Smile =) Manuel de référence

0.7.5.pre

Généré par Doxygen 1.3-rc3

Sat May 24 23 :50 :16 2003

Table des matières

1 Smile

```
SMILE README
------------

        Smile 0.7.5-pre


-------------------------------------------------------------------------------
 -> Pour toutes questions vous pouvez me contacter  chavent@imerir.com.
    Vos remarques et suggestions sont bienvenues. Merci.
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
        Rpertoires :
-------------------------------------------------------------------------------

 ./src/smileCORE : c'est smile =) .
 ./src/smileTEST : permet de tester le core.
 ./src/smileGUI  : ce sera la gui avance (pas encore utilisable svp).
 ./src/smileMAF  : c'est le module d'affichage.
 ./src/smileMOP  : c'est le module de traduction.

 ./doc/html      : vous trouverez la doc qui dcrit les classes.

 ./lib           : rpertoire o est install la librairie smile.

 ./bin           : rpertoire o sont installs les excutables.

 ./include       : rpertoire o sont places les enttes necessaire
                   au dveloppement.

-------------------------------------------------------------------------------
        Avertissement :
-------------------------------------------------------------------------------

 * Cette version n'est pas totalement stable.

 * Les essais ont t effectus sur Debian 3.0 et Slackware 9.0.
```

```
--------------------------------------------------------------------------------
        Prrequis
--------------------------------------------------------------------------------

 * make, g++ (test avec le 2.95),
 * ode pour le core ( fonctionne avec la version cvs de ode du 08/05/2003),
 * opengl et glut pour la gui de base,
 * gtk+-2.0 pour la gui avance (pas encore au point).


--------------------------------------------------------------------------------
        Construction/Installation :
--------------------------------------------------------------------------------

 >cd src
 >make clean CORE TEST MAF

--------------------------------------------------------------------------------
        Utilisation :
--------------------------------------------------------------------------------

 >cd bin
 >export LD_LIBRARY_PATH=../lib:$LD_LIBRARY_PATH
 >./testPerfCom &
 >./smileMAF


--------------------------------------------------------------------------------
        Doc :
--------------------------------------------------------------------------------

 * Accessible dans ./doc/html.

--------------------------------------------------------------------------------
        Pour les dveloppeurs :
--------------------------------------------------------------------------------

 La doc est gnre par doxygen. Un fichier d'exemple
 permet de rappeler les bases de son utilisation
 (doc/exemple.h).

SMILE CHANGELOG
---------------



--------------------------------------------------------------------------------
        0.7.5
--------------------------------------------------------------------------------

19/05/2003 Paul

        * Makefile : ajout d'un fichier de config (pour ajuster dbug/release)

15/05/2003 Paul

        * Joint.cpp :
          ajout de la methode lier.
        * Ccylindre.cpp :
          remplacement du parametre rayon par le parametre diametre.

--------------------------------------------------------------------------------
        0.7.4
--------------------------------------------------------------------------------

12/05/2003 Paul
```

```
        * Univers.cpp :
          Correction du bug lors de la destruction d'un Univers.
        * Client.cpp Serveur.cpp :
          Corrig le bug qui empechait une bonne communication
          en localhost sur les machines qui compilaient avec g++-2.95.

11/05/2003 Paul

        * Serveur.cpp :
          Mise en place des itrateurs.

10/05/2003 Paul

        * Client.cpp :
          Intgration du client au CORE.

09/05/2003 Paul

        * Affinage du protocole de communication.

xx/05/2003 Paul

        * Container.cpp :
          inversion des paramtres de la methode getListOfPropertyOfObjetsIn
          ajout de la methode getListOfObjetsIn.

        * GD_Objet.cpp :
          ajout des methodes de srialisation/dsrialisation.

        * Frustum.cpp :
          spar du fichier common.h pour ajouter les methodes (srialisation)
           la structure.


--------------------------------------------------------------------------------
        0.7.3
--------------------------------------------------------------------------------
        Rorganisation des rpertoires.
        Mise en place du serveur/client.
```

2 Smile =) Index hiérarchique

2.1 Smile =) Hiérarchie des classes

Cette liste d'héritage est, autant que possible, classée par ordre alphabétique

3 Smile =) Index des composants

3.1 Smile =) Liste des composants

Liste des classes, des strutures et des unions avec une brève description :

4 Smile =) Index des fichiers

4.1 Smile =) Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

5 Smile =) Index des pages

5.1 Smile =) Pages associées

Liste de toutes les pages de documentation associées :

6 Smile =) Documentation de la classe

6.1 Référence de la classe Boite

`#include <Boite.h>`

Graphe d'héritage de la classe Boite

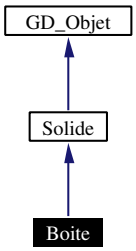
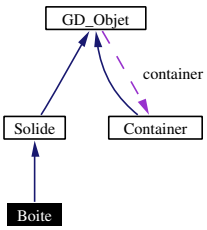


Diagramme de collaboration de Boite :



6.1.1 Description détaillée

Cette classe permet de fabriquer une boite.

Membres publics

Constructeur/Destructeur

– Boite (Container ∗_container, double lx=1.0, double ly=1.0, double lz=1.0, double _densite=1.0)
Constructeur par défaut.

– ∼Boite (void)
Destructeur.

Positionement

Fonctions relatives au positionement dans l'espace.

– void getEchelle (double ∗echelle)
Donne les dimensions.

– void setEchelle (double ∗echelle)
Ajuste les dimensions.

6.1.2 Documentation des contructeurs et destructeurs

6.1.2.1 Boite (Container * *_container*, double *lx* = 1.0, double *ly* = 1.0, double *lz* = 1.0, double *_densite* = 1.0)

Instancie une Boite.

Paramtres :

_container est un pointeur vers le conteneur de ce solide.

lx est la cote de la boite selon x.

ly est la cote de la boite selon y.

lz est la cote de la boite selon z.

_densite est la densité de la boite.

6.1.2.2 ~Boite (void)

Destructeur

6.1.3 Documentation des méthodes

6.1.3.1 void getEchelle (double * *echelle*) [virtual]

Fonction permettant de récupérer les dimensions du GD_Objet.

Paramtres :

echelle est un pointeur vers un tableau de 3 doubles.

Attention :

Cette methode donne par défaut une matrice remplie de 1 (donc une mise à l'echelle sans effet). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.1.3.2 void setEchelle (double * *echelle*) [virtual]

Fonction permettant d'ajuster les dimensions du GD_Objet.

Paramtres :

echelle est un pointeur vers un tableau de 3 doubles.

Redéfinie à partir de GD_Objet.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Boite.h

6.2 Référence de la classe Ccylindre

`#include <Ccylindre.h>`

Graphe d'héritage de la classe Ccylindre

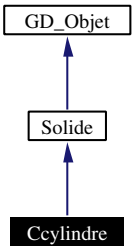
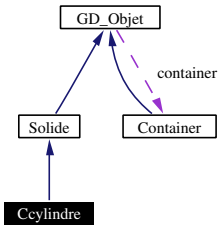


Diagramme de collaboration de Ccylindre :



6.2.1 Description détaillée

Cette classe permet de fabriquer un ccylindre (Caped Cylinder ou capsule).

Membres publics

Constructeur/Destructeur

– Ccylindre (Container ∗_container, double _diametre=1.0, double _lz=1.0, double _densite=1.0)
Constructeur par défaut.

– ∼Ccylindre (void)
Destructeur.

Positionement

Fonctions relatives au positionement dans l'espace.

– void getEchelle (double ∗echelle)
Donne les dimensions.

– void setEchelle (double ∗echelle)
Ajuste les dimensions.

6.2.2 Documentation des contructeurs et destructeurs

6.2.2.1 Ccylindre (Container ∗ *_container*, double *_diametre* = 1.0, double *_lz* = 1.0, double *_densite* = 1.0)

Instancie un Ccylindre.

Paramtres :

_container est un pointeur vers le conteneur de ce solide.

_diametre est le diametre du ccylindre.

_lz est la hauteur du ccylindre.

_densite est la densité du ccylindre.

6.2.2.2 ~Ccylindre (void)

Destructeur

6.2.3 Documentation des méthodes

6.2.3.1 void getEchelle (double * *echelle*) [virtual]

Fonction permettant de récupérer les dimensions du GD_Objet.

Paramtres :

echelle est un pointeur vers un tableau de 3 doubles.

