

### Compléments / Aide à la mise en oeuvre

Sur la base des systèmes masse-ressort 1D vus en TP (<https://nimbus.u-pem.fr/s/aTfzNXokqH6pexC>), plusieurs développements peuvent être envisagés, en prolongement de certaines propriétés et/ou défauts de ces modèles.

Avant de détailler le travail qui vous sera demandé (projet), voilà quelques éléments complémentaires pour vous aider à appréhender ces modèles, faciles à écrire, mais parfois difficiles à utiliser. Cela correspond plus ou moins à ce qui était au menu des deux dernières séances (cf. Cours. Chap.4).

### Caractéristiques des modèles de départ

- Schéma d'intégration numérique explicite d'ordre 2 **LeapFrog**, potentiellement instable.
- Construction modulaires de type "assemblage" de briques élémentaires (**masses/ressorts**)
- Deux types de briques (**PMat** et **Link**), déclinables en un nombre quelconque de modules physiques élémentaires (particule / point-fixe / ressort / frein / liaison-conditionnelle ...)
- Modèle élémentaire construit (corde) : système à topologie fixe.
- Aucun modèle de gestion de collision.
- Moteur Physique : Simulateur générique "bas-niveau" :
  - ① Calcul et distribution de toutes les forces circulant dans le système entre 2 pas de simulation
  - ② Mise à jour de toutes les variables d'états (vit./pos.) par intégration numérique (LeadFrog)
  - ③ Appel au "moteur de rendu"
- Moteur de rendu totalement indépendant du Moteur Physique. Ils se contentent de partager certaines données (états des particules).

### Particules ou Masses-Ressorts

Fondamentalement, c'est la même chose. La différence réside dans les interactions entre les masses.

- **Masses-Ressorts** : les éléments matériels (masses) sont ponctuels et reliés par un réseau de liens (ressorts) dont la topologie est "fixe" (comme la corde ou le drapeau). Ce sont ces liens qui définissent l'espace entre les masses et donnent une forme et une structure aux objets.

La complexité du système dépend de la topologie (nombre de liens), mais reste en général linéaire (par rapport au nombre de masses).

- **Particules** : elles sont souvent présentées comme des "masses ponctuelles entourées d'un champs de force" (*SPH : Smooth Particles Hydrodynamics*), mais on peut faire exactement la même chose avec un système **Masses-Ressorts** entièrement connecté : chaque masse est reliée à toutes les autres mais les liaisons sont "conditionnelles" (actives seulement lorsque la distance entre les masses est inférieure à un certain seuil – qui définit la "taille" des particules) et un peu plus élaborées que de simples ressorts.

La complexité est alors en  $n^2$  pour  $n$  particules, et le recours à des structures accélératrices (subdivision de l'espace) devient vite nécessaire.

## Modèles 1D → 2D → 3D

Les modèles en dimension 1 restent très limités pour des rendus graphiques. Il faut au minimum passer en dimension 2 ou 3. La structure générale reste exactement la même. Ce qui change c'est les caractéristiques des composants de base (2 ou 3 coordonnées), et les calculs associés :




- les calculs de forces, vitesses... deviennent vectoriels  $\Rightarrow$  2 informations : module et direction
- les calculs de distances  $\Rightarrow$  distance euclidienne  $d(A, B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$
- moteur de rendu 3D  $\Rightarrow$  utiliser la lib `<libg3x>` (ou directement `OpenGL`, ou autre chose...)

Pour tester des **Systèmes de Particules**, la version 2D est bien suffisante et beaucoup plus simple à gérer (structures accélératrices). Pour des **Systèmes Masses-Ressorts**, la version 3D est bien sûr plus intéressante.

## Quelques "principes" à respecter

### Paramétrage

On ne s'intéressera ici qu'à des modèles très simples (masses et liaisons toutes identiques, limitées dans un premier temps à de simples "ressorts-amortis").

