Newton Game Dynamics : les matériaux (Part. 2)

par Gallego Michaël (bakura.developpez.com)

Date de publication : 31/03/2007

Dernière mise à jour : 31/03/2007

Dans le tutoriel précédent, nous avons vu comment se servir des matériaux afin de définir des comportements qui changent suivant le matériel dont est constitué l'objet. Aujourd'hui, nous allons apprendre comment jouer un son lorsque deux objets se rencontrent, en utilisant les callbacks des matériaux.

Version de Newton Game Dynamics utilisée à l'écriture de ce tutoriel : 1.53

I - Introduction
II - Installation de OpenAL
    II-A - Utilisation d'OpenAL
III - Les callbacks utilisés par les matériaux
    III-A - Classe NewtMaterialPair
IV - Programme exemple
V - Conclusion
VI - Remerciements

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

I - Introduction

Ce tutoriel sera plus court que les autres, dans la mesure où il n'y a que peu de chose à ajouter par rapport au tutoriel précédent. Comme je l'ai dit en introduction, nous allons voir comment il est possible de jouer des sons lorsque deux objets se rencontrent grâce à un système de callback. En pratique ce système s'avère extrêmement puissant et permet d'aboutir à toutes sortes d'effets spéciaux très intéressants.

Pour jouer les sons, nous allons utiliser une API bien connue sobrement nommée OpenAL, puisqu'elle reprend les mêmes idées qu'OpenGL. La partie suivante sera consacrée à son installation puis comment nous allons nous en servir (je ne traiterai pas des sons 3D,... là n'est pas le but, mais sachez que c'est très facile à faire sous OpenAL).

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

II - Installation de OpenAL

OpenAL est donc une API de son multi plate-forme très puissante utilisée dans de nombreux jeux commerciaux tels que Doom 3, Ghost Recon ou encore l'excellent Psychonauts (que je vous recommande chaudement entre parenthèses !) et soutenue notamment par Creative. Contrairement à l'autre API sonore, FMod, OpenAL est plus bas niveau dans la mesure où elle ne propose aucun loader tout fait (sauf pour les fichiers WAV) et que son utilisation est nettement moins intuitive que celle de FMod. Toutefois, pour l'utilisation dont nous allons en faire, il n'y a rien de très compliqué.

Il vous faut tout d'abord télécharger le SDK à **cette adresse**. Si vous êtes sous Windows, téléchargez les fichiers OpenAL 1.1 Installer for Windows et OpenAL 1.1 SDK for Windows. Le premier contient des dll qui seront nécessaires pour que le son sorte bien, et que vous devrez redistribuer aux autres personnes qui utiliseront vos applications, tandis que le deuxième contient les fichiers d'en-tête et tout le reste. Enfin, téléchargez, dans la partie ALUT, freealut binary ZIP si vous êtes sous Windows. ALUT est à OpenAL ce que GLUT est à OpenGL, et nous permettra donc de simplifier l'utilisation d'OpenAL. Dézippez tous ça, ajoutez les fichiers "include" et "lib" de la même manière que d'habitude, et liez alut.lib et OpenAL32.lib. Pour finir, sachez que les outils indispensables de tout programmeur, la doc officiel, se trouve à **cette adresse !**

II-A - Utilisation d'OpenAL

Comme dit plus haut, l'utilisation d'OpenAL ressemble à celle d'OpenGL, pour l'utilisation des textures notamment. Après avoir initialisé OpenAL, la création d'un son se fait de cette manière :

1) On commence à créer un buffer qui servira à stocker le son, grâce à la fonction alGenBuffers.

2) On charge ensuite le son dans le buffer, soit par un loader perso, ce qui est conseillé dans une utilisation plus poussée (le format ogg notamment), ou sinon, on peut se servir de la fonction toute faite alutCreateBufferFromFile qui charge un son (WAV uniquement) à partir d'un nom de fichier.

3) On crée ensuite une source grâce à la fonction glGenSources. Un même son (donc un buffer crée via l'étape 1) peut appartenir à plusieurs sources en même temps (afin d'éviter la duplication de données par exemple), alors que l'inverse n'est pas possible.

4) Ensuite, on "attache" le buffer à la source. Cette étape se fait via la fonction alSourcei.

5) Puis on joue le son quand on le désire grâce à la fonction alSourcePlay.

