INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

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

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

N° 0319

Mars 2006

Thème NUM D

ISSN 0249-0803

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

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

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

Rapport technique n° 0319 — Mars 2006 — 52 pages

Résumé : Ce rapport technique 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 implentation 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

* 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

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Adaptative Raymarching Implementation for gyrocinetic data visualization

Abstract: This technical report describes an implementation of an efficient volume rendering method designed to visualize gyrokinetic simulations. It is based on a raymarching approach using 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. A good approximation of the rays' continuous integration is obtained by introducing a Multi-Sampled Pre-integrated table. A comparison with a another recent implementation of a gyrokinetic volume visualization demonstrates the improved efficiency of our approach in terms of framerate and quality.

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

1 Introduction

Dans la quête actuelle de nouvelles ressources énergétiques, la fusion thermonucléaire constitue un des enjeux majeurs de ce siècle. Mais la réalisation de réacteurs nécessite la détermination de champs magnétiques permettant de confiner suffisamment longtemps un plasma très chaud. La complexité du phénomène rend indispensable le recours à des simulations numériques avant de procéder à des tests en vraie grandeur. C'est pourquoi, des codes de simulations numériques sont développés pour mieux comprendre les phénoménes liés à la physique des plasmas. Mais la difficulté inhérente à ces simulations réside dans l'étude et l'analyse de la grande masse de données multidimensionnelles générées. Il est alors nécessaire d'offrir aux physiciens des outils de visualisation adaptés et performants, facilitant l'exploitation des résultats.

Notons que la visualisation scientifique de plasmas est un problème difficile en raison du caractère multidimensionnel des résultats que fournissent les simulations. Chaque particule est en effet représentée par un vecteur à six dimensions dans l'espace des phases (produit de l'espace physique et de l'espace des vitesses), engendrant des « hypervolumes » de données considérables. Ces volumes de données, de plusieurs méga-octets pour chaque pas de temps, mettent souvent à défaut les outils actuels de visualisation. Il est alors courrant d'utiliser des projection ou coupes au sein des hypervolumes pour réduire la dimension du problème. C'est également ce que nous proposons de faire dans ce travail. L'objectif est donc de concevoir et de développer un outil permettant, par le biais d'un rendu volumique, d'explorer des données plasma réduite à trois dimensions (par intégration le long de l'espace des phases). Ces données sont issues de simulations gyrocinétiques de plasma du CEA (Commissariat à l'Energie Atomique) de Cadarache.

La suite de ce rapport se compose de huit parties. Dans la section 2, un bref état de l'art des techniques de visualisation dédiées au rendu de champs scalaires 3D est dressé. Dans la section 3, la problématique des plasma est abordée afin de situer le contexte. Une implantation récente de visualisation de simulations gyrocinétiques y est aussi décrite. La section 4 explicite le pré-traitement des données numériques nécessaire à leur visualisation, notre méthode étant décrite de façon détaillée par la suite. Des optimisations techniques sont décrites dans la section 5. L'implantation machine de la méthode est abordée dans la section 6, avec une description des fonctionnalités de l'interface. La section 7 dresse une liste de résultats obtenus par l'usage de notre approche, une comparaison étant effectuée avec la plus récente implantation de rendu de simulations gyrocinétiques. Finalement, la section 8 dresse un bilan de ce travail et décrit nos objectifs futurs dans le cadre de la visualisation des plasmas.

2 Etat de l'art

2.1 Introduction

Avec l'importance croissante des simulations numériques dans certains domaines scientifiques, les communautés de la visualisation scientifique et plus particulièrement du rendu volumique se trouvent confrontées à des masse de données (champs scalaires, champs vectoriels...) de plus en plus importantes. Notons que nous nous placerons par la suite dans le domaine particulier des champs scalaires 3D et du rendu volumique. Contrairement aux méthodes de rendu de surfaces (par exemple, avec un maillage sous la forme de polygones), le rendu volumique consiste à visualiser des champs de données par transparence, permettant ainsi de visualiser ce qui se trouve à l'intérieur essentiellement par la manipulation d'une fonction de transfert.

Durant ces dernières années, de très nombreuses contributions en rendu volumique ont été apportées afin de satisfaire les besoins d'exploration et d'analyse des données. Les approches étant au départ purement logicielles puis réservées au matériel haut de gamme, l'avancée considérable de l'architecture des cartes graphiques permet depuis quelques années la visualisation de données numériques sur PC standard, donnant ainsi lieu à une nouvelle famille d'algorithmes tirant profit des possibilités offertes par ce matériel.

Dans la suite de cette partie, nous proposons un bref descriptif de quelques techniques de rendu volumique (section 2.4) après avoir presenté dans la section 2.2 la nature des données manipulées et leur transcription en

information colorée par le biais de la fonction de transfert. La section 2.5 montre comment il est possible de rendre de telles visualisations plus convaincantes par l'introduction de modèles d'illumination. Nous décrivons ensuite dans la section 2.6, le principe de la « pré-intégration ». Cet état de l'art, qui se veut bien évidemment loin d'être complet, s'adresse surtout aux lecteurs qui ne sont pas familiers avec le rendu volumique. Il offre ainsi une introduction à ce type de visualisation.

2.2 Nature des données à visualiser

En raison de traitements informatiques, les valeurs des données (supposées discrétisées) sont généralement converties en entier ou flottants codés sur 1, 2, 4, voire 8 octets. En raison des limitations mémoires des cartes graphiques ainsi qu'en raison de la taille des données souvent importante, des entiers codés sur 1, voire 2 octets sont couramment utilisés en visualisation.

Afin de bénéficier des performances des cartes graphiques, il est nécessaire de stocker les données sous la forme d'une texture 3D (tableau tridimensionnel de valeurs discrètes), qui constitue un maillage structuré de l'espace. On affecte ainsi à tout voxel (unité de volume correspondant au plus petit élément d'une texture 3D) une valeur calculée en fonction des données initiales, ce qui peut engendrer dans certains cas un rééchantillonnage. Un exemple de texture 3D est illustré dans la figure 1. Comme la texture 3D réside dans une mémoire dédiée à la carte graphique, la rapidité de traitement et de rendu en est grandement accéléré. Des fonctions internes, notamment d'interpolation trilinéaire dans l'espace des voxels, sont également disponibles.

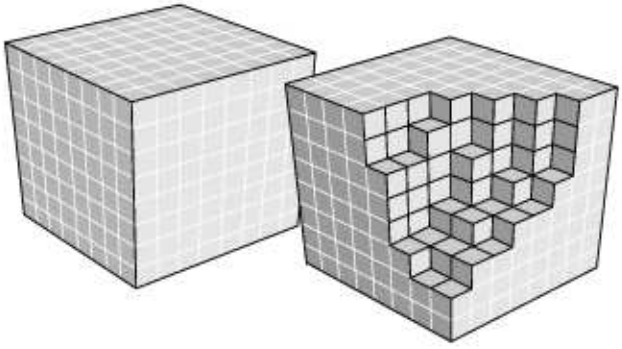


FIG. 1 – Représentation des données scalaires après discrétisation sous la forme d'une texture 3D, les unités cubiques étant les voxels d'après [EE02].

Afin d'interpréter les données numériques sous une forme visuelle, il est nécessaire avant tout de les **classifier** en leur affectant des couleurs.

2.3 Classification

Les informations de couleur étant généralement définies en Informatique Graphique comme un triplet de composantes primaires (rouge, vert et bleu), l'étape qui nous permet de passer de l'espace des scalaires ($\in \mathbb{R}$) à l'espace des couleurs et de transparence ($\in \mathbb{R}^4$) en rendu volumique se dénomme « classification ».

Ceci revient à définir une fonction allant de $\mathbb{R}$ dans $\mathbb{R}^4$ appelée **fonction de transfert**, cette dernière étant quasiment toujours transcrite sous la forme d'une texture 1D (tableau unidimensionnel de valeurs discrètes) où pour un scalaire s :

- $c(s)$ représente la couleur, soit un triplet (R, G, B),
- $\tau(s)$ la transparence (aussi appelé coefficient d'extinction).

La figure 2 montre un exemple de fonction de transfert. L'axe des abscisses dénote les valeurs numériques, l'axe des ordonnées l'intensité des composantes rouge (C), verte (B) et bleue (A). Toujours relativement à l'axe des ordonnées, le niveau de transparence est exprimé en niveau de gris dans le fond.

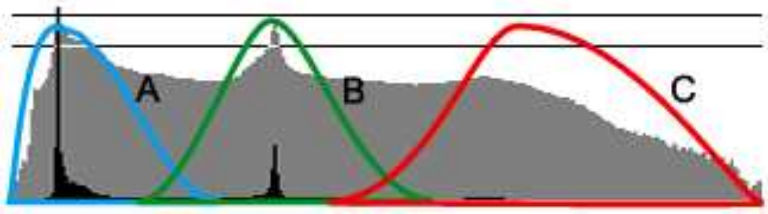


FIG. 2 – Exemple de fonction de transfert unidimensionnelle univariée d'après [KKH02].

A partir d'une fonction de transfert, il est possible de traiter les échantillons voxels de deux façons :
- par **pré-classification** : la fonction de transfert est appliquée pour chaque échantillon avant l'interpolation trilinéaire. On dit alors que l'on travaille dans l'espace des couleurs, les couleurs étant interpolées. Les hautes fréquences de la fonction de transfert (zones à forte variation d'intensité de couleur et de transparence pour deux scalaires rapprochés) peuvent ne pas être reproduites en conséquence, ce qui se traduit par l'introduction de défauts visuels, comme on peut le voir sur la figure 3, image (a).
- par **post-classification** : la fonction de transfert est appliquée après interpolation, donc au niveau du champs de données. La visualisation en découlant permet de rendre tous les détails de la fonction de transfert comme le montre l'image (b) de la figure 3. Cependant cette approche est moins performante que la précédente de par l'application de la fonction de transfert à toutes valeurs issues de l'interpolation linéaire inter-échantillons.

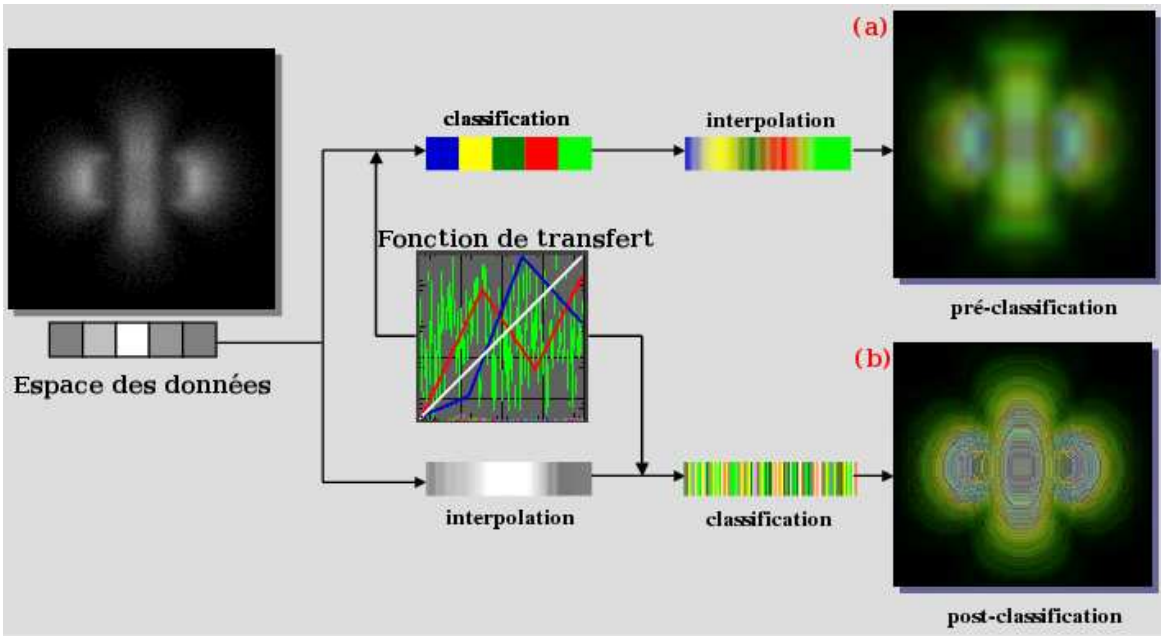


FIG. 3 – Rendu d'un noyau d'hydrogène (texture 3D de 16^3 voxels) en pré-classification et post-classification d'après [EKE01].

Notons que la création automatique d'une fonction de transfert demeure encore aujourd'hui une problématique de recherche active. Il n'existe encore aucun algorithme optimal de construction assignant couleurs et opacités pour un ensemble de données. Il existe cependant des méthodes de génération dites semi-automatique où une analyse topologique [FTAT00, WSH03, WS04] et une analyse des données [HHKP96, KD98] guide l'utilisateur dans la conception de la fonction de transfert. Il demeure cependant très courant que l'étape de

classification se fasse de façon « manuelle », la modification dynamique des attributs et l'observation en temps réel du volume rendu permettant une convergence (souvent lente) vers une fonction de transfert satisfaisante.

L'objet d'une bonne fonction de transfert consiste évidemment à faire ressortir des données, à un instant donné de l'exploration de l'information, ce qui est pertinent tout en cachant les détails inutiles. Ce processus de classification est donc d'une importance tout à fait capitale en terme de visualisation. Tout champs scalaire volumique peut être visualisé d'une quasi infinité de façon de par la liberté que l'on a à affecter couleur et transparence à chacune des valeurs scalaires.

2.4 Principe du rendu volumique

2.4.1 Du continu au discret

Soit $\overrightarrow{r}(t)$ un rayon lancé dans le volume (zone de l'espace délimitant le champs scalaire), ce dernier étant paramétré par la distance à l'observateur t (voir figure 4). Le scalaire correspondant à la position courante dans l'espace $\overrightarrow{r}(t)$ est donné par $s(\overrightarrow{r}(t))$. Etant donné que l'on se place dans un modèle optique d'absorption, l'équation propre au Rendu Volumique va intégrer l'opacité $\tau(s(\overrightarrow{r}(t)))$ ainsi que la couleur $c(s(\overrightarrow{r}(t)))$ le long du rayon, nous donnant la couleur finale de notre pixel C selon l'équation (1) :

$$C = \int_0^D c(s(\overrightarrow{r}(t)))e^{-\int_0^{t'} \tau(s(\overrightarrow{r}(t')))dt'} dt. \tag{1}$$

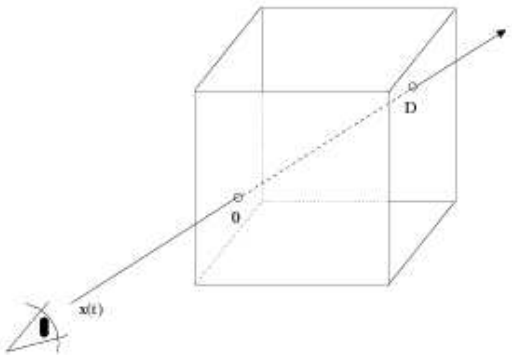


FIG. 4 – Intégration du volume le long d'un rayon.

Bien sûr, il est possible d'imaginer d'autres formes d'intégration de couleur et de transparence le long du rayon. Cette forme courante correspond à la théorie physique de transport de lumière en milieu absorbant, mais non diffusant.

Pour une position de l'observateur dans l'espace et un vecteur de vue, la résolution de l'intégrale pour chacun des rayons traversant le volume de O (point d'entrée) à D (point de sortie) nous permet d'obtenir l'image finale. En subdivisant le rayon allant de 0 à D en une succession finie d'échantillons rapprochés (ici, on considère que l'on a $n+1$ échantillons) et en substituant l'intégrale continue par une somme de Riemann, on obtient alors pour l'échantillon d du $i^{ème}$ rayon lancé :

$$C \approx \sum_{i=0}^{n} C_i \alpha_i e^{-\sum_{i'=0}^{i-1} \alpha(i')},$$

avec :

– $C_i = c(s(\overrightarrow{r}(id)))$ la couleur, soit l'émission.

– $\alpha_i = \tau(s(\overrightarrow{r}(id)))$ la transparence, soit le coefficient d'absorption.

De par la propriétés des exponentielles ($e^{x+y} = e^x e^y$), on obtient :

$$C \approx \sum_{i=0}^{n} C_i \alpha_i \prod_{i'=0}^{i-1} e^{-\alpha(i')}.$$

En considérant le développement limité de la fonction exponentielle e^x au voisinage de 0, soit $1 + x + \epsilon(x)$ et en négligeant le terme $\epsilon(x)$, on obtient finalement :

$$C \approx \sum_{i=0}^{n} C_i \alpha_i \prod_{i'=0}^{i-1} (1 - \alpha_{i'}). \tag{2}$$

2.4.2 Méthodes de rendu volumique

Lancer de rayon

Une des premières méthodes de rendu volumique fut celle du lancé de rayon pure (« Raycasting »). Pour chacun des pixels de l'image, un rayon est lancé dans l'espace en respectant la nature de la projection utilisée. Le segment d'intersection avec le volume est alors discrétisé en une suite d'échantillons pour lesquels la somme est effectuée, attribuant ainsi la couleur finale du pixel d'écran (voir figure 5).

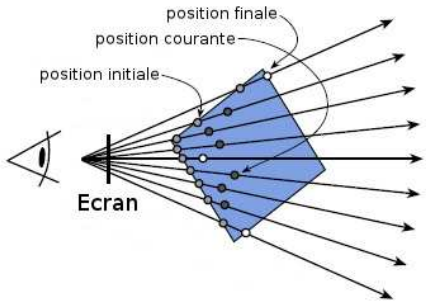


FIG. 5 – Schéma représentant un lancer de rayon classique.

Cette approche coûteuse et jusqu'à récemment non totalement transposable sur GPU a été peu développée dans le domaine du Rendu Volumique. Il existe cependant quelques approches [KW03, RGW+03] récentes simulant un lancé de rayon en exploitant certaines fonctionnalités de la carte graphique (notamment, le rendu hors écran). C'est ce type d'approche que nous développons dans ce rapport.

Méthodes basées texture 3D

Le principe de cette approche est simple. On rend dynamiquement des polygones calculés à la volée, ces derniers étant ordonnancés de l'arrière vers l'avant afin de respecter la transparence (voir figure 6). A ces polygones sont affectées des coordonnees de textures 3D permettant d'associer les valeurs du champs à visualiser.

La qualité et intéractivité sont régies par le nombre de polygones de support utilisé. Cependant, de par la nature d'une projection en perspective, l'éloignement inter-échantillon d'un même rayon augmente plus l'on s'éloigne du vecteur directeur de vision. Soit, plus un pixel est éloigné du centre de la fenêtre écran, plus la distance entre les échantillons du rayon qui lui est relatif est grande (voir figure 7).

