LAMEIRA Yannick **M2 Imagina**
GAUTHIER Silvère

Projet :
Proposition automatique de plan de coupe

Introduction

Aujourd'hui, avec les progrès de la 3D, on retrouve des problèmes tels que l'occlusion, l'encombrement visuel . En effet, les coupes sont placées manuellement, ce qui fait que tous les objets ne sont pas visibles suivant certains plans de vue. Si l'objet n'est pas situé à l'avant et à droite, il est considéré comme peu important et sera donc omis. Ce problème se produit dans de nombreux cas, lorsqu'il y a de l'interactivité entre les objets qui sont en dynamique. Il faut donc permettre à tous les objets d'être vus. On peut retrouve cet exemple dans le bâtiment : les murs des bâtiments empêchent de voir ce qui se trouve en intérieur, on peut donc difficilement positionner les objets. Il faut donc permettre aux objets d'importance primaire d'être vus même lorsqu'ils sont cachés par des objets d'importance secondaire.

Beaucoup de personnes se sont intéressées à ce problème et ont proposé des illustrations en coupe qui enlèvent de manière choisies les éléments qui ne sont pas importants. Cependant, les illustrations sont grossières et donnent l'impression d'être coupées imprécisément. De plus, le fait que les coupes soient placées manuellement et non automatiquement et de manière optimale, implique qu'il faut permettre à l'utilisateur de savoir ce qui à été enlevé . La solution au problème serait de réduire au maximum les interventions de l'utilisateur.

Le but final est donc de trouver une solution afin de régler ces soucis. Une coupe automatique permettrait de visualiser efficacement les objets importants tout en gardant le contexte et surtout l'environnement. Il faudrait donc faire en sorte que l'utilisateur puisse choisir un espace mais que l'optimisation soit automatique. Cela permettrait donc d'avoir, quelque soit le point de vue choisi, une coupe dynamique et des objets qui sont en interaction sans que ceux-ci ne soient cachés lorsque la caméra change de plan.

Synthèse d'articles :

Plans de coupe d'adaptation pour faire comprendre de scènes polygonales

Les scènes complexes peuvent contenir beaucoup d'objets, qui sont visibles directement ou qui sont imbriqués dans d'autres objets. Il faut donc pouvoir éviter l'encombrement visuel et pour cela, il existe une méthode permettant l'exposition d'objets cachés tout en gardant les informations proposées par la scène. Certains artistes ont tenté de contourner le problème d'opacité en omettant des parties d'objets secondaires qui bloque le visuel des objets importants. L'objectif de cet article est de créer des coupes de scènes complexes et dynamiques qui pourraient montrer les objets importants sans changer l'environnement. Pour cela, il faut déterminer les régions d'espace 3D qui bloquent les objets d'intérêt. Le second défi de cet article est de permettre la visualisation d'objets qui se trouvent coupés avec des objets secondaires.

Il y a eu des travaux antérieurs ont été réalisés par exemple les rendus volumétriques qui permettent de préserver les volumes rendus, l'opacité et le système entre autres. Les plans de coupe prédéfinis sont influencés par les utilisateurs et sont utilisées dans les applications d'imagerie médicale. Ensuite on retrouve les plan de coupe d'objets définis qui permettent d'atteindre des taux de trame. Pour minimiser les calculs par image le style de coupe enlève l'analyse des structures de scènes et le traitement des objets secondaires. L'article se base sur des recherches utilisant les profondeur d'image. En effet, grâce à des coques arrières qui gardent la forme de l'objet dans des mémoires tampons.

Il existe différentes méthodes de rendu : Pour créer une vu dépendante découpée en rendus il faut 3 étapes :

- la coque arrière de l'objet principal est dans un tampon de profondeur. Cela est ensuite traité pour rendre une représentation de l'image de profondeur d'une surface de coupe.
- Les objets secondaires sont rendus et coupés contre l'image de profondeur de coupe et les objets principaux sont rendus complètement.
- Les lignes sont combinées pour donner des informations supplémentaires sur la forme des surfaces de coups et des structures d'objets qui sont partiellement coupés.