6) Enfin, on oublie pas de libérer la mémoire grâce aux fonctions alDeleteSources et alDeleteBuffers.

Bien sûr, cette présentation d'OpenAL ne vaut pas une lecture attentive de la doc, puisque l'API propose évidemment des dizaines de fonctions très intéressantes, mais ici, nous nous en limiterons à l'usage présenté ci-dessus.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

III - Les callbacks utilisés par les matériaux

Newton Game Dynamics met à notre disposition trois callbacks en relation avec les matériaux. Voici le prototype de chaque callback :

int (*NewtonContactBegin) (const NewtonMaterial* material, const NewtonBody* body0, const NewtonBody* body1)

int (*NewtonContactProcess) (const NewtonMaterial* material, const NewtonContact* contact)

void (*NewtonContactEnd) (const NewtonMaterial* material)

Le premier callback est appelé chaque fois que les AABBs (Axis Align Bounding Box), c'est à dire les "boîtes englobantes" de deux objets rentrent en contact, ce qui permet de faire un premier test très approximatif (à partir de collisions de primitives simples, comme une boîte, plutôt qu'avec la primitive de collision complexe qui peut composer un corps). Si la fonction retourne 0, aucune action ne sera entreprie et les collisions seront tout simplement ignorées. Si la fonction retourne 1, ce sera alors le second callback (NewtonContactProcess) qui sera appelé. C'est dans ce callback que vous pourrez appeler toute sorte de fonctions. Si cette fonction retourne 0, le contact sera, comme pour le premier callback, ignoré, sinon le contact sera accepté et la collision gérée. Enfin, le dernier callback sera appelée une fois la collision effectuée. Dans notre cas, nous accepterons toutes les collisions (c'est à dire que NewtonContactBegin et NewtonContactProcess retournerons 1), et le callback NewtonContactEnd sera vide.

III-A - Classe NewtMaterialPair

Reprenons à présent la classe NewtMaterialPair de la dernière fois, et plus précisemment le fichier .hpp. Nous allons ajouter, juste avant la déclaration de la classe, une structure appelée StructEffetsSpeciaux, dont voici le code :

NewtMaterialID.hpp

```
struct StructEffetsSpeciaux
{
 StructEffetsSpeciaux ()
 {
  uiBuffer = uiSource = 0; // On initialise les deux variables à 0
 };

 ~StructEffetsSpeciaux ()
 {
  alDeleteSources (1, &uiSource);
      alDeleteBuffers (1, &uiBuffer);
 };

 void LoadSound (std::string nomFichier)
 {
  if (!uiBuffer && !uiSource)
  {
   alGenBuffers (1, &uiBuffer); // Création du buffer
   uiBuffer = alutCreateBufferFromFile (nomFichier.c_str()); // Chargement du son

   alGenSources (1, &uiSource); // Création d'une source
   alSourcei (uiSource, AL_BUFFER, uiBuffer); // Attache du buffer à la source
  }
  else
   std::cerr << "Ressources déjà crées";
 }
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

NewtMaterialID.hpp

```
    void PlaySound ()
    {
     alSourcePlay (uiSource);
    }

    ALuint uiBuffer; // Le buffer
    ALuint uiSource; // La source
   };
```

Le code est très simple : un constructeur qui se charge d'initialiser les deux variables à 0, le destructeur qui libère la mémoire, une fonction LoadSound qui crée un buffer, charge un son, l'attache à une source, et enfin une fonction PlaySound qui... joue le son !

Dans notre classe NewtMaterialPair, on ajoute donc une variable membre _structEffet de type StructEffetsSpeciaux, une fonction LoadSound qui se charge tout simplement d'appeler la fonction LoadSound de la structure (et évite ainsi que l'on touche directement à la structure), et enfin une fonction nommée NewtMaterialSetCollisionCallback.

NewtMaterialID.hpp

```
   class NewtMaterialPair
   {
      public:
         // Constructeur / Destructeur
         NewtMaterialPair (const NewtMaterialID * mat1, const NewtMaterialID * mat2);
         ~NewtMaterialPair ();

         /// ---------------------------------------------------------------------------
         /// FONCTIONS RELATIVES A NEWTON GAME DYNAMICS
         /// ---------------------------------------------------------------------------
         // Définit si deux matériaux rentrent ou non en contact
         void NewtMaterialSetDefaultCollidable (int etat);

