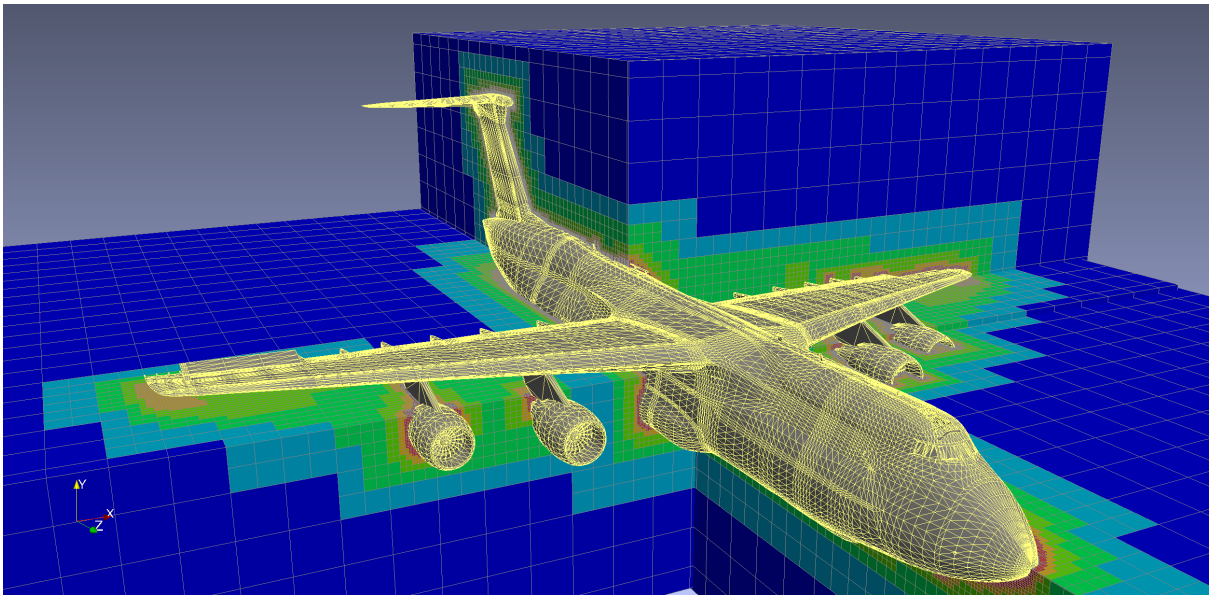
Localisation géométrique rapide
par arbre octal

La librairie libOL



Loïc MARÉCHAL / INRIA, Projet Gamma

Février 2024

Document v1.52, librairie v1.82

Table des matières

Couverture : Maillage octree d'un Boeing 747 réalisé par Thomas Døhlen.

1 Introduction

Le but de cette librairie est de facilement et rapidement retrouver certains éléments d'un maillage en effectuant des requêtes géométriques, telles que trouver le triangle le plus proche d'un point donné, ou bien de fournir la liste des triangles ou sommets inclus dans une zone rectangulaire donnée.

D'autres entités que les triangles ou sommets sont gérés tels que les arêtes, les quadrilatères et tétraèdres.

L'utilisateur a le moyen de définir des fonctions filtres afin de contraindre les critères de recherches à son propre contexte.

Une des principales forces de cette librairie est sa très grande efficacité tant en termes de mémoire, les données utilisateur ne sont pas dupliquées et l'arbre est encodé sous forme binaire, que de temps de calcul, l'appel des requêtes étant *thread-safe* et peut donc être appelé en parallèle.

2 Utilisation

La donnée d'entrée à fournir à la librairie est un maillage de surface ou de volume sous sa forme la plus simple : une liste des coordonnées de chaque sommet et une table des numéros des sommets pour chaque élément. À partir de ce maillage, la librairie génère un octree automatiquement raffiné de telle sorte que chaque octant ne contienne au plus que 20 entités de chaque type afin d'accélérer les opérations de recherche. Le temps de construction et la mémoire requis sont donc dépendants du nombre d'entités du maillage et de la forme de la géométrie. Un octree étant fondamentalement isotrope, il aura tendance à être fortement raffiné, et donc à occuper beaucoup de mémoire, dans le cas de géométries très minces (anisotropes).