Attention :

Cette methode donne par défaut une matrice remplie de 1 (donc une mise à l'echelle sans effet). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.2.3.2 void setEchelle (double * *echelle*) [virtual]

Fonction permettant d'ajuster les dimensions du Ccylindre.

Paramtres :

echelle est un pointeur vers un tableau de 3 doubles (echelle[0]=rayon, echelle[1]=rayon, echelle[2]=hauteur).

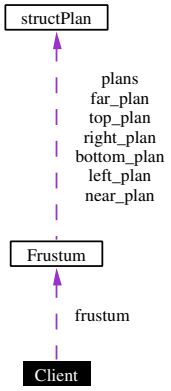
Redéfinie à partir de GD_Objet.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Ccylindre.h

6.3 Référence de la classe Client

`#include <Client.h>`

Diagramme de collaboration de Client :



6.3.1 Description détaillée

Cette classe permet d'acceder au contenu d'une scene partagée par un serveur à travers un réseau TCP/IP.

Voir galement :
Serveur.

Membres publics

Constructeur/Destructeur.

– Client (char ∗serveur=SERVEUR, unsigned short port=PORT)
Constructeur par défaut.

– ∼Client (void)
Destructeur.

Accesseurs.

– void **getFrustum** (frustum_t ∗_frustum)
– void **setFrustum** (frustum_t ∗_frustum)

Amis (friends)

– class GlRenderer

Thread

– void ∗ clientDialogThread (void ∗arg)

6.3.2 Documentation des contructeurs et destructeurs

6.3.2.1 Client (char ∗ *serveur* = SERVEUR, unsigned short *port* = PORT)

Instancie un Client.

Paramtres :

serveur est le serveur auquel se connecter..

port est le port d'écoute du serveur.

6.3.3 Documentation des fonctions amies et associées

6.3.3.1 friend class GlRenderer [friend]

Obsolète

Classe amie pour que le renderer puisse accéder aux objets repatriés depuis le serveur.

À Faire

Trouver mieux que cela pour le rendre indépendant.

6.3.3.2 void* clientDialogThread (void * *arg*) [friend]

Thread de dialogue avec le serveur.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Client.h

6.4 Référence de la classe Collider

`#include <Collider.h>`

Graphe d'héritage de la classe Collider

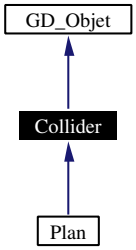
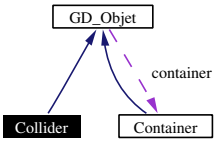


Diagramme de collaboration de Collider :



6.4.1 Description détaillée

Cette classe permet de faire des surfaces fixes.

Membres publics

Constructeur/Destructeur

– Collider (Container *container=0)
Constructeur par défaut.

– virtual ∼Collider (void)
Destructeur.

Accesseurs

– dGeomID getGeomID (void)
Renvoi le dGeomID.

Positionement

Fonctions relatives au positionement dans l'espace.

– void getAABB (double ∗aabb)
Donne l'AABB.

Attributs Protégés

– dGeomID geom
Référence vers le "geom" de ODE (qui sert aux calculs de collision).

6.4.2 Documentation des contructeurs et destructeurs

6.4.2.1 Collider (Container ∗ *container* = 0)

Instancie un Collider.

Paramtres :

container est un pointeur vers le conteneur de ce collider.

6.4.2.2 virtual ∼Collider (void) [virtual]

Le destructeur s'occupe de détruire le "geom".

6.4.3 Documentation des méthodes

6.4.3.1 void getAABB (double ∗ *aabb*) [virtual]

Fonction permettant d'obtenir la boite englobante alignée avec les axes (Axis Aligned Bounding Box) du GD_Objet.

Paramtres :

aabb matrice (1 ligne)(6 colonnes) que la fonction rempliera avec les coordonées de l'aabb du GD_Objet.

Redéfinie à partir de GD_Objet.

La documentation associée à cette classe a été générée à partir du fichier suivant :

– Collider.h

6.5 Référence de la classe Container

`#include <Container.h>`

Graphe d'héritage de la classe Container

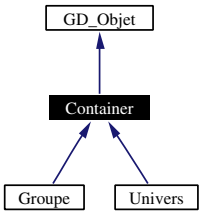
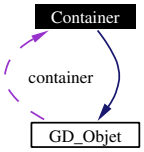


Diagramme de collaboration de Container :



6.5.1 Description détaillée

Cette classe permet de faire des groupes de GD_Objet pour accélérer les calculs de collision et faciliter les déplacements pour tout un ensemble de GD_Objet.

Membres publics

Constructeur/Destructeur

– Container (Container *container=0)
Constructeur par défaut.
Attention :
Seul un univers peut prendre en paramètre un container null.

– virtual ~Container (void)
Destructeur.
Attention :
Tout le contenu sera récursivement détruit.

Accesseurs

– dSpaceID getSpaceID (void)
Renvoi le SpaceID.

– dWorldID getWorldID (void)
Renvoi le WorldID.

– Container : :iterator begin (void)
Retourne un iterateur sur les GD_Objets.

- Container : :iterator end (void)
  Retourne un iterateur sur les GD_Objets.
- void getListOfPropertyOfObjetsIn (list< property_t > *properties, frustum_t *frustum=0)
  Donne des propriété des GD_Objets dans le frustum.
- void getListOfObjetsIn (list< GD_Objet * > *_gdobjets, frustum_t *frustum=0)
  Donne une liste de pointeur vers lee gdobjets contenus dans le frustum.
- void addGDObjet (GD_Objet *gdobjet)
  Ajoute un GD_Objet à gdobjets.
- void remGDObjet (GD_Objet *gdobjet)
  Enlève un GD_Objet à gdobjets.

Positionement

Fonctions relatives au positionement dans l'espace.

- void setPosition (double *position)
  Ajuste la position.
- void getPosition (double *position)
  Donne la position.
- void setRotation (double *rotation)
  Ajuste l'orientation.
- void getRotation (double *rotation)
  Donne l'orientation.
- void setEchelle (double *echelle)
  Ajuste les dimensions.
- void getEchelle (double *echelle)
  Donne les dimensions.
- void getAABB (double *aabb)
  Donne l'AABB.

Attributs Protégés

- list< GD_Objet * > gdobjets
  Liste du contenu.
- dWorldID monde
  Référence vers le "world" de ODE (qui sert aux calculs de dynamique).
- dSpaceID espace
  Référence vers le "space" de ODE (qui sert aux calculs de collision).
- dJointGroupID contactjointgroup

6.5.2 Documentation des contructeurs et destructeurs

6.5.2.1 Container (Container * *container* = 0)

Instancie un Container.

Paramtres :

container est un pointeur vers le conteneur de ce conteneur.

6.5.2.2 virtual ~Container (void) [virtual]

Le destructeur s'occupe de détruire le monde et le space.

6.5.3 Documentation des méthodes

6.5.3.1 dSpaceID getSpaceID (void)

Renvoi le SpaceID

6.5.3.2 dWorldID getWorldID (void)

Renvoi le WorldID

6.5.3.3 Container : :iterator begin (void)

Retourne un itérateur de type Container : :iterator. Par exemple :

```
for( Container::iterator from=leContainer.begin() ; from!=leContainer.end() ; ++from )
{
    GD_Objet *leGdobjet=(*from);
}
```

Voir galement :

end(void)

6.5.3.4 Container : :iterator end (void)

Retourne un itérateur de type Container : :iterator.

Voir galement :

begin(void)

6.5.3.5 void getListOfPropertyOfObjetsIn (list< property_t > * *properties*, frustum_t * *frustum* = 0)

Donne des propriété des GD_Objets dans le frustum.

Paramtres :

properties un pointeur vers où stocker le résutat.

frustum un pointeur vers le volme à tester. Par défaut il vaut 0, ce qui aura pour effet de renvoyer TOUT les GD_Objets.

6.5.3.6 void getListOfObjetsIn (list< GD_Objet * > * *_gdobjets*, frustum_t * *frustum* = 0)

Donne la liste de gdobjets.

Paramtres :

_gdobjets est un poiteur vers la liste où stocker le résutat.

frustum un pointeur vers le volme à tester. Par défaut il vaut 0, ce qui aura pour effet de renvoyer TOUT les GD_Objets.

6.5.3.7 void addGDObjet (GD_Objet * *gdobjet*)

Ajoute un GD_Objet à gdobjets.

