Université de Strasbourg

**D**épartement d'Informatique
Master 2 I3D et CMI IIRVIJ
Interactions dans les mondes 3D animés
Année 2019-2020

Travaux Pratiques Animation

# 1 Introduction

## 1.1 Les algorithmes de génération de mouvements de *points*

Pendant le cours, nous avons vu les différentes méthodes d'animation et, en particulier, l'animation par images clefs, qui offre une liberté infinie à l'animateur mais très peu de contrôle.

À l'opposée, avec les modèles dits *procéduraux*, le mouvement n'est pas produit par un animateur, mais par un algorithme. Ils permettent un bien meilleur contrôle, pourvu que la classe de mouvements visée soit plus restreinte. Pour caricaturer, si la classe de mouvements visée est celle des mouvements circulaires, un modèle procédural de mouvement circulaire produit des mouvements bien plus aboutis et contrôlés qu'avec des images clefs mais, bien entendu, on perd en généralité.

L'idée est de spécifier la classe de mouvements visés pour obtenir des algorithmes qui contraignent les degrés de libertés nécessaires et suffisants pour obtenir toute la classe visée et seulement la classe visée. Ainsi, quelque soit l'action de l'utilisateur, il obtient toujours des mouvements pertinents. Dans ce TP, nous allons découvrir les algorithmes les plus simples de génération de mouvement. Parmi les algorithmes de génération de mouvement, on peut distinguer les modèles analytiques, les modèles cinématiques et les modèles dynamiques. Nous en verrons quelques de chaque type.

Les mouvements créés peuvent s'appliquer à différents types de formes : des objets géométriques indéformables (solides rigides), ou déformables (avec des sommets pouvant avoir des mouvements indépendants), avec une topologie constante (polygones ou surfaces paramétriques) ou variable (surfaces implicites) ou, tout simplement, des points.

Cette partie du cours s'intéresse aux mouvements et non aux formes. Donc nous allons choisir, la primitive avec la forme minimale : le point. C'est aussi celui qui est le plus simple à manipuler et qui permet souvent de reconstituer les autres formes.

## 1.2 L'outil de visualisation

### 1.2.1 *particleView*

L'objectif de ce TP est de concevoir et implémenter des algorithmes de génération de points en mouvement. Cela implique d'une part, un travail géométrique sur des points, des vitesses et/ou des forces. Cela implique aussi un travail de visualisation. Mais l'essentiel des compétences que je voudrais vous transmettre gravitent autour du premier point. Pour vous éviter de passer votre temps sur la conception et l'élaboration d'un outil de visualisation de points en mouvements, je vous invite à découvrir l'outil *particleAnim* dont vous pouvez télécharger les sources depuis la plateforme *Moodle* et compiler en entrant `make`.

Cet outil prend en entrée un fichier texte décrivant le mouvement des points (et quelques primitives de forme très simples points, cercles et segments de droite) et permet de visualiser ces mouvements en temps-réel. *"Temps-réel"* signifie que, si la description spécifie une animation de $N$ images avec un pas de temps de $dt$ secondes, alors le mouvement visualisé durera effectivement $N.dt$ secondes dans le monde réel.

**Atelier *particleView :***

- Télécharger l'archive `particleView.tgz` sur *Moodle* ;
- Décompresser ;
- Entrer dans le répertoire et taper *make* ; Vous devriez obtenir l'exécutable `particleView`
- Télécharger le fichier d'entrée `01_vagues.pv` depuis *Moodle* ;
- Entrer la commande :
  `./particleView 01_vagues.pv`
- Essayer les commandes précédentes ;
- Pour avoir la version temps-réel, un peu moins stable, entrer la commande :
  `./particleView 01_vagues.pv -s`

Voici quelques commandes utiles :

1. Navigation spatiale (avec flèches ou souris) :
   - `r` mode rotate camera (défaut)
   - `t` mode translate camera
   - `z` mode zoome/dézoom
   - `+/-` avancer/reculer camera
   - `i` remettre la camera à sa position par défaut
2. Navigation dans le temps ;
   - `Entree` pour lancer/arrêter l'animation ;
   - `o` pour revenir au début de l'animation ;
   - `s` pour avancer pas à pas.

