Un moteur de physique
Article 5 - Les contraintes

Gabriel Peyré
nikopol0@altern.org
www.orion3d.fr.st

Le 19 septembre 2001

Table des matières

Introduction

Cet article va nous permettre d'intégrer des contraintes dans la résolution des équations du mouvement présentées dans les articules précédents.

Chapitre 1

Introduction de contraintes

Avant de rentrer dans le vif du sujet (la résolution proprement dite), il convient de comprendre comment on va modéliser une contraintes entre plusieurs *rigid body*.

1.1 Variables utilisées

Pour commencer, la chose importante à noter est que l'on ne modifie pas la façon dont nos rigid body sont modélisés. En l'occurrence, on va utiliser des coordonnées généralisées, que l'on notera X_i pour chaque objet i. Il importe guère de savoir précisément ce qu'est X_i, disons, pour fixer les idées, un vecteur (position) et un quaternion (orientation).

Par contre, nous, on s'intéresse aux vitesse, et on va tracker ici un vecteur de 6 floats (3 pour la vitesse v, et 3 pour la rotation instantanée Ω), noté V_i. La vitesse globale de notre moteur de physique, on va la noter V, qui est un grand vecteur, composé des Vi mis bout à bout.

De même, pour chaque objet, on a une matrice M_i de taille (6,6), qui représente la façon dont la masse de notre objet est répartie. Elle est composée de deux matrices (3,3) placée sur la diagonale, en haut à gauche la matrice $m * Id$ (avec m la masse), et en bas à droite, la matrice I_{world}, la matrice d'inertie dans le repère global. On regroupe toutes ces matrices (6,6) dans une grande matrice de masse M, qui a sur sa diagonale toutes les matrices M_i. On notera W l'inverse de M.

Toujours de la même façon, on regroupe les forces et moments qui s'exercent sur l'objet i dans un vecteur de 6 floats F_i (en haut la force, en bas le moment), et on regroupe tous ces vecteurs force dans un grand vecteur F.

Donc au final, les équations de la mécanique du solide s'écrivent :

$$\frac{dv}{dt} = W * F \tag{1.1}$$

1.2 Expression des contraintes

Les contraintes sont exprimées sous forme de fonctions implicites. Ces fonctions prennent comme argument l'état du système, X. On écrira ainsi l'ensemble des contraintes sous la forme $C(X) = 0$.

Dans le cas d'une simple contrainte de pivot, la fonction C(X) n'a qu'une dimension (elle renvoie un float). Mais dans le cas général, notre contrainte C est constituée de pleins de contraintes $C(i_0,i_1,i_2,...)$, agissant sur les objets i_0, i_1, ... Ces mini-contraintes peuvent renvoyer un seul float (elles enlèvent juste un degrés de liberté, comme par exemple un pivot), mais peuvent en renvoyer plusieurs (comme par exemple un pivot avec interdiction de translation, qui enlève trois degrés).

Donc, lorsque l'on va implémenter nos contraintes, il va falloir que la contrainte, étant donné l'ensemble des objets, puisse nous donner la valeur $C(X)$. Par exemple, en prenant l'exemple d'un pivot, $C(X)$ sera juste la distance entre les deux points des deux objets où se réalise le pi-

vot. Mais toujours dans le cas général, chaque mini-contraintes $C(i_0,i_1...)$ sera responsable pour le calcul de certaines lignes du résultat total de la contraintes $C(X)$.

1.3 Matrice jacobienne et multiplicateurs de Lagrange

Mais la où les choses se compliquent, c'est qu'en fait, on n'utilise les contraintes que pour agir sur les vitesse. En fait, une fois que $C(X) = 0$ et $C'(X) = 0$ (on verra comment arriver à ce point un peu plus tard), on veut conserver cet état, et on veut donc que nos objets accélèrent dans la bonne direction, de telles sorte que l'on ai $C'(X) = 0$.

Pour résoudre $C'(X) = 0$, on a besoin de la matrice [1] :

$$J = \frac{\partial C}{\partial X} \tag{1.2}$$

C'est la matrice *jacobienne* de C (matrice des *dérivées partielles*), et c'est en fait elle la chose principale que notre objet "contrainte" devra calculer. Dans le cas d'un pivot, il s'agit juste de dériver la fonction $|p_1 - p_2|^*$, ou p_1 et p_2 sont les positions du point de pivot sur les deux objets.