         // Définit la "douceur" du matériel
         void NewtMaterialSetDefaultSoftness (const float softnessValue) const;

         // Définit l'elasticité du matériel
         void NewtMaterialSetDefaultElasticity (const float elasticityValue) const;

         // Définit les valeurs de friction du matériel
         void NewtMaterialSetDefaultFriction (const float staticFriction, const float kineticFriction)
    const;

         // Définit un callback pour la pair de matériaux
         void NewtMaterialSetCollisionCallback ();

         void LoadSound (std::string nomFichier) {_structEffet.LoadSound (nomFichier);}

      private:
         const NewtWorld * _nWorld; // Pointeur vers un objet NewtWorld

         const NewtMaterialID * _mat1; // Pointeur vers le matériel n°1
         const NewtMaterialID * _mat2; // Pointeur vers le matériel n°2

         StructEffetsSpeciaux _structEffet;
   };
```

Voici à présent la définition de cette fonction NewtMaterialSetCollsiionCallback :

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

NewtMaterialID.cpp
```
void NewtMaterialPair::NewtMaterialSetCollisionCallback ()
{
      NewtonMaterialSetCollisionCallback (_nWorld->GetNewtonWorld(), _mat1->GetID(),
                                          _mat2->GetID(), &_structEffet, NewtContactBegin,
                                          NewtContactProcess, NewtContactEnd);
}
```

Rien de très compliqué ici. On se charge d'appeler la fonction correspondante NewtonMaterialSetCollisionCallback dont le premier paramètre est un pointeur vers une structure NewtonWorld, le deuxième l'identifiant du premier matériel, le troisième l'identifiant du deuxième matériel, le quatrième un pointeur vers une structure à stocker. En effet, dans le callback, il sera possible de récupérer cette structure et ainsi de jouer le son. Enfin, les trois derniers paramètres sont des pointeurs de fonctions vers les trois callbacks. Ici, je les ait appelés NewtContactBegin, NewtContactProcess et NewtContactEnd. Ces trois fonctions sont déclarées static au début du fichier, et voici sans plus attendre leur définition :

NewtMaterialID.cpp
```
static int NewtContactBegin (const NewtonMaterial * material, const NewtonBody * body1,
                             const NewtonBody * body2)
{
   return 1;
}

static int NewtContactProcess (const NewtonMaterial * material, const NewtonContact * contact)
{
   const float MIN_SPEED = 0.55;

   StructEffetsSpeciaux * effet = static_cast <StructEffetsSpeciaux*>
 (NewtonMaterialGetMaterialPairUserData (material));

   float speed = NewtonMaterialGetContactNormalSpeed (material, contact);