Bien qu'une structure octree ne contienne qu'un seul maillage, il est possible d'en créer plusieurs, ceux-ci étant différenciés par une étiquette unique retournée à la création.

Supposons que nous aillons un maillage de *NmbVer* sommets, stockés dans la table *VerCrd* et *NmbTri* triangles dans la table *TriTab* et que l'on souhaite trouver le triangle le plus proche du point de coordonnées $\{0.5, 2.3, -6.0\}$ ainsi que la liste des triangles contenus partiellement ou totalement dans la région cubique centrée en $\{2, 2, 10\}$ et de taille 2. Le déroulement est le suivant :

```
int64_t OctIdx;
int IntersectedTriangles[10];
int NmbVer, NmbTri, (*TriTab)[3];
double (*CrdTab)[3], VerCrd[3]={0.5, 2.3, -6.0};
double BoxMin[3]={1,1,9}, BoxMax[3]={3,3,11}, MinDis;
...
(allocation et lecture du maillage)
...
```

```
OctIdx = LolNewOctree(
         NmbVer, VerTab[1], VerTab[2],
              0,      NULL,      NULL,
         NmbTri, TriTab[1], TriTab[1],
              0,      NULL,      NULL,
              0,      NULL,      NULL,
              0,      NULL,      NULL,
              0,      NULL,      NULL,
              0,      NULL,      NULL,
              1, 1);

printf("l'Octree numéro %d a été construit\n", OctIdx);

TriIdx = LolGetNearest(OctIdx, TypTri, VerCrd, &MinDis, 0, NULL, NULL, 0);
printf("le triangle le plus proche de 0.5, 2.3, -6.0 est : %d\n", TriIdx);

NmbBoxTri = LolGetBoundingBox(OctIdx, TypTri, 10, TriTab, BoxMin, BoxMax, 0);
for(i=0;i<NmbTri;i++)
    printf("triangle numéro : %d\n", TriTab[i]);

LolFreeOctree(OctIdx);
```

Il est à noter que l'intersection d'un triangle avec une boîte est calculée de manière complète, c'est-à-dire qu'il y a intersection si l'un des sommets du triangle tombe dans la boîte, ou qu'une de ses arêtes intersecte une face de la boîte ou encore que le plan du triangle intersecte une arête de la boîte. Certains octrees simplifiés se contentent de tester si un sommet de triangle est contenu dans la boîte, l'approche de la libOL est plus consommatrice de ressource, mais est géométriquement exact.

La vitesse de création est d'environ 2.000.000 de sommets ou 200.000 triangles par seconde et la consommation mémoire de 30 octets par sommet ou 240 octets par triangle.

La librairie étant constituée d'un seul fichier `liboctree.c` et d'un fichier de définitions `liboctree.h` pour le C et `liboctree.ins`.

Note : par défaut la librairie utilise des entiers codés sur 32 bits, mais il est possible de les étendre à 64 bits en passant le paramètre `-Di8` au compilateur.

3 Liste des commandes

3.1 LolFreeOctree

Libère un octree et toute la mémoire associée à cette instance. Cela ne clôture pas la librairie qui peut continuer à allouer d'autres instances.

Syntaxe

```
mem = LolFreeOctree(OctIdx);
```

Commentaires

Retourne la mémoire totale utilisée en octets.

3.2 LolGetBoundingBox

Retourne une liste d'entités de maillage d'un type donné et inclus dans une boîte englobante.