Paramtres :
gdobjet est un pointeur vers le GD_Objet à ajouter

À Faire
implémenter le dSpaceAdd (dSpaceID, dGeomID) ;

6.5.3.8 void remGDObjet (GD_Objet * *gdobjet*)

Enlève un GD_Objet à gdobjets.

Paramtres :
gdobjet est un pointeur vers le GD_Objet à enlever

À Faire
implémenter le dSpaceRemove (dSpaceID, dGeomID) ;

6.5.3.9 void setPosition (double * *position*) [virtual]

Fonction permettant d'ajuster la position du GD_Objet.

Paramtres :
position matrice (1 ligne)X(3 colonnes) spécifiant les nouvelles coordonnées cartésiennes du GD_Objet

Redéfinie à partir de GD_Objet.

6.5.3.10 void getPosition (double * *position*) [virtual]

Fonction permettant d'obtenir la position du GD_Objet.

Paramtres :
position matrice (1 ligne)(3 colonnes) que la fonction rempliera avec les coordonnées cartésiennes du GD_Objet.

Attention :
Cette methode donne par défaut une position située à l'origine. Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.5.3.11 void setRotation (double * *rotation*) [virtual]

Fonction permettant d'ajuster l'orientation du GD_Objet.

Paramtres :
rotation matrice (3 lignes)(4 colonnes) spécifiant la matrice de rotation du GD_Objet

Redéfinie à partir de GD_Objet.

6.5.3.12 void getRotation (double * *rotation*) [virtual]

Fonction permettant d'obtenir l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 ligne3)(4 colonnes) que la fonction rempliera avec la matrice de rotation du GD_Objet.

Attention :

Cette methode donne par défaut une matrice identitée (donc une rotation nulle). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.5.3.13 void setEchelle (double * *echelle*) [virtual]

Fonction permettant d'ajuster les dimensions du GD_Objet.

Paramtres :

echelle est un pointeur vers un tableau de 3 doubles.

Redéfinie à partir de GD_Objet.

6.5.3.14 void getEchelle (double * *echelle*) [virtual]

Fonction permettant de récupérer les dimensions du GD_Objet.

Paramtres :

echelle est un pointeur vers un tableau de 3 doubles.

Attention :

Cette methode donne par défaut une matrice remplie de 1 (donc une mise à l'echelle sans effet). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.5.3.15 void getAABB (double * *aabb*) [virtual]

Fonction permettant d'obtenir la boite englobante alignée avec les axes (Axis Aligned Bounding Box) du GD_Objet.

Paramtres :

aabb matrice (1 ligne)(6 colonnes) que la fonction rempliera avec les coordonées de l'aabb du GD_Objet.

Redéfinie à partir de GD_Objet.

6.5.4 Documentation des données imbriquées

6.5.4.1 dJointGroupID contactjointgroup [protected]

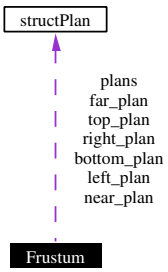
Référence vers un groupe qui sert à stocker les contacts lors des tours de simulation.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Container.h

6.6 Référence de la structure Frustum

`#include <Frustum.h>`

Diagramme de collaboration de Frustum :



6.6.1 Description détaillée

Cette structure permet de gérer un volume défini par six plans.

Voir galement :
structPlan.

Membres publics

– Frustum (void)
Constructeur..

– bool isValide (void)
Indique si le frustum est valide.

– bool serialise (char ∗chaine=0, int taille=0)
Sérialise.

– bool unSerialise (char ∗chaine=0, int taille=0)
Désérialise.

Attributs Publics

– plan_t left_plan
plan de gauche.

– plan_t right_plan
plan de droite.

– plan_t bottom_plan
plan du bas.

– plan_t top_plan
plan du haut.

– plan_t near_plan

plan proche.

– plan_t far_plan
plan éloigné.

– plan_t * plans [6]
Tableau des pointeurs vers chacun des plans.

6.6.2 Documentation des méthodes

6.6.2.1 bool isValide (void)

Fonction permettant de savoir si le frustum est valide

Renvoie :
”true” si le frustum est valide, ”false” sinon.

La documentation associée à cette structure a été générée à partir du fichier suivant :
– Frustum.h

6.7 Référence de la classe GD_Objet

`#include <GD_Objet.h>`

Graphe d’héritage de la classe GD_Objet

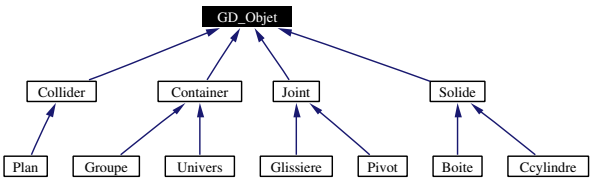
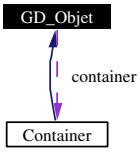


Diagramme de collaboration de GD_Objet :



6.7.1 Description détaillée

Cette classe est à la base de toutes les autres.

Accesseurs

– void setContainer (Container *_container)
Ajuste le container.

– void getContainer (Container **_container)
Donne le container.

– virtual void getTopLevelContainer (Container **_container)
Donne le container le plus haut dans la hierarchie.

– void setUserData (void *_userData)
Ajuste l'attribut userData.

– void * getUserData (void)
Renvoi l'attribut userData.

– void setType (type_t type)
Ajuste le type.

– type_t getType (void)
Renvoi le type.

– unsigned short getId (void)
Renvoie l'id.

– void getProperty (property_t *_property)
Donne les propriétés.

– bool serialise (char *chaine=0, int taille=0)
Sérialise le GD_Objet.

– bool unSerialise (char *chaine=0, int taille=0, map< GD_Objet *, GD_Objet * > *gdobjetMap=0)
Désérialize.

Membres publics

Constructeur/Destructeur

– GD_Objet (Container *container=0)
Constructeur par défaut.
Attention :
Seul les GD_Objet de type Univers peuvent être instancié sans parametre. Tous les autres doivent fournir un container non-nul ;.

– virtual ~GD_Objet (void)
Destructeur.

Positionement

Fonctions relatives au positionement dans l'espace.

À Faire
Proposer des fonctions utilisan les quaternions.

– void moveOf (double x, double y, double z)
Déplace le GD_Objet.

– virtual void setPosition (double *position)
Ajuste la position.

– virtual void getPosition (double *position)
Donne la position.

- void rotateOf (double angle, double axeX, double axeY, double axeZ)
  Tourne le GD_Objet.
- virtual void setRotation (double *rotation)
  Ajuste l'orientation.
- void setRotation (double angleRadian, double *axe)
  Ajuste l'orientation.
- void setRotation (double angleRadian, double ax, double ay, double az)
  Ajuste l'orientation.
- void setRotation (int angleDegre, double *axe)
  Ajuste l'orientation.
- void setRotation (int angleDegre, double ax, double ay, double az)
  Ajuste l'orientation.
- virtual void getRotation (double *rotation)
  Donne l'orientation.
- virtual void setEchelle (double *echelle)
  Ajuste les dimensions.
- virtual void getEchelle (double *echelle)
  Donne les dimensions.
- void scaleOf (double x=1, double y=1, double z=1)
  Redimensionne.
- virtual void getAABB (double *aabb)
  Donne l'AABB.
- bool isIn (frustum_t *frustum)
  Indique la situation.

Apparence

Fonctions relatives à l'apparence.

- bool isVisible (void)
  Teste si le GD_Objet est visible.
- void setVisible (void)
  Rend visible.
- void setInvisible (void)
  Rend invisible.
- bool isSelected (void)
  Teste la selection.
- void setSelected (void)
  Selectionne.
- void setNotSelected (void)
  Déselectionne.
- void setCouleur (unsigned char r, unsigned char v, unsigned char b)
  Ajuste la couleur.
- void setCouleur (unsigned char *couleur)
  Ajuste la couleur.

– void getCouleur (unsigned char *couleur)
Donne la couleur.

Attributs Protégés

– Container * container
C'est un pointeur vers le container.

– unsigned short id
C'est un entier identifiant de maniere unique le GD_Objet.

– type_t type
C'est le type du GD_Objet.

– void * userData
C'est un pointeur vers une donnée quelconque.

– bool visible
Indique si le GD_Objet est visible.

– bool selected
Indique si le GD_Objet est selectioné.

– unsigned char couleur [3]
C'est un tableau avec les composantes r v b du GD_Objet (0-255).

