Session 14 : Les Solveurs de Contraintes

> **Contextualisation**
>
> Dans notre moteur Verlet, nous déplacions directement les positions pour satisfaire les contraintes ($P_{\text{new}} = P_{\text{old}} + ...$). C'est ce qu'on appelle du **Position Based Dynamics (PBD)**.
>
> Les moteurs comme Rapier ou Box2D utilisent une approche différente, plus rigide, basée sur la vitesse : les **Impulsions Séquentielles**.

1. Qu'est-ce qu'une Contrainte ?

Pour un moteur physique, tout est une contrainte. C'est une règle mathématique $C(x)$ que le système doit respecter.

- **Contact (Non-pénétration) :** La distance entre deux objets doit être positive ($C \geq 0$).
- **Joint (Charnière) :** La distance entre deux points d'ancrage doit être nulle ($C = 0$).
- **Joint (Moteur) :** La vitesse angulaire doit être constante ($\delta\omega = k$).

Le rôle du solveur est de trouver les forces (ou impulsions) minimales à appliquer pour que $C(x)$ soit satisfait à la fin de la frame.

2. Rappel de "L'Impulsion" (La méthode Rapier)

Au lieu de calculer des forces (qui sont instables sur un pas de temps discret), les moteurs modernes calculent des **Impulsions**. Comme nous l'avons fait avec notre moteur Eurler. Une impulsion J est un changement immédiat de vitesse.

$$v_{\text{finale}} = v_{\text{initiale}} + \frac{J}{m}$$

L'objectif du solveur est de trouver le J exact pour que la vitesse relative au point de contact devienne nulle (ou rebondisse).

3. L'Algorithme : Projected Gauss-Seidel (PGS)

Dans votre TP1, mis en place un solveur de position. On voulait séparer les billes, mais quand elles sont toutes collées ensemble, il fallait itérer plusieurs fois pour les séparer complètement.

Résoudre toutes les contraintes d'un coup (Global Solver) demande d'inverser des matrices géantes. C'est trop lent pour le temps réel. L'industrie utilise une méthode **Itérative** extrêmement efficace: le **Projected Gauss-Seidel**.

Le concept des Impulsions Séquentielles. *Imaginez une table à 4 pieds sur un sol inégal.*

1. *Vous calez le pied A. Le pied B se lève.*
2. *Vous calez le pied B. Le pied C se lève.*
3. *Vous calez le pied C...*

*Si vous faites le tour de la table une seule fois, elle est bancale. Si vous faites le tour 10 fois (**Itérations**), elle finit par être stable partout.*

L'algorithme simplifié :

1. Pour chaque contact et chaque joint...
2. Calculer l'erreur de vitesse.
3. Calculer l'impulsion correctrice J.
4. Appliquer J immédiatement aux deux corps.

5. Répéter l'opération N fois (généralement 10 à 20 fois).

4. Les Secrets de la Stabilité

Pourquoi Rapier est-il plus stable que notre moteur Verlet ? Grâce à deux concepts avancés.

A. Warm Starting (Démarrage à chaud)

Le solveur est itératif. Il part d'une solution devinée pour s'approcher de la solution réelle.

- **Cold Start :** On suppose que l'impulsion nécessaire est 0. Il faut beaucoup d'itérations pour trouver la bonne force de support d'une pile.
- **Warm Start :** On utilise **l'impulsion de la frame précédente** comme point de départ.

Résultat : Une pile de caisses est stable immédiatement car le moteur "se souvient" de la force nécessaire pour la porter.

B. Sleeping (Endormissement)

Même avec le Warm Starting, des micro-erreurs de calcul (flottants) créent du "Jittering" (tremblement). L'Island Manager surveille l'énergie cinétique globale d'un groupe d'objets. Si elle passe sous un seuil ($E < \varepsilon$) pendant quelques frames :

- On force les vitesses à $(0, 0, 0)$.
- On désactive le solveur pour ces objets.
- L'objet passe en mode **Sleep**.

> **Comparatif des Solveurs**

Type	Avantages	Inconvénients
PBD (Notre Verlet)	Stable, impossible à "casser", facile à coder.	Aspect "mou" ou caoutchouteux. Gestion de la friction difficile.
Impulse Based (Box2D, Rapier)	Très rigide (Hard constraints), empilement stable, friction précise.	Peut exploser si dt varie ou si contraintes conflictuelles.
Penalty Based (Ressorts)	Gère bien les corps mous et tissus.	Instable pour les corps rigides (vibrations).

5. Paramétrer le Solveur (API)

Dans `Rapier` ou `Cannon.js`, vous avez souvent accès à ces paramètres dans `IntegrationParameters` :

- **dt (TimeStep) :** Doit être fixe (ex: $\frac{1}{60}$).
- **Velocity Iterations :** Nombre de passes pour résoudre les vitesses (collisions). Standard : 4 à 10.
- **Position Iterations :** Nombre de passes pour corriger le chevauchement (Pénétration). Standard : 2 à 5.
- **ERP (Error Reduction Parameter) :** À quelle vitesse corrige-t-on une erreur (ex: joint qui s'étire). Trop haut = ressort violent. Trop bas = joint mou.