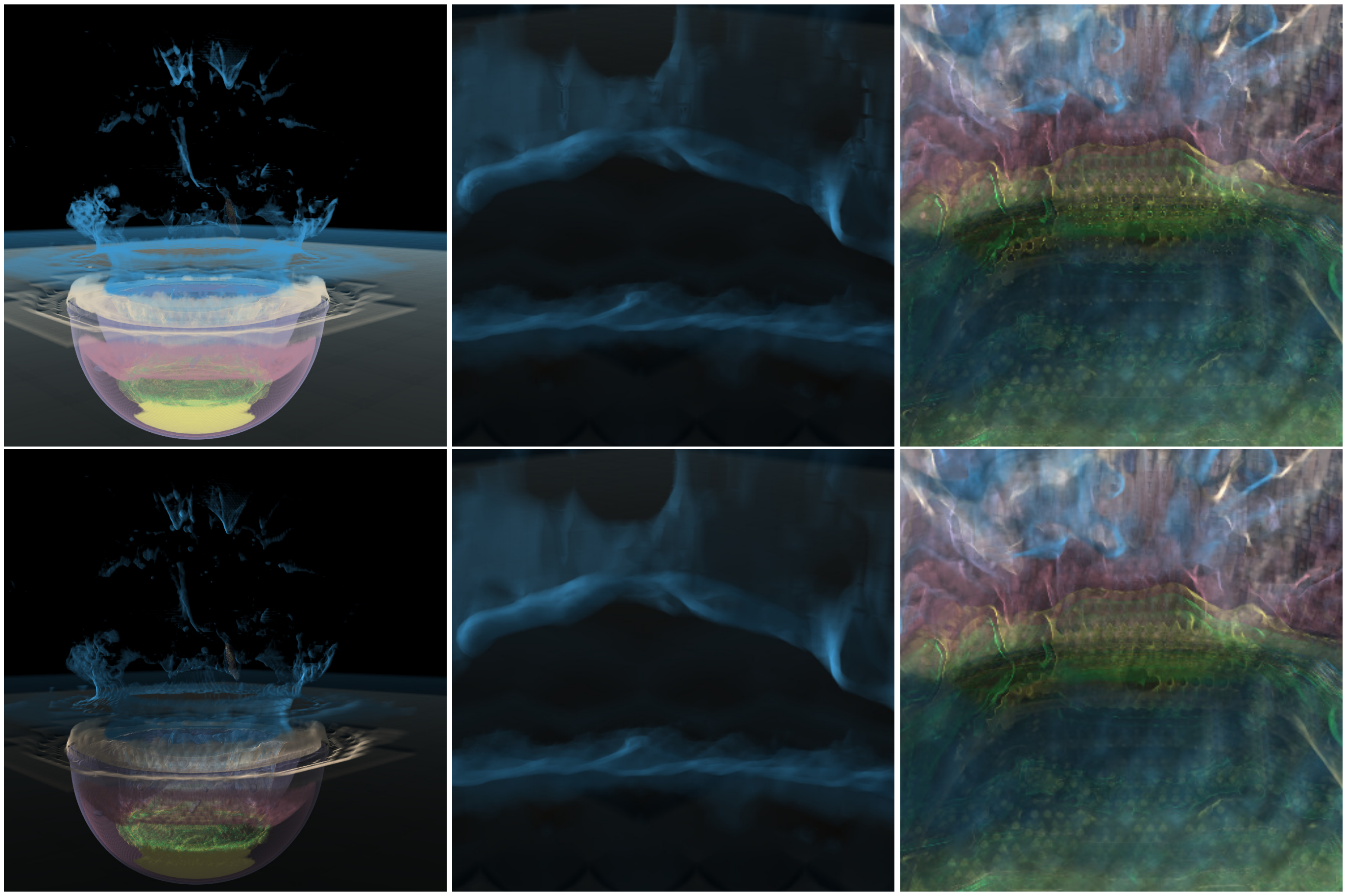

Fig. 10. Left: reference pictures computed using a 1024×1024 viewport without shading (top, 3.77 fps) and with shading (bottom, 3.62 fps). Middle and right: close-up on two parts of the datasets with 2 preintegration steps per cell (top) and our adaptive technique (bottom).

[4] Stefan Guthe, Stefan Roettger, Andreas Schieber, Wolfgang Strasser, and Thomas Ertl. High-quality unstructured volume rendering on the pc platform. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 119–125. Eurographics Association, 2002.

[5] Ralf Kähler and Hans-Christian Hege. Texture-based volume rendering of adaptive mesh refinement data. *The Visual Computer*, 18(8):481–492, 2002.

[6] Ralf Kähler, Steffen Prohaska, Andrei Hutanu, and Hans-Christian Hege. Visualization of time-dependent remote adaptive mesh refinement data. In *IEEE Visualization*, page 23, 2005.

[7] Ralf Kähler, Mark Simon, and Hans-Christian Hege. Interactive volume rendering of large sparse data sets using adaptive mesh refinement hierarchies. *IEEE Trans. Vis. Comput. Graph.*, 9(3):341–351, 2003.

[8] Martin Kraus, Wei Qiao, and David S. Ebert. Projecting tetrahedra without rendering artifacts. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 27–34, Washington, DC, USA, 2004. IEEE Computer Society.

[9] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 38, Washington, DC, USA, 2003. IEEE Computer Society.

[10] Christian Ledergerber, Gaël Guennebaud, Miriah Meyer, Moritz Bächer, and Hanspeter Pfister. Volume mls ray casting. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1372–1379, 2008.

[11] Eric Lum, Brett Wilson, and Kwan-Liu Ma. High-quality lighting and efficient pre-integration for volume rendering. The Joint Eurographics-IEEE TVCG Symposium on Visualization 2004, 2004.

[12] Nelson Max, Pat Hanrahan, and Roger Crawfis. Area and volume coherence for efficient visualization of 3D scalar functions. In *Computer Graphics (San Diego Workshop on Volume Visualization)*, volume 24, pages 27–33, 1990.

[13] Kenneth Moreland and Edward Angel. A fast high accuracy volume renderer for unstructured data. In *VV '04: Proceedings of the 2004 IEEE Symposium on Volume Visualization and Graphics (VV'04)*, pages 9–16, Washington, DC, USA, 2004. IEEE Computer Society.

[14] Steven Parker, Michael Parker, Yarden Livnat, Peter-Pike Sloan, Charles Hansen, and Peter Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):238–250, 1999.

[15] Stefan Roettger and Thomas Ertl. A two-step approach for interactive preintegrated volume rendering of unstructured grids. In *VVS '02: Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 23–28, Piscataway, NJ, USA, 2002. IEEE Press.

[16] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser. Smart hardware-accelerated volume rendering. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 231–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[17] Stefan Röttger, Martin Kraus, and Thomas Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 109–116, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[18] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. In *VVS '90: Proceedings of the 1990 workshop on Volume visualization*, pages 63–70, New York, NY, USA, 1990. ACM Press.

[19] J. E. Vollrath, T. Schafhitzel, and T. Ertl. Employing Complex GPU Data Structures for the Interactive Visualization of Adaptive Mesh Refinement Data. In *Proceedings of the International Workshop on Volume Graphics '06*, 2006.

[20] Gunther H. Weber, Oliver Kreylos, Terry J. Ligocki, John Shalf, Hans Hagen, Bernd Hamann, Kenneth I. Joy, and Kwan-Liu Ma. High-quality volume rendering of adaptive mesh refinement data. In *VMV '01: Proceedings of the Vision Modeling and Visualization Conference 2001*, pages 121–128. Aka GmbH, 2001.