Dans ce TP, votre objectif sera de créer des animations sous la forme de fichiers d'entrée du type de `01_vagues.pv` pour *particleView*. Je vous demande d'utiliser cet outil. Si vous voulez en utiliser un autre, je ne m'y opposerai pas nécessairement. Je vous invite à m'en parler et à me présenter l'outil en question.

### 1.2.2 Exemples de fichiers d'entrée

Le format des fichiers d'entrée de *particleView* est décrit en détail dans l'appendice A. Mais un exemple est peut-être plus parlant. Sur la plateforme *Moodle*, vous pourrez trouver trois exemples de fichiers de ce format : `00_noanim.pv`, `01_battement.pv` et `01_vagues.pv`. Voici le fichier `00_noanim.pv`. Qui comporte 4 points et 2 images. La position des 4 points est identique entre la première et la seconde image.

```
#PV==             // magic number.
1.0               // pas de temps de l'animation (en secondes)
4                 // nombre de points
2                 // nombre d'images dans l'animation
oLine 0:2
oLine 1-3
Sphere 2:2 0.5
Point 2:3
====
-1 0 -1           // la position du 1er pt 1er image
1 0 -1            // la position du 2eme pt 1er image
1 0 1
-1 0 1
-1 0 -1           // la position du 1er pt 2eme image
1 0 -1
1 0 1
-1 0 1
```

Comme tous les fichiers de ce format, il est divisé en trois partie :

1. Une **entête** composée des quatre premières lignes. Seul le premier mot de chaque ligne est lu. Le reste est ignoré
2. La **visualisation** qui indique comment les points seront représentés. (Cette visualisation reste constante pour toute l'animation).
3. L'**animation** commence après la ligne qui commence par un caractère `'='` et comporte la position de tous les points à chaque image à raison d'un point (3 chiffres) par ligne.

Dans le fichier ci-dessus :

— `oLine 0:2` signifie que les points d'indice 0, 1 et 2 devront toujours être reliés par des segments de droite. Il y aura donc toujours deux segments : 0-1 et 1-2.
— `oLine 1-3` signifie que le point d'indice 1 devra toujours relié au point d'indice 3 par un segment de droite ;
— `Sphere 2:2 0.5` signifie que le point d'indice 2 devra être représenté par une sphère de rayon 0.5 ;
— `Point 2:3` signifie que les points d'indice 2 à 3 (2 et 3 donc) devront être représentés par un point.

**Atelier *format pv :***

— Télécharger le fichier d'entrée `00_noanim.pv` depuis *Moodle* ;
— Modifier ce fichier (dans un éditeur de texte) et relancer `particleView`, pour vous assurer que vous maîtrisez bien le format.

### 1.2.3 Générer des fichiers `.pv`

Vous pouvez générer les fichiers au format `pv` en C, en C++, en *Ocaml* ou en *Python*. Si vous voulez utiliser d'autres langages, je ne m'y opposerai pas nécessairement. Je vous invite à m'en parler. Vous pouvez télécharger, depuis *Moodle*, les fichiers `00_noanim.c, 01_battement.c, 01_vagues.c` ou les fichiers `00_noanim.py, 01_battement.py, 01_vagues.py` ayant servi à générer les fichiers `.pv` correspondants. Vous trouverez également les fichiers `Vecteur.h` et `Vecteur.c` ainsi que `Vecteur.py`.

**Atelier génération d'animation**

— Modifier `01_vagues.py` ou `01_vagues.c` pour ralentir ou accélérer le rythme des vagues.
— Selon les cas, entrer :
`gcc Vecteur.c 01_vagues.c -lm`
ou
`python3 01_vagues.py`
pour générer un nouveau fichier et l'essayer sur *particleView*.

