

# Reconstruction tomographique en utilisant le GPU

## Introduction

Il existe de nombreux algorithmes de **reconstruction tomographique**, selon le type de scanner, la qualité ou le temps de reconstruction voulue. Mon programme se concentre sur l'algorithme de reconstruction OSEM à partir d'images de scanner TEMP, mais l'orientation objet du code devrait le rendre assez flexible à des évolutions futures.

Il est d'ailleurs prévu à terme de prendre en compte la réponse impulsionnelle du capteur.

Concernant l'utilisation du **GPU**, on peut dire que les progrès récents sur [les formats de données](#) font tomber les dernières difficultés pratiques, et rendent les GPUs tout à fait adéquats pour ce genre de calculs.

Möller et Haines, dans leur étude comparative du fonctionnement des GPUs et des CPUs[1], nous disent qu'à technologie équivalentes un GPU pourra être jusqu'à 100 fois plus **performant** sur des algorithmes de rendu; en fait on approche ici des performances d'une puce dédiée, tout en gardant les avantages d'évolutivité et de prix.

## Les algorithmes de reconstruction

Je n'entrerais pas dans le détail ici. Une excellente référence est *Analytic and Iterative Reconstruction Algorithms in SPECT*[2] par Philippe P. Bruyant. Dans tous les cas, l'algorithme de reconstruction doit partir d'un ensemble de "projections mesurées" -délivrées par le scanner médical- et **reconstruire un volume**. Je ne traite que le cas des projections **orthogonales**.

L'algorithme de **rétroprojection filtrée** peut s'écrire simplement par:

$Volume = \text{Rétroprojection}(\text{Filtrage}(\text{Projections mesurées}))$

Et l'algorithme **OSEM** est un processus itératif défini par:

$Volume(k+1) = Volume(k) * \text{Rétroprojection}(\text{Projections mesurées} / \text{Projections du Volume}(k))$

Les formules sont plutôt simples mais les difficultés vont venir d'autres points:

- difficulté à maintenir la **précision** voulue dans les calculs
- difficulté à prendre en compte **les erreurs de mesure** dans la reconstruction, principalement la réponse impulsionnelle du capteur et l'atténuation.

## Le programme

Les algorithmes de reconstruction font intervenir 2 objects, *Volume* et *ProjectionsSet*, et 2 opérateurs, *projection* et *backprojection*.

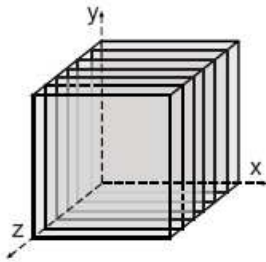
Volume
<code>+projection(): ProjectionsSet</code>

ProjectionsSet
<code>+backprojection(): Volume</code>

Ce sont les éléments de base de mon programme, je vais les décrire plus en détails.

## Volume

C'est un ensemble de valeurs sur 3 dimensions que l'on cherche à retrouver. Mon programme ne supporte que les dimensions cubiques; en pratique on utilisera surtout des volumes  $64^3$ ,  $128^3$  et  $256^3$ . Si on veut reconstruire un espace de dimensions arbitraires il faudra le partitionner en sous-ensembles cubiques.



Volume representation as a stack of 2D textures.

Le volume est stocké directement en mémoire graphique sous forme de textures (le but du jeu étant de minimiser le temps d'accès aux données et de limiter au maximum les transferts GPU <-> mémoire centrale).

Il y a 2 choix à faire pour la représentation de ce volume:

- doit-on utiliser une texture 3D ou une pile de textures 2D ?
- doit-on utiliser des textures 8 bits ou 16 bits?

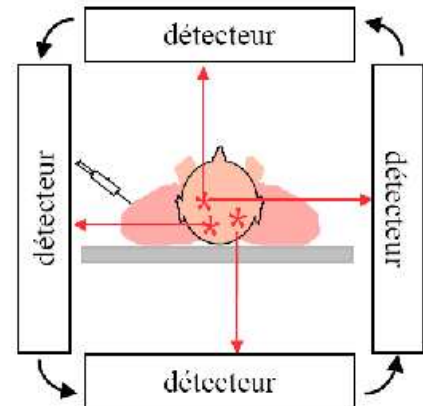
Ces questions seront discutées en détail dans les paragraphes suivants.