- Nombre de masses mobiles = Nombre de modes de déformation (cf. Cours Chap.4)  
Quelle que soit la topologie de connexion, un système à  $n$  masses libres se décompose en  $n$  modes de déformation (indépendants en 1D) : le *modèle modal*. En 2D ou 3D, ces modes ne sont plus réellement indépendants (interdépendance des dimensions dues au calcul de distances).
-  **Ce qu'il faut retenir** : augmenter le nombre de masses augmente le nombre de modes et donc le risque que les modes les plus élevés passent en zone de divergence (cf. Cours p.60).
- le paramètre de masse  $m$  n'est pas très utile en première approche
-  mettre toutes les masses à  $m = 1$ .
- les paramètres de raideur  $k$  et amortissement  $z$  des liaisons "ressort-amorti" sont liés au pas d'échantillonnage  $h = 1/F_e$  (cf. Cours p.37) : pour garder une "cohérence" physique à la simulation, il faut ajuster conjointement les 3 paramètres en gardant "constants" les paramètres algorithmiques ( $K_a = h^2 \frac{k}{m}$ ) et ( $Z_a = h \frac{z}{m}$ ).
-  **Ce qu'il faut retenir** : il est pratique de paramétrer les modèles de la manière suivante :
  - fixer  $h$  (ou  $F_e$ )
  - fixer  $K_a, Z_a$  (et  $M_a = m = 1$ ).
  - en déduire les paramètres des modules à simuler : ( $k = K_a * m * h^2$ ) et ( $z = Z_a * m * h$ )
- **attention** : ces relations sont établies pour des modèles 1D linéaires. Elles peuvent s'appliquer également en 2D et 3D mais la non linéarité (dimensions "liées") de ces modèles peuvent introduire des écarts de comportement.
- **divergence** : cela arrive très souvent !  
Lorsqu'une simulation diverge, cela peut avoir plusieurs origines :
  - tout explose dès le départ : cause la plus probable, le paramètre de viscosité est trop élevé. Essayer d'abord avec  $z = 0$ , si ça ne diverge pas, augmenter et observer.
  - ça diverge avec des oscillations : cause la plus probable, le paramètre de raideur est trop élevé ou la fréquence de calcul ( $F_e = 1/h$ ) est trop faible.
  - rien n'y fait, ça explose : un bug dans le code.... là, il n'y a plus qu'à le trouver et le corriger !

De manière générale :

- commencer par un modèle TRES simple (deux masses, un ressort), trouver une zone de paramètres à peu près stable, puis construire petit à petit en ajustant les paramètres.

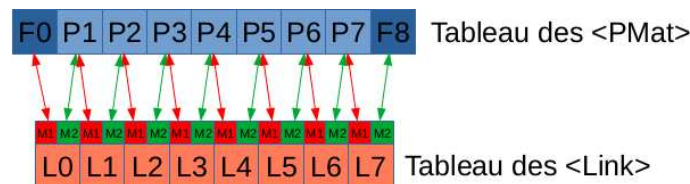
- autre point à retenir : si le nombre de modes de déformation ne dépend que du nombre de particules, la distribution de ces modes dépend lui de la topologie de connexion. Pour un même nombre de particules, plus la topologie est "dense" (beaucoup de liaisons), plus les modes seront rapprochés. Cela "abaisse" le mode le plus élevé qui est aussi le premier à provoquer la divergence (cf. Cours p.59-61).
- ☞ le bon paramétrage des modèles est, de loin, la partie la plus délicate dans l'usage des modèles physiques. C'est d'autant plus difficile que les meilleurs résultats sont obtenus en bordure des zones de stabilité (cf. Cours p.41).

## Modularité

Les exemples donnés en TP (en dimension 1) ne sont qu'une façon de faire parmi bien d'autres. Elle présente néanmoins un avantage certain : la modularité algorithmique. Cette modularité permet de construire des modèles génériques et homogènes, rend le moteur physique indépendant du modèle (exécute en aveugle) et permet une parallélisation efficace (*multithreads*).