Pour pallier à ce problème, il existe une méthode où le volume est décomposé selon une logique sphérique [GHY98, LHJ99] (voir figure 8). Ainsi, la distance entre les données prise en compte est maintenue constante indépendamment du rayon traité.

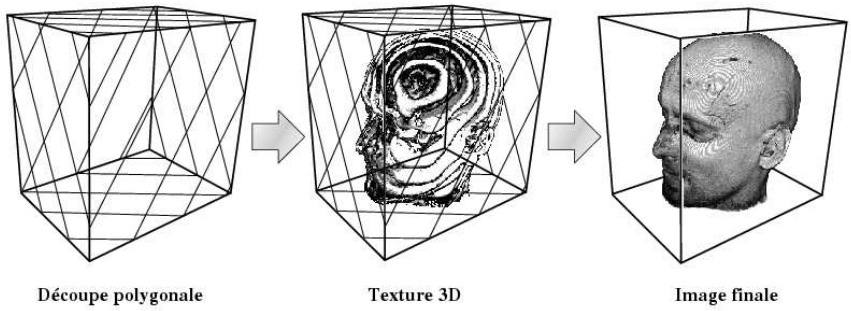


FIG. 6 – Exemple de rendu par découpe dirigée par le vecteur de vue en utilisant une texture 3D d'après [EE02].

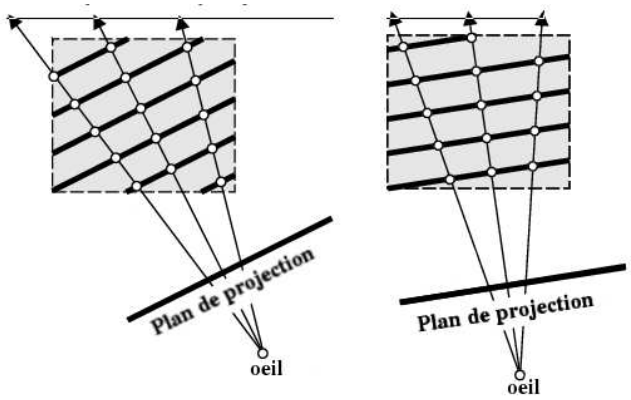


FIG. 7 – L'éloignement inter-échantillon augmente avec les rayons limitrophes lors d'une projection en perspective comme transformation d'après [EE02].

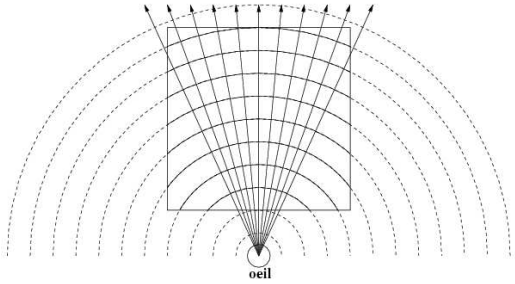


FIG. 8 – Dans le cas d'un lancé de rayon suivant une logique sphérique, la distance est maintenue constante pour tous les rayons d'après [EE02].

Projection de cellules
Une autre méthode consiste en une décomposition d'un volume de données en un maillage de l'espace. A partir des scalaires et de leur répartitions spatiales, le volume est exprimé en une union de cellules (tétraèdres, hexaèdres...), chacun des sommets ayant une valeur numérique qui lui est affectée (voir figure 9).

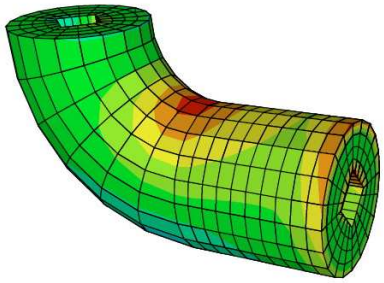


FIG. 9 – Décomposition d'un volume en un ensemble d'unités volumiques.

Lors du rendu, les cellules sont triées selon la profondeur et affichées en fonction de la transparence. Lorsqu'une approche basée texture 3D n'est pas possible (répartition spatiale des échantillons trop parsemée, insuffisance mémoire...), il est commun d'avoir recours à ce type d'approche [SM97, FMS00].

2.5 Illumination et ombrage

Comme dans toute application graphique (notamment tridimensionnelle), l'ajout d'un éclairage local par l'usage d'une fonction de réflectance bidirectionnelle (Bidirectional Reflectance Distribution Function en anglais, BRDF) décrivant l'interaction locale de la lumière relativement à une surface, améliore grandement la sensation de forme et de profondeur.

Dans le cas des maillages surfaciques, les intensités et réflexions de la lumière perçues sont calculées grâce à l'usage des vecteurs normaux. Dans le cadre d'un rendu volumique, il est possible d'appliquer le même principe en définissant localement une surface en tout point de l'espace, par le biais d'une normale définie selon le vecteur gradient. Ce vecteur gradient est déterminé en explorant le voisinage des points dans le champs de voxels (voir figure 10).

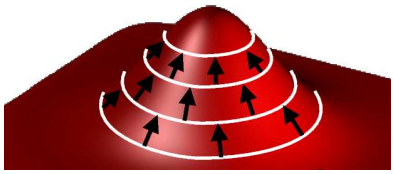


FIG. 10 – Exemple d'isorsuface en rouge avec les gradients (flèches noires) montrant l'évolution du champs scalaire.

Pour un point (x, y, z) de la discrétisation du champs scalaire f, le gradient se définit par :

$$\nabla f(x, y, z) = (\frac{\partial f}{x}, \frac{\partial f}{y}, \frac{\partial f}{z}).$$

Etant donnée la nature discrète du champs, il existe plusieurs méthodes de calcul approchant le gradient de précision proportionnelle à leur complexité. Parmi les approches les plus courantes, celle de la différence centrale

tient compte du 6-voisinage. De complexité faible et offrant de bons résultats visuels, son utilisation est courante en rendu volumique :

$$\nabla f(x_i, y_j, z_k) \approx \begin{cases} \frac{1}{2}(f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)) \\ \frac{1}{2}(f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)) \\ \frac{1}{2}(f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})) \end{cases}$$

Précalculer le gradient est chose courante. Pour une texture 3D donnée, on stocke alors les 3 composantes normalisées du gradient dans les composantes R, G et B de la texture, la composante A étant destinée à la valeur du voxel en question. Si l'on décide de ne stocker que les champs scalaires, calculer le gradient à la volée est devenu chose possible grâce aux capacités en terme de programmation des cartes graphiques. Pour un voxel donné, les voisins sont alors déterminés dynamiquement.

En utilisant le gradient, une fonction de réflectance surfacique peut être appliquée [GK96, MHS99, RSEB+00, MG02, LWM04] comme par exemple :

$$L_r(x, w_r) = L_e(x, w_r) + \int_\Omega f_r(w_i, w_r) L_i(x, w_i) \cos(\theta) dw_i,$$

avec

- $L_r(x, w_r)$ la luminance quittant le point x dans la direction w_r,
- $L_e(x, w_r)$ la luminance propre émise en un point x dans la direction w_r,
- $L_i(x, w_i)$ la luminance incidente au point x depuis la direction w_i,
- Ω l'ensemble des directions w_i dans l'hémisphère couvrant la surface au point x,
- $f_r(w_i, w_r)$ la fonction BRDF qui décrit les propriétés de réflexion de la surface au point x.

La figure 11 illustre un exemple où un crâne humain est rendu sans (à gauche) et avec (à droite) une BRDF de type Phong [Pho75], le calcul du gradient étant basé sur une différence centrale.

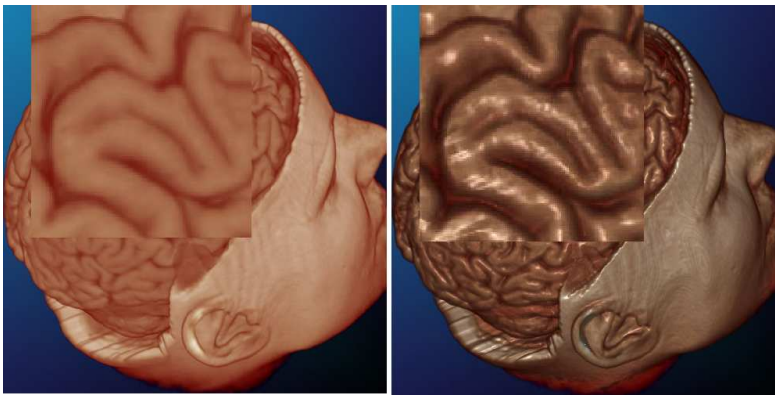


FIG. 11 – Rendu volumique d'un crane humain sans (à gauche) et avec (à droite) éclairage de Phong d'après [EHK+03]. Le type de gradient utilisé est celui de la différence centrale.

Afin de compléter les effets induits par la lumière ainsi que son atténuation dans le volume, des modèles d'ombrage ont également été proposés [BR98, KKH02, KPHE02, KPH+03]. Ceux-ci permettent des rendus de haute qualité visuelle :

1. En stockant un second volume (texture 3D) représentant l'intensité lumineuse accumulée par transport dans le volume initial, on détermine pour chaque voxel du champs de données leur contribution relative à la position de la source lumineuse [BR98]. Ainsi, après application de la fonction de transfert, on multiplie le quadruplet (R, G, B, A) par la valeur d'ombre correspondante. Cependant, de par l'interpolation trilinéaire effectuée, les ombres apparaissent de façon floue avec un rendu final souvent trop obscur, ce phénomène étant connu en tant que « fuite d'atténuation ».

2. Une autre méthode prend en considération la position de l'observateur et de la source de lumière lors du processus de segmentation du volume [KKH02]. On choisit cette fois le plan de découpe comme étant à mi-chemin entre le plan orthogonal au vecteur de vue et le plan orthogonal au vecteur directionnel de la lumière. Bien plus efficace que l'approche précédente, cette méthode a le mérite d'être plus performante, une texture 2D remplaçant avantageusement la texture 3D précédente ; de plus la qualité du rendu est grandement amélioré.

La figure 12 illustre un exemple d'ombrage.

FIG. 12 – Une numérisation de carpe visualisée avec lissage de Phong (à gauche), l'ombrage ayant été rajouté pour les deux autres (seules la localisation et la direction de la source lumineuse changent) d'après [KPHE02].

2.6 Pré-intégration

Bien qu'une post-classification (section 2.3) permette de capturer les hautes fréquences d'une fonction de transfert, seules les variations sur une même coupe sont prises en compte, négligeant ainsi l'échantillonnage le long du rayon. Lorsque l'échantillonnage le long du rayon est trop faible, ceci se traduit par des défauts visuels importants. Afin de remédier à ce problème, la notion de pré-intégration a été développée [EKE01, MG02, LWM04]. On pré-calcule entre tout couple de scalaires, les couleurs et les opacités issues de l'équation (1) en supposant une variation linéaire de la fonction de champs et en se fixant une distance d entre couples. Le pré-calcul aboutit à un tableau à deux dimensions appelé table de pré-intégration.

Connaissant les valeurs de deux scalaires, dits avant $s_f = x(id)$ et arrière $s_b = x(id+1)$, pour deux echantillons id et $id+1$ consecutifs sur le rayon (voir figure 13), la transparence est approchée par :

$$\begin{aligned} \alpha_i &= 1 - e^{-\int_{id}^{(i+1)d} \tau(s(x(\lambda)))\mathrm{d}\lambda}, \\ &\approx 1 - e^{-\int_0^1 \tau((1-\omega)s_f + \omega s_b)d \ \mathrm{d}\omega}. \end{aligned}$$

De même, pour les couleurs, on obtient :

$$\tilde{C}_i \approx \int_0^1 \tilde{c}((1-\omega)s_f + \omega s_b) e^{-\int_0^1 \tau((1-\omega')s_f + \omega' s_b)d \ \mathrm{d}\omega' d \ \mathrm{d}\omega}.$$

L'intégrale du rendu volumique est alors approchée en utilisant une classification pré-intégrée. Cette intégrale peut alors s'écrire selon :

$$I \approx \sum_{i=0}^{n} \tilde{C}_i \alpha_i \prod_{j=0}^{i-1} (1 - \alpha_j).$$

On ne considère donc non plus de simples coupes mais des tranches, ces dernières étant définies par les deux polygones de support l'encadrant (voir figure 14). On garantit ainsi pour un champ scalaire continu, la capture des hautes fréquences dans la fonction de transfert ; en conséquence moins de coupes le long du volume peuvent être utilisées.

Avec une telle table, les rendus sont grandement améliorés (voir figures 15 et 16). Notons que si l'espace des scalaires est défini sur plus d'un octet, il devient difficile de stocker dans la carte graphique la table de pré-intégration en raison de contraintes de taille (dans le cas de 2 octets, on a 65536 valeurs numériques possibles, une fonction de transfert nécessite alors 0.25 Mo et sa table de pré-intégration 16 Go).

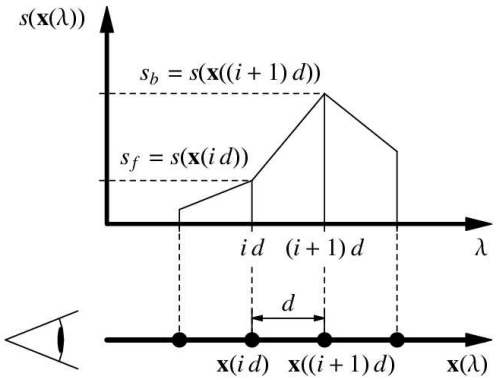


FIG. 13 – Schéma représentant les paramètres déterminant la couleur et la transparence du i^{eme} segment du rayon d'après [EKE01].

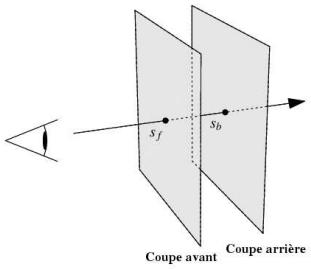


FIG. 14 – Une tranche du volume délimitée par deux coupes d'après [EKE01].

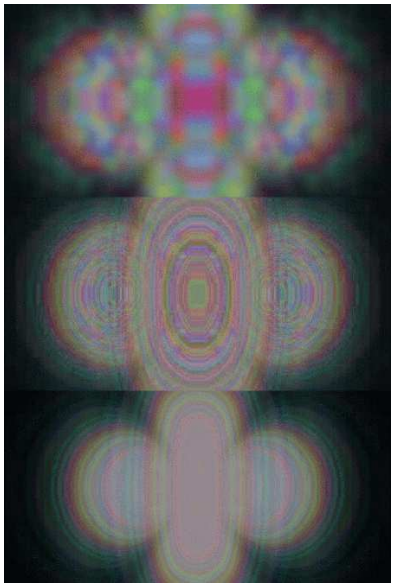


FIG. 15 – 3 rendus différents : en haut, pré-classification, au milieu post-classification et en bas avec pré-intégration d'après [EKE01].

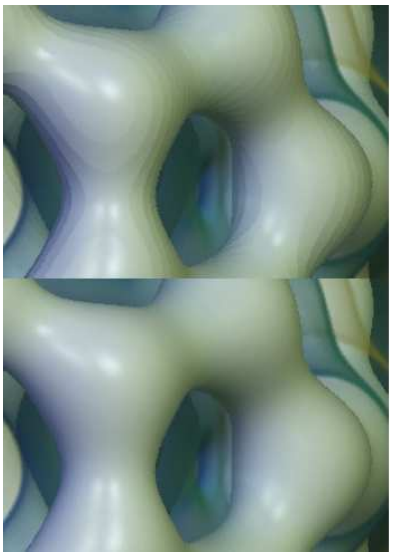


FIG. 16 – Visualisation d'une molécule sans pré-intégration (en haut) et avec pré-intégration (en bas) d'après [LWM04]. Les artefacts liés à la découpe ne sont plus visibles dans le deuxième cas.

3 Problématique des plasmas

Nous proposons ici un aperçu de la physique des plasmas avec une description d'une méthode existante de visualisation dédiée au rendu de simulations gyrocinétiques.

3.1 Intérêt scientifique

La construction du dispositif physique nécessaires à la création et au contrôle d'un plasma est extrêmement coûteuse (voir figure 17). En conséquent, des modèles théoriques ont été développés pour accompagner ces efforts expérimentaux .

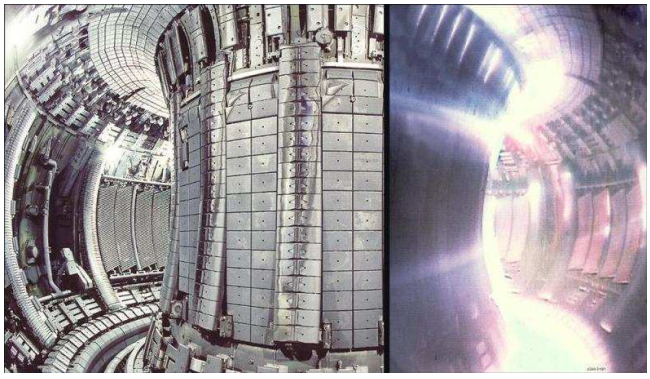


FIG. 17 – L'intérieur d'un tokamak.

Par ailleurs, une partie importante de l'étude théorique repose sur des simulations numériques. Notons que ces simulations permettent à la fois de réduire les coûts et les risques technologiques, mais permettent également d'améliorer l'efficacité et la performance des réacteurs.

3.2 Modélisation physique d'un plasma

Les plasmas sont obtenus en chauffant à très haute température un gaz conduisant à l'obtention d'un nouveau gaz composé de particules chargées et de particules neutres. C'est pourquoi le plasma peut être considéré comme étant le $4^{ième}$ état de la matière. Nous venons de voir que les conditions nécessaires à ce changement d'état sont obtenues à l'intérieur d'un réacteur appelé tokamak.

3.2.1 Description cinétique

Le modèle généralement utilisé pour étudier le comportement des particules du plasma se base sur une description cinétique du plasma en utilisant l'**équation de Vlasov**. Cette dernière est couplée aux équations de Maxwell ou de Poisson permettant de décrire l'évolution des champs électriques et magnétiques. L'équation de Vlasov caractérise l'évolution dans le temps et l'espace de la distribution des particules d'un plasma non collisionnel.

La résolution de cette équation cinétique permet ensuite de calculer, par intégration dans l'espace des vitesses, les densités de charge. En principe, cette équation devrait être résolue en 6 dimensions (3 dimensions dans l'espace physique et 3 dimensions dans l'espace des vitesses). Cependant, les machines de calcul ne permettent pas, à l'heure actuelle, de résoudre cette équation en 6D dans des temps acceptables. Une simplification permet d'aboutir à une équation **gyrocinétique**. Notons que la résolution de cette dernière reste encore lourde en terme de calculs numériques, mais permet néanmoins de réaliser des simulations numériques de plasmas.