# 2 L'animation analytique

Nous entrons enfin dans le vif du sujet. L'animation paramétrée n'a pas besoin de la notion de vitesse ou de force. La position des particules est déterminée par une expression mathématique ou un enregistrement et ne dépend absolument pas de la position des autres points. Les animations `01_battement` et `01_vagues` sont de ce type. La position des points en $x$ et en $z$ est constante. Seule la composante $y$ est déterminée par une fonction sinusoïdale dont on peut contrôler la pulsation, l'amplitude et le déphasage.

**Atelier mouvement circulaire**
Écrire un nouveau programme qui génère un nouveau fichier `02_cercle.pv` représentant un point qui tourne autour de l'origine dans le plan (x,z) avec un mouvement circulaire uniforme (au moins un tour). Assurez-vous que vous pouvez contrôler le rayon, la vitesse de rotation et la position de départ sur le cercle.

Pour cela, je vous rappelle que l'équation paramétrique d'un cercle centré à l'origine et de rayon $R$, situé dans le plan $xy$ est la suivante :

$$\begin{cases} x = R\cos\theta \\ y = R\sin\theta \end{cases}$$

Ainsi lorsque $\theta$ parcourt l'intervalle $[0, 2\pi]$, le point $P$ dont les coordonnées sont décrites ci-dessus parcourt tout le cercle. Reste à décider la relation que vous voulez établir entre $\theta$ et le temps $t$ et le numéro de l'image $fr$ pour que le mouvement de $P$ soit circulaire uniforme. Autrement dit, calculer $\theta$ en fonction de $fr$.

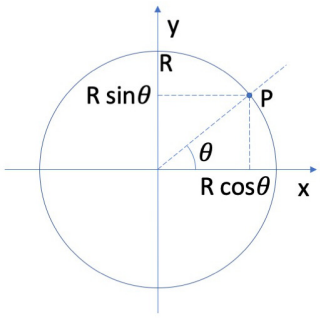


FIGURE 1: Un point $P$ situé sur le cercle a pour coordonnées $R\cos\theta$ et $R\sin\theta$.

**Atelier vortex**

Écrire un nouveau programme qui génère un nouveau fichier `03_tourbillon.pv` représentant au moins 200 points qui tournent autour de l'origine dans le plan (x,z) avec un mouvement circulaire uniforme sur des rayons choisis au hasard, les positions étant uniformément répartis sur le cercle. Comme sur un cyclone, on impose que la vitesse de rotation des particules soit inversement proportionnelle à la distance de la particule à l'origine. Autrement dit, plus on s'approche de l'origine et plus les points tournent vite. La cerise sur le gâteau serait que la densité de points ne soit pas plus grande au centre que sur les bords.

# 3 Les modèles de mouvement *cinématiques*

## 3.1 L'*advection* dans un champ de vitesses

Dans les modèles d'animation analytiques, il n'est pas nécessaire d'utiliser la notion de vitesse. Il suffit d'une expression mathématique qui gouverne la position des points. Dans ce paragraphe, nous allons voir un autre type de modèle de mouvement.

En temps discret, la vitesse $\overrightarrow{V}_n$ d'un point $P$ à l'instant $n$ est définie en fonction des positions $\overrightarrow{X}_n$ et $\overrightarrow{X}_{n-1}$ de ce point respectivement aux instants $n$ et $n-1$ et du pas de temps $T$ par la relation (1).

$$\overrightarrow{V}_n = \frac{\overrightarrow{X}_n - \overrightarrow{X}_{n-1}}{T} \quad (1)$$

$$\overrightarrow{X}_n = \overrightarrow{X}_{n-1} + \overrightarrow{V}_n.T \quad (2)$$

La relation (2) qui ensuit définit la notion *d'advection*. Supposons qu'on définisse, plus ou moins arbitrairement, un *champs de vitesses*. Autrement dit, à chaque point $P$ de l'espace on associe un vecteur vitesse $\overrightarrow{V}(P)$. Cela signifie que toute particule qui se trouverait en $P$ devrait avoir cette vitesse $\overrightarrow{V}(P)$. Ainsi si, en chaque point $P$, on connaît la vitesse des particules, alors, grâce à la relation (2), à partir de chaque position d'une particule, on peut avoir sa position à l'instant suivant.

