Adaptive Raymarching and Multi-Sampled Pre-integration on Graphics Hardware: Application to the Visualization of Gyrokinetic Plasmas Simulations

Jean-François El Hajjar — Florence Zara — Jean-Michel Dischler — Virginie Grandgirard

No 5974

Septembre 2006

Thème NUM D

N 0249-6399

Adaptive Raymarching and Multi-Sampled Pre-integration on Graphics Hardware: Application to the Visualization of Gyrokinetic Plasmas Simulations

Jean-François El Hajjar[*] , Florence Zara [†] , Jean-Michel Dischler [‡] , Virginie Grandgirard [§]

Thème NUM D — Modélisation, simulation et analyse numérique
Projet CALVI

Rapport de recherche no 5974 — Septembre 2006 — 17 pages

Abstract: The study of plasmas is a rising research topic in the field of physics. It implies the need for specific algorithms allowing physicists to easily visualize and thus correctly interpret related complex datasets. This paper describes an innovative and highly efficient volume rendering method designed for gyrokinetic simulations. Based on a Raymarching approach that benefits from the latest functionalities offered by current graphics programmable hardware, we exploit the geometrical properties of the tokamak (a physical device used to create plasmas) in order to achieve a rendering at interactive framerates. We therefore introduce a novel and general method for Raymarching volumes with an adaptive step. A good approximation of the rays' continuous integration is obtained by introducing a Multi-Sampled Pre-integrated table. A comparison with a recent gyrokinetic simulations visualization technique demonstrates the improved efficiency of our approach in terms of framerate and quality.

Key-words: graphics hardware, plasma physics, texture-based methods, volume visualization, raymarching, pre-integration

[*] XLIM, UMR CNRS-UNILIM 6172, Limoges, France
[†] LIRIS, UMR CNRS-UCBL 5205, Lyon, France
[‡] LSIIT-IGG, UMR CNRS-ULP 7005, Strasbourg, France
[§] CEA-DSM-DRFC, Association Euratom-CEA, Cadarache, France

Implantation d'un raymarching adaptatif pour la visualisation de données volumiques issues de simulations gyrocinétiques

Résumé : Ce rapport décrit la mise en oeuvre d'une méthode de rendu volumique efficace adaptée aux données volumiques issues d'une simulation gyrocinétique. Cette méthode est basée sur un lancer de rayon adaptatif implanté en utilisant la programmation des cartes graphiques. La géométrie spécifique d'un tokamak (dispositif physique dédié aux plasmas) est exploitée dans le but d'obtenir un rendu interactif. Une bonne approximation de l'intégration du rayon continu est obtenue en introduisant une table de pré-intégration échantillonnée. Une comparaison avec une implantation récente de visualisation de simulations gyrocinétiques est également faite et démontre l'efficacité de notre approche en terme de qualité et de fréquence d'affichage.

Mots-clés : Matériel graphique, physique des plasmas, visualisation volumique, raymarching, pré-intégration

1 Introduction

Due to the continuous technological advances of 3-D imaging devices (scanner and sensors) and to the need for higher precision as well as accuracy in computational science (numerical simulations), designing real-time visualization applications is currently a real challenge for the scientific visualization and computer graphics communities. Exploring large volumetric scalar fields has been made possible with "Direct Volume Rendering" methods, most of them exploiting graphics hardware capabilities to reach interactive frame rates. We can mainly classify such methods as either *forward projective* (shear-warp factorisation, cell projection, splatting, ...) or *backward projective* (raycasting).

The formers have been quite popular due to their performance achievements and to their tighter correlation with graphics hardware functionalities. Consequently, much research has been driven on improving the approximation of the ray integration throughout the volume as well as on improving visual quality [7, 16, 5, 17, 15, 13].

Although the latter have existed for quite some time in software, Raycasting-like approaches applied to Direct Volume Rendering have suffered from their conceptual incompatibility with the graphics hardware range of functionalities, thus making interactivity out of reach on standard PCs. It is just recently that some papers transposed backward projective methods to the GPU [18, 11].

This paper discusses the visualization of volumetric data resulting from a gyrokinetic simulation. Visualizing such simulations is a new active research topic for which the need for adapted tools is urging, and for which common visualization algorithms do not perform to their fullest due to the spatial arrangement of the data. Recently, Crawford *et al.* introduced the problematic by proposing an approach designed for visualizing gyrokinetic simulations [3] (we describe these simulations in section 2). But due to an expensive geometrical transformation executed per fragment (even for irrelevant ones) interactivity is hardly achieved. Consequently, neither pre-integration nor any shading model has been integrated in their approach, thus leading to residual visual artefacts.

By inspiring ourselves of their resampling scheme, we present in this paper a novel hybrid rendering technique that we define as a combination of forward projective and backward projective methods, where the tokamak's torus-like geometry is taken into account. By resampling in a 3-D texture the unwrapped toroidal data and building a mesh representing the tokamak's crust (see section 4), rendering the volume is achieved in three passes independently of the data's size. The tokamak's crust allows us to do smart empty space skipping by Raymarching only the voxels contributing to the fragments of interest, textures' coordinates being computed in the fragment shader by a world space (wrapped) to texture space (unwrapped) transform operation (see section 4). To allow us the use of adaptive Raymarching steps, while benefiting from pre-integration [5, 15, 13], we introduce a novel Multi-Sampled Pre-integration technique. The latter consists in precomputing multiple pre-integrated tables, each table corresponding to a given slab's thickness. Stacking these tables hierarchically up in a 3-D texture sorted by thickness, allows our fragment shader to dynamically determine and use the appropriate level of detail depending on the ray's length. Note that since each ray is rendered with its own step size, the latter being independent of the others, we preferred calling our algorithm Raymarching instead of the usual term Raycasting.

A complete lighting model (ambiant, diffuse and specular component) further increases the quality of the rendered images.