Elle s'exprime de 2 façons :

- **modularité de construction** : le système repose sur 2 classes d'objets, les "masses" (<PMat>) et les "liaisons" (<Link>), et peut être vu comme un graphe :
  - une masse ("noeud" du graphe) ignore totalement son environnement. Elle se contente d'accumuler des informations (ici, des forces) via les liaisons qui la connaissent, puis de mettre à jour son état, toujours indépendamment de tout le reste.
  - une liaison ("arc" du graphe) ne peut exister dans la simulation que si elle est connectée à 2 "masses" : elle récupère les états de ses masses, calcule une force, et la distribue sur ses masses. Chaque liaisons est donc totalement indépendante des autres.
  - une masse peut être connectée à un nombre arbitraire (y compris 0!) de liaisons.



structure algorithmique de "corde" : 2 points fixes, 7 particules, 8 liaisons

- **modularité d'exécution** : le modèle final se réduit à deux tableaux (un pour chaque type de brique, masse ou liaison).
  - ☞ Sauf cas particulier (cf. beaucoup plus loin), ces tableaux n'ont à être soumis à aucun ordre précis : toutes les briques peuvent être stockées "en vrac", quelle que soit leur nature.

Le "moteur" de simulation se contente alors d'exécuter séquentiellement, et indépendamment de tout ordre, l'ensemble des algorithmes de chaque module de liaison (distribution des forces élémentaires), puis l'ensemble des mises à jours des états des modules de masse.

L'indépendance des traitements de chaque module permet une parallélisation efficace : chaque processeur (ou *thread*) gère une portion de chacun des tableaux. Seule contrainte : les *threads* gérant les liaisons doivent attendre que les *threads* gérant les masses se soient tous terminés.

**attention** : cette modularité à l'exécution n'est possible qu'avec des schémas d'intégration explicites (ou semi-explicites, comme LeapFrog). Avec un schéma tel que Euler Implicite, ce n'est plus possible (ou plus difficile) car la mise à jour des états doit prendre en compte l'intégralité de l'état du système.

## Jeu de construction

La classe d'objets <PMat> restera toujours très modeste : elle contient a priori seulement un objet, la *particule* mobile, et sa version "dégénérée", le *point fixe* (particule de masse infinie ou particule immobile).

On peut inventer des objets de type <PMat> plus exotiques (cf. plus loin) tant qu'ils respectent une règle de base : un objet <PMat> peut constituer un modèle simulable à lui seul.

☞ un modèle constitué uniquement de <PMat>, sans aucune liaison, doit être simulable (i.e. il ne doit pas faire "planter" le moteur).

La classe <Link> en revanche peut être aussi fournie qu'on le souhaite et contenir toutes sortes de liaisons plus ou moins complexes.

Toute contiennent au minimum deux références vers des points matériels  $M_1$  et  $M_2$  **connectés** et un algorithme renvoyant une force.

Nous avons déjà vu :

- le **ressort de Hook** : produit une force proportionnelle à son allongement
  - paramètres :  $k$  (raideur),  $l_0$  (longueur à vide, calculée sur les positions initiales de  $M_1$  et  $M_2$ )
  - force calculée :  $\vec{F} = -k * (d - l_0) * \frac{\overrightarrow{M_1 M_2}}{d}$  où  $d = ||\overrightarrow{M_1 M_2}||$
  - la masse  $M_1$  reçoit la force  $(+\vec{F})$  alors que  $M_2$  reçoit  $(-\vec{F})$
- le **frein cinétique** : produit une force proportionnelle à la vitesse relative des 2 masses.
  - paramètres :  $z$  (viscosité)
  - force calculée :  $\vec{F} = -z * (\vec{V}_{M_2} - \vec{V}_{M_1})$
  - la masse  $M_1$  reçoit la force  $(+\vec{F})$  alors que  $M_2$  reçoit  $(-\vec{F})$
- le **ressort-frein** : combinaison (montage "parallèle") des deux précédents.
  - paramètres :  $k$  (raideur),  $z$  (viscosité),  $l_0$  (longueur à vide)
  - force calculée :  $\vec{F} = -k * (d - l_0) * \frac{\overrightarrow{M_1 M_2}}{d} - z * (\vec{V}_{M_2} - \vec{V}_{M_1})$
  - la masse  $M_1$  reçoit la force  $(+\vec{F})$  alors que  $M_2$  reçoit  $(-\vec{F})$
- les **liaisons conditionnelles** : n'importe quelle liaison, agrémentée d'un test qui la rend active ou inactive. Quelques exemples utiles :
  - la liaison de type "inter-particules" : c'est par exemple un simple ressort-frein dont la force n'est calculée que si la distance entre les masses est inférieure à un certain seuil. Ce seuil d'interaction définit la "taille" des particules. Il peut être un paramètre de la liaison ou être calculé à partir des données lues dans les particules.
  - la liaison de rupture : c'est à peu près la même chose, mais la liaison est détruite ou rendue inactive si la distance entre les particules devient trop grande ou si c'est la force produite par la liaison qui dépasse un certain seuil.
    - ☞ permet de construire un modèle (très très élémentaire) de fracture.
  - à partir de là, on invente ce que l'on veut.... Quelques modules de ce type sont fournis dans le "kit" de démarrage 1D fourni en TP.