Deux types de méthodes ont été utilisés jusqu'à aujourd'hui pour étudier les turbulences dans les régimes gyrocinétiques :

- Les premières sont basées sur une approche Lagrangienne. Les méthodes Particles-in-Cell (PIC) consiste par exemple à décrire le plasma par un nombre fini de macro-particules. Les trajectoires de ces particules sont alors les caractéristiques de l'équation de Vlasov, tandis que les champs consistants sont calculés en rassemblant les charges et densités courantes des particules sur un maillage de l'espace physique. Notons que des travaux récents ont été proposés pour visualiser l'évolution de ces particules au cours du temps [MSW$^+$02, CFG$^+$04].
- Les secondes se basent sur une approche Eulérienne. Elles s'appuient sur une discrétisation de l'équation de Vlasov sur un maillage de l'espace des phases qui demeure fixe dans le temps. Par exemple la Méthode de Balance des Flux (FBM, Flux Balance Method) utilise une méthode à volumes finis permettant de calculer la moyenne de l'équation de Vlasov sur chacune des cellules de la grille statique.

3.2.2 Méthode Semi-Lagrangienne

La simulation dont sont issues nos données est basée sur une méthode intermédiaire appelée Semi-Lagrangienne (SL) [FSB01]. Le but de cette méthode est de tirer partie des deux approches, Lagrangienne et Eulérienne, afin d'obtenir à la fois une bonne description de l'espace des phases (même pour les régions de faibles densités) et une meilleure stabilité numérique. Dans cette approche [GBB$^+$], le maillage est fixe au cours du temps dans l'espace des phases (méthode Eulérienne), et l'équation de Vlasov est intégrée le long des trajectoires (méthode Lagrangienne) en utilisant l'invariance de la fonction de distribution le long des trajectoires (appelées caractéristiques). Des interpolations selon un schéma de splines cubiques permettent alors d'évaluer les nouvelles valeurs de la fonction de distribution aux points de contrôle de la grille.

Dans le cadre de cette méthode numérique, la fonction de distribution f complète est calculée, à la différence des codes PIC qui ne calculent que la fonction de distribution perturbée. La simulation que nous utilisons est de plus appliquée à une géométrie cylindrique avec une réduction de l'espace des phases à la dimension 4. Même en se restreignant à 4 dimensions, la quantité de données demeure très importante puisqu'une grille régulière de résolution 64 demande déjà 128Mo de mémoire en précision double pour chacun des pas de temps de la simulation.

3.2.3 Equation de Vlasov

Dans la version 4D, un plasma cylindrique et périodique de rayon a et de périmètre $2\pi R$ est considéré comme étant un cas limite d'un tore étiré. Le plasma est confiné par un fort champ magnétique uniforme $\vec{B} = B\vec{e_z}$ où $\vec{e_z}$ représente le vecteur unitaire dans la direction toroïdale z. Concernant les ions, les effets de bords du rayon de Larmor sont négligés de manière à ce que les trajectoires puissent être gouvernées par les trajectoires du « centre-guide »(guiding-center, GC). Avant de présenter en détail l'équation de Vlasov, nous introduisons quelques notations :

$$\frac{dr}{dt} = v_{GC_r} \ ; \ r\frac{d\theta}{dt} = v_{GC_\theta} \ ; \ \frac{dz}{dt} = v_\parallel \ ; \ v_\parallel^\cdot = \frac{q}{m_i}\vec{E}_z, \tag{3}$$

où v_{GC_r} et v_{GC_θ} sont les composantes radiales et poloïdales de la vélocité dérivante $E\times B$ avec $\vec{v}_{GC} = \frac{\vec{E}\times\vec{B}}{B^2}.\vec{E}$, où $\vec{E}$ est le champs électrique, $q = Ze$ la charge ionique, m_i la masse ionique et $v_\parallel$ la vélocité le long des lignes du champ magnétique. Notons que cette configuration cylindrique simplifiée ne prend pas en compte les effets toroïdals, mais permet l'étude des modes des gradients de la température ionique.

A partir de ces hypothèses, la fonction de distribution f est une fonction de dimension 4 dans l'espace des phases, dépendant des 3 coordonnées cylindriques (r, θ, z) et de la vitesse parallèle $v_\parallel$. L'évolution de cette fonction de distribution $f(r, \theta, z, v_\parallel, t)$ est alors décrite par l'équation cinétique de Vlasov :

$$\frac{\partial\vec{f}}{\partial t} + \vec{v}_{GC}\,.\vec{\bigtriangledown}_\perp f + v_\parallel\frac{\partial f}{\partial z} + v_\parallel^\cdot\frac{\partial f}{\partial v_\parallel} = 0, \tag{4}$$

où $\vec{\bigtriangledown}_\perp = (\frac{\partial.}{\partial r}, \frac{1}{r}\frac{\partial.}{\partial\theta})$. Cette équation couple le mouvement $\vec{E}\times\vec{B}$ au travers du champ magnétique du mouvement parallèle au champ magnétique.

3.3 Visualisation d'un plasma

Le but de notre travail consiste à fournir un outil de visualisation interactive des données issues de la simulation gyrocinétique *e. g.* un outil destiné à visualiser, pour un pas de temps donné et une vitesse parallèle fixée, la fonction de distribution $f_{v_\parallel}(r, \theta, z)$ discrétisée sur un cylindre de résolution $64 \times 128 \times 64$.

3.3.1 Méthode de Crawford *et al.*

Les approches classiques de visualisation volumique se sont révélées assez inefficaces lors de leur application à la visualisation des résultats issus d'une simulation gyrocinétique. En effet, de par les propriétés géométriques du tokamak qui sont celles d'un tore, les échantillons découlant de la simulation numérique sont en conséquence répartis spatialement selon une structure toroïdale. Dans le cadre d'une approche basée texture 3D, cette contrainte a un fort impact en terme d'implantation matérielle et de gestion des ressources. Or les cartes graphiques actuelles ne pouvant disposer de plus de 256Mo de mémoire vive, la disposition spatiale des échantillons en question impliquent une perte mémoire non négligeable lors de leur transposition sous forme de texture 3D. Dans la figure 18, on visualise une répartition de scalaires (en bleu) issus d'une simulation gyrocinétique stockée dans une texture 3D, la perte mémoire étant représentée en rouge. On se retrouve donc confronté face à un problème de taille mais aussi de précision pour les données destinées à être visualisée (en raison du ré-échantillonnage).

Afin de pallier à ce problème, Crawford *et al.* [CMH+04] utilisent un déroulement des données (voir figure 19) selon l'angle majeur qui s'identifie à un changement de répère (en l'occurence, passage d'un repère cartésien à un repère mixte).

Ce déroulement de texture présente l'intérêt imminent d'une occupation mémoire moindre comparé à la boite englobante originelle du tokamak, mais nécessite cependant l'application d'une formule de changement de repère par échantillon afin de retrouver les coordonnées de texture correcte selon la localisation de ce dernier.

Afin de formaliser le changement de repère en question, considérons l'équation paramétrique d'un tore :

$$\begin{cases} x(\alpha,\beta) = (R + r.cos(\theta)).cos(\phi), \\ y(\alpha,\beta) = r.sin(\theta), \\ z(\alpha,\beta) = (R + r.cos(\theta)).sin(\phi). \end{cases} \tag{5}$$

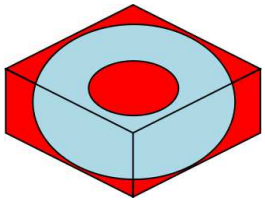


FIG. 18 – Un tore délimité par sa boîte englobante représentant la perte mémoire dans le cas d'une simulation gyrocinétique.

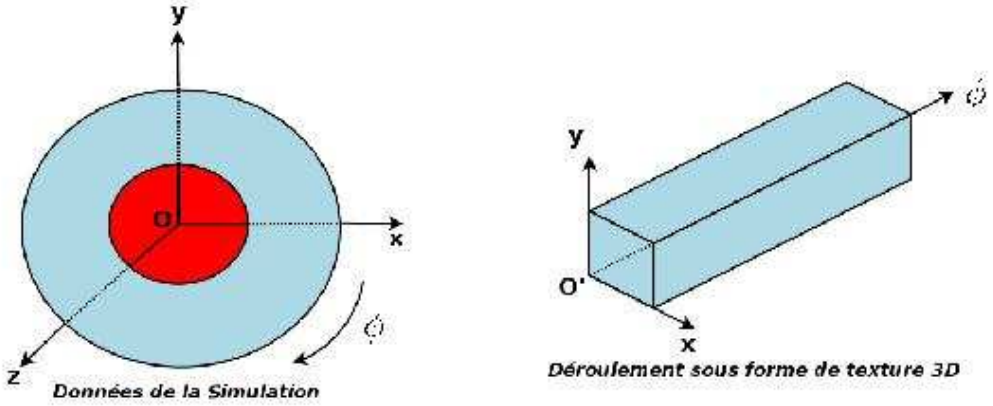


FIG. 19 – Déroulement selon l'angle majeur des données issues de la simulation, ces dernières étant stockée dans une texture 3D standard.

où R est le rayon majeur, r le rayon mineur, ϕ l'angle majeur et θ l'angle mineur.

Le tokamak étant centré en l'origine de notre repère O, pour tout plan poloidal (coupe selon le plan Oyz) les échantillons sont répartis selon des rayons mineurs r tel que $r \in]0, r_{max}]$ avec $\forall r$, $\theta \in [0, 2\pi[$, en respect avec l'équation paramétrique du tore (5) (voir figure 20).

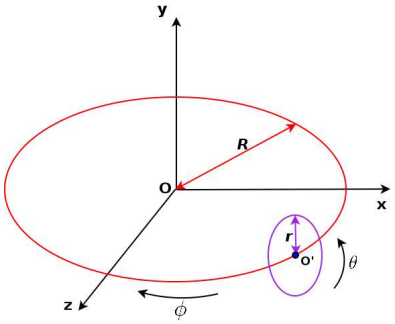


FIG. 20 – Représentation visuelle de l'équation paramétrique d'un tore dans l'espace.

Le passage des coordonnées monde (x, y, z) d'un échantillon à ses coordonnées de texture normalisées (s, t, u) est alors régit par la formule suivante :

$$\begin{cases} s = \frac{\sqrt{x^2+z^2}-(R-r_{max})}{2.r_{max}} \\ t = \frac{y}{r_{max}} + 0.5 \\ u = \frac{atan(z,x)}{2\pi}. \end{cases} \quad (6)$$

En effectuant une découpe de la boite englobante du tokamak dans l'espace monde et en rendant hiérarchiquement selon la profondeur les coupes obtenues, une simulation gyrocinétique est en mesure d'être visualisée (voir figure 21) grâce à un programme graphique (dit *shader*) effectuant le changement de repère pour chaque fragment des polygones de support. Les coordonnées de texture dans la texture 3D déroulée sont alors déterminées correctement, permettant ainsi d'utiliser les voxels adéquats lors du rendu.

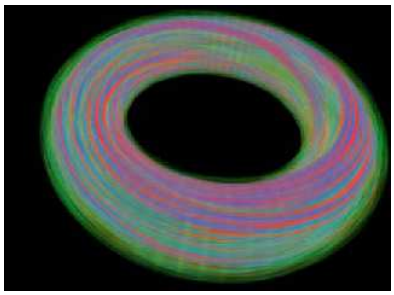


FIG. 21 – Rendu volumique d'une simulation gyrocinétique (image issue de l'article [CMH+04]).

La lourdeur de cette dernière en terme d'arithmétique afflige une sérieuse perte en terme de performance, notamment par le fait que la décharge des fragments ne contribuant pas au rendu final ne pourra s'effectuer qu'après changement de repère. Il convient également d'effectuer un test d'appartenance au tokamak :

$$\begin{cases} s \in [0,1], \text{ soit } \sqrt{x^2+z^2} \in [R-r_{max}, R+r_{max}] \\ t \in [0,1], \text{ soit } y \in [-r_{max}, r_{max}] \\ u \in [0,1], \text{ soit } atan(z,x) \in [0,2\pi] \end{cases} \tag{7}$$

La figure 22 montrent deux exemples de polygones de support lors d'un rendu volumique de plasma. Les pixels en bleu sont ceux contribuant à l'image finale et inversement, ceux en rose sont inutiles, la formule de changement de repère étant malgré tout appliquée.

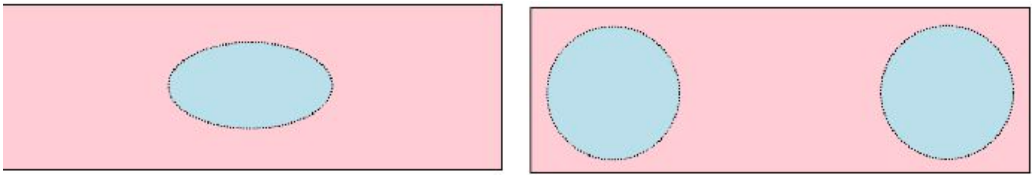


FIG. 22 – Exemple de deux polygones de support lors d'un rendu volumique de plasma montrant la contribution des fragments à l'image finale.

Afin d'améliorer les performances en terme d'images seconde mais aussi en terme de qualité de rendu, nous proposons une nouvelle implantation. C'est cette approche que nous allons décrire dans ce qui suit.

4 Adaptive Raymarching

4.1 Données de la simulation

4.1.1 Description

Les données issues des simulations numériques du CEA sont échantillonnées selon l'équation paramétrique d'un tore (voir l'équation (5) et la figure 20) : tout scalaire est alors défini à un pas de temps donné par ses coordonnées (r, θ, ϕ) ainsi que sa valeur numérique s.

La discrétisation des résultats à partir des équations physiques décompose le tokamak en un nombre fini de plans poloïdaux (nombre de coupes selon l'angle ϕ), la progression angulaire entre deux coupes étant maintenue constante (voir figure 23).

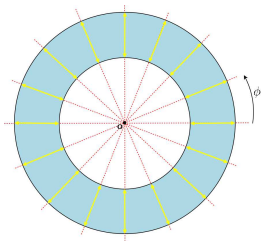


FIG. 23 – Projection sur le plan (O, x, z) du tore. La progression angulaire des plans poloïdaux (en jaune) est maintenue constante selon l'angle ϕ.

Pour chacun de ces plans poloïdaux, des échantillons sont distribués sur des anneaux de rayon r linéairement décroissants avec $r \in]0, r_{max}]$. La répartition des scalaires sur chacun des anneaux évolue linéairement selon l'angle θ et cela indépendamment du rayon de l'anneau traité (on a autant d'échantillons sur le périmètre du cercle de plus grand rayon que sur le cercle de plus petit rayon, comme on peut le voir sur la figure 24).

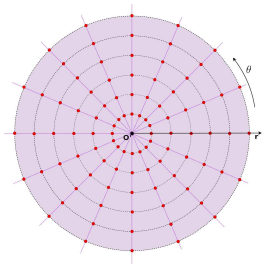


FIG. 24 – Vue exclusive d'un plan poloïdal centré en O'. L'espacement radial entre anneaux successifs est constant, la progression angulaire des échantillons pour tous les anneaux étant commune et linéaire.

4.1.2 Pré-traitement des données

Un rééchantillonnage des données issues de la simulation est nécessaire pour effectuer un rendu volumique basé sur une approche texture 3D.

Les scalaires issus de simulations gyrocinétiques de plasmas sont des réels codés en précision double (flottant sur 8 octets). Ce codage permet en effet de conserver au mieux une précision numérique, les valeurs des échantillons étant inférieures à 10^{-3}.

Cependant, dans le cadre de la conversion des champs numériques en une texture 3D standard, où chaque voxel se doit d'avoir une valeur tenant sur un octet (par soucis de gain mémoire et afin de bénéficier de pré-intégration), il nous faut :

- effectuer une mise à l'échelle de nos scalaires avant l'affectation des voxels (problème de taille mémoire),
- calculer les valeurs au centre des voxels par interpolation trilinéaire.

En parcourant au préalable tous les échantillons s calculés lors de la simulation et en déterminant la valeur minimale s_{min} ainsi que la valeur maximale s_{max} rencontrée, on détermine de nouveaux scalaires s_{voxel} qui

serviront au rendu volumique de la façon suivante :

$$s_{voxel} = \frac{s\_s_{min}}{s_{max} - s_{min}} * 255.$$

Notons que les valeurs sont converties pour tenir sur un octet, soit $s_{voxel} \in [0, 255]$.

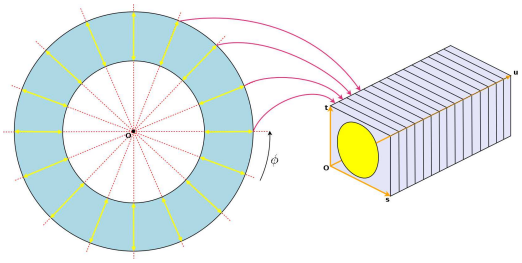


FIG. 25 – Chaque plan poloïdal de la simulation numérique est stocké hiérarchiquement en tant que tranche de la texture $3D$ selon la profondeur (axe u).

Il nous faut maintenant créer la texture 3D. Le déroulement des données consiste à faire correspondre chaque coupe à un plan poloïdal selon la profondeur de la texture (axe u) . On déroule alors la forme toroïdale en un parallélépipède adapté en terme d'alignement mémoire à la carte graphique (voir figure 25). Ainsi, dans l'idéal, on devrait avoir autant de plans de profondeur dans l'espace de texture que dans l'espace angulaire relatif à ϕ de la simulation.

Détaillons le processus de rééchantillonage, en définissant auparavant les paramètres liés à la simulation numérique :

- nb_ϕ le nombre de pas angulaires majeurs, soit le nombre de plans poloïdaux.
- nb_θ le nombre de pas angulaires mineurs pour tout anneau d'une coupe poloïdale.
- nb_r le nombre de rayons déterminant ainsi le nombre d'anneaux par coupe poloïdale.

On introduit alors les notations suivantes :

$$\begin{cases} \forall k \in [0, nb_\phi[,\ \phi(k) = \frac{k}{nb_\phi} \\ \forall j \in [0, nb_\theta[,\ \theta(j) = \frac{j}{nb_\theta} \\ \forall i \in [0, nb_r[,\ r = \frac{i}{2*nb_r} \end{cases}$$

De même, pour un voxel de coordonnées de texture normalisée (s, t, u), on définit :

$$\begin{cases} \phi(s,t,u) = u \\ \theta(s,t,u) = atan(t - 0.5, s - 0.5) * \frac{1}{2\pi} \\ r(s,t,u) = \sqrt{(s-0.5)^2 + (t-0.5)^2} \end{cases}$$

Ainsi, pour un voxel de coordonnée (s, t, u), on est en mesure de déterminer sa valeur issue d'une succession d'interpolations linéaires des données initiales.