Syntaxe

```
NmbTri = LolGetBoundingBox(
         OctIdx, typ, MaxTri, TriTab,
         BoxMin[3], BoxMax[3], ThrIdx);}
```

Paramètres

Paramètre	type	description
OctIdx	int64_t	index de l'octree retourné par LolNewOctree
typ	int	type d'entité à rechercher : LolTypVer, LolTypEdg, LolTypTri, LolTypQad ou LolTypTet
MaxTri	int	nombre maximum d'éléments que la table suivante peut contenir
TriTab	int *	pointeur sur une table qui contiendra la liste des éléments contenus dans cette boîte englobante
BoxMin	double [3]	coordonnées du coin inférieur de la boîte englobante
BoxMin	double [3]	coordonnées du coin supérieur de la boîte englobante
ThrIdx	int	numéro du thread ou process appelant ou 0 dans le cas séquentiel

Retour	type	description
index	int	retourne le nombre d'entités contenues dans la boîte

Exemple

Alloue une table de 10 triangles et demande à la librairie de retourner les 10 premiers triangles intersectant le cube de coordonnés $\{0,0,0\} - \{1,1,1\}$ et affiche leur numéro.

```
int TriTab[10];
double box[2][3]={{0,0,0}, {1,1,1}};
NmbTri = LolGetBoundingBox(OctIdx, TypTri, 10, TriTab, box[0], box[1], 0);
for(i=0;i<NmbTri;i++)
    printf("triangle numéro : %d\n", TriTab[i]);
```

3.3 LolGetNearest

Recherche l'entité de type spécifiée la plus proche d'une coordonnée géométrique.

Syntaxe

```
index = LolGetNearest(
        OctIdx, typ, crd, PtrMinDis, MaxDis,
        UsrPrc, UsrDat, ThrIdx);}
```

Paramètres

Paramètre	type	description
OctIdx	int64_t	index de l'octree retourné par LolNewOctree
typ	int	type d'entité à rechercher : LolTypVer, LolTypEdg, LolTypTri, LolTypQad ou LolTypTet
crd	double [3]	coordonnées du vertex de référence
PtrMinDis	double *	pointeur sur une valeur dans laquelle la distance à l'entité la plus proche sera retournée
MaxDis	double	limiter la recherche à des entités distance au maximum de MaxDis du point de référence (0 si aucune limite n'est souhaitée)
UsrPrc	int ()(void *, int)	pointeur sur une routine de filtrage prenant en entré un pointeur sur les données utilisateur et un entier contenant l'entité en cours de test et qui retourne un entier booléen : 0 pour exclure du test et 1 pour effectuer le test
UsrDat	void *	pointeur sur les données de l'utilisateur qui sera transmis à la routine de test
ThrIdx	int	numéro du thread ou process appelant ou 0 dans le cas séquentiel

Retour	type	description
index	int	retourne l'index de l'entité la plus proche du sommet de référence fourni ou 0 en cas d'erreur

Commentaires

Il est tout à fait possible de donner un point de référence en dehors de la boîte englobante de l'octree qui est taillé au plus juste autour de l'objet fourni en entrée. Plus le point est éloigné du triangle le plus proche, plus le temps de recherche sera long.

La routine de filtrage optionnelle est appelée en aveugle avec comme premier paramètre le pointeur `UsrDat` fourni par l'utilisateur en tant que `void *`, et le numéro de l'entité de maillage que la librairie souhaite évaluer au cours d'une requête. Le rôle de cette routine est de retourner un résultat booléen, 0 pour refuser de considérer cette entité dans la liste à tester et 1 pour l'envoyer au test.

L'index `ThrIdx` est un identifiant unique du process ou thread de l'utilisateur qui appellent la procédure de recherche et dont le numéro doit être compris entre 0 et $MaxThr-1$ où $MaxThr$ est le nombre maximum de processus pouvant appeler les fonctions de la librairie de manière concurrente.

L'exemple suivant génère un octree autour d'un maillage de quatre sommets et deux triangles et cherche lequel des deux est le plus proche du point d'origine $\{0, 0, 0\}$.