Attributs Protégés Statiques

– unsigned short numGDObjets
C'est le compteur de GD_Objets instanciés.

6.7.2 Documentation des contructeurs et destructeurs

6.7.2.1 GD_Objet (Container * *container* = 0)

Instancie un GD_Objet.

À Faire
ameliorer le controle du paramètre.

Paramtres :
container est un pointeur vers le conteneur du GD_Objet instancié.

6.7.3 Documentation des méthodes

6.7.3.1 void setContainer (Container * *_container*)

Fonction permettant d'ajuster le container du GD_Objet.

Paramtres :
_container nouveau container

6.7.3.2 void getContainer (Container ** _container)

Fonction permettant d'obtenir le container du GD_Objet.

Paramtres :
_container pointeur vers un pointeur de container

6.7.3.3 virtual void getTopLevelContainer (Container ** _container) [virtual]

Fonction permettant d'obtenir le container du GD_Objet le plus haut dans la hierarchie.Ce container sera un univers. D'ailleur cette fonction est redéfinie dans la classe Univers.

Paramtres :
_container pointeur vers un pointeur de container

Redéfinie dans Univers.

6.7.3.4 void setUserData (void * _userData)

Fonction permettant d'associer à un GD_Objet des données personnelles.

Paramtres :
_userData pointeur vers les donner à enregistrer.

6.7.3.5 void* getUserData (void)

Fonction permettant de récupérer les données personnelles.

Renvoie :
un pointeur vers un type quelconque.

6.7.3.6 void setType (type_t type)

Fonction permettant d'ajuster le type du GD_Objet.

Paramtres :
type est le nouveau type.

6.7.3.7 type_t getType (void)

Fonction permettant de récupérer le type du GD_Objet.

Renvoie :
une valeur correspondante au type de type type_t.

6.7.3.8 unsigned short getId (void)

Fonction permettant de récupérer l'id du GD_Objet.

Renvoie :
l'id du GD_Objet.

6.7.3.9 void getProperty (property_t * _property)

Fonction permettant de récupérer les propriétés du GD_Objet.

Paramtres :

_property un pointeur dans lequel la fonction stockera le résultat.

6.7.3.10 bool serialise (char * *chaine* = 0, int *taille* = 0)

Fonction permettant de ranger dans une chaine de 256 charactères les propriétés du GD_Objet afin de l'envoer à travers le réseau par exemple.

Renvoie :

true si la sérialisation a réussie, false sinon.

Paramtres :

chaine est un pointeur vers la chaine à écrire.

taille est la longeur de la chaine vers laquelle chaine pointe.

Attention :

La taille doit etre correctement fixée.

6.7.3.11 bool unSerialise (char * *chaine* = 0, int *taille* = 0, map< GD_Objet *, GD_Objet * > * *gdobjetMap* = 0) [static]

Fonction permettant d'ajuster les propriété du GD_Objet en fonction d'une chain de charactère. Le premier champ (type) n'est pas pris en compte.

Renvoie :

true si la désérialisation a réussie, false sinon.

Paramtres :

chaine est un pointeur vers la chaine à lire.

taille est la longeur de la chaine vers laquelle chaine.

gdobjetMap est une liste de correspondance entre pointeur de GD_Objets.

Attention :

La taille doit etre correctement fixée.

6.7.3.12 void moveOf (double *x*, double *y*, double *z*)

Fonction permettant de déplacer le GD_Objet suivant x, y et z.

Paramtres :

x déplacement selon x.

y déplacement selon y.

z déplacement selon z.

6.7.3.13 virtual void setPosition (double * *position*) [virtual]

Fonction permettant d'ajuster la position du GD_Objet.

Paramtres :

position matrice (1 ligne)X(3 colonnes) spécifiant les nouvelles coordonnées cartésiennes du GD_-Objet

Redéfinie dans Container, Glissiere, Pivot, Plan, et Solide.

6.7.3.14 virtual void getPosition (double * *position*) [virtual]

Fonction permettant d'obtenir la position du GD_Objet.

Paramtres :

position matrice (1 ligne)(3 colonnes) que la fonction rempliera avec les coordonnées cartésiennes du GD_Objet.

Attention :

Cette methode donne par défaut une position située à l'origine. Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie dans Container, Glissiere, Pivot, Plan, et Solide.

6.7.3.15 void rotateOf (double *angle*, double *axeX*, double *axeY*, double *axeZ*)

Fonction permettant de tourner le GD_Objet autour de son centre de gravité selon un axe et un angle donné.

Paramtres :

angle angle en radian.

axeX orientation de l'axe de rotation selon x

axeY orientation de l'axe de rotation selon y

axeZ orientation de l'axe de rotation selon z

6.7.3.16 virtual void setRotation (double * *rotation*) [virtual]

Fonction permettant d'ajuster l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 lignes)(4 colonnes) spécifiant la matrice de rotation du GD_Objet

Redéfinie dans Container, Glissiere, Pivot, Plan, et Solide.

6.7.3.17 void setRotation (double *angleRadian*, double * *axe*)

Fonction permettant d'ajuster l'orientation du GD_Objet en fonction de l'axe de rotation et de l'angle en radian.

Paramtres :

angleRadian est l'angle de rotation en radian

axe est le vecteur définissant l'axe de rotation.

6.7.3.18 void setRotation (double *angleRadian*, double *ax*, double *ay*, double *az*)

Fonction permettant d'ajuster l'orientation du GD_Objet en fonction de l'axe de rotation et de l'angle en radian.

Paramtres :

angleRadian est l'angle de rotation en radian

ax est la composant x du vecteur de rotation.

ay est la composant y du vecteur de rotation.

az est la composant z du vecteur de rotation.

6.7.3.19 void setRotation (int *angleDegre*, double * *axe*)

Fonction permettant d'ajuster l'orientation du GD_Objet en fonction de l'axe de rotation et de l'angle en radian.

Paramtres :

angleDegre est l'angle de rotation en degré

axe est le vecteur définissant l'axe de rotation.

6.7.3.20 void setRotation (int *angleDegre*, double *ax*, double *ay*, double *az*)

Fonction permettant d'ajuster l'orientation du GD_Objet en fonction de l'axe de rotation et de l'angle en radian.

Paramtres :

angleDegre est l'angle de rotation en degré

ax est la composant x du vecteur de rotation.

ay est la composant y du vecteur de rotation.

az est la composant z du vecteur de rotation.

6.7.3.21 virtual void getRotation (double * *rotation*) [virtual]

Fonction permettant d'obtenir l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 ligne3)(4 colonnes) que la fonction rempliera avec la matrice de rotation du GD_Objet.

Attention :

Cette methode donne par défaut une matrice identitée (donc une rotation nulle). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie dans Container, Glissiere, Pivot, Plan, et Solide.

6.7.3.22 virtual void setEchelle (double * *echelle*) [virtual]

Fonction permettant d'ajuster les dimensions du GD_Objet.

Paramtres :

echelle est un pointeur vers un tableau de 3 doubles.

Redéfinie dans Boite, Ccylindre, Container, Glissiere, Pivot, et Plan.

6.7.3.23 virtual void getEchelle (double * *echelle*) [virtual]

Fonction permettant de récupérer les dimensions du GD_Objet.

Paramtres :

echelle est un pointeur vers un tableau de 3 doubles.

Attention :

Cette methode donne par défaut une matrice remplie de 1 (donc une mise à l'echelle sans effet). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie dans Boite, Ccylindre, Container, Glissiere, Pivot, et Plan.

6.7.3.24 void scaleOf (double *x* = 1, double *y* = 1, double *z* = 1)

Fonction permettant de redimensionner le GD_Objet.

Paramtres :

x proportion du redimensionnement selon x.

y proportion du redimensionnement selon y.

z proportion du redimensionnement selon z.

6.7.3.25 virtual void getAABB (double * *aabb*) [virtual]

Fonction permettant d'obtenir la boite englobante alignée avec les axes (Axis Aligned Bounding Box) du GD_Objet.

Paramtres :

aabb matrice (1 ligne)(6 colonnes) que la fonction rempliera avec les coordonées de l'aabb du GD_Objet.

Redéfinie dans Collider, Container, Joint, et Solide.

6.7.3.26 bool isIn (frustum_t * *frustum*)

Fonction permettant de savoir si le GD_Objet est dans le volume délimité par le frustum.

Paramtres :

frustum corresponant au volume à tester.