## ProjectionsSet

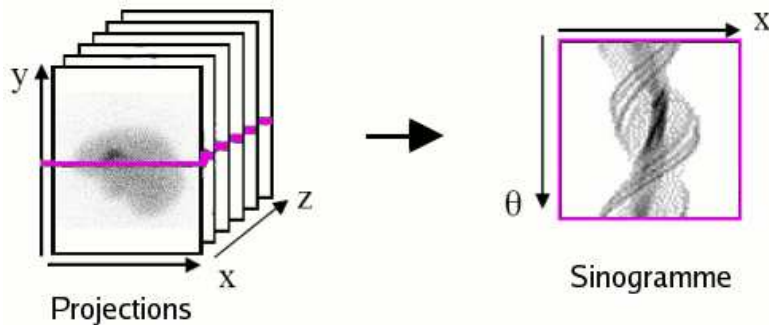
C'est un ensemble de projections du volume selon différents angles.

C'est typiquement ce que nous délivre le scanner médical: le patient étant allongé, le détecteur tourne autour de lui en prenant des images tout les quelques degrés.

Cette opération est simulé par l'opérateur *projection()* qui part d'un *Volume* pour obtenir un *ProjectionsSet*.

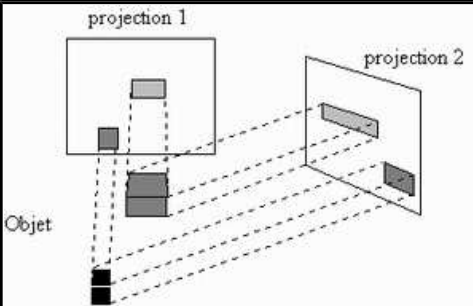
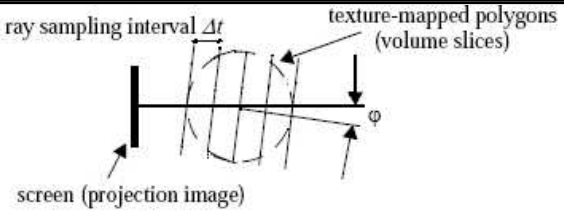


Typiquement un *ProjectionsSet* est composé de 60 à 120 images. Là encore on peut choisir de les stocker dans un ensemble de textures 2D ou dans une texture 3D. Si on les stocke dans une texture 3D, un avantage est qu'il suffit de prendre une tranche horizontale de la texture pour obtenir ce qu'on appelle un **sinogramme**: l'évolution d'une ligne de pixels en fonction de l'angle de projection.

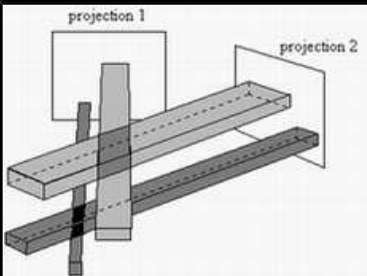
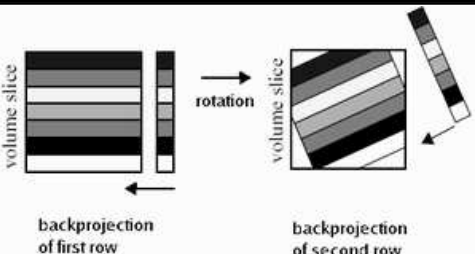


Un sinogramme est intéressant car il contient toutes les informations relatives à une coupe transaxiale du volume. Autrement dit, il suffit juste d'un sinogramme pour reconstruire la coupe correspondante.

## projection

Concept	Implémentation
 <p>Cet opérateur projette orthogonalement les valeurs du volume pour toute une série d'angles. On parle d'<b>accumulation</b> car les valeurs projetées sont additionnées dans une image, chaque pixel contenant la somme des valeurs du volume le long de sa ligne de projection.</p>	 <p>On oriente la caméra suivant l'angle de projection <math>\varphi</math>, et on dessine le volume tranche par tranche avec la fonction de <b>mélange</b> <code>GL_ADD</code> activée. En fonction de <math>\varphi</math>, le pas d'intégration <math>\Delta t</math> varie légèrement et il faut corriger l'image finale en la multipliant par <math>1/\cos(\varphi)</math>. <i>Voir <a href="#">[3]</a> pour plus d'informations.</i></p>