The remaining parts of the paper are organised as follows: section 2 describes the gyrokinetic simulations process used to generate our data. Section 3 briefly overviews some related works. The next section 4 details how the data have been pre-processed. Section 5 explains how our pre-integrated tables are computed and exploited. Section 6 reviews the three rendering passes of our Raymarching technique applied for visualizing the tokamak. Finally, before concluding, we present in section 7 some results and discuss some implemented optimisations. We also propose in this section a comparative study with previous works, especially [3].

2 Gyrokinetic Simulations

Before describing our method, let us first introduce the physical context of this application, since this is what has driven our main motivations and subsequent choices. The present work deals with the visualization of a simulation computing the ion turbulence in tokamak plasmas [8].

Plasma (the fourth state of the matter) which is found in the stars and the interstellar environment, makes up most of our universe (99%). On Earth, it does not exist in a natural form, apart from lightning and the Aurora Borealis, but it is produced artificially by applying magnetic fields powerful enough to separate the kernel from its electrons in gases. The torus-shaped tokamak configuration is ideal to obtain controlled thermonuclear fusion based on magnetic confinement. The thermal confinement of a magnetized fusion plasma is essentially determined by turbulent heat conduction across the equilibrium magnetic field. Understanding the turbulent transport in a tokamak configuration is therefore a key issue for controlled fusion. In practice, the study of plasma turbulence requires to solve Maxwell's equations coupled to the calculation of the plasma response to the perturbed electromagnetic field (Vlasov equation). This response can be computed by using a kinetic description of the plasma. In principle, one has to solve a 6-D kinetic equation (3-D in space and 3-D in velocity) to determine a distribution function, which yields current and charge densities once integrated over the velocity space. Nowadays computing facilities are not get able to handle this 6-D kinetic problem.
Using the fact that turbulence frequencies are lower than the cyclotron frequency, scientists in the fusion community have developed a new kinetic equation, called gyrokinetic, describing the distribution function in the 4-D phase space, parameterized by the adiabatic invariant, $\mu = \frac{mv_\perp^2}{2B}$ the action variable associated to the gyrophase. This 5-D gyrokinetic problem is still very demanding in terms of numerics.

Two methods have been used up to now to investigate turbulence in the gyrokinetic regime. The first method is based on a Lagrangian approach. Particle In Cell (PIC) codes, which are the most widely used in this category, consists in describing the plasma with a finite number of macro-particles. The trajectories of these particles are the characteristic curves of the Vlasov equation, whereas self-consistent fields are computed by gathering the charge and current densities of the particles on a mesh of the physical space. Some recent work has been proposed to visualize the large amount of generated time-space particles [14, 2].

The second method is Eulerian. It consists in discretizing the Vlasov equation on a mesh of the phase space that remains fixed in time. The Flux Balance Method (FBM) uses a finite volume method for computing the average of the Vlasov equation on each cell on a fixed grid.

The simulation that we use is based on an intermediate Semi-Lagrangian method [6]. The purpose of the SL method is to take advantage of both the Lagrangian and Eulerian approaches, to have a good description of the phase space, in particular in regions where the density is low, as well as an enhanced numerical stability. In this approach, the grid is kept fixed in time in the phase space (Eulerian method) and the Vlasov-equation is integrated along the trajectories (Lagrangian method) using the invariance of the distribution function along the trajectories. Cubic spline interpolations are performed to evaluate the new value of the distribution function on the grid points.

Here the full distribution function f is calculated in contrast with δf PIC codes, used in gyrokinetic simulations, that only calculate the perturbed distribution function. The simulation used for this paper is applied to a cylinder geometry with a reduction of the phase space to 4-D. Such simulations have been performed to prove the interests of the SL method in [8]. The goal in the future is to investigate turbulent transport in 5-D in a realistic tokamak geometry together with a relevant physics of low frequency turbulent activity. But even in 4-D, the amount of data remains very large. Regular grids of resolution 64 already require 128MB in double floating point precision for each time step.

In the 4-D version, a periodic cylindrical plasma of radius a and length $2\pi R$ is considered as a limit case of a stretched torus. The plasma is confined by a strong magnetic field which is uniform $\vec{B} = B\vec{e_z}$ where $\vec{e_z}$ stands for the unit vector in the toroidal direction z. Concerning the ions, finite Larmor radius effects are neglected so that the trajectories are governed by the guiding-center (GC) trajectories. Let us introduce some notations:

$$\frac{dr}{dt} = v_{GC_r} \; ; \; r\frac{d\theta}{dt} = v_{GC_\theta} \; ; \; \frac{dz}{dt} = v_\parallel \; ; \; \frac{dv_\parallel}{dt} = \frac{q}{m_i}\vec{E}_z, \tag{1}$$

where v_{GC_r} and v_{GC_θ} are the radial and poloidal components of the $E \times B$ drift velocity $\vec{v}_{GC} = \frac{\vec{E}\times\vec{B}}{B^2}$, $\vec{E}$ being the electric field, $q = Ze$ the ion charge and m_i the ion mass and $v_\parallel$ corresponds to the velocity along the magnetic field lines. This simplified cylinder configuration does not take into account the toroidal effects but allows one to study ion temperature gradients driven modes.