   if (effet->uiBuffer && speed > MIN_SPEED)
   {
      effet->PlaySound ();
   }

   return 1;
}

static void NewtContactEnd (const NewtonMaterial * material)
{
}
```

Le premier callback renvoie toujours 1, ce qui a pour effet d'appeler le second callback, NewtContactProcess, ou toutes les choses intéressantes se passent. Notez que le premier callback peut vous permettre par exemple d'ignorer certaines collisions à des moments bien précis entre deux corps d'un certain type.

Le second callback est nettement plus intéressant. On commence tout d'abord par définir une constante nommée MIN_SPEED, que j'expliquerai plus tard. On récupère ensuite la structure stockée auparavant (rappelez-vous !) grâce à la fonction **NewtonMaterialGetMaterialPairUserData**, qui prend en paramètre un pointeur vers la structure NewtonMaterial qui est en paramètre du callback. Une variable nommée speed est ensuite créée et initialisée avec la valeur de retour de la fonction **NewtonMaterialGetContactNormalSpeed**. Cette fonction ne peut être appelée QUE dans ce callback (puisque c'est le seul à avoir comme paramètre un pointeur vers une structure NewtonContact. Cette fonction, très intéressante, permet de récupérer, comme son nom l'indique, la vitesse du contact d'après le vecteur normal au contact (le vecteur orthogonal donc).

Newton propose de nombreuses autres fonctions qui peuvent vous être utiles, comme la fonction **NewtonMaterialGetContactTangentSpeed** qui permet de récupérer la vitesse du contact d'après le

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

vecteur tangeant au contact et plein d'autres... Il est également possible de modifier les valeurs de "douceur", d'élasticité et les coefficients de friction que vous auriez préalablement définies (pour effectuer certains effets par exemple) grâce aux fonctions **NewtonMaterialSetContactSoftness**, **NewtonMaterialSetContactElasticity**, **NewtonMaterialSetContactStaticFrictionCoef** (friction statique) et enfin **NewtonMaterialSetContactKineticFrictionCoef**, respectivement. Pour avoir une description plus détaillé de chaque fonction, jetez un coup d'oeil à la doc du SDK, ou encore sur cette doc "améliorée" : **ici**, rubrique "Contact Behavior Control Interface".

Revenons à notre callback. On a ensuite un bloc if qui vérifie en premier lieu que le buffer uiBuffer soit différent de 0 (donc qu'un son ait été chargé), puis que la valeur de la vitesse stockée dans la variable speed soit supérieur à la constance MIN_SPEED. En effet, lorsque la vitesse est très faible (vers la fin du mouvement), Newton génère de très nombreux contacts, ce qui signifie que le son sera joué à chaque contact, et provoquera un résultat très laid. C'est pourquoi en dessous d'une certaine vitesse que j'ai défini, le son ne sera plus joué. La variable speed permettrait également, par exemple, de faire varier le volume du son via les fonctions proposées par OpenAL. Si les deux conditions demandées par le if sont vraies, alors on joue le son !

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

IV - Programme exemple

Allons à présent dans notre classe Affichage, et plus précisemment la fonction InitScene. Celle-ci est assez semblable à celle du tutoriel précédent, à quelques différences prêt. Tout d'abord l'initialisation d'OpenAL, dont nous n'avons pas encore traitée :

Affichage.cpp
```
// INIT DU SON
alutInit (NULL, NULL);
```

Je pense qu'il est difficile de faire plus simple ! On se contente d'un appel à alutInit qui se charge de tout initialiser à notre place. Il est possible de créer soit même ses contextes de périphérique en utilisant la fonction alutInitWithoutContext.

Pour le reste, on ajoute juste un appel pour spécifier les fonctions callbacks :

Affichage.cpp
```
    // 1-2
    pair2->NewtMaterialSetDefaultElasticity (0.9f);
    pair2->NewtMaterialSetCollisionCallback ();
    pair2->LoadSound ("boxSound.wav");

...
```

Ceci signifie que lorsque un objet de matériel 1 (ici c'est le sol), rentrera en contact avec un objet de matériel 2 (la boîte de droite, celle avec la plus forte élasticité), puisque j'ai effectué un appel à NewtMaterialSetCollisionCallback, le premier callback sera appelé. Puisqu'il retourne 1, le second callback sera appelé. Ayant spécifié un son et donc mis la variable _sound à true, le if sera vraie (jusqu'à ce que la vitesse descende en dessous de la limite spécifiée) et donc le son sera joué. Enfin, le dernier callback sera appelé.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

V - Conclusion

Ce tutoriel vous aura montré une utilisation simple des callbacks de "contact". Bien évidemment, Newton Game Dynamics propose d'autres fonctions qui permettent d'aboutir à d'autres effets, et je vous invite évidemment à les essayer. De plus, sachez que ce programme a été, comme pour les autres tutoriaux, conçu de façon a être facilement compréhensible, et que le système tel qu'il a été implémenté ici trouvera vite ses limites. Il sera par exemple bien plus aisé dans un programme plus important d'avoir la possibilité d'ajouter des callbacks "personnalisés" qui seront différents pour plusieurs objets, plutôt que d'avoir recours à ce genre de système de structure et utiliser pleins de bloc if. Dans le prochain tutoriel, nous verrons comment utiliser le "picking" sous Newton Game Dynamics, c'est à dire la possibilité de "prendre" un objet avec la souris, et de le déplacer, comme s'il s'agissait d'une main, ce qui s'avérera plutôt utile pour attaquer ensuite les joints, grosse partie que nous avons pas encore vu et qui permettent de "lier" plusieurs objets entre eux via certaines contraintes physiques (une poignée reliée à un sceau,...).

Source du programme (.zip) : **ici** (**miroir**)

Source du programme (.7z) : **ici** (**miroir**)

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

VI - Remerciements

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.