Le problème de cela est que le matériel informatique ne supporte pas les tests de tampon en profondeur. Il faut donc comparer la profondeur d'un fragment avec la valeur de profondeur de coupe. Il peut être réglé grâce à un écrêtage partiel qui permet de voir l'intérieur des objets.

Le rendu permet également de créer des impressions solides sur les objets. En effet, il y a des arêtes sur les objets qui lui donnent, de loin, un rendu sculpté. Le calcul de la surface de la coupe est réalisé par un algorithme qui fonctionne en O(n2 log n) fois. Il permet de prendre les objets d'intérêt mis dans les mémoires tampons , de coder les coordonnées de l'espace et de la profondeur sur tous les pixels. On peut également définir les angles de coupe de chaque pixel, afin de mieux positionner les éléments.

Cependant, il existe plusieurs limites :

- les objets importants qui sont hors de l'écran ne sont pas intégrés dans la surface de coupe. Cela entraîne des discontinuité lorsque l'objet est interactif. Pour régler ce souci, il faudrait utiliser un tampon de coupe plus grand que le tampon de mémoire, cela prendrait donc en compte la s ;ortie des objets de l'écran mais augmenterait le temps de traitement et la taille de la mémoire.
- Il existe également des problèmes de coupe de plancher lorsqu'il y a des objets sur ce

plancher.
Les résultats obtenus sont accompagnés par des vidéos. L'algorithme est intéressant car il n'impose pas de frais généraux en plus. Il ne repose pas non plus sur les informations structurelles de la scène.

Illustrations tronqués intelligemment

Introduction : Il est difficile de localiser et visualiser des caractéristiques importantes et cela devient un problème lorsque celles-ci ne sont pas situées à droite où devant. De ce fait, une bonne coupe d'illustration doit répondre à plusieurs exigences : maximisation de la visibilité des caractéristiques importantes, respect des spécification de l'utilisateur et la forme de la coupe doit être compréhensible et l'utilisateur doit pouvoir comprendre quelles parties ont été omises.

Il existe différentes techniques pour produire des images de coupe à partir de maillages polygonaux ou de données volumétrique, qui permettent de faire apparaître les données importantes. Il existe beaucoup d'approches, on retrouve par exemple l'approche proposée par Correa, Ma, Wu et Qu qui a permis d'optimiser la fonction de transfert qui permet une meilleure visibilité et une meilleure qualité des termes. Ou encore celle proposée par Pelizzari et McGuffin, on retrouve les spécifications interactives des plans de coupe ou le positionnement manuel de ceux-ci.

La méthode qui est proposée dans cet article consiste à créer une étape interactive où l'utilisateur va sélectionner des parties intéressantes de données basées sur des idées ou des connaissances. Ensuite on aura une étape non interactive où l'ordinateur détectera automatiquement l'emplacement optimal pour la coupe en se basant sur le choix de l'utilisateur. L'utilisateur doit définir une valeur pour chaque voxel, une fois fait, l'ordinateur fait le rapprochement entre meilleure taille et position des primitives grâce à des étapes. Le processus itératif d'optimisation est répété afin de maximiser la mesure de qualité. Grâce au choix de l'utilisateur, le système produit une image qui code les pixels et montre les caractéristiques importantes. Cette approche à plusieurs avantages : les formes découpées sont moins complexes et les paramètres tels que l'ombrage et la transparence sont préservés.

L'algorithme recherche l'optimum dans l'espace de configuration discrétisé (chaque zone est divisée en N morceaux et l'espace n'est pas influencé par la complexité des données). La fonction objectif doit prendre en compte la position de la caméra et mesurer la qualité des solutions. Pour mieux trouver cette fonction, les données sont colorées différemment selon leur importance et sont projetées su le plan, ce qui donne une image transformée en numéro unique. Cette image est la moyenne des valeurs d'une couche de couleur sous tous les pixels.
Simulated Annealing (SA) permet de résoudre les problèmes d'optimisation complexes. Il est applicable à des problèmes généraux. L'algorithme gère aussi les mouvements de caméra et l'exploration des données en fonction du temps.