Renvoie :

"true" si le GD_Objet est à l'interieur du frustum, "false" sinon.

6.7.3.27 bool isVisible (void)

Fonction permettant de savoir si le GD_Objet est visible.

Renvoie :

"true" si le GD_Objet est visible, "false" sinon.

6.7.3.28 void setVisible (void)

Fonction permettant de rendre visible le GD_Objet.

6.7.3.29 void setInvisible (void)

Fonction permettant de rendre invisible le GD_Objet.

6.7.3.30 bool isSelected (void)

Fonction permettant de savoir si le GD_Objet est selectionne ou pas.

Renvoie :

"true" si le GD_Objet est selectionné, "false" sinon.

6.7.3.31 void setSelected (void)

Fonction permettant de selectionner le GD_Objet.

6.7.3.32 void setNotSelected (void)

Fonction permettant de deselectionner le GD_Objet.

6.7.3.33 void setCouleur (unsigned char *r*, unsigned char *v*, unsigned char *b*)

Fonction permettant d'ajuster la couleur du GD_Objet.

Paramtres :

r composante rouge (0-255).

v composante verte (0-255).

b composante bleue (0-255).

6.7.3.34 void setCouleur (unsigned char * *couleur*)

Fonction permettant d'ajuster la couleur du GD_Objet.

Paramtres :

couleur tableau de composantes rvb (0-255).

6.7.3.35 void getCouleur (unsigned char * *couleur*)

Fonction permettant d'obtenir la couleur du GD_Objet.

Paramtres :

couleur matrice (1 ligne)(3 colonnes) que la fonction rempliera avec la couleur du GD_Objet.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– GD_Objet.h

6.8 Référence de la classe Glissiere

`#include <Glissiere.h>`

Graphe d'héritage de la classe Glissiere

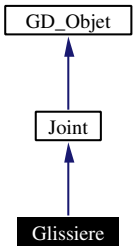
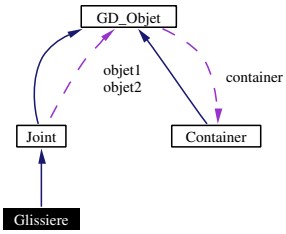


Diagramme de collaboration de Glissiere :



6.8.1 Description détaillée

Cette classe permet de fabriquer une glissiere.

Membres publics

Constructeur/Destructeur

– Glissiere (Container ∗_container, GD_Objet ∗_objet1=0, GD_Objet ∗_objet2=0)
Constructeur par défaut.

– ∼Glissiere (void)
Destructeur.

Positionement

Fonctions relatives au positionement dans l'espace

– void setPosition (double ∗position)
Ajuste la position.

– void getPosition (double ∗position)
Donne la position.

– void setRotation (double ∗rotation)
Ajuste l'orientation.

– void getRotation (double ∗rotation)
Donne l'orientation.

– void setEchelle (double ∗echelle)
Ajuste les dimensions.

– void getEchelle (double ∗echelle)
Donne les dimensions.

Spécificités des glissieres

Fonctions spécifiques au fonctionement des glissieres

– void setAnchor (double ∗position)
N'a pas d'effet.

– void getAnchor (double ∗position, bool voisins=false) const
Donne le centre de la glissière.

– void setAxis (double ∗axis)
Ajuste l'axe de la glissiere.

- void getAxis (double *axis) const
  Donne l'axe de la glissiere.
- void setForce (double force)
  Impose une force.
- void setVel (double vel)
  Impose une vitesse.
- void getExtension (double *extension) const
  Donne l'extension.
- void getExtensionRate (double *rate) const
  Donne la variation de l'extension.

6.8.2 Documentation des contructeurs et destructeurs

6.8.2.1 Glissiere (Container * *_container*, GD_Objet * *_objet1* = 0, GD_Objet * *_objet2* = 0)

Instancie une glissiere.

Paramtres :

_container est un pointeur vers le conteneur de cette glissiere.

_objet1 est le premier objet à lier.

_objet2 est le second objet à lier.

6.8.2.2 ~Glissiere (void)

Destructeur

6.8.3 Documentation des méthodes

6.8.3.1 void setPosition (double * *position*) [virtual]

Fonction permettant d'ajuster la position du GD_Objet.

Paramtres :

position matrice (1 ligne)X(3 colonnes) spécifiant les nouvelles coordonnées cartésiennes du GD_Objet

Redéfinie à partir de GD_Objet.

6.8.3.2 void getPosition (double * *position*) [virtual]

Fonction permettant d'obtenir la position du GD_Objet.

Paramtres :

position matrice (1 ligne)(3 colonnes) que la fonction rempliera avec les coordonnées cartésiennes du GD_Objet.

Attention :

Cette methode donne par défaut une position située à l'origine. Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.8.3.3 void setRotation (double * *rotation*) [virtual]

Fonction permettant d'ajuster l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 lignes)(4 colonnes) spécifiant la matrice de rotation du GD_Objet

Redéfinie à partir de GD_Objet.

6.8.3.4 void getRotation (double * *rotation*) [virtual]

Fonction permettant d'obtenir l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 ligne3)(4 colonnes) que la fonction rempliera avec la matrice de rotation du GD_Objet.

Attention :

Cette methode donne par défaut une matrice identitée (donc une rotation nulle). Elle devra donc impérativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.8.3.5 void setEchelle (double * *echelle*) [virtual]

Fonction permettant d'ajuster les dimensions du GD_Objet.

Paramtres :

echelle est un pointeur vers un tableau de 3 doubles.

Redéfinie à partir de GD_Objet.

6.8.3.6 void getEchelle (double * *echelle*) [virtual]

Fonction permettant de récupérer les dimensions du GD_Objet.

Paramtres :

echelle est un pointeur vers un tableau de 3 doubles.

Attention :

Cette methode donne par défaut une matrice remplie de 1 (donc une mise à l'echelle sans effet). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.8.3.7 void setAnchor (double * *position*)

Cette methode n'a pas d'effet sur une glissière.

6.8.3.8 void getAnchor (double * *position*, bool *voisins* = false) const

Cette methode s'utilise de la meme manière que getPosition. Cependant elle offre en plus la possibilité d'obtenir les coordonnées relatives des objets reliés.

Paramtres :

position matrice (1 ligne)(3 colonnes) que la fonction rempliera avec les coordonnées cartésiennes du centre de la glissière.

voisins si cette valeur vaut true il faut passer une matrice 3∗(1 ligne)(3 colonnes) en premier paramètre.

6.8.3.9 void setAxis (double ∗ *axis*)

Cette methode permet d'ajuster l'axe de la glissiere.

Paramtres :

axis matrice (1 ligne)(3 colonnes) définissant l'axe.

6.8.3.10 void getAxis (double ∗ *axis*) const

Cette methode permet d'obtenir l'axe de la glissiere.

Paramtres :

axis matrice (1 ligne)(3 colonnes) que la fonction rempliera avec un vecteur définissant l'axe.

6.8.3.11 void setForce (double *force*)

Cette methode permet d'imposer une force, et ainsi d'activer le "moteur" de la glissiere. Si la vitesse est mise à zero (setVel(0)), définir la force revient à définir un coefficient de friction sur l'axe. Sinon, la force correspondra au couple maximum que le "moteur" utilisera pour atteindre la vitesse définie. Si la force est nulle, le "moteur" est éteint.

Paramtres :

force force appliquée par le "moteur" sur la liaison.

Voir galement :

setVel.

6.8.3.12 void setVel (double *vel*)

Cette methode permet d'imposer une vitesse linéaire pour le "moteur".

Paramtres :

vel vitesse imposée au moteur.

Voir galement :

setForce.

6.8.3.13 void getExtension (double ∗ *extension*) const

Cette methode donne l'extension entre les deux objets reliés par la glissiere. Cette longueur vaut zero dans la configuration initiale des deux objets. Lorsque l'axe de la glissiere est modifié, la position courante des objets reliés est prise en compte pour remettre à zero la longueur.

Paramtres :

extension est un pointeur vers la variable à affecter.

Voir galement :

getExtensionRate.

6.8.3.14 void getExtensionRate (double * *rate*) const

Cette methode donne la variation de l'extension par rapport au temps.

Paramtres :
rate est un pointeur vers la variable à affecter.