Afin de rééchantillonner, on cherche l'indice k nous donnant les deux plans poloidaux entourants le voxel de coordonnée (s, t, u) tel que :

$$\phi(k) < \phi(s,t,u) < \phi(k+1)$$

Pour chacun de ces deux plans, on trouve l'encadrement angulaire mineur déterminant ainsi l'indice j :

$$\theta(j) < \theta(s,t,u) < \theta(j+1)$$

Finalement, on détermine les deux anneaux renfermant le voxel, soit i est affecté tel que :

$$r(i) < r(s,t,u) < r(i+1)$$

On effectue ensuite pour chacun des deux plans poloïdaux k et $k+1$ et pour chacun de leurs anneaux i et $i+1$ une interpolation selon l'angle mineur. On s'aligne ainsi sur le centre de notre voxel :

$$\begin{cases} \delta_\theta = \frac{\theta(s,t,u)-\theta(j)}{\theta(j+1)-\theta(j)} \\ s_{i,k} = s(i,j,k) + \delta_\theta \times (s(i,j+1,k) - s(i,j,k)) \\ s_{i+1,k} = s(i+1,j,k) + \delta_\theta \times (s(i+1,j+1,k) - s(i+1,j,k)) \\ s_{i,k+1} = s(i,j,k+1) + \delta_\theta \times (s(i,j+1,k+1) - s(i,j,k+1)) \\ s_{i+1,k+1} = s(i+1,j,k+1) + \delta_\theta \times (s(i+1,j+1,k+1) - s(i+1,j,k+1)) \end{cases} \tag{8}$$

A partir des deux valeurs précédemment calculées pour chacun des deux plans poloïdaux, on atteint le centre des deux voxels par interpolation linéaire radiale :

$$\begin{cases} \delta_r = \frac{r(s,t,u)-r(i)}{r(i+1)-r(i)} \\ s_k = s_{i,k} + \delta_r \times (s_{i+1,k} - s_{i,k}) \\ s_{k+1} = s_{i,k+1} + \delta_r \times (s_{i+1,k+1} - s_{i,k+1}) \end{cases} \tag{9}$$

La figure 26 représente un exemple de rééchantillonage dans le cadre bilinéaire.

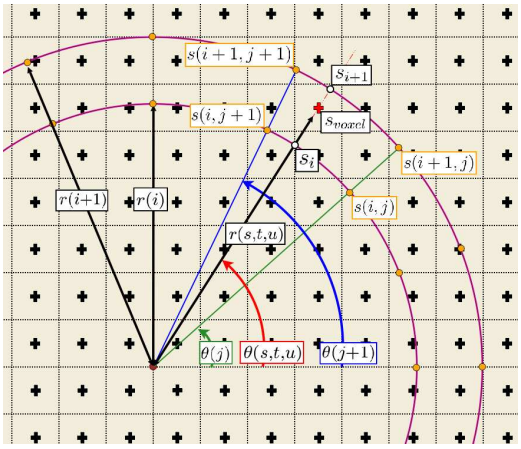


FIG. 26 – Au sein de toute tranche selon l'axe u dans l'espace des coordonnées de texture, un rééchantillonnage est effectué afin d'affecter les valeurs au centre des voxels.

Nos données ayant été converties en une texture tridimensionnelle (la figure 27 illustre un exemple sur les données du CEA), déterminons comment il nous est possible de l'utiliser pour la visualisation de la simulation gyrocinétique.

4.1.3 Changement de repère

Afin de trouver la correspondance entre un point de l'espace dont les coordonnées monde sont connues et ses coordonnées de textures dans la simulation déroulée selon l'angle majeur, on applique la formule de changement de repère (6) proposée par Crawford *et al.*.

Cependant, afin d'alléger les calculs et améliorer les performances par la suite, on propose de normaliser l'enveloppe du tokamak tel que le rayon mineur maximal r_{max} soit égal à $0,5$, une translation de $+0,5$ unités selon l'axe (O,y) s'ensuivant.

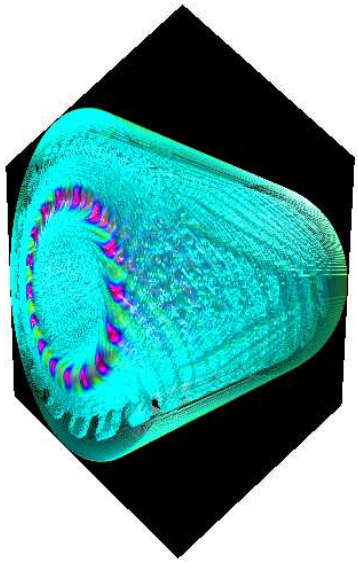


FIG. 27 – Une simulation gyrocinétique du CEA déroulé selon l'angle majeur et visualisée par rendu volumique.

Ainsi, le changement de repère devient :

$$\begin{cases} s = x^2 + z^2 - (R - r_{max}) \\ t = y \\ u = \frac{atan(z,x)}{2\pi}. \end{cases} \tag{10}$$

4.2 Algorithme de rendu

4.2.1 Considérations matérielles

Dans le cadre de la conception d'un outil de rendu volumique, il est primordial d'étudier au préalable les capacités des cartes graphiques. En effet, l'aspect matériel est aujourd'hui indissociable des méthodes de visualisation.

Le rendu hors écran

Les cartes graphiques offrent aujourd'hui la possibilité de rendre des primitives graphiques dans des zones mémoires non affichables (*i. e.* pas uniquement la zone écran). De plus, on dispose d'autant de formats numériques (entier sur 1, 2 ou 4 octets, flottants...) que sur machine standard, permettant ainsi de ne pas se limiter uniquement à des valeurs représentant des couleurs.

L'extension GL_EXT_framebuffer_object disponible depuis OpenGL 1.5 permet la définition de zones hors écran dans lesquels on peut écrire sans contrainte, notamment en utilisant des programmes graphiques pour la GPU (communément appelés *shaders*). La lecture et l'utilisation des résultats ainsi calculés se fait en déclarant la zone (en anglais *buffer*) en tant que texture et en allant ainsi chercher les éléments adéquats (les éléments de texture 2D sont appelés *texels*).

Shader Model 3.0

Les Shader Model définissent une norme qui respecte les GPU des cartes graphiques. Plus précisément, elles définissent les capacités en terme de programmation de ces dernières ainsi que leur jeu d'instructions. On distingue deux types de programmes dédiés à la GPU :

– Les vertex shaders, destinés aux sommets. Ce dernier gère les transformations géométriques des sommets et dispose de capacités d'interpolation linéaires lors de l'assemblage par primitive. De plus, il est en mesure de communiquer ses résultats aux fragments obtenus.

– Les fragments shaders, destinés aux pixels (fragments). Un fragment est quasi analogue à la notion de pixel. Les opérations affectant (parmi d'autres) la couleur, l'opacité, l'application de texture sont effectuées à ce niveau. On est aussi en mesure de rejeter un fragment dès le début de son traitement si sa contribution n'est pas utile.

A ce jour, la dernière version est le Shader Model 3.0, qui apporte avec lui un éventail d'innovations présentant un fort intérêt pour notre implantation :
– Les vertex et fragment shaders peuvent contenir jusqu'à 65535 instructions.
– Tous les shaders bénéficient de façon interne d'une précision flottante de 32 bits.
– Possibilité de branchement dynamiques à faible coût dans les vertex et fragment shaders.
– Possibilité de boucler avec condition d'arrêt dynamique dans les vertex shaders et avec condition d'arrêt statique dans les fragments shaders.
– L'interruption du traitement d'un fragment et sa non contribution au rendu sont devenues des opérations à faible coût.

4.2.2 Le maillage du Tokamak

Intérêt

Afin d'effectuer un lancer de rayon matériel (utilisation exclusive de la carte graphique), il nous faut être en mesure dans un premier temps de déterminer les coordonnées de départ et de fin du rayon. Ses points se situent sur un maillage du bord du tokamak.

Construction

En ne tenant compte que des scalaires limitrophes (*i. e.* pour chaque plan poloïdal, on ne considère que l'anneau de plus grand rayon r_{max}), la construction du maillage du tokamak s'effectue en complexité linéaire, les résultats issues des simulations numériques étant organisés hiérarchiquement (*i. e.* pour chaque plan poloïdal d'angle majeur croissant, les anneaux de rayon linéairement décroissant sont définis, et pour chaque anneau les échantillons sont distribués par progression angulaire mineure croissante).

Pour chaque scalaire de l'espace défini par sa localisation $s(r, \theta, \phi)$ dans l'équation paramétrique d'un tore (équation (5)), on détermine deux faces triangulaires en tenant compte de l'orientation de ces dernières comme le montre la figure 28 (ici, le sens trigonométrique direct a été choisi) :

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

La figure 29 présente la construction complète de l'enveloppe du tokamak. On construit ainsi une enveloppe avec $nb_\phi \times nb_\theta$ sommets et $2 * nb_\phi \times nb_\theta$ faces triangulaires (le nombre d'anneaux ici n'a pas d'importance.

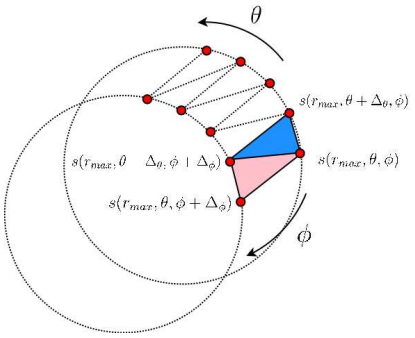


FIG. 28 – Construction du maillage.

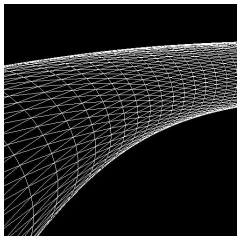


FIG. 29 – Visualisation du maillage.

Utilisation

Le maillage, précédemment calculé et stocké sous forme de liste d'affichage, va nous permettre d'effectuer les deux premières passes inhérentes à notre approche inspiré d'un raycasting.

En utilisant un vertex shader qui interpole les coordonnées monde des sommets de notre maillage (*i. e.* on s'intéresse à la localisation statique des points avant la transformation perspective) ainsi qu'un fragment shader écrivant en virgule flottante dans un buffer écran les coordonnées précédemment interpolées, on dispose de deux textures contenants les informations de début et de terminaison de chacun de nos rayons lancés.

Voyons comment utiliser ces informations pour effectuer le rendu des scalaires du tokamak.

4.2.3 Raymarching

Principe

Contrairement au raycasting (lancer de rayon) standard où l'intégration des rayons est effectuée selon un schéma de subdivision homogène (nombre d'échantillons proportionnel à la norme du rayon), l'approche proposée impose un échantillonnage constant et cela indépendant de la taille du rayon traité.

Un tel choix est justifié de par l'architecture actuelle des cartes graphiques, ces dernières ne supportant que les conditions d'arrêt statiques dans les boucles s'exécutant dans un fragment shader.

Une alternative a néanmoins été développée et évaluée. En fixant la condition d'arrêt de la boucle principale de rendu à une valeur maximale (dans notre implantation, 256 s'est révélé être un bon choix) ainsi qu'en déterminant dans le fragment shader le nombre d'échantillons pour le rayon en cours d'intégration (nombre d'échantillons par unité de distance multiplié par la norme du rayon), on est en mesure d'effectuer un raycasting. Afin de sortir de la boucle lorsque l'on a atteint notre réelle condition d'arrêt, un test est effectué à chaque itération afin de forcer la sortie au moment adéquat par le biais de l'instruction « break ».

Un tel test par échantillon se révèle très coûteux en terme de cycles GPU, ce qui a guidé le choix final vers une subdivision non homogène de nos rayons. Le taux d'échantillonnage de notre intégration a donc ainsi été fixé indépendamment de la norme des rayons.

Voici les 3 passes nécessaires au rendu final d'une simulation gyrocinétique.

Passe 1

En activant l'élimination des faces avants (*i. e.* on ne considère que les faces arrières), on rend le maillage de l'enveloppe du tokamak en écrivant dans un premier buffer hors écran les coordonnées mondes des faces interpolées par fragment, en bénéficiant de la précision des flottants sur 32 bits. De plus, en activant le z-buffer, on effectue un tri selon la profondeur. Dans le cas où l'on a plusieurs points de l'espace appartenants à des faces arrières pour un même fragment, on ne garde alors que le sommet le plus proche de l'observateur. La raison d'un tel tri sera explicité par la suite.

Lorsque l'on efface le buffer destiné aux faces arrières, on utilise un triplet de flottants permettant de déterminer si le fragment courant nécessite un raymarching (un fragment peut très bien correspondre à un rayon qui n'aura pas d'intersection avec le tokamak).

En tirant parti de la normalisation du tokamak qui impose que toute coordonnée $y \in [0, 1]$, un choix adéquat pour notre triplet d'effacement est $(x, y_{discard}, z)$ où $y_{discard} \notin [0, 1]$. Notons que dans le cadre de l'implantation du moteur de rendu, on a fixé $y_{discard} = 2$.

Ainsi, après parcours des faces arrières, tout pixel dont la seconde composante sera égale à $y_{discard}$ est ignoré dans la troisième passe.

Passe 2

Afin de rendre les faces avants dans un deuxième buffer hors écran, un quadrilatère de la taille de la fenêtre écran (dit *viewport*) et représentant le plan avant de la pyramide de projection en perspective, doit être rendu

au préalable avec le z-buffer désactivé. Ce plan doit permettre un rendu lorsque la camera se situe à l'intérieur du tokamak, la projection du maillage n'étant plus suffisante dans ce cas.

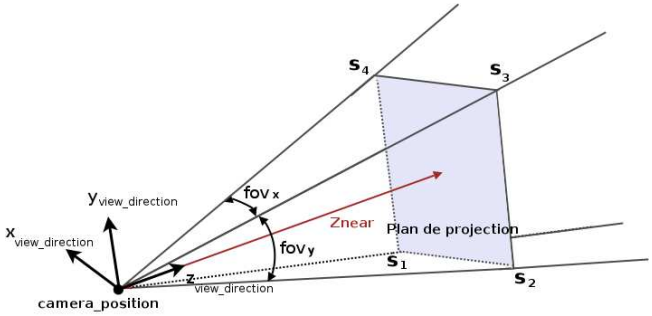


FIG. 30 – Paramètre d'une projection en perspective.

Connaissant les paramètres de la matrice de projection en perspective ainsi que la localisation de la camera dans l'espace, il est possible de déterminer le quadrilatère de la manière suivante (voir la figure 30) :

$$\begin{cases} s_1 = & camera\_position + z_{near}.tan(\frac{fov_x}{2}).x_{view\_direction} \\ & -z_{near}.tan(\frac{fov_y}{2}).y_{view\_direction} + z_{near}.z_{view\_direction} \\ s_2 = & camera\_position - z_{near}.tan(\frac{fov_x}{2}).x_{view\_direction} \\ & -z_{near}.tan(\frac{fov_y}{2}).y_{view\_direction} + z_{near}.z_{view\_direction} \\ s_3 = & camera\_position - z_{near}.tan(\frac{fov_x}{2}).x_{view\_direction} \\ & +z_{near}.tan(\frac{fov_y}{2}).y_{view\_direction} + z_{near}.z_{view\_direction} \\ s_4 = & camera\_position + z_{near}.tan(\frac{fov_x}{2}).x_{view\_direction} \\ & +z_{near}.tan(\frac{fov_y}{2}).y_{view\_direction} + z_{near}.z_{view\_direction} \end{cases}$$

où s_i avec $i \in [1,4]$ sont les sommets du quadrilatère s_1, s_2, s_3, s_4 tels que le sens trigonométrique direct soit respecté. $camera\_position$ représente la localisation de la camera dans l'espace, ($x_{view\_direction}, y_{view\_direction}$, $z_{view\_direction}$) définissant un repère orthonormé de visualisation mobile dans l'espace. Finalement, fov_x et fov_y sont les angles d'ouverture de la pyramide de vision selon l'axe des x et selon l'axe des y.

Dans le cadre de l'implantation, il a été choisi que $fov_x = 90^\circ$ et $fov_y = 90^\circ$, nous donnant la formule finale :

$$\begin{cases} s_1 = & camera\_position + z_{near}.x_{view\_direction} \\ & -z_{near}.y_{view\_direction} + z_{near}.z_{view\_direction} \\ s_2 = & camera\_position - z_{near}.x_{view\_direction} \\ & -z_{near}.y_{view\_direction} + z_{near}.z_{view\_direction} \\ s_3 = & camera\_position - z_{near}.x_{view\_direction} \\ & +z_{near}.y_{view\_direction} + z_{near}.z_{view\_direction} \\ s_4 = & camera\_position + z_{near}.x_{view\_direction} \\ & +z_{near}.y_{view\_direction} + z_{near}.z_{view\_direction} \end{cases}$$

Après projection de ce quadrilatère dans le buffer relatif aux faces avants, on rend une nouvelle fois l'enveloppe du tokamak (le maillage) en activant cette fois l'élimination des faces arrières ainsi que le z-buffer qui a gardé les indices de profondeur de la passe 1. Avec les mêmes vertex et fragments shaders que dans la passe 1, l'écriture des coordonnées monde est effectuée pour chacun des fragments appartenant à la projection du maillage.

A la fin de ces deux premières passes, on a :
– Une texture contenant les coordonnées monde des fragments relatifs aux faces arrières (ces derniers étant les plus proches de l'observateur), ou bien un fragment vide ($y_{discard}$).

– Une texture contenant les coordonnées monde des fragments relatifs aux faces avants, ces derniers étant situés devant ceux relatifs aux faces arrières.

Ainsi, on distingue quatre principaux scénarios de rendu possible (des cas hybrides peuvent apparaître comme le montre la figure 31) :

1. Utilisation des faces avants et arrières les plus proches de l'observateur (figure 32(a)).
2. Utilisation du plan de projection et des faces arrières les plus proches de l'observateur (figure 32(b)).
3. Utilisation des faces avants et arrières les plus éloignées de l'observateur (figure 32(c)).
4. Utilisation du plan de projection et des faces arriéres les plus éloignées de l'observateur (figure 32(d)).