Given these assumptions, the distribution function f is a 4-D phase space function that depends on the three cylindrical coordinates (r, θ, z) and on the parallel velocity $v_\parallel$. The evolution of this distribution function $f(r, \theta, z, v_\parallel, t)$ is then described by the drift-kinetic Vlasov equation:

$$\frac{\partial \vec{f}}{\partial t} + \vec{v}_{GC} \, .\vec{\nabla}_\perp f + v_\parallel \frac{\partial f}{\partial z} + \frac{q}{m_i} E_z \frac{\partial f}{\partial v_\parallel} = 0, \tag{2}$$

where $\vec{\nabla}_\perp = (\frac{\partial .}{\partial r}, \frac{1}{r}\frac{\partial .}{\partial \theta})$. This equation couples the $\vec{E} \times \vec{B}$ motion across the magnetic field to the motion parallel to the magnetic field. Self-consistency is ensured by the quasineutrality equation that relates the electric potential Φ to the first moment of the distribution function, namely :

$$-\nabla_\perp \cdot \left[\frac{n_0(r)}{B\Omega_0} \nabla_\perp \Phi \right] + \frac{e \, n_0(r)}{T_e(r)} \left(\Phi - \langle \Phi \rangle \right) = n_i - n_0 \tag{3}$$

where $\Omega_0 = \frac{q_i B_0}{m_i}$ is the ion cyclotron frequency, and T_e and n_0 are respectively the electron temperature and density profiles. The ion density profile is given by $n_i(r, \theta, z, t) = \int dv_\parallel \, f(r, \theta, z, v_\parallel, t)$ and $\langle \cdot \rangle$ represents the average on the magnetic field lines.

This paper presents the visualization of this gyrokynetic simulation *i.e* a volume rendering technique designed to visualize for a given t and a fixed $v_\parallel$, the distribution function $f_{v_\parallel}(r, \theta, z)$ discretized on a cylinder of resolution $64 \times 128 \times 64$.

3 Previous Work

Since Lacroute and Levoy [12] introduced the notion of texture based approaches for volume rendering, many extensions have been developed on the shear warp factorisation scheme. Conceptually, trilinear interpolation of the volume's scalar set is substituted by rendering object aligned slices in an ordered way. Cabral *et al.* [1] exploit 3-D hardware acceleration based on the same principle, followed by others [20, 4, 19].

Much work has been done on improving the integration of the rays through the volume [5, 15, 13] and on adding realism by integrating lighting models [7, 16, 9, 10].

In comparison, raycasting approaches have been less explored as their implementations don't trivially transpose onto the graphics hardware. However, some papers addressed this matter with original concepts as well as interesting results [18, 11].

By taking into account the scheme used for sampling the scalars during a gyrokinetic simulation, Crawford *et al.* [3] introduced a novel method for resampling the data to meet the memory limitations of current graphics hardware. Furthermore, they describe a hardware based volume rendering technique designed to visualize plasmas whereas before, gyrokinetic simulations were mainly analyzed using isosurfaces.

However, applying the shear warp factorisation to the bounding box of the tokamak lacks of performance in terms of frame rate as the world space to texture space expensive transformation is done per fragment, even for the pixels that will not contribute to the final image. Given the nature of the physical device where samples are collected (the center being hollow), more slicing planes must be used to achieve a correct rendering, coming at the expense of more computations and thus a lower framerate.

4 Preprocessing the Data

We place ourselves in a right handed coordinate system for the rest of the paper.

4.1 Data caracteristics

The tokamak's center is placed at the origin of our coordinate system, bringing symmetry relatively to the xy, yz and xz planes. Samples issue from the simulation are arrayed evenly along the y axis, slicing the volume in poloidal planes with a constant angular step Δ_γ, each describing an angle γ relatively the xy plane.

The poloidal planes are futhermore decomposed into rings of linearly decreasing radii $r.\Delta_r$. Along this rings the samples are distributed with a constant angular step Δ_θ (see Figure 1). Note that our numerical simulation's sampling scheme is different from the one established by the Princeton Plasma Physics Laboratory[3], since the meshes along a ring are distributed with respect to the current ring's radius.

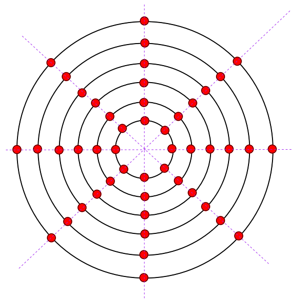

Figure 1: Samples of a typical poloidal plane. Constant radial progression of the meshes for all the rings can be observed, implying oversampling for the rings of low radius.

4.2 Building the crust and normalisation

For each poloidal plane, we consider only the rings of maximum radius r_{max}. We then loop through the concerned samples for building correctly oriented faces (in the case of our implementation, we chose the counter clockwise orientation). Since we can omit the r coordinate (the rings of lowest radii are ignored during this phase), vertices can be parameterised by their major angle γ (determining the current poloidal plane) and their minor angle θ (determining the samples located on the ring of radius r_{max}). With $v(r_{max}, \theta, \gamma)$ being the vertex actually processed, we triangulate two faces of the crust (see Figure 2):

$$\begin{cases} v(r_{max}, \theta, \gamma) & v(r_{max}, \theta + \Delta_\theta, \gamma) & v(r_{max}, \theta + \Delta_\theta, \gamma + \Delta_\gamma) \\ v(r_{max}, \theta, \gamma) & v(r_{max}, \theta + \Delta_\theta, \gamma + \Delta_\gamma) & v(r_{max}, \theta, \gamma + \Delta_\gamma) \end{cases}$$

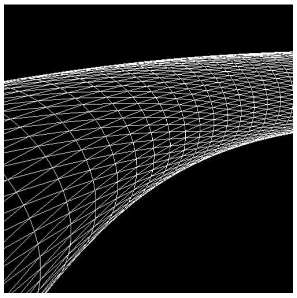

Figure 2: Visualizing in wireframe the tokamak's crust builded with the furthest rings' samples.

Building the crust of the tokamak can be achieved in linear complexity. We only store the vertices world space coordinates which we index in our list of faces.

For further optimisation, we apply some geometrical transformations to the mesh previously built:

- We scale our object by a factor of $\frac{1}{2.r_{max}}$, so that the revelant surface of the poloidal planes is the surface of the unit circle, π.
- A positive shift of $\frac{1}{2}$ along the y axis is then applied to the whole mesh.