## backprojection (rétroprojection)

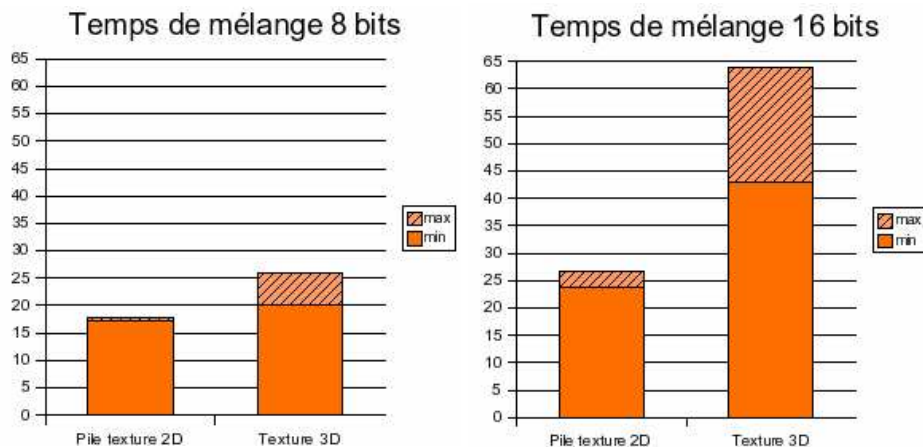
Concept	Implémentation
 <p>Pour chaque projection, la valeur des pixels est épanchée orthogonalement dans le volume. Les valeurs rétroprojetées sont additionnées, et on normalise à la fin par le nombre de projections. Ce n'est pas exactement l'opérateur inverse de la projection, mais cela constitue une bonne approximation.</p>	 <p>Mon programme effectue la rétroprojection tranche par tranche, autrement dit 1 sinogramme à la fois. On place la caméra en vue de dessus, et on épanche chaque ligne du sinogramme séparée à chaque fois par une petite rotation axiale. Tout comme pour la projection, la fonction de mélange <code>GL_ADD</code> est activée. Cette méthode est très rapide car on accède seulement à des lignes de texture, donc la bande-passante des textures est très peu sollicitée. <i>Voir <a href="#">[4]</a> pour plus d'informations.</i></p>

## Pile de textures 2D ou texture 3D ?

Comparons ces 2 méthodes de représentation du volume:

avec une <b>Pile de textures 2D</b>	avec une <b>Texture 3D</b>
<b>Avantages:</b> - Accès texture plus rapide.	<b>Avantages:</b> - peut représenter le volume depuis n'importe quel angle. - facile à mettre à jour. - plus besoin de changement d'unité de texture durant les rendus. - globalement, rend l'algorithme plus simple à implémenter.
<b>Inconvénients:</b> - il faut au moins 2 piles pour pouvoir représenter le volume sous tous les angles (cela prend 2 fois plus de mémoire). - difficile à mettre à jour de façon optimisé.	<b>Inconvénients:</b> - il n'est pas clair que les drivers actuels supportent toutes les fonctionnalités des texture 3d flottantes, comme par exemple la possibilité de faire un rendu directement dans la texture.

Toutes les cartes graphiques sont optimisées pour l'utilisation de textures 2D, c'est donc normal que les lectures sur une texture 3D soient plus lentes. Ce petit [programme](#) permet de mesurer le temps nécessaire pour projeter un volume(rempli aléatoirement) en fonction du type de texture. Voici les résultats pour une geforce 6800:



Selon les coordonnées de texture utilisées, il y a des variations (parties hachurées) dans le temps de projection.

Globalement la méthode texture 2d est 1,4x à 2x plus rapide en ce qui concerne le mélange. Mais si on prend en compte tout les avantages de la texture 3D, dont le principal est de pouvoir représenter le volume sous n'importe quel angle, je crois que cette dernière solution est largement préférable quand les drivers graphiques le permettent.

## Données 8 bits ou données 16 bits ?

Cette question a occupé une place assez importante dans mes recherches, je vais donc essayer de développer ce que j'ai pu trouver. On va d'abord essayer de voir les conséquences de ce choix en termes de **précision** et de **performances**.

### Précision

#### **\* Pour les données du volume**

En imagerie médicale, les données sont codées le plus souvent sur 8 bits, parfois sur 12 (sachant que les écrans ne peuvent afficher que 8 bits par canal RVB, les données 12 bits doivent être réparties sur aux moins 2 canaux pour pouvoir être représentées). Mais pour qu'une telle précision soit utile encore faut-il que l'erreur due à l'algorithme de reconstruction ne soit pas trop importante.