Il y a eu plusieurs problèmes avec l'algorithme : l'évaluation est la partie qui est souvent exécutée par l'algorithme, il y a donc eu un problème d'optimisation rapide. Il fallait donc éviter l'envoi d'image en calculant les valeurs moyennes directement par le processeur graphique (GPU).

Les algorithmes comme le SA sont basés sur le hasard, il est donc très important de les évaluer. Pour faire cela, un processus d'optimisation naïf a été mis en œuvre. Les valeurs obtenues par SA et ce processus ont été comparées .
Pour mesurer la convergence de l'algorithme vers une configuration fixe, il fallait calculer le chevauchement des primitives. Le résultat d'une prise de mesure est représenté en figure 9 et il permet de voir que l'algorithme converge vers la solution optimale.

Résultats

Voici les deux algorithmes utilisés ici. Pour la translation, l'utilisateur doit faire un clic gauche à l'aide de la souris sur un voxel de la ROI. Pour la rotation, l'utilisateur devra effectuer un clic droit.

Translation

```
|
| tmp ← valeur du voxel à la position du clic gauche souris
| count ← 0
| bestCount ← 0
|
| Pour chaque translation tx
| | count ← nombre de voxel = tmp dans le plan composés de tous les points avec abscisse = tx
| | si count > bestCount
| | | alors garder en mémoire tx comme meilleure translation
| | fin si
| fin pour
|
| Pour chaque translation ty
| | count ← nombre de voxel = tmp dans le plan composés de tous les points avec ordonnée = ty
| | si count > bestCount
| | | alors garder en mémoire ty comme meilleure translation
| | fin si
| fin pour
|
| Pour chaque translation tz
| | count ← nombre de voxel = tmp dans le plan composés de tous les points avec profondeur = tz
| | si count > bestCount
| | | alors garder en mémoire tz comme meilleure translation
| | fin si
| fin pour
|
| Renvoyer le plan correspondant à la meilleure translation
|
```

fin algorithme

Rotation

```
|
| tmp ← valeur du voxel à la position du clic gauche souris
| count ← 0
| bestCount ← 0
|
| Pour chaque rotation {rx, ry, rz}
| | currentPlan ← points dont les coordonnées correspondent au plan actif vu par l'utilisateur
| | currentPlan ← rotation de degré rx autour de l'axe x de currentPlan
| | currentPlan ← rotation de degré ry autour de l'axe y de currentPlan
| | currentPlan ← rotation de degré rz autour de l'axe z de currentPlan
| | count ← nombre de voxel = tmp dans currentPlan
| | si count > bestCount
| | | alors garder en mémoire {rx, ry, rz} comme meilleure rotation
| | fin si
| fin pour
|
| Renvoyer le plan correspondant à la meilleure rotation
|
```

fin algorithme

Explications :

Pour le choix du meilleur plan de coupe, nous nous basons sur la valeur des voxels. En effet, l'utilisateur peut cliquer sur l'image afin de définir un voxel d'intérêt sur lequel les algorithmes se baseront. Une fois ce voxel sélectionné, sa valeur et sa position sont récupérée. On peut ensuite définir de nouveaux plans de coupes, soit par translation, soit par rotation autour du point choisi. Parmis tous ces plans, nous allons en choisir un, qui sera affiché dans une seconde fenêtre.

Soit la valeur du voxel sélectionné « value », le plan retourné à l'utilisateur sera le plan ayant le plus de voxels compris entre « value - sigma » et « value + sigma », sigma étant un coeffiscient donné en paramètre des fonctions et définissant la taille de l'intervalle de valeurs possibles pour notre comparaison.