Exemple

```
double crd[5][3] = { {2,-3,5.2}, {3.4,6,8.2}, {5,1,3}, {3,4,1}, {0,0,0} };
double MinDis;
int tri[2][3] = { {1,2,3}, {2,3,4} };
OctIdx = LolNewOctree(4, crd[0], crd[1], 2, tri[0], tri[1], 0);
TriIdx = LolGetNearest(OctIdx, TypTri, crd[4], &MinDis, 0, NULL, NULL, 0);
printf("le triangle le plus proche de l'origine est %d\n", TriIdx);
```

3.4 LolIntersectSurface

Recherche le premier triangle intersecté par le vecteur donné. Cette routine, qui ne fonctionne pour l'instant qu'avec les triangles, permet de rechercher une entité proche, mais par lancer de rayon plutôt que par distance comme LolGetNearest().

Syntaxe

```
index = LolIntersectSurface(
        OctIdx, crd, tng, PtrMinDis, MaxDis,
        UsrPrc, UsrDat, ThrIdx);}
```

Paramètres

Paramètre	type	description
OctIdx	int64_t	index de l'octree retourné par LolNewOctree
crd	double [3]	coordonnées du point d'origine du vecteur
tng	double [3]	tangente du vecteur
PtrMinDis	double *	pointeur sur une valeur dans laquelle la distance à l'entité la plus proche sera retournée
MaxDis	double	limiter la recherche à des entités distance au maximum de MaxDis du point de référence (0 si aucune limite n'est souhaitée)
UsrPrc	int ()(void *, int)	pointeur sur une routine de filtrage prenant en entré un pointeur sur les données utilisateur et un entier contenant l'entité en cours de test et qui retourne un entier booléen : 0 pour exclure du test et 1 pour effectuer le test
UsrDat	void *	pointeur sur les données de l'utilisateur qui sera transmis à la routine de test
ThrIdx	int	numéro du thread ou process appelant ou 0 dans le cas séquentiel

Retour	type	description
index	int	retourne l'index de l'entité la plus proche intersecté par le vecteur fourni ou 0 en cas d'erreur

Commentaires

Il est tout à fait possible de donner un point de référence en dehors de la boîte englobante de l'octree qui est taillé au plus juste autour de l'objet fourni en entrée. Plus le point est éloigné du triangle le plus proche, plus le temps de recherche sera long.

La routine de filtrage optionnelle est appelée en aveugle avec comme premier paramètre le pointeur `UsrDat` fourni par l'utilisateur en tant que `void *`, et le numéro de l'entité de maillage que la librairie souhaite évaluer au cours d'une requête. Le rôle de cette routine est de retourner un résultat booléen, 0 pour refuser de considérer cette entité dans la liste à tester et 1 pour l'envoyer au test.

L'index `ThrIdx` est un identifiant unique du process ou thread de l'utilisateur qui appellent la procédure de recherche et dont le numéro doit être compris entre 0 et $MaxThr-1$ où $MaxThr$ est le nombre maximum de processus pouvant appeler les fonctions de la librairie de manière concurrente.

3.5 LolNewOctree

Syntaxe

```
retour = LolNewOctree(
         NmbVer, pVer1, pVer2,
         NmbEdg, pEdg1, pEdg2,
         NmbTri, pTri1, pTri2,
         NmbQad, pQad1, pQad2),
         NmbTet, pTet1, pTet2,
         NmbPyr, pPyr1, pPyr2,
         NmbPri, pPri1, pPri2,
         NmbHex, pHex1, pHex2,
         BasIdx, MaxThr);}
```

Paramètres

Paramètre	type	description
NmbVer	int	nombre de sommets à insérer dans l'octree
pVer1	double *	pointeur sur la table des coordonnées du premier sommet du maillage
pVer1	double *	pointeur sur la table des coordonnées du second sommet du maillage
NmbEdg	int	nombre d'arêtes à insérer dans l'octree
pEdg1	int *	pointeur sur la table des indices de sommets de la première arête du maillage
pEdg2	int *	pointeur sur la table des indices de sommets de la seconde arête du maillage
NmbTri	int	nombre de triangles à insérer dans l'octree
pTri1	int *	pointeur sur la table des indices de sommets du premier triangle du maillage
pTri2	int *	pointeur sur la table des indices de sommets du second triangle du maillage
NmbQad	int	nombre de quads à insérer dans l'octree
pQad1	int *	pointeur sur la table des indices de sommets du premier quad du maillage
pQad2	int *	pointeur sur la table des indices de sommets du second quad du maillage