Pour implémenter le calcul de J (et de J', par la suite), on va faire comme pour le calcul de C : modulariser. Ainsi, notre grosse contrainte est en fait découpées en petites contraintes $C_k(i_0,i_1,...)$ qui agissent sur un petit nombre d'objets. Chacune de ces mini-contraintes k va calculer des petits bloc de J, noté $J_k(i_0,i_1...)$, qui se situent sur la ligne k (la ligne de la mini-contrainte), aux colones i_0, i_1, ...

Donc, comme en général chaque mini-contrainte agit sur 2 ou 3 objets, la matrice J est très creuse : ceci va nous permettre d'accélérer les calculs grandement.

1. Il faut bien faire attention à ne pas se mélanger entre les dérivées par rapport au temps ($C'(X)$ par exemple), et par rapport aux position de l'objet ($\frac{\partial C}{\partial X}$ par exemple).

Chapitre 2

Résolution des contraintes

On s'attaque maintenant au coeur du problème : la résolution des contraintes, autrement dit, comment faire en sorte qu'elle soient toujours vérifiées, alors que les objets de notre système bougent de façon indépendantes.

2.1 Forces de contraintes

Il y a une chose évidente : si on laisse nos objets bouger tout seuls, les contraintes ne seront plus vérifiées, par exemple le point de pivot va lâcher. Pour contrer ceci, et faire que $C'(X) = 0$, on va agir très simplement : en ajoutant de nouvelles forces, dites forces de contraintes. Ainsi, on va modifier l'accélération, et les contraintes seront vérifiées. Dans la suite, on va noter ces forces Q (qui est composé, comme F, de pleins de petits vecteurs de taille 6, les Q_i, un par rigid body).

Notre but est donc de calculer les forces Q. Pour ce faire, il faut utiliser l'équation qui régit l'évolution des contraintes :

$$C'(X) = 0 \tag{2.1}$$

On obtient facilement $C'(X)$ par la formule de dérivation composée :

$$C' = J' * X' + J * X' \tag{2.2}$$

Notre objet "contrainte" va donc aussi devoir calculer J', par exemple par la formule :

$$J' = \frac{\partial C'}{\partial X} * \frac{\partial X}{\partial t} \tag{2.3}$$

Comme on a l'équation de la mécanique des solides :

$$\frac{\partial V}{\partial t} = W * (Q + F) \tag{2.4}$$

On a donc l'équation $C'(X) = 0$ qui s'écrit :

$$C' = J' * X' + J * X' \tag{2.5}$$

$$C' = J' * X' + J * W * (Q + F) = 0 \tag{2.6}$$

D'où la formule :

$$J * W * Q - J' * X' - J * W * F = 0 \tag{2.7}$$

2.2 Principe du travail virtuel

Le hic, c'est que ce problème a plus d'inconnues (6, celles de Q) que d'équations (nombre de lignes de C et de J). On doit dons ajouter des conditions.

La plus logique est de choisir des forces Q qui ne travaillent pas (ie. qui n'ajoute pas d'énergie). Ceci signifie qu'elles sont orthogonales aux vitesses, ie :

$$\overrightarrow{Q} * \overrightarrow{X'} = 0 \tag{2.8}$$

Les forces qui vérifient ceci peuvent en fait s'écrire comme combinaison linéaire des directions interdites, qui sont celles orthogonales aux gradients (les colonnes de J). Ceci signifie que la force est dans l'image de la transposé J^T de la matrice J. Donc on peut écrire :

$$Q = J^T * \lambda \tag{2.9}$$

Où lambda est un vecteur inconnu (de taille le nombre de lignes de J), nommé *multiplicateurs de lagrange*, qu'il nous faut calculer.

En combinant les équations 2.9 et 2.7, on obtient l'équation à résoudre :

$$J * W * J^T * \lambda = -J * X' - J * W * F \quad (2.10)$$

2.3 Stabilité et forces élastiques

Une chose importante à noter : en général, au départ, les objets sont peut être placés n'importe où. Donc on n'a pas les deux conditions préliminaires :

$$C(X) = C'(X) = 0 \quad (2.11)$$

Pour palier ce problème, on va ajouter des forces élastiques amortie dans l'équation (???) pour obtenir :

$$J * W * J^T * \lambda = -J * X' - J * W * F - k_s * C - k_d * C' \quad (2.12)$$

Où k_s est la raideur du ressort, et k_d le frottement. On peut les choisir plus ou moins au hazard, ils n'ont pas un rôle très important (juste rendre le système plus stable).

Chapitre 3

Implémentation concrète

Une fois préciser les équations qui régissent nos contraintes, voyons comment les résoudre en utilisant une implémentation orientée objet.