La translation est ici un parcours de tous les plans pouvant être visualisés par l'utilisateur. On choisit ainsi automatiquement le plan qui pourrait avoir le plus d'intérêt pour celui-ci. La translation ainsi effectuée est affichée dans une nouvelle fenêtre et les coordonnées du plan affiché sont données dans la console pour que l'utilisateur puisse le retrouver au besoin par parcours manuel.

Pour améliorer le choix du plan, on pourrait ajouter une contrainte de connexité avec le voxel cliqué. Mais ici, nous risquerions de trop augmenter la complexité donc nous n'exploiterons pas cette solution.

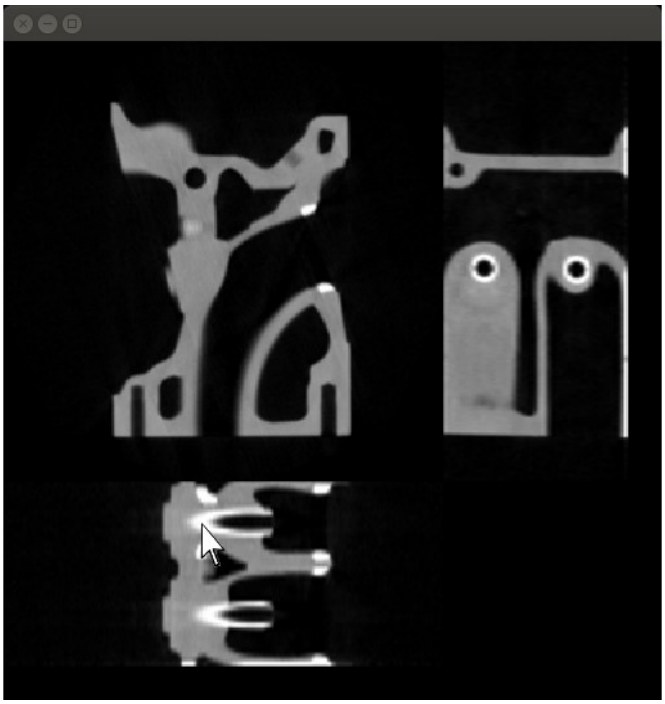
La rotation, quant à elle, se base autour du point cliqué. En effet, nous effectuons trois rotations successives, respectivement en abscisses, ordonnées et profondeurs, autour de ce point. Pour cela, nous devons changer de repère : sinon nous tournerions autour du point {0,0,0}. Après translation du repère, nous réalisons les rotations et estimons la meilleure par comparaison des points du plan de rotation avec la valeur du voxel cliqué.

Enfin, nous reproduisons le meilleur plan dans une nouvelle fenêtre en repassant dans le repère de centre {0,0,0}. Attention, ce plan n'a pas toujours la même taille, il faut alors prévoir plus grand ou calculer cette taille, ce qui est complexe et inutile ici.

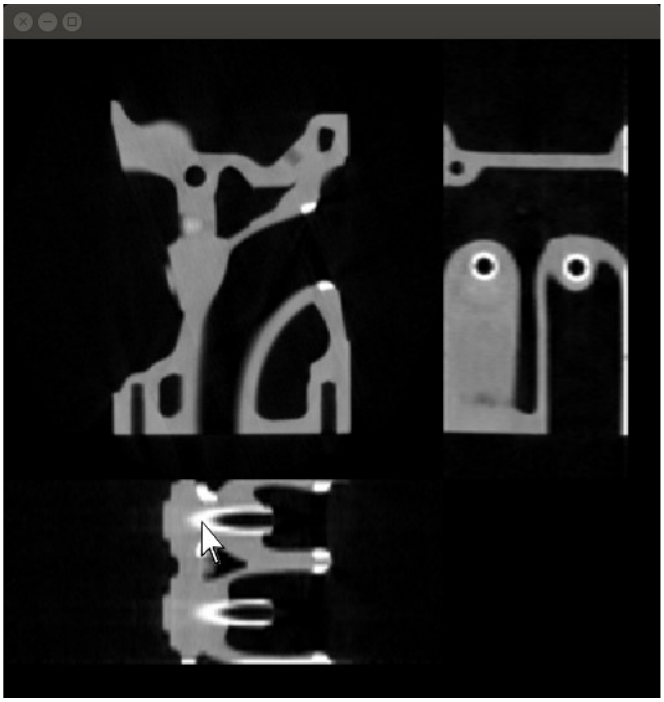
Pour améliorer tout cela, il faudrait réduire le pas (en degrés) de la rotation (ici 30°) ainsi que prévoir une rotation autour de plusieurs points d'une sphère englobant le voxel cliqué. Ce qui augmente significativement la complexité, c'est pour cela que nous n'irons pas plus loin vu nos contraintes de puissance de calcul et de temps.