In this way, the y components of our meshes are bounded to $[0, 1]$, which will allow us later to reduce computations with regard to the world space coordinate to texture space coordinate transformation, as explained in the next subsection 4.3.

4.3 Unwrapping the data

Considering the spatial distribution of the samples collected by the gyrokinetic simulation, a standard 3D texture resampling approach would imply a waste of memory storage, as well as a loss of information due to memory limitations on graphic hardware. This is due to the fact that the center of the tokamak and the corners of its bounding box are free from samples. Crawford *et al.* [3] use a smart coordinate system change to map the toroidal data to a basic rectangular 3-D texture.

Basically, we take advantage of the poloidal planes being sampled linearly by δ_γ to create a 3-D texture where γ is directly mapped to the u texture coordinate. That is, each slab of our texture along the height represents the samples contained in one poloidal plane. Resampling occurs internally of these slabs by re-interpolating bilinearly along the r and θ component, so that consistent values are affected to the voxels' centers (see Figure 3).

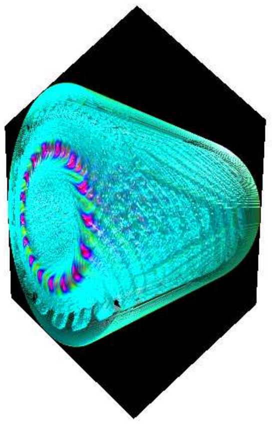

Figure 3: The data of the Gyrokinetic Simulation after being re-sampled and unwrapped, resulting in a 3D texture. This capture has been done using our Raymarching together with some shading, as described in this paper.

During rendering, texture coordinates can be easily recovered by applying the inverse transformation to the world space coordinate of the fragment. Indeed, considering the torus parametric equation (which we apply to the crust we previously built), with r_{max} and R being respectively the minor and the major radii, and α and β being respectively the minor and major angles, we have:

$$\begin{cases} x(\alpha, \beta) = (R + r_{max}.cos(\beta)).cos(\alpha), \\ y(\alpha, \beta) = r_{max}.sin(\beta), \\ z(\alpha, \beta) = (R + r_{max}.cos(\beta)).sin(\alpha). \end{cases}$$

Recovering texture coordinates from a vertex world space coordinates (x, y, z) is now given by the formula:

$$\begin{cases} s = \sqrt{x^2 + z^2} - (R - r_{max}), \\ t = y, \\ u = \frac{atan(z,x)}{2\pi}. \end{cases} \tag{4}$$

In order to be able to incorporate a lighting model, zero scalars are added to the borders of the textures's width and height all along the depth. These are used to enable the computation of a gradient necessary for lighting, also on these border samples.

5 Multisampled Pre-Integration Table

5.1 Motivations

Pre-integrated Volume Rendering [5, 15, 13] is a technique that has the advantage of capturing much more features inside a volume with less samples. However, it is necessary that the step along the casted rays be constant in order to have a correct approximation. Recently, Shaders Model 3.0 made loops core part of the GPU, allowing us to do Hardware Raycasting. When using such an approach, care must be taken with the length of the Shaders' programs as well as their complexity for the sake of interactivity, leading us to the next problem:

As the length of a ray may vary significantly depending on the region we are in and on the view vector's direction, a compromise must be done:

- By choosing a large step, we can satisfy the integration of the longest rays. But, when raycasting smaller regions, the volume integration will be done with very few samples, which may lead to some more or less important visual artefacts.
- On the opposite, by choosing a small step, the rendering will be improved for narrow regions, but we will not be able to fully integrate the longest rays due to the hardware limitations. This may have serious consequences when the traversed part of the ray is translucent and the rest of it contains very opaque data, thus giving the impression of a Z-far clipping plane, especially when moving the viewpoint.

In order to benefit from Pre-integration as well as from a hardware Raycasting approach, we propose a method where we march along the rays with an adaptive step and where color and opacity information is fetched in what we call a Multi-Sampled Pre-integrated Transfer Function Set (MSPITFS).

5.2 Precomputing the Multi-Sampled Pre-integrated Transfer Function Set

We consider the case of one-dimensional and univariate transfer functions. Since such a transfer function can be stored as a one-dimensional texture, its related pre-integrated table is stored as a two-dimensional texture [5, 15, 13]. Subsequently, our Multi-Sampled Pre-integrated Transfer Function Set will be represented as a three-dimensional texture; that is, a stacked set of two-dimensional tables, each table representing a given integration step. Conceptually, one may consider this set of tables as a sort of LOD (Level Of Detail) representation:

- The lowest level value corresponds to the pre-integrated transfer function that has been computed using the maximum number of samples we allow along one ray (this corresponds to the maximal size of the integration step).
- The highest level value represents the transfer function itself, that is an integration step of size zero.

As is common for LOD, we apply powers of two for the different integration steps in the MSPITFS.

We also impose that the maximum number of samples along a ray be a power of two, $2^{lod_{max}}$. Furthermore, the value of lod_{max} (which is consequently a positive integer) determines the number of pre-integrated transfer function tables in our 3-D table stack, which is $nb_{lod} = (lod_{max} + 1)$. Figure 4 illustrates a stack example of eight pre-integration tables, stored on the GPU as a 3-D texture of height eight. The integration step grows as a power of two. The left bottom image corresponds to an integration step of size zero, thus matching the transfer function. Note that the tables become more and more similar as the integration step increases.