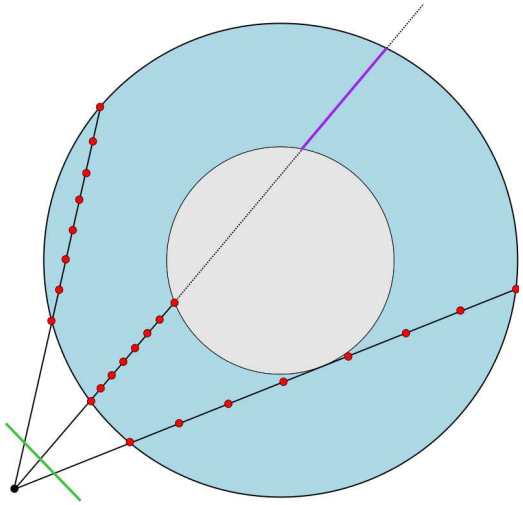
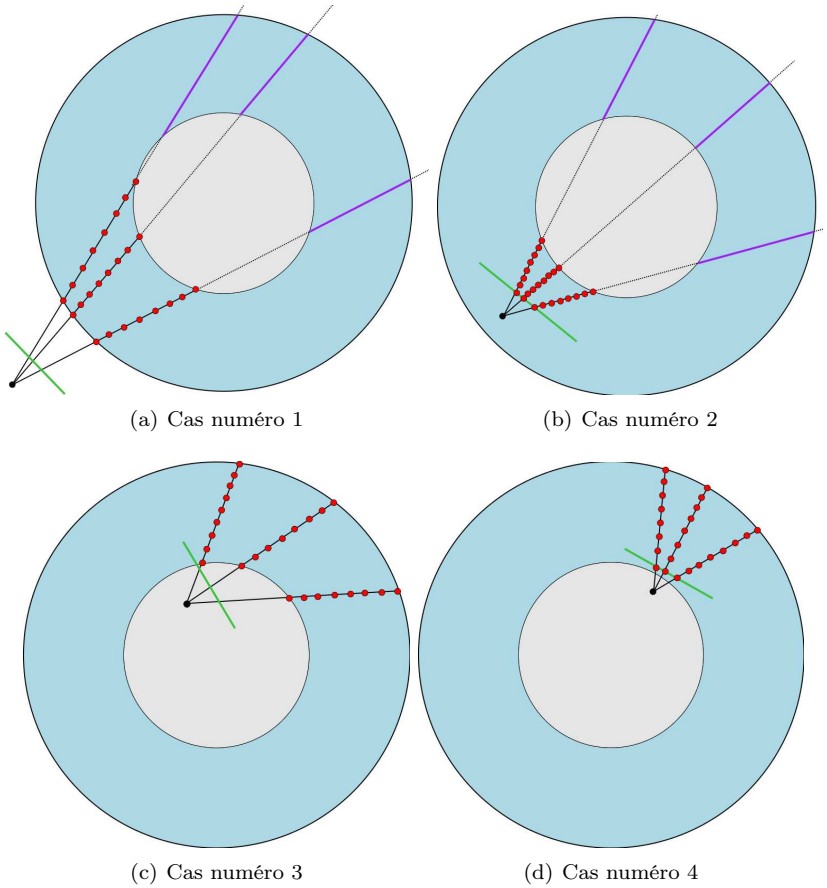


FIG. 31 – Cas hybride rencontré lors d'un raymarching d'une simulation gyrocinétique.



(a) Cas numéro 1 (b) Cas numéro 2

(c) Cas numéro 3 (d) Cas numéro 4

FIG. 32 – Les quatres principaux cas rencontrés lors d'un raymarching sur les données issues d'une simulation gyrocinétique.

Passe 3

La troisième et dernière passe sert à produire l'image finale.

En utilisant une projection orthogonale, on rend un quadrilatère de la taille exacte du viewport auquel on a affecté des coordonnées de texture normalisées telles que chaque fragment soit en mesure de pointer sur les coordonnées correctes de ses sommets de départ et de terminaison dans les deux textures (voir figure 33) .

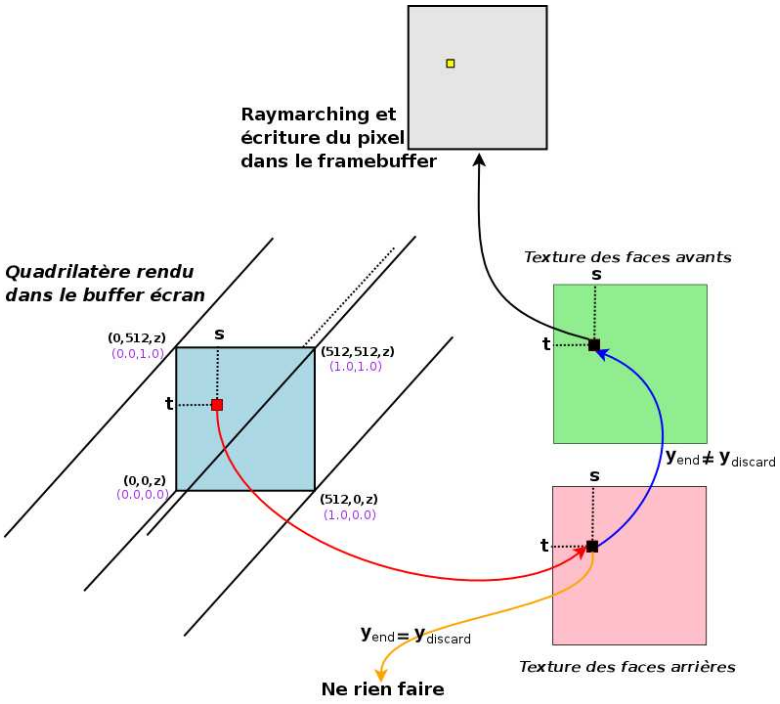


FIG. 33 – Lecture des textures relatives aux faces arrières et avants lors de la passe 3 du raymarching.

Dans le cas d'un viewport de 512×512 pixels, on définit une matrice de transformation tel que le coin inférieur gauche du parallélépipède de vision soit en $(0, 0)$ selon les axes (O, x) et $(0, y)$, la largeur et hauteur de ce dernier faisant 512 unités. Pour la profondeur z, grâce à la nature d'une projection orthogonale, il nous suffit de prendre la même composante z pour tous les sommets.

Un quadrilatère ainsi défini remplit alors l'écran :

x	y	u	v
0	0	0.0	0.0
512	0	1.0	0.0
512	512	1.0	1.0
0	512	0.0	1.0

où x et y sont respectivement l'abscisse et l'ordonnée du sommet, u et v étant les coordonnées de texture normalisées associées au sommet (dans le cas d'une texture de largeur 512, $u = 0$ correspond au premier texel alors que $u = 1.0$ correspond au $512^{ième}$ texel).

Avant de déclencher la phase de raymarching, on doit alors tester si le fragment en cours de traitement n'est pas issu d'une zone vide de l'espace. En allant chercher la valeur du sommet d'arrêt $(x_{end}, y_{end}, z_{end})$ dans la texture relative aux faces arrières, on effectue un test sur sa deuxième composante. Si $y_{end} = y_{discard}$, alors on arrête prématurément l'exécution du fragment shader. Si le test se révèle négatif, alors on sait qu'un rendu est nécessaire en ce fragment. En conséquence, on lit le texel correspondant dans la texture relatives aux faces

avants, $(x_{start}, y_{start}, z_{start})$.

A partir de ces informations, on initialise l'algorithme. Le vecteur directeur de rayon est dans un premier temps calculé :

$$ray = (x_{end}, y_{end}, z_{end}) - (x_{start}, y_{start}, z_{start})$$

Le nombre d'échantillons nb_{steps} à intégrer ayant été fixé au préalable et étant indépendant de la norme du vecteur directionnel (les sommets de départ et d'arrivée sont comptabilisés dans nb_{steps}), on détermine le vecteur d'avancement ray_{inc} dans l'espace monde :

$$ray_{inc} = \frac{ray}{nb_{steps} - 1}$$

Ensuite, on effectue une intégration de l'avant vers l'arrière en respectant la cohérence d'un rendu arrière vers l'avant :

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

où $C(i)$ est la couleur du $i^{ième}$ échantillon du rayon (les 3 composantes primaires rouge, vert et bleu), $\alpha(i)$ étant l'opacité. Les échantillons sont strictement ordonnés selon la profondeur afin d'obtenir un rendu volumique correct en respectant la transparence.

Comme dans tout rendu volumique, le passage de l'espace des données (valeur numérique unidimensionnelle) à l'espace des couleurs et opacités (quadruplets d'entier symbolisant leurs intensités respectives) est réalisé par l'emploi d'une fonction de transfert. Le champs scalaire étant composé d'entiers codés sur 1 octet (256 valeurs possibles) dans notre texture tridimensionnelle de simulation gyrocinétique, une telle fonction peut se représenter sous forme de texture unidimensionnelle où la coordonnée de texture est analogue à la valeur normalisée du scalaire recherché, le texel pointé stockant l'information de couleur et de transparence.

L'algorithme 1 explicite l'ensemble de ces opérations.

Algorithme 1 Raymarching utilisant une fonction de transfert.

```
1 :   (x_end,y_end,z_end) = texture2D(back_faces, (u, v))
2 :   Si y_end = y_discard Alors
3 :    rejeter le fragment, sortie du fragment shader
4 :   Fin de Si

5 :   (x_start,y_start,z_start) = texture2D(front_faces, (u, v))
6 :   ray = (x_end,y_end,z_end) - (x_start,y_start,z_start)
7 :   ray_inc = ray / (nb_steps - 1)
8 :   (x,y,z) = (x_start,y_start,z_start)
9 :   FragmentColor.rgba = (0.0, 0.0, 0.0, 1.0)

10 :   Pour (i=0, i<nb_steps, i++) Faire
11 :     texture_coord.x = sqrt(x^2 + z^2) - (R - r_max)
12 :     texture_coord.y = y
13 :     texture_coord.z = atan(z,x) / 2π
14 :     voxel = texture3D(tokamak_texture, texture_coord.xyz)
15 :     pixel.rgba = texture1D(transfer_function, voxel)
16 :     FragmentColor.rgb = FragmentColor.rgb + pixel.a*pixel.rgb*FragmentColor.a
17 :     FragmentColor.a *= (1.0 - pixel.a)
18 :     (x,y,z) = (x,y,z) + ray_inc
19 :   Fin de Pour
```

Une propriété intéressante de cette méthode et non présente dans l'approche de Crawford *et al.* est celle du respect des zones à intégrer dans le tokamak. En effet, lorsque l'on se place face au tore dans le plan Oxz, les deux parties opposées du tokamak n'ont pas à être rendues ensemble. Ce mélange (en violet dans la figure 32(a)) introduit une erreur dans le sens où dans la réalité,ces parties sont indépendantes. Ainsi, intégrer entièrement un rayon traversant le centre du tokamak est erroné et peut potentiellement induire en erreur un physicien dans l'analyse des données de la simulation gyrocinétique.

Grâce au filtre effectué lors des passes 1 et 2 avec le z-buffer, on est en mesure d'éviter ceci. La figure 34 illustre ce propos, où dans l'image de gauche (Crawford *et al.*) la partie arrière du tokamak en principe non visible est néanmoins prise en compte, tandis qu'à droite, un rendu de la même scène est effectué avec notre méthode (la partie arrière n'étant pas rendue).

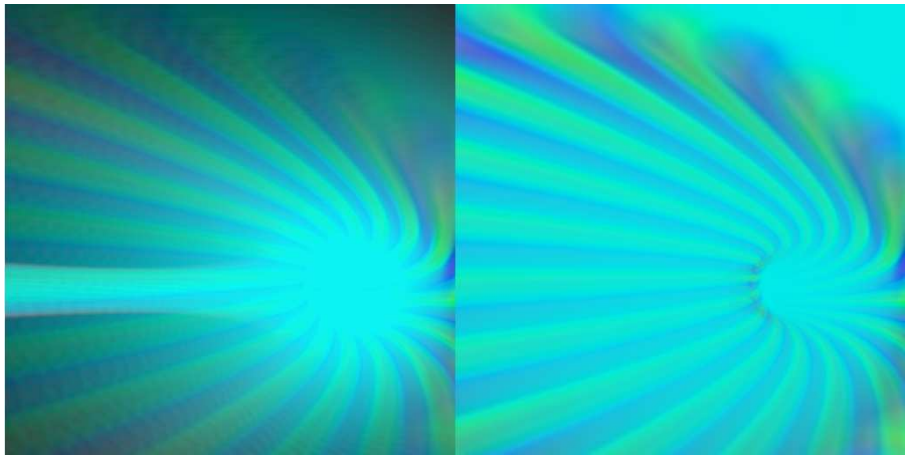
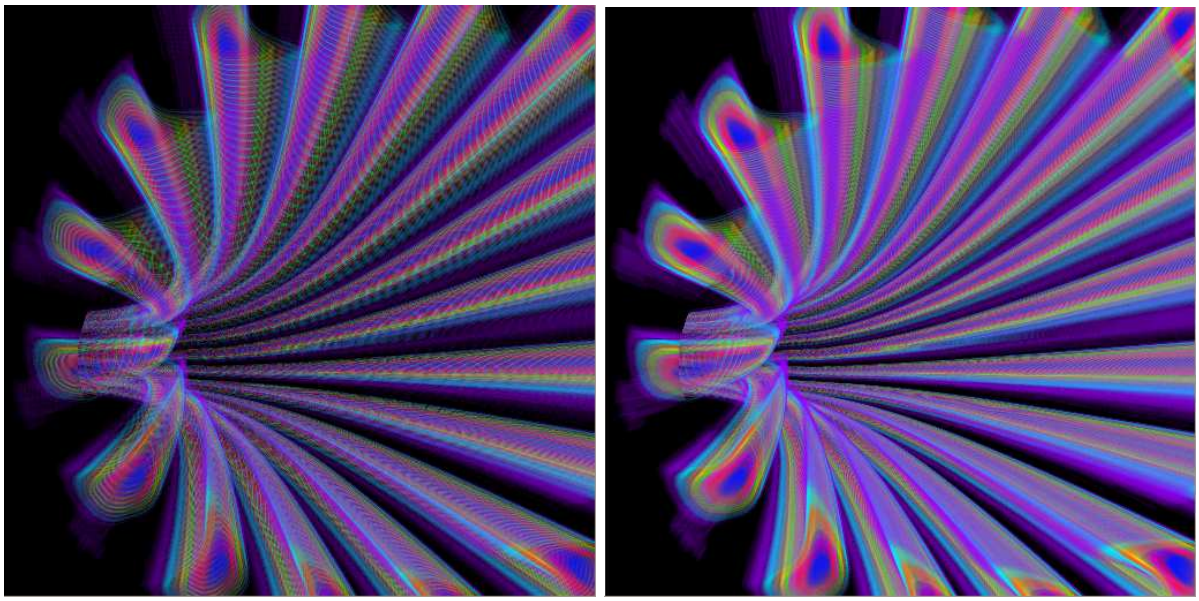


FIG. 34 – Comparaison entre une approche par découpe (à gauche) où l'information est érronée et un raymarching (à droite) où l'intégration du volume s'effectue correctement.

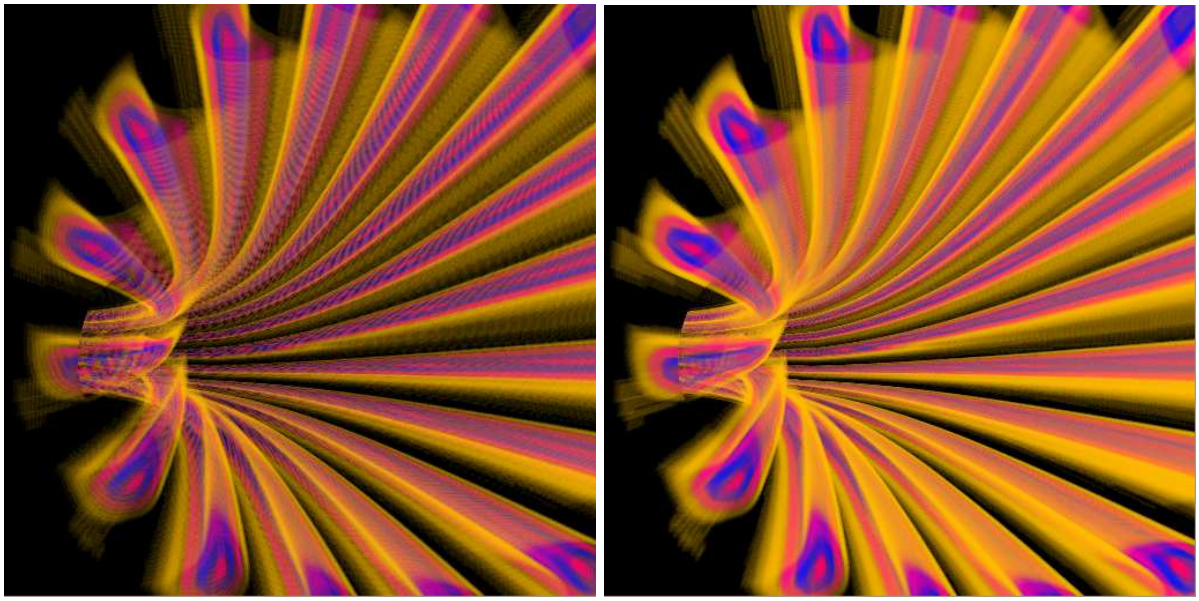
Les figures 35 et 36 montrent la visualisation intérieure d'une simulation gyrocinétique avec l'algorithme présenté, les figures 37 et 38 montrant des captures extérieures.

Afin d'améliorer en terme de qualité et de performance le rendu par raymarching, des optimisations ont été étudiées et testées comme le détaille la section suivante.



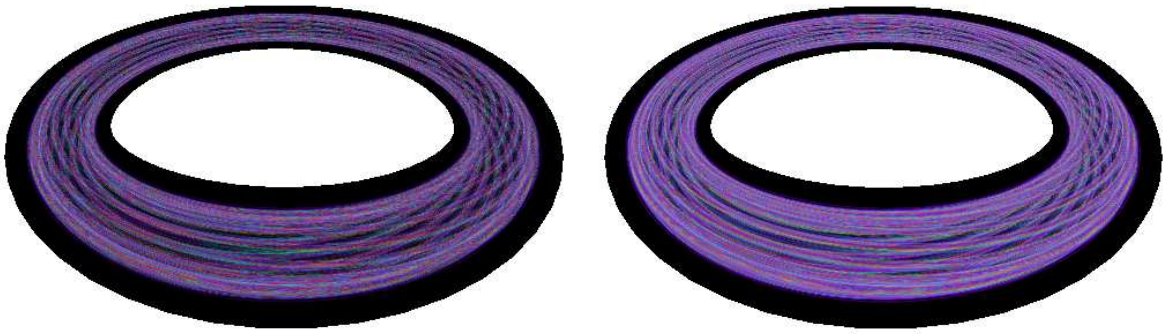
(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 35 – Visualisation de l'intérieur d'un tokamak par raymarching et en utilisant une fonction de transfert.