## Quelques cas particuliers de liaisons

Avec le système précédent, il semble difficile d'intégrer quelque chose d'aussi simple qu'une force de gravité s'appliquant sur les particules. Voici plusieurs options, de la moins bonne à la moins mauvaise.

- la particule "à gravité intégrée" :

cette méthode, très classique et très tentante (parceque très simple) consiste à ajouter cette force extérieure constante directement dans l'algorithme d'intégration :

$\vec{V}_M = (\vec{V}_M + \frac{h}{m_M}(\vec{F}_M + \vec{G}))$  où  $\vec{F}_M$  est le "buffer" d'accumulation de force de la particule M, alimenté par les liaisons connectées à M)

Cette méthode **n'est pas conseillée** car elle rompt la "modularité" et la "physicalité" du modèle : *une particule seule et isolée ne subit aucune contrainte.*

- le module <Link> "force externe" :

consiste à créer une liaison à peu près standard, dont le seul rôle est de distribuer une force constante (c'est le cas de la gravité) ou "pilotée" par une fonction externe (pour un modèle de vent, par exemple).

Le seul point de vigilance est que cette force n'a *a priori* qu'un seul point d'application dans le système. Il suffit alors de faire pointer l'autre vers une masse "poubelle" (NULL), ou vers un dispositif interactif (position/vitesse de la souris, joystick...).

C'est ce modèle qui est fourni dans le "kit" de démarrage 1D fourni en TP.

Principal défaut de cette méthode, pour un modèle de gravité ou de vent : sur un système à n particules mobiles, il faut créer n liaisons de ce type (une par particule).

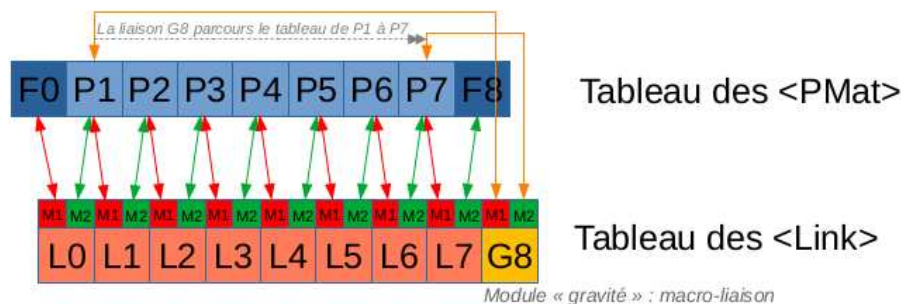
Ce type de module reste néanmoins **très** pratique pour interagir avec le système.

- le module <Link> "macro-liaison" :

c'est le plus pratique et le plus économique. Il consiste à créer une liaison connectée à un ensemble de particules (tout ou partie du tableau contenant tous les <PMat>). C'est donc la liaison elle-même qui "choisit" les particules sur lesquelles elle agit.

Concrètement ça ressemblerait à quelque-chose comme ça, pour une modèle de gravité simple :

- connexions :  $M_1 = 1^{\circ}$  particule,  $M_2 =$  dernière particule
- algorithme :  $P = M_1$ ; tant que  $P < M_2$  faire :  $\{\vec{F}_P + = \vec{G}; P = \text{suivante}; \}$



structure algorithmique de "corde" avec macro-liaison "gravité"