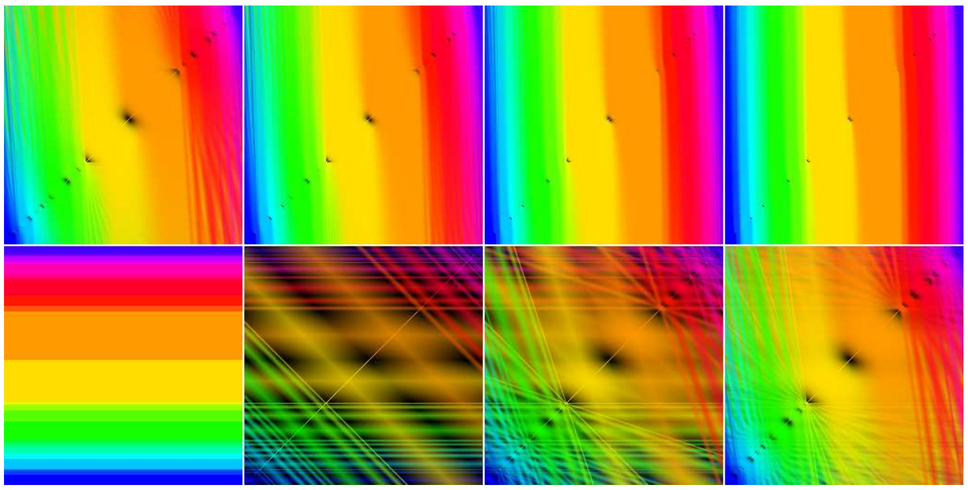
Figure 4: Eight consecutive tables of the Multi-Sampled Pre-Integrated Transfer Function Set are shown, increasing in terms of step size from bottom to top and from left to right. Note that the bottom left image corresponds to the transfer function and represents the highest LOD level.

5.3 Using the Multi-Sampled Pre-Integrated Transfer Function Set

The purpose of the MSPITFS is to fetch well approximated color and opacity values according to the current integration step size related to the currently processed pixel. Before determining the depth component (Level Of Detail) u in the table stack, we need to precompute the length of the longest ray we may encounter for a specific volume, let it be $length_{max}$.

In the case of the tokamak, we have $length_{max} = 4\sqrt{r.R}$, with r and R being the minor and major angles from the torus parameteric formula (see Figure 5).

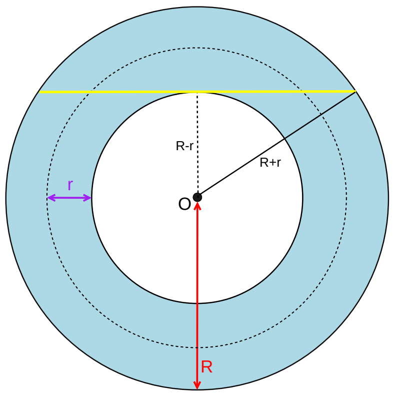

Figure 5: Viewing a torus from top. We show how the longest ray's length (in yellow) can be easily computed from the tokamak's parameteric equation. r is the minor radius and R the major radius of the torus.

During rendering, the correct table in the stack is accessed as follows: for a given ray, knowing the spatial localisation of its starting and ending vertices, we compute the length of its directionnal vector : $length_{ray}$.

Determining the table coordinate u within the 3-D stack is done by applying the following equation $u = \frac{\log \frac{length_{max}}{length_{ray}}}{nb_{lod}}$, log being the logarithmic function in base 2. We notice that :

$$\begin{cases} \text{if } length_{ray} \;=\; length_{max}, \;\; u = 0 \\ \lim length_{ray} \rightarrow 0, \;\; u = \;\; +\infty \end{cases}$$

By enabling clamping, we make sure that we always have $u \in [0, 1]$. In the case of rays with very small length $length_{ray}$, this allows us to force the lowest level of detail in our MSPITFS, to be 1 (that is the table on the bottom left of Figure 4). The value 0, on the contrary, represents the highest level of detail (that is the table on the top right of Figure 4).

Futhermore, by enabling 3-D texture linear interpolation in hardware, the Pre-integrated tables for each ray will result from averaging its two nearest neighbours. This provides in many cases (depending on the transfer function and the current depth) smooth transitions between neighboring pixels. Figure 6 illustrate a comparison between different level of details.

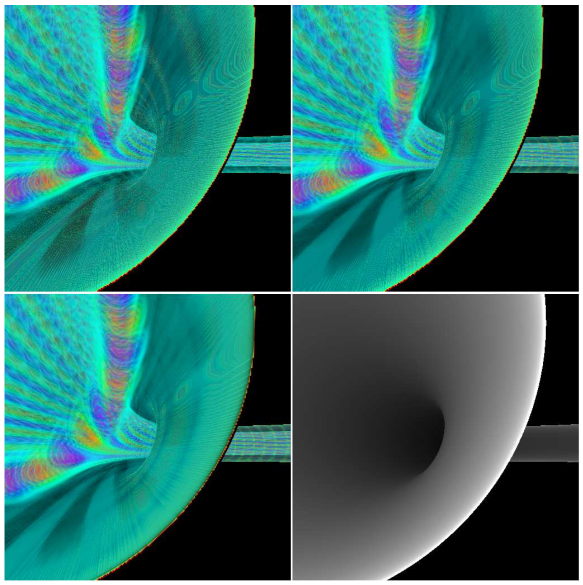

Figure 6: Visualizing the same scene with three different data space to color space transformations, from top to bottom and from left to right : Using a transfer function without pre-integration, stantard pre-integration, the multi-sampled pre-integrated transfer function set (12 levels of details). The last image (lower right) shows the level of details that has been determined in the shader by assigning darker colors to higher level of details. The number of steps per ray has been fixed to 32.

6 Rendering Passes

6.1 Hardware

We take advantage in our algorithm of the `GL_EXT_framebuffer_object` openGL 1.5 extension for the purpose of drawing to rendering destinations other than the buffers provided by the window-system. We though use for offscreen rendering a virtual framebuffer having two auxiliaries buffers of the size of our viewport, each fragment being defined by three components (commonly named R, G and B) in the standard IEEE 32bit floating-point format. We also make use of a zbuffer for depth sorting.