En effet, disons que l'on cherche à reconstruire des valeurs flottantes entre 0 et 1. Avec 12 bits de précision la différence entre 2 valeurs consécutives est  $1/2^{12} = 2E-4$ . En 8 bits, on tombe à  $1/2^8 = 4E-3$ . Donc si l'erreur moyenne due à l'algorithme de reconstruction est supérieure à  $4E-3$ , l'erreur de précision deviendra marginale. Or, d'après mes calculs l'erreur se situe au mieux à  $8E-3$  pour l'algorithme OSEM-2D, et ce nombre augmente énormément si on tient compte du bruit. Un codage des données sur 12 bits n'apporte donc pas dans ce cas de gain de précision.

#### **\* Pour les données des projections**

Même si on n'utilise que 8 bits pour représenter les données du volume, si l'on mélange ces valeurs dans le cadre d'une *projection*, on va vite déborder de la plage 0-255. Là, on n'aura pas d'autres choix que de passer en précision 16 bits (on retrouve d'ailleurs cette précision de 16 bits dans les fichiers issus d'un scanner médical).

On pourrait cependant faire une normalisation des valeurs **après** le mélange, pour ramener les valeurs sur 8 bits avant de les stocker. Sans doute la perte de précision ne serait pas trop importante, mais d'une part cela complique l'algorithme, et d'autre part cela n'apporte pas de gain de performances. En effet, lorsque les données de projection sont rétroprojetées, la bande passante des textures est peu utilisée (cf. paragraphe précédent), si bien que l'utilisation de textures 16 bits ne provoque pas de ralentissement. J'utilise donc toujours des textures 16 bits pour stocker les projections.

## \* Influence de la précision des données du volume sur la convergence de l'algorithme

On a vu qu'étant donné l'erreur de l'algorithme de reconstruction on pouvait encoder le résultat en 8 bits seulement. Mais peut-être que durant la reconstruction une précision de 16 bits permet à l'algorithme de converger plus vite ou vers un meilleur résultat? Mon programme utilise pour l'instant des *templates* ce qui me permet de tester les 2 solutions.

Pour en savoir plus j'ai fait des tests de reconstruction à partir de phantomes numériques:



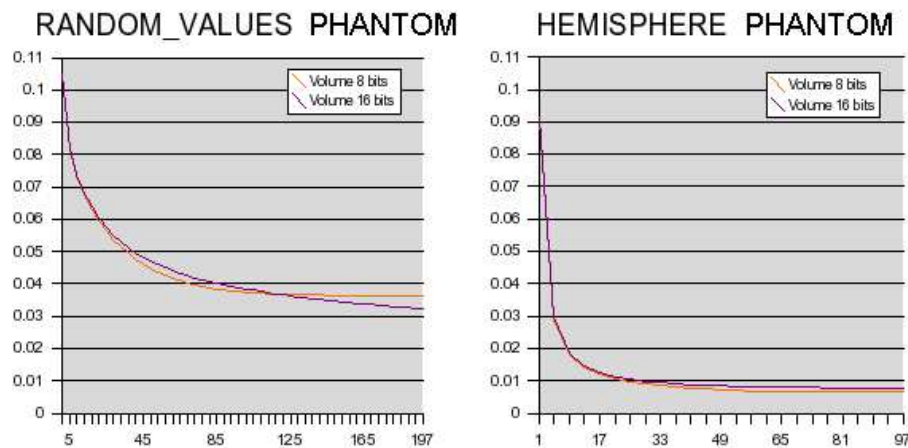
Phantome rempli de valeurs aléatoires.



Phantome représentant des morceaux de sphère imbriqués.

Si on reconstruit ces phantomes à partir de leurs projections, la différence entre le volume initial et le volume final nous donne l'erreur de reconstruction.

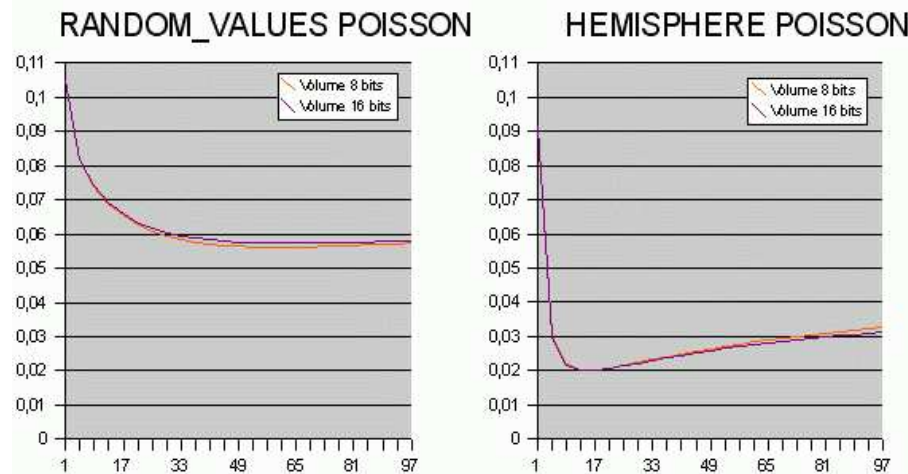
Voici le graphe de l'erreur relative en fonction du nombre d'itérations de l'algorithme, et de la précision des données:



On voit que pour les hémisphères, l'erreur de reconstruction est très faible et à peu près identique quelle que soit la précision. Pour les valeurs aléatoires par contre, l'algorithme utilisant une précision 16 bits converge vers une erreur plus petite.

Note: Il est normal que l'erreur soit plus importante pour les valeurs aléatoires puisque le procédé de reconstruction comporte une petite **erreur spatiale** qui rend difficile la reconstruction de volume à haute variabilité spatiale, mais ce n'est pas vraiment le cas des données médicales.

Ces mesures ont été faites dans des conditions idéales. En pratique, les images médicales contiennent un bruit de type poisson. Si on introduit ce bruit dans la simulation on voit qu'il n'y a plus tellement de différence entre les 2 précisions:



En résumé, une précision de 16 bits n'apporte pas vraiment de meilleur résultat.

## Performances

On a déjà vu que l'opération de rétroprojection prenait le même temps avec des données 16 ou 8 bits.

Pour ce qui est de la projection, on peut regarder la page sur le format des données dans le GPU, et en particulier la partie buffer 16 bits du [graphe 3](#). D'après ce test, le mélange (donc la projection) sera environ 40% plus rapide en 8 bits. Le temps de calcul étant à peu près réparti entre projection et rétroprojection, globalement on ne gagne que **20-25%** sur le temps total de reconstruction, ce qui n'est pas énorme.



## Conclusions

Finalement les critères de performances et de précision ne sont pas déterminants, mais ce ne sont pas les seuls; voici un tableau qui résume les différences:

	Avantages données 8 bits	Avantages données 16 bits
Performances	(+) plus rapide	(x) relativement rapide
Précision	(x) suffisante en l'état actuel de l'algorithme	(+) très bonne
Simplicité	(-) besoin de normaliser les images à différentes étapes de l'algorithme (-) problème des débordements plus fréquents (+) peut être affiché directement sur 1 canal 8bits	(+) avec tout en 16 bits l'algorithme est globalement plus simple (+) correspondance directe avec les fichiers d'images médicales
Mémoire	(+) peu gourmand en mémoire	(x) nécessite le double de mémoire, mais cela n'empêche pas les gros volumes: 32Mo seulement pour un volume $256^3$

Bien sûr, je pourrais garder la possibilité de choisir la précision mais cela complique le code. Je crois qu'à défaut d'avantage décisif il vaut mieux préférer la solution la plus simple, et dans ce cas ça serait d'utiliser des données 16 bits afin d'avoir un pipeline entièrement flottant.

Mais j'attendrais de voir comment se présente la prise en compte de la réponse impulsionnelle du capteur avant de changer le code.

## Perspectives

Dans un premier temps, il y a une phase d'**optimisation** à faire: passer en texture 3D, distribuer les calculs sur les 4 canaux RVBA, et utiliser les *rendus-dans-une-texture* quand c'est possible. Je ne pense pas que cela prendra beaucoup de temps car je connais maintenant bien mon code et les différentes commandes OpenGL utilisées.

Ensuite, essayer d'intégrer la réponse impulsionnelle du capteur. Je pense dans un premier temps essayer d'utiliser des convolutions via *fragment programs* car c'est l'approche qui s'intégrerait le mieux à mon code.

## Références

- [1] [Real-Time Rendering](#), Graphics Hardware - Tomas Akenine-Möller and Eric Haines
- [2] [Analytic and Iterative Reconstruction Algorithms in SPECT](#) - Philippe P. Bruyant
- [3] [Accelerating Popular Tomographic Reconstruction Algorithms On Commodity PC Graphics Hardware](#) - Fang Xu, Klaus Mueller
- [4] [Ultra-fast 3D filtered backprojection on commodity graphics hardware](#) - Fang Xu, Klaus Mueller