Pour prendre une image, on peut imaginer le champs de vitesses comme un ruisseau. Quand on dit que les particules sont *advectés* par le champs de vitesses, c'est un peu comme si on mettait des particules flottant sur ce ruisseau et que celui-ci les emportait dans son mouvement.

## 3.2 Ateliers

**Atelier feu d'artifice**

- Écrire un nouveau programme qui génère un nouveau fichier `04_boom1.pv` représentant au moins 200 points placés au hasard dans le plan $y = 0$ ;
- Soit un point $I$ qu'on peut placer initialement à l'origine.
- On définit le champs de vitesses suivant : pour chaque point $P$ de l'espace, le vecteur vitesse est dans la direction $\overrightarrow{IP}$ (tend à éloigner les particules du point $I$) et dont l'intensité est inversement proportionnelle à la distance de $P$ à $I$. Autrement dit, plus un point $P$ est proche de $I$ et plus le vecteur vitesse l'éloigne de $I$ rapidement (cf figure 2).
- Si vous avez réussi, essayer aussi en plaçant $I$ à une auteur $y < 0$ (`04_boom2.py`).
- Une autre version : au lieu de relire le nouveau vecteur vitesse à chaque nouvelle position, on peut la lire une seule fois au départ de la simulation. Ensuite, à chaque image, on peut multiplier le vecteur vitesse par un nombre $k$ compris entre 0 et 1. Comment évolue la vitesse pour $k = 0.1$ ? et pour $k = 0.95$ (`04_boom3.py` et `04_boom4.py`) ?

**Atelier vortex**

Essayer d'écrire un programme qui génère un nouveau fichier `05_cercleCinematique.pv` créant un mouvement circulaire (1 seul point suffit) avec un champs de vitesses.

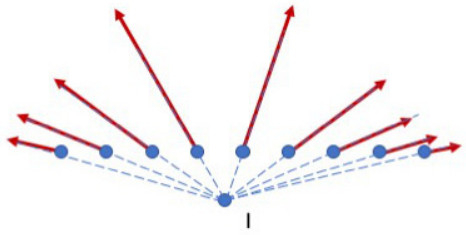

FIGURE 2: Un champs de vitesses répulsif d'autant plus intense qu'on s'approche de sa source $I$.

Ainsi, le mouvement d'une particule qui traverserait différents champs de vitesses, qui ne sont pas nécessairement constants peut produire des mouvements assez complexes. En tout cas, ils produisent facilement des mouvements pour lesquels on n'a aucune description par une fonction mathématique usuelle. Mais il peut s'avérer difficile d'obtenir exactement un trajectoire donné, comme par exemple des trajectoires circulaires comme dans la partie précédente.
D'un autre côté, les champs de vitesses que nous construisons sont, a priori, arbitraires. Ils peuvent produire des mouvements réalistes ou parfaitement incongrus. Dans le cas de l'atelier ci-dessus, le champs de vitesses ressemble fort à un champs de force. Mais il n'en est pas toujours ainsi. Avec un champs de vitesses, il n'est pas possible de rendre compte de plusieurs points qui intéragissent entre eux et avec leur environnement.

# 4 Les modèles dynamiques

Pour une vraie interaction, nous avons besoin de la notion de *force*, de la notion de *masse*. Les modèles ainsi créés sont appelés des modèles dynamiques (ou des modèles physiques). Dans ces modèles, on ne décrit plus les mouvements, mais la cause des mouvements. Le mouvement en résulte automatiquement.

## 4.1 Les lois fondamentales de la dynamique

*Isaac Newton* a établi trois lois de la dynamique :

1. Si la somme des forces appliquées à un objet est nulle, alors le mouvement de cet objet est un mouvement rectiligne uniforme ;

2. Si la somme des forces $\Sigma F$ appliquées à un objet de masse $m$ est non-nulle, alors l'objet subit une accélération $a$ telle que $\Sigma F = ma$
3. Si un objet $O_1$ exerce une force $F$ sur l'objet $O_2$, alors nécessairement l'objet $O_2$ exerce une force $-F$ sur $O_1$.