Voir galement :
getExtension.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Glissiere.h

6.9 Référence de la classe Groupe

`#include <Groupe.h>`

Graphe d'héritage de la classe Groupe

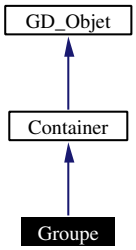
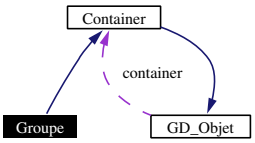


Diagramme de collaboration de Groupe :



6.9.1 Description détaillée

Cette classe permet de faire des groupes de GD_Objet pour accélérer les calculs de collision et faciliter les déplacements pour tout un ensemble de GD_Objet.

Membres publics

Constructeur/Destructeur

– Groupe (Container *container=0)
Constructeur par défaut.

– ~Groupe (void)
Destructeur.

6.9.2 Documentation des contructeurs et destructeurs

6.9.2.1 Groupe (Container * *container* = 0)

Instancie un Groupe.

Paramtres :

container est un pointeur vers le conteneur du groupe instancié.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Groupe.h

6.10 Référence de la classe Joint

`#include <Joint.h>`

Graphe d'héritage de la classe Joint

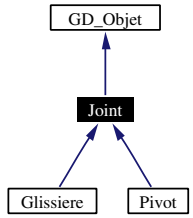
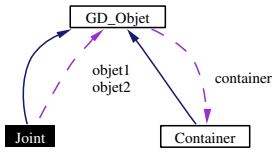


Diagramme de collaboration de Joint :



6.10.1 Description détaillée

Cette classe permet de lier deux GD_Objets et de leur imposer des contraintes. Attention, certaines précautions sont à prendre pour bien utiliser les joints.

Membres publics

Constructeur/Destructeur

– Joint (Container *container=0, GD_Objet *_objet1=0, GD_Objet *_objet2=0)
Constructeur par défaut.

– virtual ~Joint (void)
Destructeur.

Accesseurs

– dJointID getJointID (void)

Renvoi le dJointID.

– void getAABB (double *aabb)
Donne l'AABB.

Outils

– void relie (GD_Objet *_objet1, GD_Objet *_objet2=0)

Attributs Protégés

– dJointID joint
Référence vers le "joint" de ODE (qui sert aux calculs de dynamique).

6.10.2 Documentation des contructeurs et destructeurs

6.10.2.1 Joint (Container * *container* = 0, GD_Objet * *_objet1* = 0, GD_Objet * *_objet2* = 0)

Instancie un Joint.

Paramtres :

container est un pointeur vers le conteneur de ce joint.

_objet1 pointeur vers le premier objet.

_objet2 pointeur vers le deuxième objet.

Attention :

Les joints ne relient que des Solide pour l'instant.

À Faire

Résoudre le problème de relie : il faudrait que la methode soit appelée dans le constructeur, or ce n'est pas possible puisque le joint n'a pas encore été créé par ODE.

6.10.2.2 virtual ~Joint (void) [virtual]

Le destructeur.

6.10.3 Documentation des méthodes

6.10.3.1 void getAABB (double * *aabb*) [virtual]

Fonction permettant d'obtenir la boite englobante alignée avec les axes (Axis Aligned Bounding Box) du GD_Objet.

Paramtres :

aabb matrice (1 ligne)(6 colonnes) que la fonction rempliera avec les coordonées de l'aabb du GD_Objet.

Redéfinie à partir de GD_Objet.

6.10.3.2 void relie (GD_Objet * *_objet1*, GD_Objet * *_objet2* = 0)

Permet de relier deux objets entre eux.

Paramtres :

_objet1 pointeur vers le premier objet qui ne peut etre nul.

_objet2 pointeur vers le deuxième objet.

Attention :

Cette methode ne fonctionne qu'avec des Solide en parametre.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Joint.h

6.11 Référence de la classe Pivot

`#include <Pivot.h>`

Graphe d'héritage de la classe Pivot

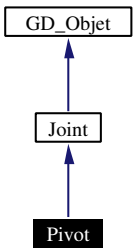
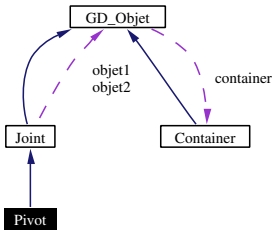


Diagramme de collaboration de Pivot :



6.11.1 Description détaillée

Cette classe permet de fabriquer un pivot.

Spécificités des pivots

Fonctions spécifiques au fonctionement des pivots.
– void setAxis (double *axis)
Ajuste l'axe de rotation du pivot.

– void getAxis (double *axis)
Donne l'axe de rotation du pivot.

– void setForce (double force)
Impose une force.

– void setVel (double vel)
Impose une vitesse.

– void getAngle (double *angle)
Donne l'angle d'ouverture en radian.

– void getAngleRate (double *rate)
Donne la variation de l'angle d'ouverture.

Membres publics

Constructeur/Destructeur

– Pivot (Container *_container, GD_Objet *_objet1=0, GD_Objet *_objet2=0)
Constructeur par défaut.

– ~Pivot (void)
Destructeur.

Positionement

Fonctions relatives au positionement dans l'espace

– void setPosition (double *position)
Ajuste la position.

– void getPosition (double *position)
Donne la position.

– void setRotation (double *rotation)
Ajuste l'orientation.

– void getRotation (double *rotation)
Donne l'orientation.

– void setEchelle (double *echelle)
Ajuste les dimensions.

– void getEchelle (double *echelle)
Donne les dimensions.

6.11.2 Documentation des contructeurs et destructeurs

6.11.2.1 Pivot (Container * *_container*, GD_Objet * *_objet1* = 0, GD_Objet * *_objet2* = 0)

Instancie un pivot.

Attention :

Le pivot a par défaut son axe de rotation selon l'axe des x et son positionement à l'origine. Les methodes de parametrage (set/get Position...) devraient etre appelées aprés avoir relié les GD_Objet (sinon elles n'ont pas d'effet).

Paramtres :

_container est un pointeur vers le conteneur de ce pivot.

_objet1 est le premier objet à lier.

_objet2 est le second objet à lier.

Voir galement :

manuel de ODE : `http ://opende.sourceforge.net/ode-latest-userguide.html#ref54`

6.11.2.2 ~Pivot (void)

Destructeur

6.11.3 Documentation des méthodes

6.11.3.1 void setPosition (double * *position*) [virtual]

Fonction permettant d'ajuster la position du GD_Objet.

Paramtres :

position matrice (1 ligne)X(3 colonnes) spécifiant les nouvelles coordonnées cartésiennes du GD_Objet

Redéfinie à partir de GD_Objet.

6.11.3.2 void getPosition (double * *position*) [virtual]

Fonction permettant d'obtenir la position du GD_Objet.

Paramtres :

position matrice (1 ligne)(3 colonnes) que la fonction rempliera avec les coordonnées cartésiennes du GD_Objet.

Attention :

Cette methode donne par défaut une position située à l'origine. Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.11.3.3 void setRotation (double * *rotation*) [virtual]

Fonction permettant d'ajuster l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 lignes)(4 colonnes) spécifiant la matrice de rotation du GD_Objet

Redéfinie à partir de GD_Objet.

6.11.3.4 void getRotation (double * *rotation*) [virtual]

Fonction permettant d'obtenir l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 ligne3)(4 colonnes) que la fonction rempliera avec la matrice de rotation du GD_Objet.

Attention :
Cette methode donne par défaut une matrice identitée (donc une rotation nulle). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.11.3.5 void setEchelle (double * *echelle*) [virtual]

Fonction permettant d'ajuster les dimensions du GD_Objet.

Paramtres :
echelle est un pointeur vers un tableau de 3 doubles.

Redéfinie à partir de GD_Objet.

6.11.3.6 void getEchelle (double * *echelle*) [virtual]

Fonction permettant de récupérer les dimensions du GD_Objet.

Paramtres :
echelle est un pointeur vers un tableau de 3 doubles.

Attention :
Cette methode donne par défaut une matrice remplie de 1 (donc une mise à l'echelle sans effet). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.11.3.7 void setAxis (double * *axis*)

Cette methode permet d'ajuster l'axe de rotation du pivot. Il faut que l'axe ai une longeur unitaire.

Paramtres :
axis matrice (1 ligne)(3 colonnes) définissant l'axe de rotation du pivot.

À Faire
assouplir la methode en acceptant un axe de longeur quelconque.

6.11.3.8 void getAxis (double * *axis*)

Cette methode permet d'obtenir l'axe de rotation du pivot.