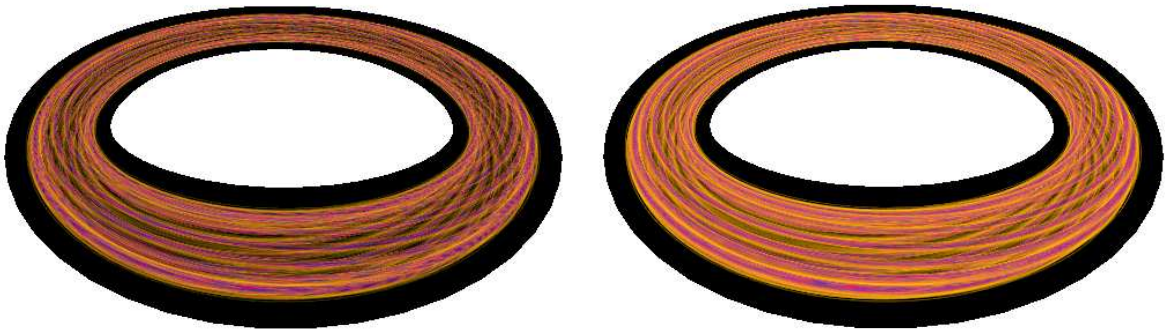
(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 36 – Visualisation de l'intérieur d'un tokamak par raymarching et en utilisant une autre fonction de transfert.



(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 37 – Visualisation de l'extérieur d'un tokamak par raymarching et en utilisant une fonction de transfert.



(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 38 – Visualisation de l'extérieur d'un tokamak par raymarching et en utilisant une autre fonction de transfert.

5 Améliorations et optimisations

5.1 MSPITFS

5.1.1 Intérêt

Une table de pré-intégration ne peut être utilisée correctement que lorsque le pas d'avancement dans l'espace entre les échantillons successifs est maintenu constant.

Cette dernière contrainte pose problème dans la nouvelle méthode proposée où le pas d'avancement peut varier grandement entre deux fragments distincts (la subdivision des rayons étant invariante selon la norme du rayon).

Pour bénéficier des avantages apportés par la pré-intégration standard, plusieurs niveaux de détails de tables de pré-intégration on été précalculés. Cette méthode a été nomée : Multi-Sampled Pre-Integrated Transfer Function Set (MSPITFS).

5.1.2 Calcul

Lors du calcul d'une table de pré-intégration classique, pour tout couple de scalaires (s_f, s_b) on détermine n scalaires intermédiaires par interpolation linéaire. Plus n est grand, plus on convergera vers l'intégration continue recherchée.

On calcule ensuite le rendu ordonné en prenant en compte la transparence selon l'équation d'intégration volumique discrète (2) ; l'intégration allant de s_b vers s_f. Les résultats ainsi trouvés sont stockés sous la forme d'une texture bidimensionnelle de taille $nb_{scalaire} \times nb_{scalaire}$ où $nb_{scalaire}$ représente l'éventail des scalaires possibles dans la texture 3D (pour une fonction de transfert de 256 valeurs, sa table de pré-intégration aura 256 texels en largeur ainsi qu'en hauteur). Dans notre implantation, l'axe des abscisses de la table de pré-intégration représente l'échantillon avant s_f, l'axe des ordonnées représentant celui des échantillons arrières s_b.

Dans le cas du calcul de la MSPITFS où l'on va déterminer plusieurs tables de pré-intégration à différents niveaux de détails pour une même fonction de transfert, un schéma de progression logarithmique de base 2 a été jugé le plus adéquat pour l'évolution du taux d'échantillonnage car :

- plus on converge vers un fort taux d'échantillonnage, moins les tables de pré-intégration voisines diffèrent.
- Déterminer le niveau de détail à sélectionner pour un fragment nécessite une formule dédiée, de préférence de faible coût. Le logarithme de base 2 étant une opération rapide sur les cartes graphiques récentes, cela à renforcé le choix de son usage.

En tirant parti du fait que l'interpolation inter-échantillon est linéaire et que l'on se place dans une logique d'évolution par puissance de deux, l'ensemble des tables de pré-intégration peut se calculer en utilisant uniquement les valeurs issues de l'échantillonnage maximal des scalaires.

Les tables résultantes de ce précalcul sont finalement stockées sous la forme d'une texture 3D où chaque tranche selon la profondeur représente une table de pré-intégration (voir figure 39). La tranche de coordonnée de texture la plus faible selon la profondeur correspond au plus haut niveau de détail ; inversement, celle de coordonnée de texture maximale correspondant à la table de pré-intégration de plus faible précision (la transposée de la fonction de transfert).

5.1.3 Utilisation

Afin d'utiliser les tables de la MSPITFS de façon optimale, il nous faut au préalable déterminer le plus long rayon que l'on sera amener à effectuer lors d'un raymarching.

Dans le cas d'un tokamak, en se basant sur l'analogie avec un tore et en utilisant le théorème de Pythagore, ray_{max} est trivialement calculé par $(R+r)^2 = (R-r)^2 + (\frac{ray_{max}}{2})^2$, soit $ray_{max} = 4\sqrt{Rr}$ (voir la figure 40).

Etant donné que l'on a un échantillonnage constant pour tous les rayons, plus la norme de ces derniers est élevé plus l'espace inter-échantillon croît, nécessitant une table de pré-intégration plus précise.

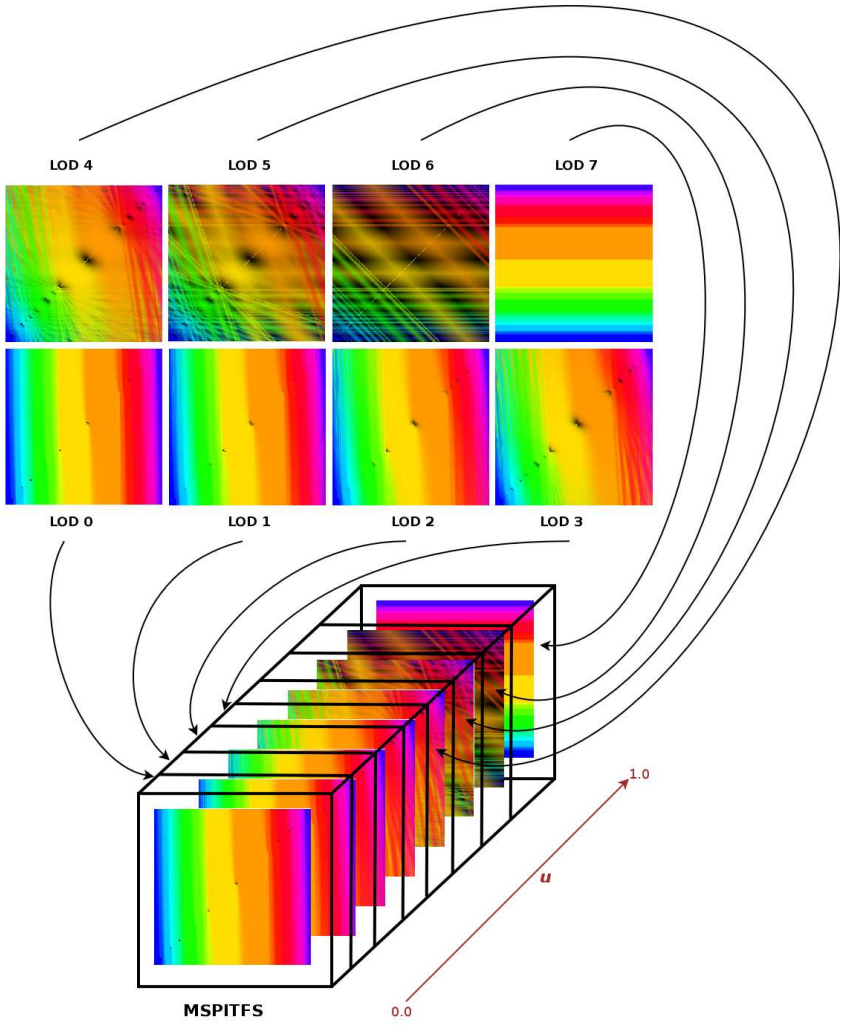


FIG. 39 – Exemple de MSPITFS avec 8 niveaux de détails.

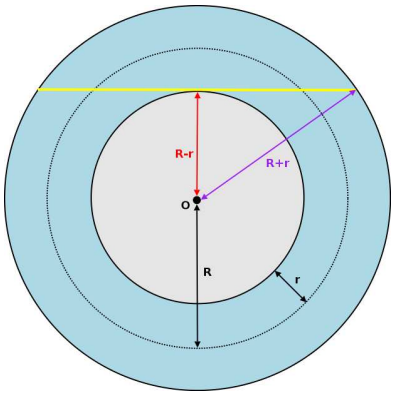


FIG. 40 – Visualisation du plus long rayon dans un tore.

La formule suivante met en relation la norme du rayon à intégrer pour un fragment avec l'indice de niveau de détail dans la MSPITFS (soit la coordonnée de texture selon la profondeur, $u_{MSPITFS}$) :

$$u_{MSPITFS} = \log \frac{ray_{max}}{\|ray\|} * \frac{1}{nb_{lod}} + \frac{1}{2nb_{lod}}$$

avec :

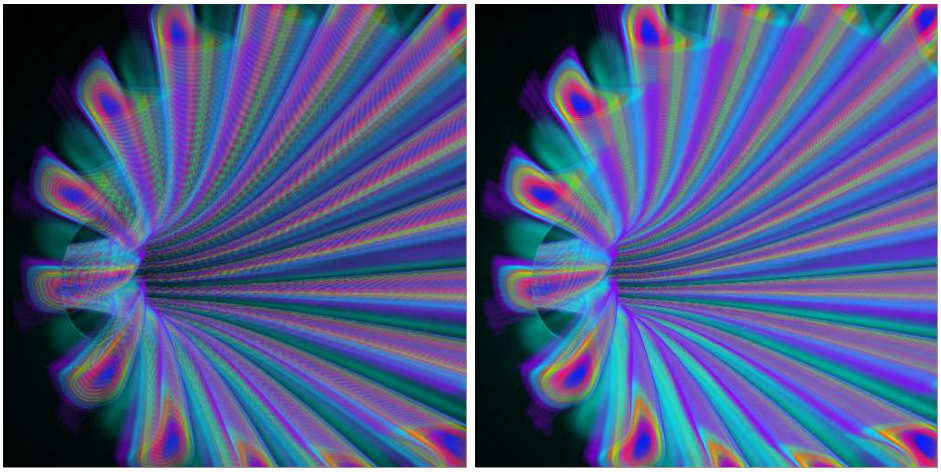
- ray le vecteur représentant le rayon lancé pour le fragment couramment traité, $\|ray\|$ étant sa norme,
- nb_{lod} le nombre de niveau de détail de tables de pré-intégration stocké dans la texture 3D de la MSPITFS.

On remarquera que le fait de rajouter $\frac{1}{2nb_{lod}}$ lors du calcul du niveau de détail de la MSPITFS est dû à l'expression des coordonnées de textures dans OpenGL. En effet, afin de s'aligner sur les centres des texels pour obtenir avec précision la table de pré-intégration adéquate, cette opération est nécessaire de par la convention qui a été adopté lors du développement de l'API. L'algorithme 2 présente le nouveau fragment shader relatif à la passe 3.

Algorithme 2 Raymarching avec MSPITFS.

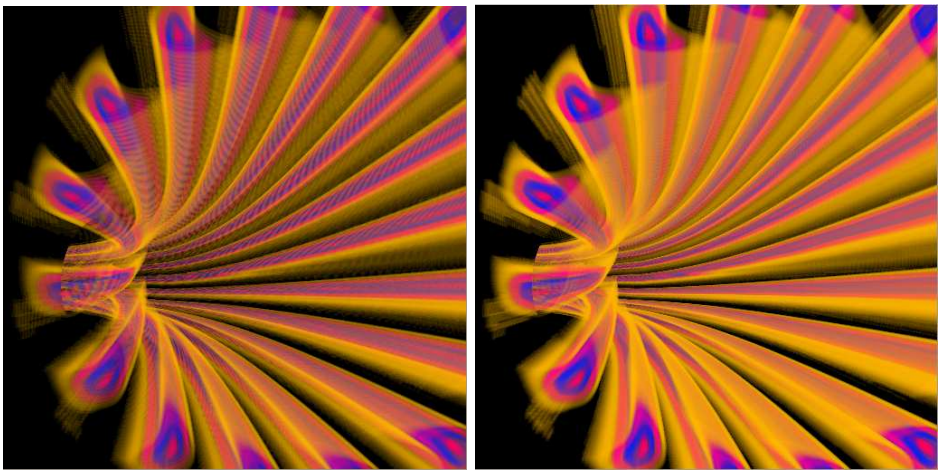
```
1 :   (x_end,y_end,z_end) = texture2D(back_faces, (u, v))

2 :   Si y_end = y_discard Alors
3 :    rejeter le fragment, sortie du fragment shader
4 :   Fin de Si

5 :   (x_start,y_start,z_start) = texture2D(front_faces, (u, v))
6 :   ray = (x_end,y_end,z_end) - (x_start,y_start,z_start)
7 :   ray_inc = ray / (nb_steps-1)
8 :   u_MSPITFS = log(ray_max/||ray||) × 1/nb_lod + 1/(2nb_lod)
9 :   (x,y,z) = (x_start,y_start,z_start)
10 :   texture_coord.x = sqrt(x^2 + z^2) - (R - r_max)
11 :   texture_coord.y = y
12 :   texture_coord.z = atan(z,x)/(2π)
13 :   s_f = texture3D(tokamak_texture, texture_coord.xyz)
14 :   (x,y,z) = (x,y,z) + ray_inc
15 :   FragmentColor.rgba = (0.0, 0.0, 0.0, 1.0)

16 :   Pour (i=0, i<nb_steps, i++) Faire
17 :     texture_coord.x = sqrt(x^2 + z^2) - (R - r_max)
18 :     texture_coord.y = y
19 :     texture_coord.z = atan(z,x)/(2π)
20 :     s_b = texture3D(tokamak_texture, texture_coord.xyz)
21 :     pixel.rgba = texture3D(MSPITFS, vec3(s_f, s_b, u_MSPITFS))
22 :     FragmentColor.rgb = FragmentColor.rgb + pixel.a*pixel.rgb*FragmentColor.a
23 :     FragmentColor.a *= (1.0 - pixel.a)
24 :     (x,y,z) = (x,y,z) + ray_inc
25 :     s_f = s_b
26 :   Fin de Pour
```

Les figures 41 et 42 montrent la visualisation intérieure d'une simulation gyrocinétique avec l'utilisation d'une MSPITFS de 12 niveaux de détails pour un suréchantillonage maximal de 4096, les figures 43 et 44 illustrant l'extérieur du Tokamak.



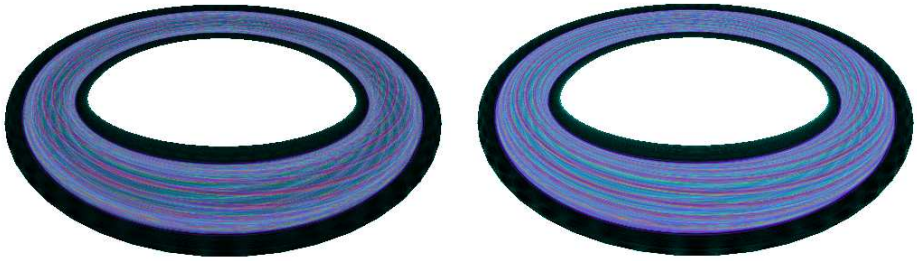
(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 41 – Visualisation intérieure d'une simulation gyrocinétique avec la MSPITFS d'une première fonction de transfert.



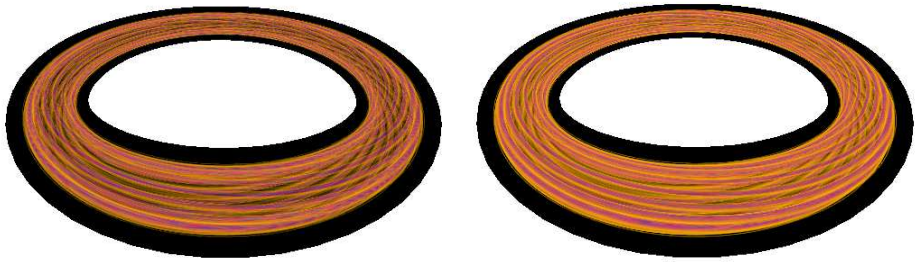
(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 42 – Visualisation intérieure d'une simulation gyrocinétique avec la MSPITFS d'une seconde fonction de transfert.



(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 43 – Visualisation extérieure d'une simulation gyrocinétique avec la MSPITFS d'une première fonction de transfert.



(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 44 – Visualisation extérieure d'une simulation gyrocinétique avec la MSPITFS d'une seconde fonction de transfert.

En utilisant un raymarching avec une MSPITFS, la présence des artefacts s'est vu diminuée et le rendu amélioré en conséquence.

5.2 Eclairage

Dans le cadre de l'ajout d'un éclairage, un modèle de Phong a été choisi comme BRDF, notamment pour sa complexité raisonnable et sa qualité de rendu :

$$I = I_a + I_d * (-\overrightarrow{L}.\overrightarrow{\nabla f}) + I_s * ((\overrightarrow{L} - 2 * (\overrightarrow{L}.\overrightarrow{\nabla f}) * \overrightarrow{\nabla f}).\overrightarrow{\nabla f})^n \tag{11}$$

avec :

$$\begin{cases} I_a \text{ la composante ambiante} \\ I_d \text{ la composante diffuse} \\ I_s \text{ la composante spéculaire} \\ \overrightarrow{L} \text{ le vecteur incident de lumière normalisé} \\ \overrightarrow{\nabla f} \text{ le gradient normalisé de notre localisation courante} \end{cases}$$

Le gradient $\overrightarrow{g} = \overrightarrow{\nabla f}$ est calculé à la volée selon un schéma de différence centrale. Soit, pour un voxel de coordonnées de texture (s, t, u), on a :

$$\overrightarrow{g} = \overrightarrow{(g_x, g_y, g_z)} = \begin{cases} g_x = (s + \frac{1}{width}, t, u) - (s - \frac{1}{width}, t, u) \\ g_y = (s, t + \frac{1}{height}, u) - (s, t - \frac{1}{height}, u) \\ g_z = (s, t, u + \frac{1}{depth}) - (s, t, u - \frac{1}{depth}) \end{cases} \tag{12}$$

où *width*, *height* et *depth* sont respectivement la largeur, la hauteur et la profondeur de la texture 3D. En utilisant les décalages tel qu'explicités dans la formule 12, on est sûr de visiter les voisins connexes à notre voxel actuel, cela étant induit par le système de normalisation des coordonnées de texture dans OpenGL.