Recently, Programmable Hardware made a significant advance with the Shader Model 3.0, bringing new features of interests without whom our implementation would not have been possible :

- Vertex and Fragment shaders can have up to 65535 instructions.
- Shaders benefit internally from 32bit floating point precision.
- Dynamic flow control allows efficient branching in both Vertex and Fragment Shader.

- Dynamic loops for the Vertex Shader as well as static loops in the Fragment Shader are made possible.
- Early-outs and fast fragment discarding allows several optimisations.

We now detail each of the passes.

6.2 Rendering the tokamak

6.2.1 First Pass

We clear the first auxiliary buffer with values that will allow us to determine if the fragment shader has written or not to a pixel. As world space coordinates of the tokamak's faces will be written in the framebuffer, we take advantage of the normalisation that occurred when building the crust. Knowing that all the y components are bounded to $[0,1]$, a good choice for our clearing triplet would be $(0, discard\_value, 0)$ with $discard\_value \notin [0,1]$. As a matter of fact, we chose for our implementation $(0,2,0)$. We then render offscreen the back faces of the tokamak's crust in the first auxiliary buffer, the ztest being enabled to keep the nearest fragments. The vertex shader computes the view space coordinate induced by our current "ModelView" matrix so that the correct fragments are enabled (correct in the sens of localisation in the framebuffer). Furthermore, we take advantage of the vertex shader capabilities in terms of linear interpolation to interpolate the current world space coordinate which is later on passed to the fragment shader by the mean of a varying variable. The vectors $(x_{back}, y_{back}, z_{back})$ are written in the current framebuffer.

6.2.2 Second Pass

We now render the front faces of the tokamak's crust in the second auxiliary buffer. Before doing so, a special pass is carried out for handling the case of inner volume exploration. Indeed, the crust being hollow, further computations are required so that the near clipping plane world space coordinates are determined per fragments. This is achieved by completely projecting the front clipping plane into our current buffer:

By knowing, in the world space coordinate system, our camera location $camera\_position$, its normalised vectors $x_{view\_direction}$, $y_{view\_direction}$, $z_{view\_direction}$ defining our viewing coordinate system and the value of the z_{near} clipping plane, we render a quad intersecting our view frustum in the second auxiliary buffer with depth testing being disabled. Let fov_x, respectively fov_y, be the field of view in degrees in the x direction, respectively in the y direction. The quad is defined by four vertices as follows:

$$\left\{\begin{array}{rl} vertex\_1 = & camera\_position + z_{near}.atan(\frac{fov_x}{2}).x_{view\_direction} \\ & -z_{near}.atan(\frac{fov_y}{2}).y_{view\_direction} + z_{near}.z_{view\_direction} \\ vertex\_2 = & camera\_position - z_{near}.atan(\frac{fov_x}{2}).x_{view\_direction} \\ & -z_{near}.atan(\frac{fov_y}{2}).y_{view\_direction} + z_{near}.z_{view\_direction} \\ vertex\_3 = & camera\_position - z_{near}.atan(\frac{fov_x}{2}).x_{view\_direction} \\ & +z_{near}.atan(\frac{fov_y}{2}).y_{view\_direction} + z_{near}.z_{view\_direction} \\ vertex\_4 = & camera\_position + z_{near}.atan(\frac{fov_x}{2}).x_{view\_direction} \\ & +z_{near}.atan(\frac{fov_y}{2}).y_{view\_direction} + z_{near}.z_{view\_direction} \end{array}\right. \tag{5}$$

We chose a field of view of 90 degrees for both x and y, which is a common perspective tranform in 3D applications. We now re-enable the depth test and render the front faces of the tokamak's crust, writing the world space coordinate of our fragments as done in the first pass.

6.2.3 Third Pass

We now use an orthogonal viewing transformation and render a quad face of the size of the viewport, passing to each of its vertices adapted texture coordinates. The two previous auxiliary buffers from our offscreen framebuffer are binded as two distinct textures units in which we are going to read the previous interpolated values.

For every processed fragment, the interpolated texture coordinates are used to fetch the correct texels in our textures representing the back faces and front faces. A first texture look up is done with the back face texture. By reading the y component of our texel, we discard the current fragment if $y \notin [0, 1]$ meaning no fragment has been written, otherwise we launch the raymarching algorithm.

We compute the ray direction knowing the location in world space of the front and back fragments, $ray = front_{position} - back_{position}$ The number of raymarching steps nb_{step} being predetermined in our fragment shader, we divide all three components of the vector ray by n to obtain the ray's vector in world space coordinate, $ray_{step} = \frac{ray}{nb_{step}}$.

Once the spatial information for the current samples has been collected, we use the previously described procedure to determine the correct index u into the MSPITFS. Once this operation has been done for a given fragment, we know what pre-integrated table has to be used for completely integrating the pixel, since once computed, the steps's length is kept constant along the ray. Figure 7 illustrates an example of raymarching.

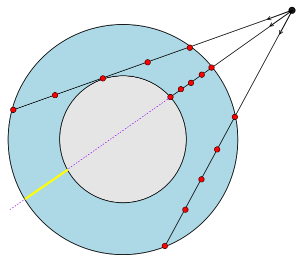

Figure 7: This figure illustrates how rays from a fixed point of view are marched adaptively depending on their length and on the number of samples that has been predefined (for illustration purpose, we chose 5 samples per ray). Notice the ray traversing two times the tokamak, only the nearest integration is taken into account, the yellow sub part being ignored.

Before entering our main loop, a last operation is required. Since we want to benefit from the pre-integration (which uses the scalar data of two consecutive slices), we need to fetch for our front (first) sample its related data in the 3D texture and store it temporarily. We know begin the integration:

Starting from the second sample along the ray, we raymarch nb_{step} times along our vector, the current world space coordinate allowing us to recover the normalised texture coordinate of our unwrapped toroid and thus fetching the correct voxel (which scalar value has been interpolated by the hardware).