Résultats :

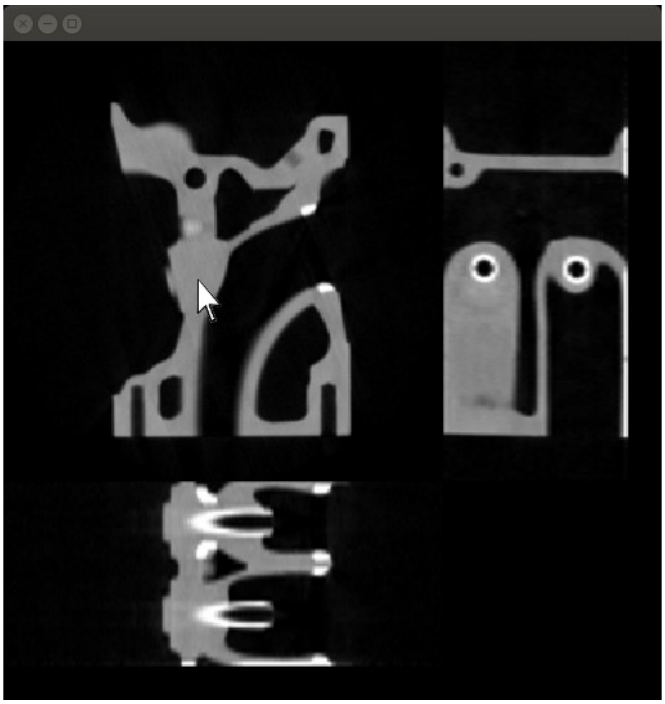
Voici quelques résultats sur le fichier engine.hdr du site personnel de Monsieur Subsol.