3.1 Implémentation orientée objet

Avant d'aborder la résolution de l'équation 2.12 la chose importante est de bien comprendre, comment, en philosophie orientée objet, tout ceci peut être implémenter de façon homogène.

Toutes les mini-contraintes sont stockées dans un manager de contraintes. Ce manager a pour rôle de résoudre les contraintes, c'est à dire :

- de faire calculer à chacune des contraintes qui le compose les quantités dont il a besoin (W, $J(i_0,...)$, $J'(i_0,...)$, $C(i_0,...)$, $C'(i_0,...)$).
- d'assembler ces résultats dans des grands vecteurs et des matrices creuses.
- de demander au moteur de physique les forces qui agissent sur chaque objet, et de les assembler dans un grand vecteur F.
- de calculer λ en résolvant l'équation 2.12. Il en déduit Q.
- d'ajouter a chaque objet les forces et moments qui composent Q.
- de rendre la main, pour que le moteur de physique résolve les équations de la mécanique du solide.

Chacune des contraintes doit donc être capable de :

- donner sa dimension (nombre de lignes qu'elle utilise dans C)
- calculer les bonnes lignes de C et C'.
- les blocs aux colonnes i_0, i_1, ... des matrices J et J'.

Pour faire tout ça, on a besoin des structures de données :

- des vecteurs de taille N quelconque.
- des matrices de taille (N,N) quelconque.
- des matrices creuses, qui sont des collections de matrices de taille quelconque, avec pour chacune sa position dans la matrice creuse.

3.2 Résolution du système

la dernière chose à savoir est comment résoudre l'équation 2.12. Cette équation peut être ré-écrite sous la forme :

$$M * x = b \tag{3.1}$$

Avec la matrice M qui est un produit de matrices creuses, et qui est définie symétrique positive.

Pour utiliser les matrices creuses, on va utiliser une technique de *gradient conjugué*, qui possède l'avantage de ne jamais accéder directement à la matrice M. A la place, elle a besoin de pouvoir faire les opérations $M * y$ et $M^T * y$, ce qui est facile à faire avec des matrices creuses. La technique

de gradient conjugué pour une matrice définie positive est très bien expliquée dans les **numérical recipies** [1], Il s'agit juste (!) de minimiser la quantité :

$$f(x) = \frac{1}{2} * |M * x - b|^2 \tag{3.2}$$

par améliorations successives. En effet, quand le minimum est atteint, le gradient df/dx est nul, ce qui signifie :

$$\frac{df(x)}{dx} = M * x - b \tag{3.3}$$

on a donc bien le résultat voulu.

3.3 Implémentation dans Orion3D

Nous venons donc de voir comment calculer les forces de contraintes qui permettent d'avoir en permanence de "bonnes" accélérations, qui permettent de toujours respecter les contraintes.

Voici maintenant un bref aperçu de la façon dont tout ceci est agencé dans **Orion3D**, de manière à vous donner des idées sur la façon dont les classes réalisant les différentes étapes de la résolution doivent s'organiser.

Chaque contrainte hérite d'une classe abstraite, `OR_RigidBodyConstraint_ABC`, qui stocke les rigid body sur lesquels la contraintes s'applique. Cette classe de base définie bien sûr les différentes fonctions pour calculer les données dont on a besoin pour résoudre les contraintes (nombre de degrés de libertés enlevés, matrices J_k, J'_k, etc.).

Des contraintes concrètes ont été écrite (pivot, glissière, rotule, etc.), qui constitue en quelques sortes des "briques de base" pour relier entre eux des rigid body.

Toutes ces contraintes sont rassemblées dans un manager de contraintes, de classe `OR_ConstraintManager`, qui, lorsqu'on lui demande de résoudre les contraintes, réalise l'algorithme détaillé un peu plus haut.

De plus, des classes de matrices quelconques (`OR_MatrixNxN`), vecteur quelconques (`OR_VectorNxN`) et matrices creuse (`OR_SparseMatrix`) ont été écrites.

1. http://www.nr.com

Conclusion

Comme vous avez pu le constater, la résolution de ces contraintes n'est pas aisée, et demande principalement :

- De bien répartir les fonctionnalités du moteur de contraintes entre les différentes classes.
- D'écrire une vraie librairie de gestion de matrices creuses et de vecteur de taille quelconque.

Cependant, le jeu en vaut la chandelle, car la résolution exacte des contraintes par cette méthode permet d'adopter de grand *time step* (contrairement aux méthodes par *pénalités*), et apporte beaucoup de stabilité.

Bibliographie

todo