NmbTet	int	nombre de tétraèdres à insérer dans l'octree
pTet1	int *	pointeur sur la table des indices de sommets du premier tétraèdre du maillage
pTet2	int *	pointeur sur la table des indices de sommets du second tétraèdre du maillage
NmbPyr	int	nombre de pyramides à insérer dans l'octree
pPyr1	int *	pointeur sur la table des indices de sommets de la première pyramide du maillage
pPyr2	int *	pointeur sur la table des indices de sommets de la seconde pyramide du maillage
NmbPri	int	nombre de prismes à insérer dans l'octree
pPri1	int *	pointeur sur la table des indices de sommets du premier prisme du maillage
pPri2	int *	pointeur sur la table des indices de sommets du second prisme du maillage
NmbHex	int	nombre d'hexaèdres à insérer dans l'octree
pHex1	int *	pointeur sur la table des indices de sommets du premier hexaèdre du maillage
pHex2	int *	pointeur sur la table des indices de sommets du second hexaèdre du maillage
BasIdx	int	index de l'indice de début de chaque table : 0 ou 1
MaxThr	int	nombre maximum de processus ou thread pouvant effectuer des requêtes de manière concurrente sur cet octree : vaut 1 ou plus

Retour	type	description
index	int	retourne l'index d'un octree à fournir aux commandes de la librairie ou 0 en cas d'erreur

Commentaires

Si vous avez du mal à jongler avec les déclarations de tableaux multidimensionnels, plutôt cryptiques, du C, vous pouvez passer un simple tableau unidimensionnel de double (double *), où VerTab[0] est la coordonnée x_1 du premier vertex, VerTab[1] est y_1, VerTab[2] est z_1, VerTab[3] est x_2 et ainsi de suite. Notez que la mémoire prise par l'octree dépend aussi du paramètre MaxThr, car une petite quantité de mémoire de travail est allouée pour chaque thread (plus 10%). De plus, si vous compilez la librairie en passant l'option `-DWITH_FAST_MODE` au compilateur, la structure octree occupera 2,5 fois plus de mémoire, mais toutes les requêtes seront 35% plus rapides.

L'exemple suivant génère un octree autour d'un maillage de quatre sommets et deux triangles.

Exemple

```
double crd[4][3] = { {2,-3,5.2}, {3.4,6,8.2}, {5,1,3}, {3,4,1} };
int tri[2][3] = { {1,2,3}, {2,3,4} };
OctIdx = LolNewOctree(
         4, crd[0], crd[1],
         0, NULL, NULL,
         2, tri[0], tri[1]);
         0, NULL, NULL,
         0, NULL, NULL,
         0, NULL, NULL,
         0, NULL, NULL,
         0, NULL, NULL,
         0, 1);
```

3.6 LolProjectVertex

Projection géométrique et topologique d'un point sur une entité de maillage. La distance et le projeté sur une arrête, un triangle, un quadrilatère ou un tétraèdre est calculé, ainsi que la nature de ce projeté, si le projeté se confond exactement avec le sommet ou l'arrête d'un triangle par exemple.

Syntaxe

```
statut = LolProjectVertex(OctIdx, VerCrd, SrfTyp, SrfIdx, PrjCrd, ThrIdx);
```

Paramètres

Paramètre	type	description
OctIdx	int64_t	index de l'octree retourné par LolNewOctree
VerCrd	double [3]	coordonnées du vertex à projeter
SrfTyp	int	type d'entité sur laquelle projeter le vertex : LolTypVer, LolTypEdg, LolTypTri, LolTypQad ou LolTypTet
SrfIdx	int	index de l'entité de surface
PrjCrd	double [3]	pointeur sur les coordonnées qui recevront la projection
ThrIdx	int	numéro du thread ou process appelant ou 0 dans le cas séquentiel

Retour	type	description
statut	int	0 : erreur, 1 : projection confondue avec un vertex, 2 : idem avec une edge, 3 : projection à l'intérieur du triangle ou quadrilatère