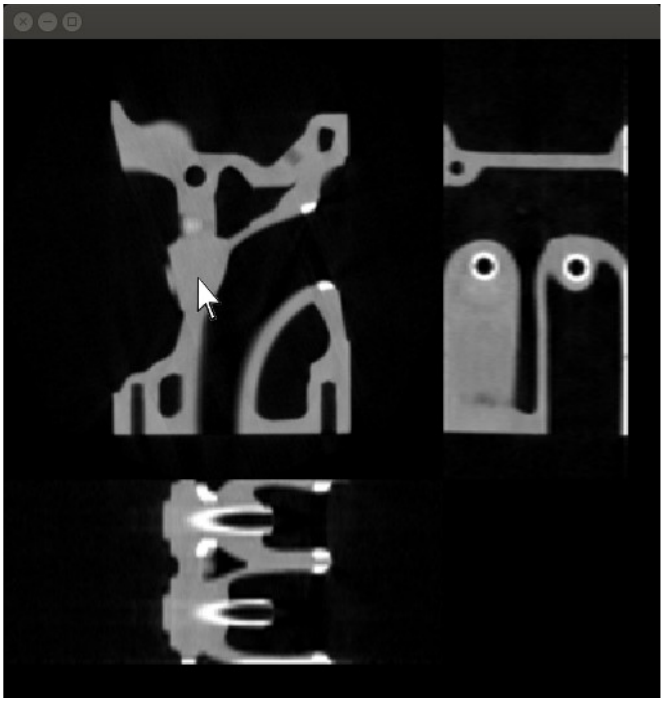
Meilleure translation : 0, 0, 106



Meilleure rotation : 60, 60, 120

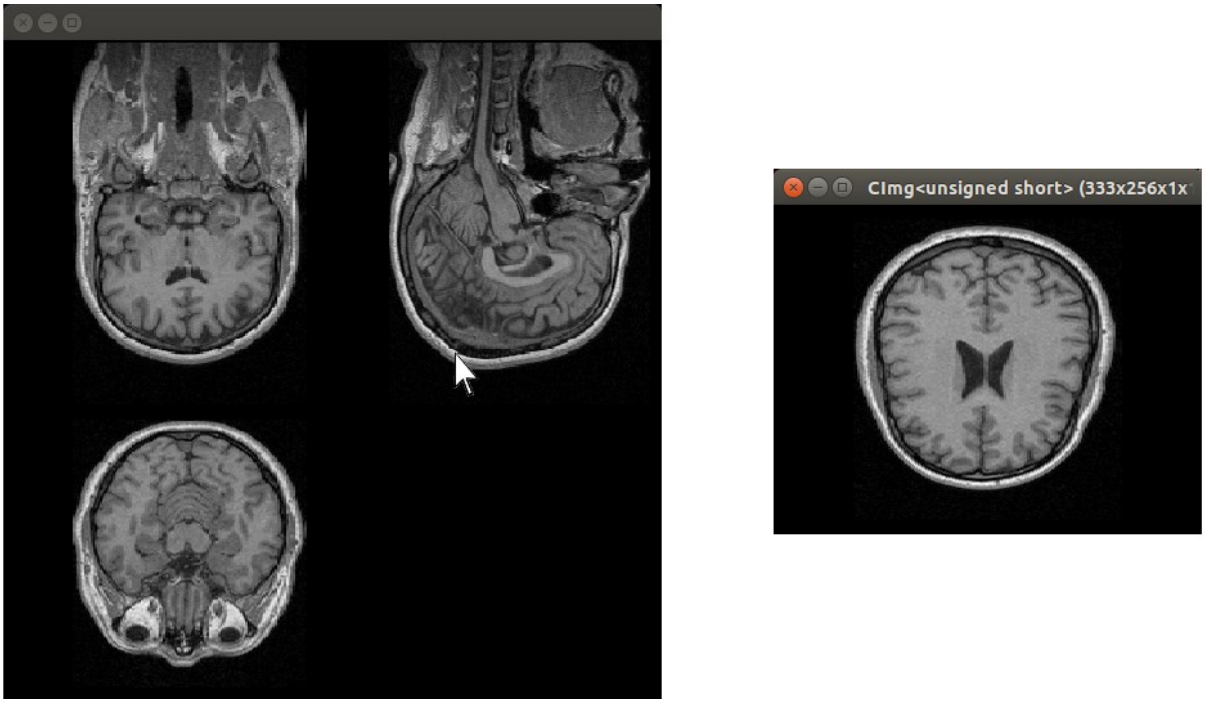


Meilleure translation : 0, 0, 1