De plus, on prend soin de normaliser le gradient afin de réduire au maximum les artefacts visuels liés à l'application de la BRDF :

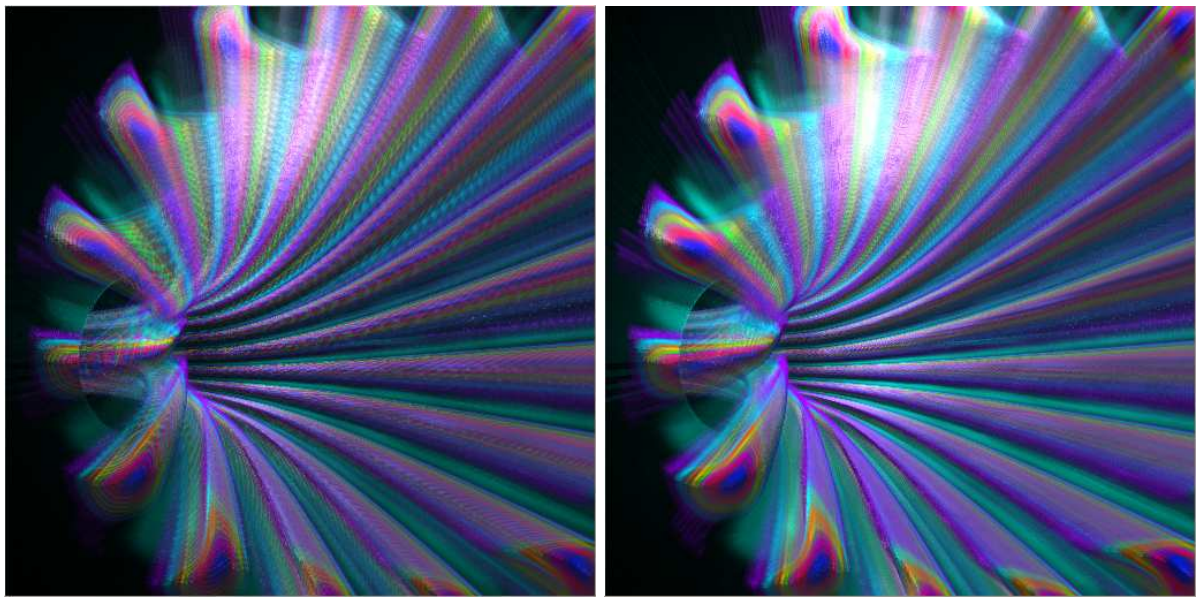
$$\overrightarrow{g} = \overrightarrow{(\frac{g_x}{\|\overrightarrow{g}\|}, \frac{g_y}{\|\overrightarrow{g}\|}, \frac{g_z}{\|\overrightarrow{g}\|})} \text{ avec } \|\overrightarrow{g}\| = \sqrt{{g_x}^2 + {g_y}^2 + {g_z}^2}.$$

L'algorithme 3 présente l'algorithme obtenu, pour un rendu par fonction de transfert, quand on considére que $n = 20$ dans l'équation 11.

Les figures 45 et 46 montrent la visualisation intérieure d'une simulation gyrocinétique avec l'utilisation d'une MSPITFS de 12 niveaux de détails pour un suréchantillonage maximal de 4096 ainsi qu'avec une illumination de Phong, les figures 47 et 48 illustrant l'extérieur du Tokamak.

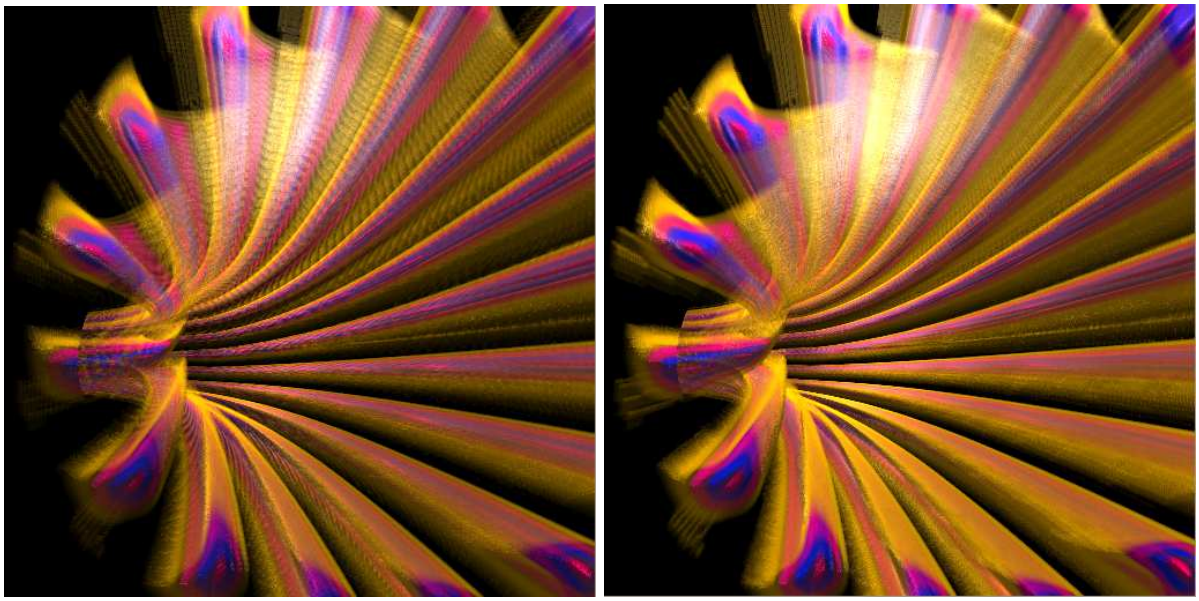
Algorithme 3 Raymarching avec éclairage de Phong.

```
1 :   (x_end,y_end,z_end) = texture2D(back_faces, (u, v))

2 :   Si y_end = y_discard Alors
3 :    rejeter le fragment, sortie du fragment shader
4 :   Fin de Si

5 :   (x_start,y_start,z_start) = texture2D(front_faces, (u, v))
6 :   ray = (x_end,y_end,z_end) - (x_start,y_start,z_start)
7 :   ray_inc = ray / (nb_steps-1)
8 :   (x,y,z) = (x_start,y_start,z_start)
9 :   FragmentColor.rgba = (0.0, 0.0, 0.0, 1.0)

10 :   Pour (i=0, i<nb_steps, i++) Faire
11 :     texture_coord.x = sqrt(x^2 + z^2) - (R - r_max)
12 :     texture_coord.y = y
13 :     texture_coord.z = atan(z,x) / 2π
14 :     voxel = texture3D(tokamak_texture, texture_coord.xyz)
15 :     pixel.rgba = texture1D(FT, voxel)
16 :     gradient.x = texture3D(tokamak_texture, vec3 (texture_coord.x+1/width,
       texture_coord.yz) - (texture_coord.x - 1/width, texture_coord.yz))
17 :     gradient.y = texture3D(tokamak_texture, vec3((texture_coord.x,texture_coord.y+1/height,
       texture_coord.z) - (texture_coord.x, texture_coord.y - 1/height,texture_coord.z)))
18 :     gradient.z = texture3D(tokamak_texture, vec3((texture_coord.xy,texture_coord.z+1/depth)
     - (texture_coord.xy,texture_coord.z+1/depth)))
19 :     normalize(gradient.xyz)
20 :     light_direction = normalize(LIGHT_POSITION.xyz - (x,y,z))
21 :     light_reflected = reflect(-light_direction.xyz, gradient.xyz)
22 :     view_vector = normalize(CAMERA_POSITION.xyz - (x,y,z))
23 :     pixel.rgb *= (I_a + max(light_direction · gradient,0)*I_d)
24 :     pixel.rgb += ((light_reflected · view_vector)^20 * I_s)
25 :     FragmentColor.rgb = FragmentColor.rgb + pixel.a*pixel.rgb*FragmentColor.a
26 :     FragmentColor.a *= (1.0 - pixel.a)
27 :     (x,y,z) = (x,y,z) + ray_inc
28 :   Fin de Pour
```



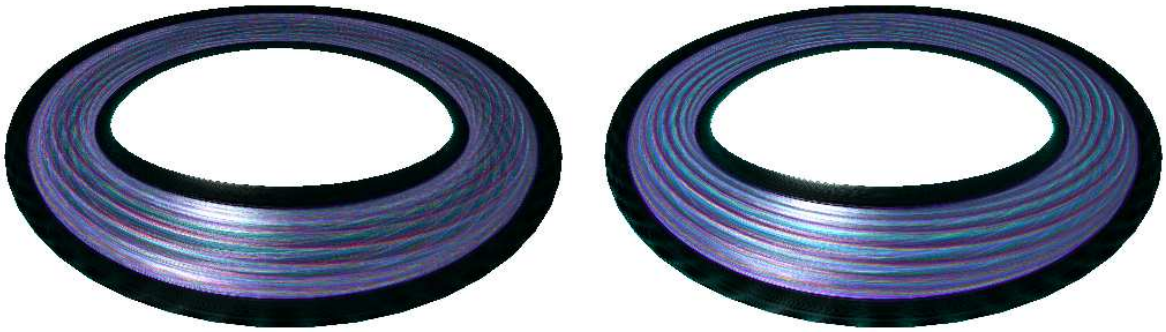
(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 45 – Visualisation de l'intéreur d'une simulation gyrocinétique par Raymarching avec MSPITFS et éclairage de Phong.



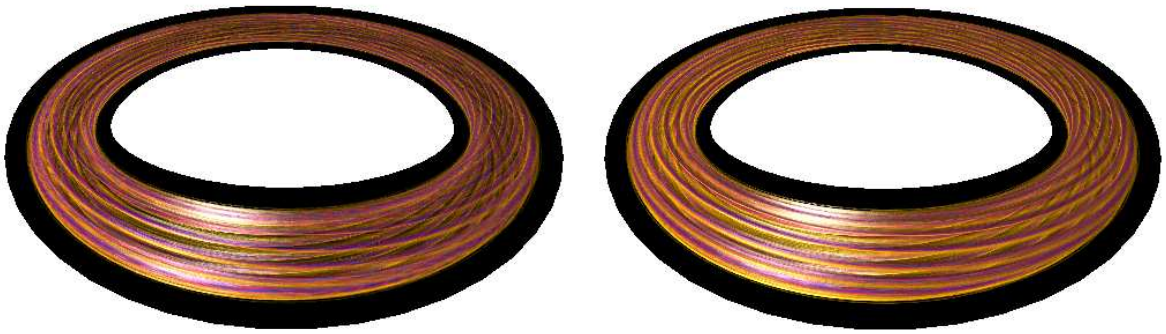
(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 46 – Visualisation de l'intéreur d'une simulation gyrocinétique par Raymarching avec MSPITFS et éclairage de Phong, une autre fonction de transfert étant utilisée.



(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 47 – Visualisation de l'extérieur d'une simulation gyrocinétique par Raymarching avec MSPITFS et éclairage de Phong.



(a) avec 64 échantillons par rayon (b) avec 128 échantillons par rayon

FIG. 48 – Visualisation de l'extérieur d'une simulation gyrocinétique par Raymarching avec MSPITFS et éclairage de Phong, une autre fonction de transfert étant utilisée.

5.3 Optimisations

Afin d'augmenter le nombre d'images par seconde obtenues lors du rendu, certaines optimisations ont été testées.

5.3.1 Table d'arc tangente

L'arc tangente est une fonction encore couteûse dans l'architecture des cartes graphiques récentes, cette dernière étant approchée par un développement limité qui totalise environ 30 cycles GPU. Une texture 2D a donc été précalculée afin de se passer de l'appel de cette fonction lors du changement de repère par échantillon.

Selon la taille de texture utilisée, on bénéficie d'une bonne approximation de la fonction arc tangente tel que pour une résolution de 256×256, il n'existe aucune différence notable entre le calcul à la volée et le précalcul avec texture.

5.3.2 Alpha early out

Etant donné que l'on effectue un rendu de l'avant vers l'arrière, il nous est possible d'arrêter prématurément une intégration pour un fragment en fonction de l'opacité accumulée.

A partir du moment où un échantillon de très faible transparence est intégré avec ceux qui l'ont été précédemment, on peut faire l'hypothèse que les échantillons suivants contribueront de façon négligeable à l'intensité finale du fragment.

6 Implantation

Une interface a été implantée en langage C, OpenGL, GTK (Gimp Tool Kit) et GTKGLext (OpenGL Extension to GTK). Elle permet notamment la visualisation volumique de simulation gyrocinétique, mais aussi de tout volume standard.

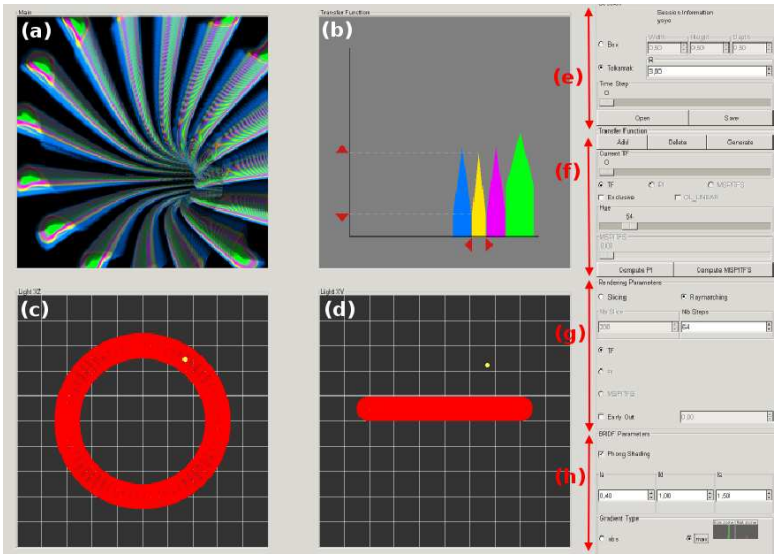


FIG. 49 – Interface de l'application.

Nous présentons ici une description de ces fonctionnalités en prenant appui sur la figure 49.

Quatres fenêtres OpenGL sont présentes, chacune ayant leur fonctionnalité :
- La fenêtre (a) est celle où le rendu est affiché. Il est possible de se déplacer en utilisant les touches directionnelles et la souris.
- La fenêtre (b) permet l'édition de la fonction de transfert ainsi que la visualisation de sa table de pré-intégration et de sa MSPITFS.
- La fenêtre (c) montre le maillage du volume avec la source de lumière (sphère jaune) positionnée dans le plan Oxz. En manipulant la souris, on peut déplacer la source lumineuse dans le plan.
- La fenêtre (d) est identique à la fenêtre (c), mais on se place cette fois ci dans le plan Oxy

6.1 Options de la session (e)

Dans la zone (e), il est possible d'ouvrir et de sauver une session. Dans le cadre d'un rendu standard d'une texture 3D, sauvegarder la session revient à stocker la texture dans un fichier avec ses propriétés (dimensions de la texture) ainsi que les fonctions de transfert lui étant associées. La création d'une nouvelle session demande à l'utilisateur de rentrer les dimensions de la texture, et elle commence avec une fonction de transfert nulle.

Dans le cas d'une simulation gyrocinétique, la création d'une session demande à l'utilisateur les dimensions de la texture à créer, chose qui influencera directement le rééchantillonage (les paramètres propres à la simulation sont disponibles dans les fichiers du CEA). Lors d'une sauvegarde, la texture rééchantillonnée, les fonctions de transfert ainsi que les divers paramètres (dimensions de la texture, nombre de plans poloïdaux, nombre de pas angulaires mineurs, nombre de pas radial) sont sauvegardés, ce qui permet notamment de ne pas réefectuer un rééchantillonnage à chaque visualisation d'une simulation.

Toujours dans le cas de la visualisation d'une simulation, il est possible de rendre la texture 3D dans sa forme torique ou bien dans une forme standard (une boite), les dimensions de ces dernières étant entèrement paramétrables.

Si la simulation en question possède plusieurs pas de temps, il est possible de visualiser chacun de ces derniers, le passage de l'un à l'autre étant immédiat.

6.2 Edition de la fonction de transfert (f)

Afin de pouvoir gérer librement l'apparence du rendu, l'édition interactive de la fonction de transfert est possible en utilisant exclusivement la souris. En utilisant des rampes linéaires pour définir des zones dans l'espace des scalaires (flèches rouges du bas pouvant être déplacées horizontalement), la couleur est assignée en choisissant un indice de hue variant de 0 à 360 (selon l'algorithme classique de conversion HSV à RGB). La transparence de cette zone est gérer par deux paliers (paramétrables avec les flèches verticales rouges). Il est possible de déclarer jusqu'à 32 zones dans l'espace des scalaires, ces dernières pouvant être constamment ajoutées, modifiées et supprimées (usage du bouton droit de la souris).

Lorsqu'une fonction de transfert est active, il est possible de rendre une seule de ces zones actives (mode exclusif), permettant l'exploration du champs scalaires et le contrôle sur les données auxquelles on affecte des propriétés.

Une table de pré-intégration ainsi qu'une MSPITFS peuvent être calculées avec comme choix le taux de sur-échantillonnage à utiliser, ainsi que le nombre de niveaux de détails pour la MSPITFS. Le résultat de ces dernières peut également être visualisé, dans le cas de la MSPITFS, une barre verticale permettant l'exploration selon la profondeur des tables. A noter la possibilité d'utiliser une interpolation trilinéaire sur la MSPITFS a été rendu possible, permettant de bénéficier du lissage par interpolation entre les différents niveau de détails.

6.3 Choix de la méthode de rendu (g)

Quelque soit la texture courrament visualisée (texture standard ou issue d'une simulation gyrocinétique), il est possible d'observer le volume par découpe ou par raymarching.

Dans le cas de la découpe, seul le nombre de polygones de support est paramétrable. Dans le cas d'un raymarching, le nombre d'échantillons par rayon est paramétrable par l'utilisateur. Il est possible de classifier le volume avec une simple fonction de transfert, avec une table de pré-intégration ou avec une MSPITFS (ces deux dernières se doivent d'être déja calculée). Il est également possible d'activer l'optimisation basée sur le test de transparence (alpha early out) avec comme réglage la valeur seuil de transparence accumulée.

6.4 Eclairage (h)

Il est possible d'activer l'éclairage de Phong, la source de lumières étant librement positionnable dans l'espace par les deux fenêtres de controles (c) et (d). La valeur des composantes ambiante, diffuse et spéculaire sont modifiables et permettent de tester plusieurs configurations.