Paramtres :
axis matrice (1 ligne)(3 colonnes) que la fonction rempliera avec un vecteur définissant l'axe de rotation du pivot.

6.11.3.9 void setForce (double *force*)

Cette methode permet d'imposer une force, et ainsi d'activer le "moteur" du pivot. Si la vitesse est mise à zero (setVel(0)), définir la force revient à définir un coefficient de friction sur l'axe. Sinon, la force correspondra au couple maximum que le "moteur" utilisera pour atteindre la vitesse définie. Si la force est nulle, le "moteur" est éteint.

Paramtres :
force force appliquée par le "moteur" sur la liaison.

Voir galement :
setVel.

6.11.3.10 void setVel (double *vel*)

Cette methode permet d'imposer une vitesse angulaire pour le "moteur".

Paramtres :
vel vitesse imposée au moteur.

Voir galement :
setForce.

6.11.3.11 void getAngle (double * *angle*)

Cette methode donne l'angle entre les deux objets reliés par le pivot. Ce angle vaut zero dans la configuration initiale des deux objets l'un par rapport à l'autre. Lorsque le centre ou l'axe du pivot est modifié, la position courante des objets reliés est prise en compte pour remettre à zero l'angle.La valeur retournée est en radian. Elle vaut entre -pi et pi.

Paramtres :
angle est un pointeur vers la variable à affecter.

Voir galement :
getAngleRate.

6.11.3.12 void getAngleRate (double * *rate*)

Cette methode donne la variation de l'angle par rapport au temps.

Paramtres :
rate est un pointeur vers la variable à affecter.

Voir galement :
getAngle.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Pivot.h

6.12 Référence de la classe Plan

`#include <Plan.h>`

Graphe d'héritage de la classe Plan

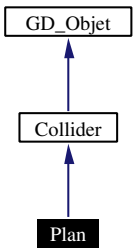
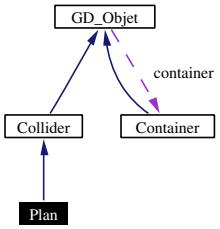


Diagramme de collaboration de Plan :



6.12.1 Description détaillée

Cette classe permet de fabriquer un plan définit par sa normale et une constante.

Membres publics

Constructeur/Destructeur

– Plan (Container ∗container, double a=0, double b=0, double c=1, double d=0)
Constructeur par défaut.

– ∼Plan (void)
Destructeur.

Positionement

Fonctions relatives au positionement dans l'espace.

– void setPosition (double ∗position)
Ajuste la position.

– void getPosition (double ∗position)
Donne la position.

– void setRotation (double ∗rotation)
Ajuste l'orientation.

– void getRotation (double ∗rotation)
Donne l'orientation.

– void setEchelle (double ∗echelle)
Ajuste les dimensions.

– void getEchelle (double ∗echelle)
Donne les dimensions.

6.12.2 Documentation des contructeurs et destructeurs

6.12.2.1 Plan (Container ∗ *container*, double *a* = 0, double *b* = 0, double *c* = 1, double *d* = 0)

Instancie un Plan.

Paramtres :

container est un pointeur vers le conteneur de ce conteneur.

a est la composante en x de la normale.

b est la composante en y de la normale.

c est la composante en z de la normale.

d est la constante du plan.

6.12.3 Documentation des méthodes

6.12.3.1 void setPosition (double * *position*) [virtual]

Fonction permettant d'ajuster la position du GD_Objet.

Paramtres :

position matrice (1 ligne)X(3 colonnes) spécifiant les nouvelles coordonnées cartésiennes du GD_Objet

Redéfinie à partir de GD_Objet.

6.12.3.2 void getPosition (double * *position*) [virtual]

Fonction permettant d'obtenir la position du GD_Objet.

Paramtres :

position matrice (1 ligne)(3 colonnes) que la fonction rempliera avec les coordonnées cartésiennes du GD_Objet.

Attention :

Cette methode donne par défaut une position située à l'origine. Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.12.3.3 void setRotation (double * *rotation*) [virtual]

Fonction permettant d'ajuster l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 lignes)(4 colonnes) spécifiant la matrice de rotation du GD_Objet

Redéfinie à partir de GD_Objet.

6.12.3.4 void getRotation (double * *rotation*) [virtual]

Fonction permettant d'obtenir l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 ligne3)(4 colonnes) que la fonction rempliera avec la matrice de rotation du GD_Objet.

Attention :

Cette methode donne par défaut une matrice identitée (donc une rotation nulle). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.12.3.5 void setEchelle (double * *echelle*) [virtual]

Fonction permettant d'ajuster les dimensions du GD_Objet.

Paramtres :
echelle est un pointeur vers un tableau de 3 doubles.

Redéfinie à partir de GD_Objet.

6.12.3.6 void getEchelle (double * *echelle*) [virtual]

Fonction permettant de récupérer les dimensions du GD_Objet.

Paramtres :
echelle est un pointeur vers un tableau de 3 doubles.

Attention :
Cette methode donne par défaut une matrice remplie de 1 (donc une mise à l'echelle sans effet). Elle devra donc imperativement être redéfinie dans la descendance.

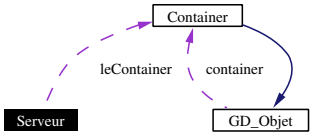
Redéfinie à partir de GD_Objet.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Plan.h

6.13 Référence de la classe Serveur

`#include <Serveur.h>`

Diagramme de collaboration de Serveur :



6.13.1 Description détaillée

Cette classe permet de partager le contenu de la scene à travers un réseau TCP/IP.

Voir galement :
Client.

Membres publics

Constructeur/Destructeur

– Serveur (Container *leContainer, unsigned short port=PORT)
Constructeur par défaut.

– ~Serveur (void)
Destructeur.

Amis (friends)

Thread
– void ∗ serveurDialogThread (void ∗arg)

6.13.2 Documentation des contructeurs et destructeurs

6.13.2.1 Serveur (Container ∗ *leContainer*, unsigned short *port* = PORT)

Instancie un Serveur.

Paramtres :
leContainer est généralement l'univers à partager.
port est le port d'écoute.

6.13.3 Documentation des fonctions amies et associées

6.13.3.1 void∗ serveurDialogThread (void ∗ *arg*) [friend]

Thread de dialogue avec les clients.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Serveur.h

6.14 Référence de la classe Solide

`#include <Solide.h>`

Graphe d'héritage de la classe Solide

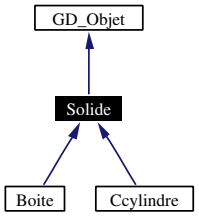
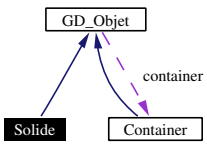


Diagramme de collaboration de Solide :



6.14.1 Description détaillée

Cette classe permet de faire des solides (boites, cylindres...).

Membres publics

Constructeur/Destructeur

– Solide (Container *_container=0)
Constructeur par défaut.

– virtual ~Solide (void)
Destructeur.

Accesseurs

– dGeomID getGeomID (void)
Renvoi le dGeomID.

– dBodyID getBodyID (void)
Renvoi le dBodyID.

Positionement

Fonctions relatives au positionement dans l'espace.

– void setPosition (double *position)
Ajuste la position.

– void getPosition (double *position)
Donne la position.

– void setRotation (double *rotation)
Ajuste l'orientation.

– void getRotation (double *rotation)
Donne l'orientation.

– void getAABB (double *aabb)
Donne l'AABB.

Attributs Protégés

– dGeomID geom
Référence vers le "geom" de ODE (qui sert aux calculs de collision).

– dBodyID body
Référence vers le "body" de ODE (qui sert aux calculs de dynamique).

– dMass masse
Référence vers la masse du ""body" de ODE.

6.14.2 Documentation des contructeurs et destructeurs

6.14.2.1 Solide (Container * *_container* = 0)

Instancie un Solide.

Paramtres :

_container est un pointeur vers le conteneur de ce solide.

6.14.2.2 virtual ~Solide (void) [virtual]

Le destructeur s'occupe de détruire le "geom" et le "body" ainsi que de se retirer du container.

6.14.3 Documentation des méthodes

6.14.3.1 void setPosition (double * *position*) [virtual]

Fonction permettant d'ajuster la position du GD_Objet.

Paramtres :

position matrice (1 ligne)X(3 colonnes) spécifiant les nouvelles coordonnées cartésiennes du GD_Objet

Redéfinie à partir de GD_Objet.