By keeping track at each iteration of the previously collected sample's scalar value s_1, we retrieve the correct color and opacities information in the MSPITFS for the current scalar s_2 by indexing the 3D texture containing our pre-integrated tables with (s_1, s_2, u), u being held constant along the ray.

Futhermore, blending the collected color intensities is done in the shader, allowing us not to use the fixed pipeline functionality and thus avoiding multiple read and write access to the framebuffer. The front to back rendering is done with respect to the traditional back to front formula, that is:

$$C = \sum_{i=0}^{n} \alpha(i)\, C(i) \prod_{j=0}^{i-1} (1 - \alpha(j)). \tag{6}$$

Notice that in our case no duplicates of the first and last poloidal planes have to be stored at the textures opposite borders, as opposed to Crawford *et al.*'s approach [3].

Figure 8 summarizes the three passes.

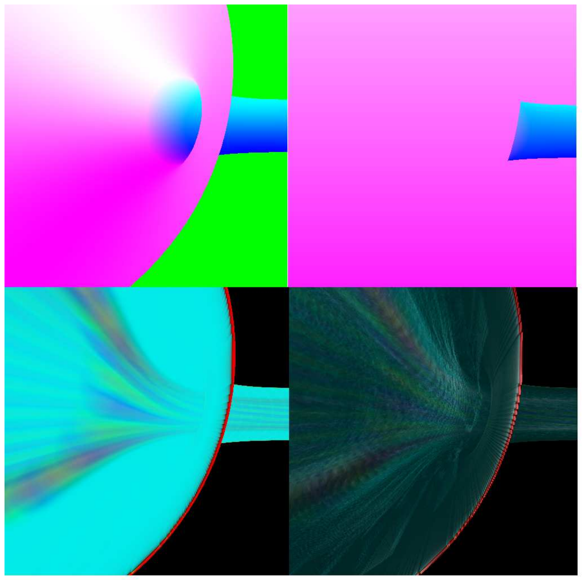

Figure 8: The Tokamak seen half inside and half outside. On the top line, we visualize the two offscreen passes. The left one shows the rendering of the back faces and the right one shows the front faces. The green color on the left image represents our $y_{discard}$ value which is used to clear the buffer. On the right, most of the colors are pink due to the z_{near} plane being projected and its localisation. On the bottom, the same scene is rendered with our Raymarching approach without (left image) and with a lighting model (right image). 32 samples are used per ray.

7 Results

We have implemented our method on a AthlonXP1800 with a GeForce6800 Ultra graphic card using the OpenGL API as well as the OpenGL Shading Language. Note that Shader Model 3.0 has only been made accessible by the use of beta drivers (Nvidia's Forceware 76.50) as GLSL shaders are compiled by the graphic card itself.

For a viewport of 512×512 pixels and a plasma dataset resampled into a 512^3 8bit precision 3-D texture, we observe the following results:

	Ray-TF	Ray-PI	Ray-MSPITFS	SW-TF
Unshaded	29	23	22	14
Shaded	22	16	11	4

With Ray standing for Raymarching, SW for shear warp, TF for transfer function, PI for pre-integration and MSPITFS for multi-sampled pre-integrated transfer function set. We used in the case of the Raymarching approach 12 levels of details for the MSPITFS and 32 steps per ray. For the slicing approach[3], 200 object aligned planes have been used on the bounding box of the tokamak.

Notice that the atan instruction wich is used once per sample comes at the cost of 29 cycles, using a precalculted table stored in a 2D texture would certainly lead to speed improvements as there is currently no hardware support for it.

One interesting point of our method can be observed in Figure 9, where blending is only done for the first continuous rays traversing the tokamak, and thus respecting in terms of visualization the informations driven by the physical device. Figure 10 illustrates an external view of the tokamak.

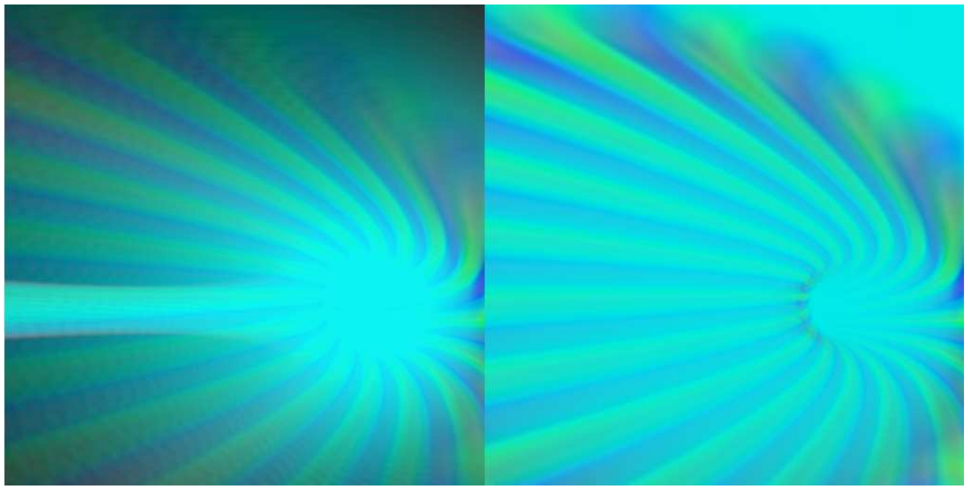

Figure 9: We visualize a scene inside the Tokamak. On the left, a standard slicing has been done as on the right we use our Gyrokinetic specific Raymarching approach. We notice on the left capture that the furthest part of the tokamak is rendered due to translucency. However, such results may lead physicans to misanalyse the data. Our approach allows the blending of the revealing volume by discarding the theorically hidden samples.