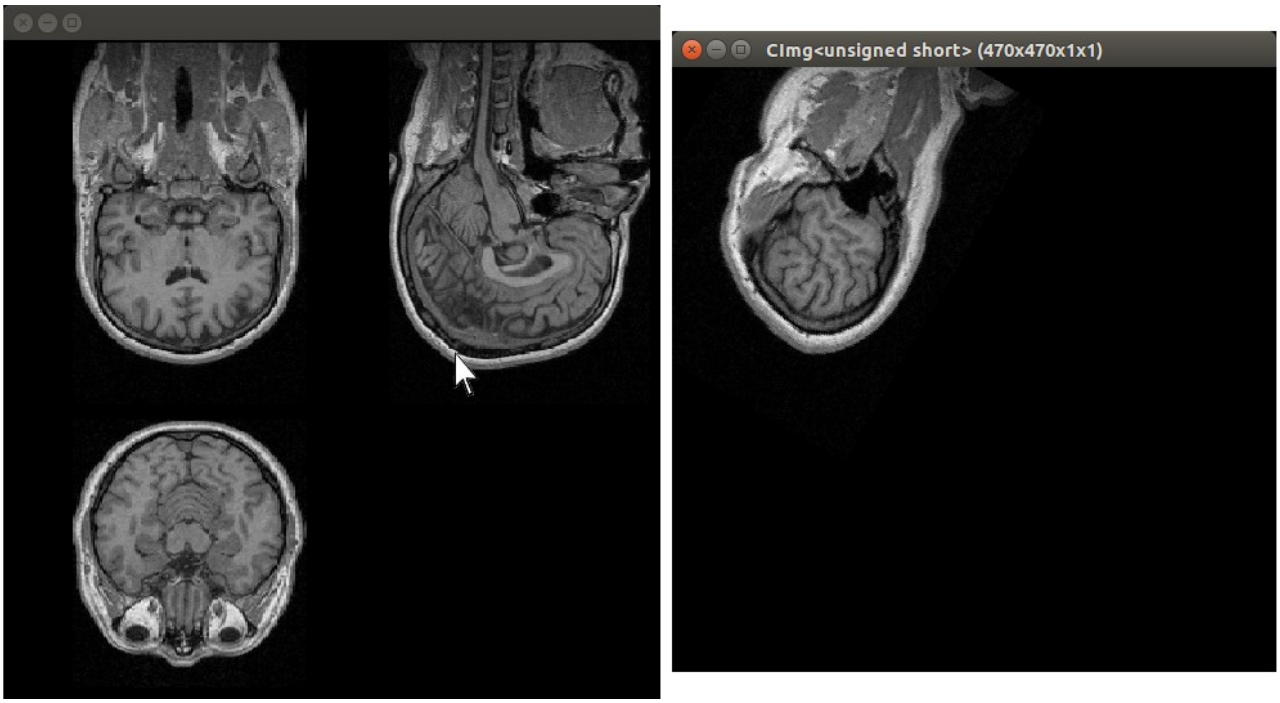


Meilleure rotation : 150, 150, 120

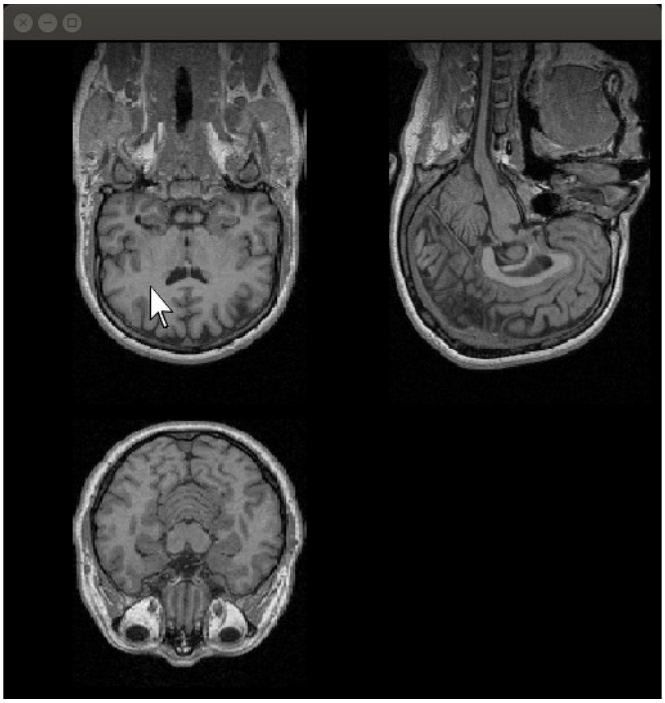
Voici quelques résultats sur le fichier M.R.-head_Coronal.hdr du site personnel de Monsieur Subsol.