6.14.3.2 void getPosition (double * *position*) [virtual]

Fonction permettant d'obtenir la position du GD_Objet.

Paramtres :

position matrice (1 ligne)(3 colonnes) que la fonction rempliera avec les coordonnées cartésiennes du GD_Objet.

Attention :

Cette methode donne par défaut une position située à l'origine. Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.14.3.3 void setRotation (double * *rotation*) [virtual]

Fonction permettant d'ajuster l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 lignes)(4 colonnes) spécifiant la matrice de rotation du GD_Objet

Redéfinie à partir de GD_Objet.

6.14.3.4 void getRotation (double * *rotation*) [virtual]

Fonction permettant d'obtenir l'orientation du GD_Objet.

Paramtres :

rotation matrice (3 ligne3)(4 colonnes) que la fonction rempliera avec la matrice de rotation du GD_Objet.

Attention :

Cette methode donne par défaut une matrice identitée (donc une rotation nulle). Elle devra donc imperativement être redéfinie dans la descendance.

Redéfinie à partir de GD_Objet.

6.14.3.5 void getAABB (double * *aabb*) [virtual]

Fonction permettant d'obtenir la boite englobante alignée avec les axes (Axis Aligned Bounding Box) du GD_Objet.

Paramtres :

aabb matrice (1 ligne)(6 colonnes) que la fonction rempliera avec les coordonées de l'aabb du GD_Objet.

Redéfinie à partir de GD_Objet.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Solide.h

6.15 Référence de la structure structPlan

`#include <Frustum.h>`

6.15.1 Description détaillée

Cette structure permet de définir un plan par son equation.

Voir galement :

Frustum.

Attributs Publics

– double a
Composante x de la normale.

– double b
Composante y de la normale.

– double c
Composante z de la normale.

– double d
Constante définissant la position du plan.

La documentation associée à cette structure a été générée à partir du fichier suivant :
– Frustum.h

6.16 Référence de la structure structProperty

`#include <common.h>`

6.16.1 Description détaillée

Définit une structure comportant les propriétées necessaires à la representation graphique d'un GD_Objet.

Attributs Publics

– type_t type
Définit le type du GD_Objet.

– unsigned char couleur [3]
Définit la couleur (rouge, vert, bleu [0-255]).

– double position [3]
Définit la matrice de transformation du GD_Objet (3lignes x 4colones).

– double rotation [12]
Définit la matrice de rotation du GD_Objet (3x4).

– double echelle [3]
Définit les dimensions du GD_Objet (3).

La documentation associée à cette structure a été générée à partir du fichier suivant :
– common.h

6.17 Référence de la classe Univers

`#include <Univers.h>`

Graphe d'héritage de la classe Univers

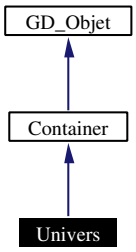
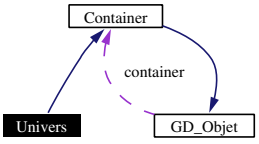


Diagramme de collaboration de Univers :



6.17.1 Description détaillée

Cette classe permet de creer un Univers.

Accesseurs

– void getTopLevelContainer (Container **_container)

Donne le container le plus haut dans la hierarchie.

– void setGravite (double gx=0, double gy=0, double gz=-9.81)
Ajuste la gravitée (par défaut à la gravitée terrestre).

– void simLoops (double step=0.05, int repeat=1)
Avance la simulation de un ou plusieurs pas.

– void **nearCallback** (void *data, dGeomID geom1, dGeomID geom2)

Membres publics

Constructeur/Destructeur

– Univers (double gx=0, double gy=0, double gz=-9.81, bool hashed=0)
Constructeur par défaut.

– ~Univers (void)
Destructeur.

6.17.2 Documentation des contructeurs et destructeurs

6.17.2.1 Univers (double *gx* = 0, double *gy* = 0, double *gz* = -9.81, bool *hashed* = 0)

Instancie un Univers.

Paramtres :

gx gravitée selon x

gy gravitée selon y

gz gravitée selon z

hashed valeur booléenne permettant d'utiliser une structure de hashage pour gérer les entitées de l'espace.

6.17.2.2 ~Univers (void)

Destructeur.

6.17.3 Documentation des méthodes

6.17.3.1 void getTopLevelContainer (Container ** *_container*) [virtual]

Fonction permettant d'obtenir le container du GD_Objet le plus haut dans la hierarchie.Ce container sera un univers. D'ailleur cette fonction est redéfinie dans la classe Univers.

Paramtres :

_container pointeur vers un pointeur de container

Redéfinie à partir de GD_Objet.

6.17.3.2 void setGravite (double *gx* = 0, double *gy* = 0, double *gz* = -9.81)

Ajuste la gravitée

Paramtres :

gx gravitée selon x

gy gravitée selon y

gz gravitée selon z

6.17.3.3 void simLoops (double *step* = 0.05, int *repeat* = 1)

Permet d'effectuer un ou plusieurs pas de simulation.

Paramtres :

step Définit en ms l'intervalle de temps à écouler entre chaque pas

repeat Définit le nombre de pas à effectuer.

La documentation associée à cette classe a été générée à partir du fichier suivant :
– Univers.h

7 Smile =) Documentation du fichier

7.1 Référence du fichier common.h

7.1.1 Description détaillée

Cette entête rassemble les déclarations communes à toutes les parties de smile.

Espaces de nommage (namespaces)

– namespace **std**

Composants

– struct structProperty

Types utilitaires.

Déclarations des types et structures standards.
– typedef double vecteur_3_1 [4]
– typedef double vecteur_4_1 [4]
– typedef double matrice_3_3 [4 ∗3]
– typedef double matrice_4_3 [4 ∗3]

Propriétés.

Déclarations des types et structures utiles. à la définition des propriétées.
– typedef enum gdobjetTypes type_t
– typedef structProperty property_t
– enum gdobjetTypes { **RIEN** = 0xffff, **CONTAINER** = 0, **UNIVERS**, **GROUPE**, **COLLIDER** = 10, **PLAN**, **SOLIDE** = 20, **BOITE**, **CCYLINDRE**, **SPHERE**, **JOINT** = 30, **PIVOT**, **ROTULE**, **GLISSIERE**, **RESSORT** }

7.1.2 Documentation du type

7.1.2.1 typedef double vecteur_3_1[4]

vecteur ligne permettant de définir une position, une direction...

7.1.2.2 typedef double vecteur_4_1[4]

vecteur ligne permettant de définir une position, une direction...

7.1.2.3 typedef double matrice_3_3[4∗3]

matrice permettant de définir des rotations ...

7.1.2.4 typedef double matrice_4_3[4∗3]

matrice permettant de définir des rotations ...

7.1.2.5 typedef enum gdobjetTypes type_t

Déclaration du type gdobjet (alias enum gdobjetTypes).

7.1.2.6 typedef struct structProperty property_t

Déclaration du type property (alias struct structProperty)

7.1.3 Documentation du type de l'énumeration

7.1.3.1 enum gdobjetTypes

Enumeration des types de GD_Objets existant.

8 Smile =) Documentation de la page

8.1 Liste obsolète

Membre GlRenderer Classe amie pour que le renderer puisse accéder aux objets repatriés depuis le serveur.

8.2 Liste des choses à faire

Membre GlRenderer Trouver mieux que cela pour le rendre indépendant.

Membre addGDObjet (GD_Objet ∗gdobjet) implémenter le dSpaceAdd (dSpaceID, dGeomID) ;

Membre remGDObjet (GD_Objet ∗gdobjet) implémenter le dSpaceRemove (dSpaceID, dGeomID) ;

Membre GD_Objet (Container ∗container=0) ameliorer le controle du paramètre.

Membre moveOf (double x, double y, double z) Proposer des fonctions utilisan les quaternions.

Membre Joint (Container ∗container=0, GD_Objet ∗_objet1=0, GD_Objet ∗_objet2=0) Résoudre le problème de relie : il faudrait que la methode soit appelée dans le constructeur, or ce n'est pas possible puisque le joint n'a pas encore été créé par ODE.

Membre setAxis (double ∗axis) assouplir la methode en acceptant un axe de longeur quelconque.

Membre plan_t A enlever.

Attention :

Ne plus utiliser. Utiliser plutot structPlan comme type.

Membre frustum_t A enlever.

Attention :

Ne plus utiliser. Utiliser plutot Frustum comme type.

Index