A noter que l'emploi d'une valeur absolue ainsi que d'un maximum est un choix utilisateur dans le calcul de l'intensité lumineuse (ces deux approches sont répandues en rendu volumique.

7 Résultats

Les tests de performance ont été effectués sur une machine dotée d'un processeur AthlonXP 1800 ainsi que d'une carte graphique GeForce 6800 Ultra. Une fenêtre de 512 sur 512 pixels a été utilisée pour le rendu des images. Les textures 3D utilisées sont issues d'un rééchantillonnage des données du CEA. Elles ont pour résolution 256^3 voxels, chaque voxel étant codé sur un octet.

La méthode de Crawford *et al.* a été ré-implantée selon la description faite dans leur papier [CMH+04], cela afin que les résultats des deux approches puissent être comparés sur la même machine. Les résultats sont exprimés dans les tableaux en nombre d'images par seconde.

Méthode de Crawford *et al.* :

Nombre de coupes	500	1000	1500
Standard	6	3.8	1.6
Eclairage	2.4	1.1	0.1
Standard,Atan	11	6	3.7
Eclairage,Atan	4.8	2.3	0.9

où :
- « Nombre de coupes » est le nombre de polygones de support utilisés pour la découpe du volume.
- « Standard » tient pour un rendu simple avec application de la fonction de transfert.
- « Eclairage » signifie que la BRDF de Phong est appliquée.
- « Atan » signifie qu'une table précalculée d'arc tangente a été utilisée à la place du développement limité fait par la carte graphiqu.e

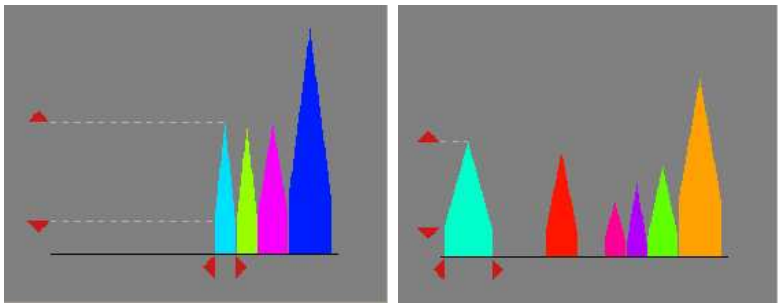
Adaptive Raymarching :

Nombre d'échantillons	32	64	128
TF	31	26	19
MSPITFS	28	22	16
TF,Atan	31	27	21
MSPITFS,Atan	29	23	19
TF,EO	33	29	25
MSPITFS,EO	30	27	25
TF,Eclairage	22	16	11
MSPITFS,Eclairage	22	15	10
TF,Eclairage,EO	31	27	21
MSPITFS,Eclairage,EO	23	18	14

où :
- « Nombre d'échantillons » est le nombre d'échantillons utilisés dans la subdivisions des rayons.
- « TF » tient pour un rendu avec application de la fonction de transfert.
- « MSPITFS » tient pour un rendu avec utilisation des tables de la MSPITFS.
- « Atan » signifie qu'une table précalculée d'arc tangente a été utilisée à la place du développement limité fait par la carte graphique
- « EO » tient pour l'utilisation de l'optimisation basé sur l'accumulation de la transparence le long d'un rayon.
- « Eclairage » signifie que la BRDF de Phong est appliquée.

On constate qu'une approche basée sur un raymarching adaptatif est plus performante que la méthode de Crawford *et al.* [CMH+04], et offre la possibilité de visualiser interactivement une simulation gyrocinétique avec un rendu de meilleure qualité.

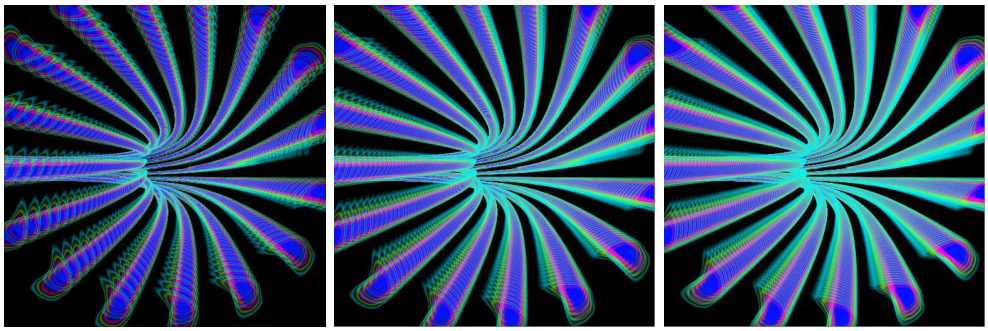
Voici quelques captures effectuant une comparaison entre la visualisation d'une simulation gyrocinétique par Raymarching adpatatif et par la méthode de Crawford *et al.*. La figure 50 illustre les deux fonctions de transfert utilisées lors des captures.



(a) Une première fonction de transfert. (b) La seconde fonction de transfert.

FIG. 50 – Edition des fonctions de transfert utilisées dans les rendus.

Dans la figure 51, on visualise l'intérieur du tokamak par découpe (Crawford *et al.*) avec 500 découpes éffectuée sur la boite englobante du tokamak (image 51(a)), 1000 découpe (image 51(b) et 1500 découpes (image 51(c), la fonction de transfert utilisée étant la première.

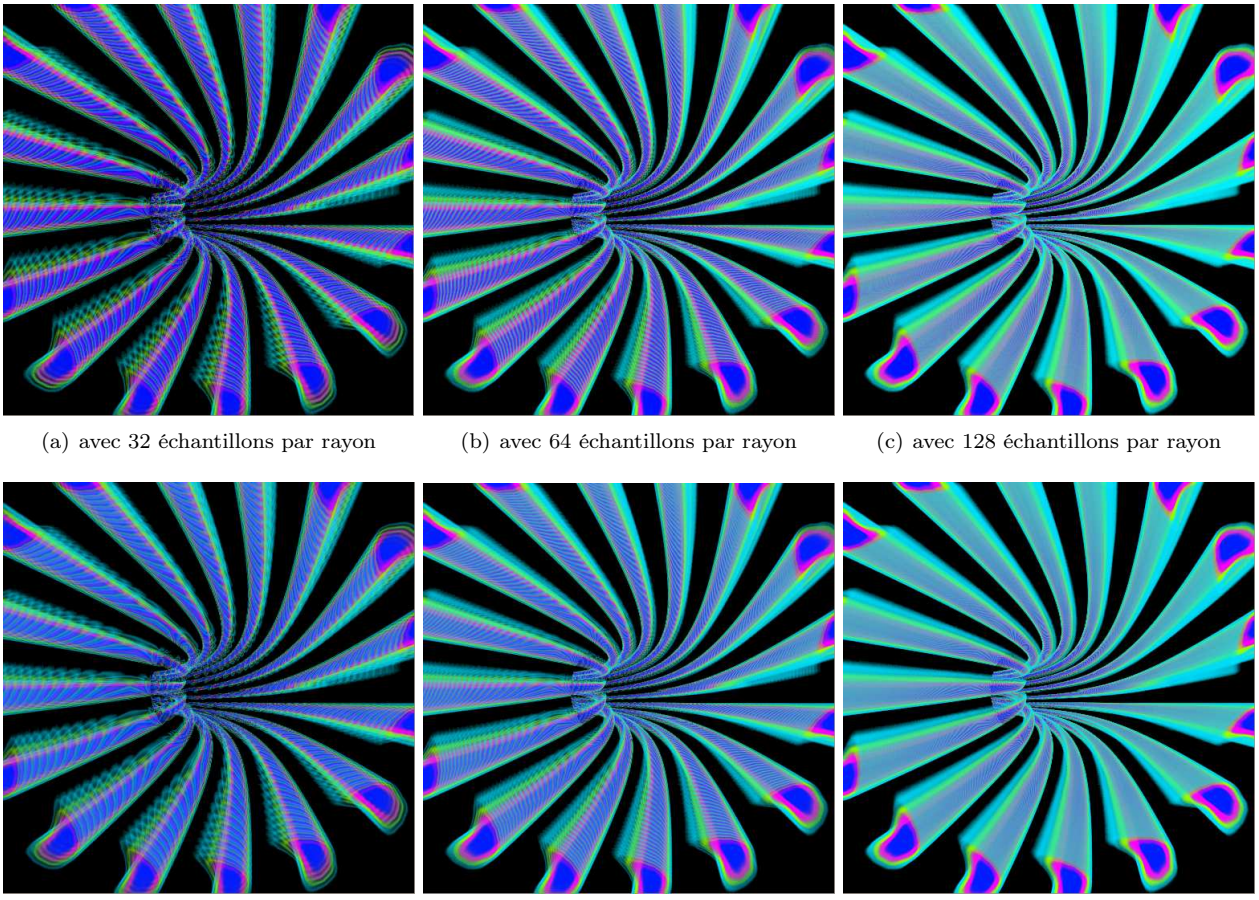


(a) avec 500 coupes (b) avec 1000 coupes (c) avec 1500 coupes

FIG. 51 – Rendu d'une simulation gyrocinétique par découpe, avec la première fonction de transfert.

(a) avec 500 coupes (b) avec 1000 coupes (c) avec 1500 coupes

FIG. 52 – Rendu d'une simulation gyrocinétique par découpe, avec la seconde fonction de transfert.



(a) avec 32 échantillons par rayon (b) avec 64 échantillons par rayon (c) avec 128 échantillons par rayon

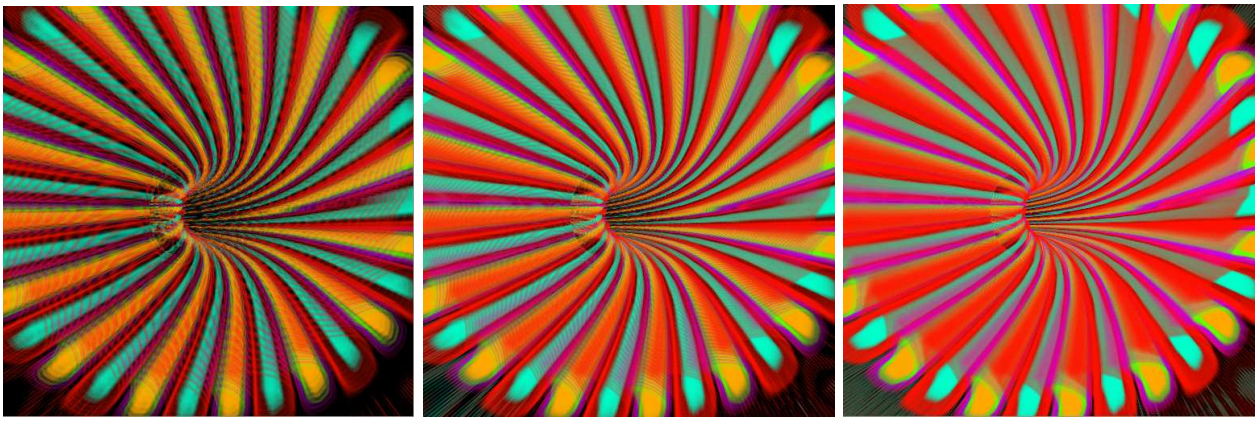
(d) avec 32 échantillons par rayon et MSPITFS (e) avec 64 échantillons par rayon et MSPITFS (f) avec 128 échantillons par rayon et MSPITFS

FIG. 53 – Visualisation d'une simulation gyrocinétique par raymarching, la première fonction de transfert étant utilisée.

La figure 52 illustre toujours la méthode de Crawford mais cette fois-ci avec la seconde fonction de transfert. L'image 52(a) est le résultat de l'utilisation de 500 polygones de support, l'image 52(b) et l'image 52(c) l'étant respectivement pour 1000 et 1500 plans de découpes.

La figure 53 illustre une approche par raymarching. Avec l'application simple de la première fonction de transfert , le volume est rendu avec 32 (image 53(a)), 64 (image 53(b)) et finalement 128 échantillons (image 53(c)). De même, mais avec l'utilisation d'une MSPITFS relative à la première fonction de transfert ayant 12 niveaux de détails et d'interpolation maximales interscalaire 4096 valeurs, la simulation gyrocinétique est visualisé avec 32 (image 53(d)), 64 (image 53(e)) et 128 échantillons (image 53(f)).

Finalement la figure 54 expose les même captures que la figure 53 (les paramètres de la MSPITFS sont conservés) mais pour la seconde fonction de transfert. Ainsi, on a un rendu par fonction de transfert pour 32 (image 54(a)), 64 (image 54(b)) et 128 échantillons (image 54(c)), et par MSPITFS pour 32 (image 54(d)), 64 (image 54(e)) et 128 échantillons (image 54(f)).



(a) avec 32 échantillons par rayon (b) avec 64 échantillons par rayon (c) avec 128 échantillons par rayon

(d) avec 32 échantillons par rayon et MSPITFS (e) avec 64 échantillons par rayon et MSPITFS (f) avec 128 échantillons par rayon et MSPITFS

FIG. 54 – Visualisation d'une simulation gyrocinétique par raymarching avec la seconde fonction de transfert.

8 Conclusion

Ce travail technique a permis de concevoir ainsi que d'implanter une application permettant la visualisation de simulations gyrocinétiques de plasma, mais aussi de façon plus générale, elle permet la visualisation de données volumiques en utilisant une approche basée sur les textures 3D.

La précédente approche[CMH+04] ne permettait pas la visualisation de simulations gyrocinétiques de plasmas à des performances interactives, ainsi aucune amélioration en terme de rendu et en terme de réalisme n'était possible. De plus, des données non corrélées (zones du tokamak diamétralement opposées) étaient fusionnées lors du rendu, ce qui constitue une erreur pouvant préter à confusion l'analyse des physiciens.

Notre implantation a été conçue dans l'esprit de pallier à ces problèmes. Ainsi, en proposant de visualiser des simulations gyrocinétiques de plasmas par raymarching, la contrainte de l'interactivité est largement satisfaite. En conséquence, un moyen d'améliorer la qualité du rendu a été étudié, et réalisé grâ à l'utilisation de niveaux de détails de tables de pré-intégration, « MSPITFS ».

De par la nature d'un plasma, les données issues des simulations numériques sont au moins dépendantes de deux dimensions supplémentaires en sus des trois dimensions relatives à la localisation des particules dans l'espace ; à savoir le temps et l'espace des phases. Le matériel graphique ne pouvant bénéficier actuellement de plus de 256Mo de RAM, trouver des moyens de compression efficaces des données dont la décompression et le rendu s'effectueront à même la GPU est loin d'être trivial.

L'approche qui a été exposé dans ce rapport est en mesure de rendre des simulations de plasmas 3D+t, cela sans compression nécessaire, à condition que les données n'atteignent pas de hautes résolutions (en terme de nombre de plans poloïdaux, de nombre de pas angulaires mineur et en terme de progression radiale). Dans ce cas, la taille est suffisamment faible pour qu'un transfert de la mémoire centrale à la mémoire GPU puisse se faire en temps réel.

Références

[BR98] Uwe Behrens and Ralf Ratering. Adding shadows to a texture-based volume renderer. In *VVS '98 : Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 39–46, New York, NY, USA, 1998. ACM Press.

[CFG+04] 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.

[CMH+04] 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.

[EE02] K. Engel and T. Ertl. Interactive high-quality volume rendering with flexible consumer graphics hardware. In *EuroGraphics 2002 State-of-the-Art (STAR) Report*, September 2002.

[EHK+03] K. Engel, M. Hadwiger, J. M. Knis, A. Lefohn, and D. Weiskopf. Interactive visualization of volumetric data on consumer pc hardware. In *Tutorial Notes, IEEE Visualization 2003*, October 2003.

[EKE01] 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.

[FMS00] Ricardo C. Farias, Joseph S. B. Mitchell, and Cláudio T. Silva. Zsweep : an efficient and exact projection algorithm for unstructured volume rendering. In *Volviz*, pages 91–99, 2000.

[FSB01] F. Filbet, E. Sonnendrücker, and P. Bertrand. Conservative numerical schemes for the Vlasov equation. *J. Comput. Phys.*, 172(1) :166–187, 2001.

[FTAT00] Issei Fujishiro, Yuriko Takeshima, Taeko Azuma, and Shigeo Takahashi. Volume data mining using 3d field topology analysis. *IEEE Comput. Graph. Appl.*, 20(5) :46–51, 2000.

[GBB+] 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.*

[GHY98] R. Grzeszczuk, C. Henn, and R. Yagel. Advanced geometric techniques for ray casting volumes. *SIGGRAPH '98*, 1998.

[GK96] 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.

[HHKP96] Taosong He, Lichan Hong, Arie Kaufman, and Hanspeter Pfister. Generation of transfer functions with stochastic search techniques. In *VIS '96 : Proceedings of the 7th conference on Visualization '96*, pages 227–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[KD98] Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *VVS '98 : Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 79–86, New York, NY, USA, 1998. ACM Press.

[KKH02] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3) :270–285, 2002.

[KPH+03] 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.

[KPHE02] J. Kniss, S. Premoze, C. Hansen, and D. Ebert. Interactive translucent volume rendering and procedural modeling. In *VIS '02 : Proceedings of the conference on Visualization '02*, pages 109–116, Washington, DC, USA, 2002. IEEE Computer Society.

[KW03] J. Krueger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings IEEE Visualization 2003*, 2003.

[LHJ99] Eric LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *VIS '99 : Proceedings of the conference on Visualization '99*, pages 355–361, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[LWM04] E. B. Lum, B. Wilson, and K-L. Ma. High-quality lighting for pre-integrated volume rendering. In *VisSym*, pages 25–34, 2004.

[MG02] 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.

[MHS99] 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.

[MSW+02] 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.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6) :311–317, 1975.

[RGW+03] 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.

[RSEB+00] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *HWWS '00 : Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 109–118, New York, NY, USA, 2000. ACM Press.

[SM97] Claudio T. Silva and Joseph S. B. Mitchell. The lazy sweep ray casting algorithm for rendering irregular grids. *IEEE Transactions on Visualization and Computer Graphics*, 3(2) :142–157, 1997.

[WS04] Gunther H. Weber and Gerik Scheuermann. *Automating Transfer Function Design Based on Topology Analysis*. 2004.

[WSH03] Gunther H. Weber, Gerik Scheuermann, and Bernd Hamann. Detecting critical regions in scalar fields. In Georges-Pierre Bonneau, Stefanie Hahmann, and Charles D. Hansen, editors, *Data Visualization 2003 (Proceedings of VisSym)*, pages 85–94, New York, New York, 2003. EUROGRAPHICS and IEEE TCVG, Association for Computing Machinery.

Table des matières

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803