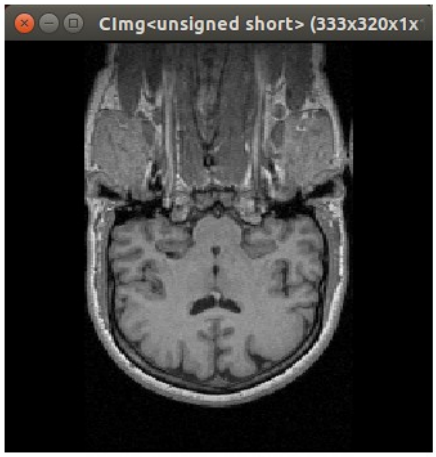


Meilleure translation : 0, 206, 0

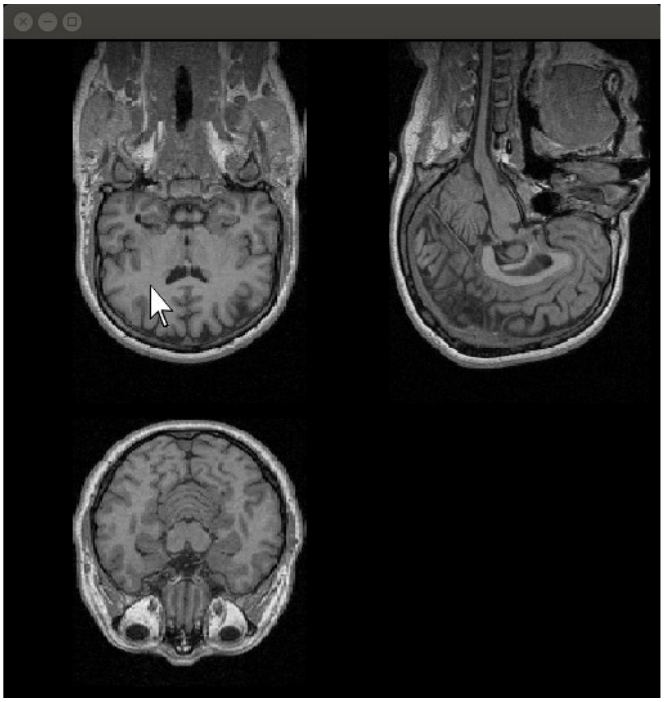


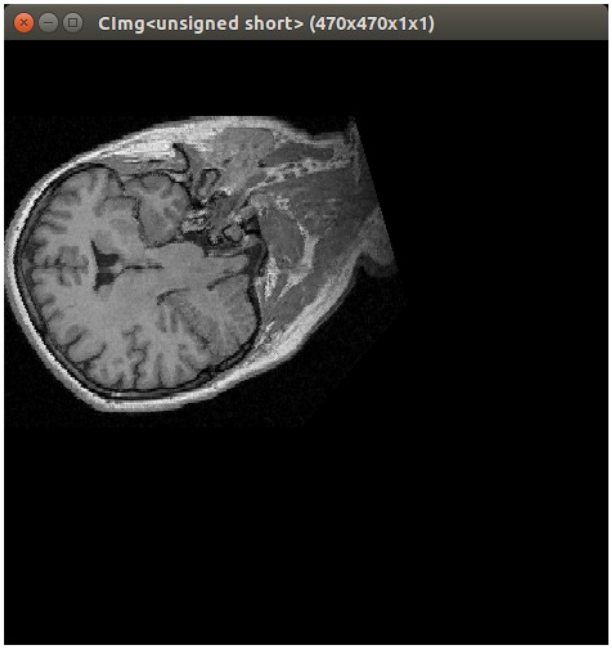
Meilleure rotation : 90, 120, 90





Meilleure translation : 0, 0, 122





Meilleure rotation : 150, 150, 90