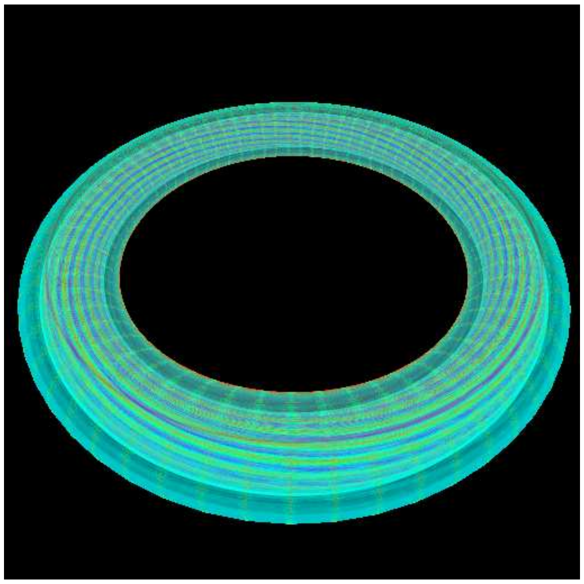

Figure 10: Outside view of the tokamak using raymarching along with multi-sampled pre-integrated transfer function set, 32 steps of raymarching.

8 Discussion and Conclusions

We have presented in this paper a novel method for Direct Volume Rendering using a Raymarching approach where color space attributes are determined in an adaptive way using a Multi-Sampled Pre-integrated Transfer Function Set. Although such a technique can be applied for standard Volume Rendering (*i. e.* using a projected bounding box for instance), it has been developed for the purpose of visualizing efficiently Gyrokinetic simulations.

The study of plasmas by computer scientists being at its early beginning, huge data sets are to be analyzed requiring appropriate visualization tools for the physics science community. Ideally, such applications should provide high tuning capabilities with the generated visual information being displayed immediately. In addition, for better data understanding, the overall quality of the rendering is primordial, which usually further increases the complexity of the scientific visualization.

Previous Gyrokinetic simulations visualization techniques (isosurface extraction, shear warp factorization) could not handle the constraint of interactivity along with high quality rendering. By Raymarching the volume using the tokamak's crust, we have been able to visualize Gyrokinetic data sets using adaptive pre-integration, thus leading to improved results from a qualitative point of view. However, since we used a texture-based approach, data re-sampling implies a loss of precision, while current hardware memory specifications limit the size of the numerical simulations to be visualized. Due to the nature of plasmas, data evolves in time-space as well as in phase-space, making the challenge for visualizing interactively time-varying samples even greater. Our future research includes the survey of different compression techniques as well as researching new time-varying Volume Rendering approaches.

References

[1] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization*, pages 91–98, New York, NY, USA, 1994. ACM Press.

[2] C.S. Co, A. Friedman, D.P. Grote, JL. Vay, and E. Wes Bethel. Interactive methods for exploring particle simulation data. *IEEE Visualization*, 2, 2004.

[3] D. Crawford, K-L. Ma, M-Y. Huang, S. Klasky, and S. Ethier. Visualizing gyrokinetic simulations. In *Proceedings of the IEEE Visualization 2004 Conference*, 2004.

[4] F. Dachille, K. Kreeger, B. Chen, I. Bitter, and A. Kaufman. High-quality volume rendering using texture mapping hardware. In *HWWS '98: Proceedings of the ACM Siggraph/Eurographics workshop on Graphics hardware*, pages 69–ff., New York, NY, USA, 1998. ACM Press.

[5] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *HWWS '01: Proceedings of the ACM Siggraph/Eurographics workshop on Graphics hardware*, pages 9–16, New York, NY, USA, 2001. ACM Press.

[6] F. Filbet, E. Sonnendrücker, and P. Bertrand. Conservative numerical schemes for the Vlasov equation. *J. Comput. Phys.*, 172(1):166–187, 2001.

[7] A. Van Gelder and K. Kim. Direct volume rendering with shading via three-dimensional textures. In *VVS '96: Proceedings of the 1996 symposium on Volume visualization*, pages 23–ff. IEEE Press, 1996.

[8] V. Grandgirard, M. Brunetti, P. Bertrand, N. Besse, X. Garbet, P. Ghendrih, G. Manfredi, Y. Sarazin, O. Sauter, E. Sonnendrücker, J. Vaclavik, and L. Villard. A drift-kinetic Semi-Lagrangian 4d code for ion turbulence simulation. *submitted to Journal of Computational Physics.*

[9] J. Kniss, S. Premoze, C. Hansen, and D. Ebert. Interactive translucent volume rendering and procedural modeling, 2002.

[10] J. Kniss, S. Premoze, C. Hansen, P. Shirley, and A. McPherson. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):150–162, 2003.

[11] J. Krueger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings IEEE Visualization 2003*, 2003.

[12] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 451–458, New York, NY, USA, 1994. ACM Press.

[13] E. B. Lum, B. Wilson, and K-L. Ma. High-quality lighting for pre-integrated volume rendering. In *VisSym*, pages 25–34, 2004.

[14] K.-L. Ma, G. Schussman, B. Wilson, K. Ko, J. Qiang, and R. Ryne. Advanced visualization technology for terascale particle accelerator simulations. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–11. IEEE Computer Society Press, 2002.

[15] M. Meissner and S. Guthe. Interactive lighting models and pre-integration for volume rendering on pc graphics accelerators. In *Graphics Interface*. Canadian Information Processing Society, 2002. to appear.

[16] M. Meißner, U. Hoffmann, and W. Straßer. Enabling classification and shading for 3d texture mapping based volume rendering using opengl and extensions. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 207–214, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[17] M. Meißner, S. Guthe, and W. Straßer. Higher quality volume rendering on pc graphics hardware. In *Graphics Interface*, 2002.

[18] S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, and W. 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.

[19] R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 169–177, New York, NY, USA, 1998. ACM Press.

[20] O. Wilson, A. VanGelder, and J. Wilhelms. Direct Volume Rendering Via 3D Textures. Technical Report UCSC-CRL-94-19, 1994.

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399