En temps discret, l'accélération $\overrightarrow{A}_n$ d'un point $P$ à l'instant $n$ est définie en fonction des vitesses $\overrightarrow{V}_n$ et $\overrightarrow{V}_{n+1}$ de ce point respectivement aux instants $n$ et $n+1$ et du pas de temps $T$ par la relation (3).

$$\overrightarrow{A}_n = \frac{\overrightarrow{V}_{n+1} - \overrightarrow{V}_n}{T} \quad (3)$$

$$\overrightarrow{V}_{n+1} = \overrightarrow{V}_n + \overrightarrow{A}_n.T \quad (4)$$

$$\overrightarrow{V}_{n+1} = \overrightarrow{V}_n + \frac{\Sigma \overrightarrow{F} T}{m} \quad (5)$$

Par ailleurs, on sait aussi exprimer la position en fonction de la vitesse (cf relation (2)). Il s'ensuit que :

$$\overrightarrow{X}_{n+1} = \overrightarrow{X}_n + \overrightarrow{V}_{n+1} T \quad (6)$$

$$\overrightarrow{X}_{n+1} = \overrightarrow{X}_n + \overrightarrow{V}_n T + \frac{\Sigma \overrightarrow{F} T^2}{m} \quad (7)$$

La relation (7) est à rapprocher de (2). On voit qu'au terme d'advection, toujours présent, s'ajoute un terme prenant en compte l'effet des forces appliquées. Mais à cela, il faut ajouter le fait que, contrairement à ce qui se passe dans les modèles dynamiques, les particules ont une inertie. Donc la relation entre la vitesse d'une particule à un instant et à l'instant suivant n'est pas arbitraire et est définie par la relation (5).

En résumé, on suppose que chaque particule est caractérisée par :
— une masse $m$ ;
— une position $X$ ;
— une vitesse $V$ ;
— la somme des forces $F$ qui lui est appliquée.

l'algorithme général de la simulation dynamique est le suivant pour chaque pas de temps.

1. Pour chaque particule :
   — Initialiser la somme des forces $F$ à $\vec{0}$ ;
   — Une par une, ajouter les forces appliquées à $F$ ;
2. Pour chaque particule :
   — Appliquer la relation (7) : réaliser une advection et y ajouter le terme correspondant aux forces ;
   — Appliquer la relation (5) : remettre à jour la vitesse de la particule ;

## 4.2 La gravité

**Atelier particle**

— Mettre au point nouvelle structure ou classe `Particle` ;
— Écrire une fonction permettant d'ajouter une force à celles déjà appliquées à la particule ;
— Écrire une fonction permettant de calculer la nouvelle position d'une masse en appliquant l'algorithme ci-dessus.

**Atelier gravite**

En utilisant l'objet particule et ses fonctions/méthodes :

— Écrire un programme avec une scène contenant une seule particule ;
— La seule force qui s'applique à cette particule est la force de gravité (masse x (0, -9.81, 0)) ;
— Calculer le mouvement des masses ;
— Même chose, mais en donnant une vitesse initiale à la particule : vous devrez obtenir la trajectoire parabolique habituelle.

## 4.3 Le frottement de l'air

La gravité est un champs. Nous n'avons pas effectué quelque chose de fondamentalement différent du modèle cinématique. À présent, appliquons une force qui dépend de la vitesse de la particule : la résistance à l'air. On suppose que la résistance à l'air est une force de même direction que le vecteur vitesse, mais de sens opposé. Souvent, on admet que la force $\overrightarrow{F_v}$ de résistance à l'air (qu'on appelle aussi la force de viscosité) est égale à :

$$\overrightarrow{F_v} = -\mu V^2 \overrightarrow{u} \tag{8}$$

où $V$ est la valeur de la vitesse et où $\vec{u}$ est un vecteur unitaire correspondant au vecteur vitesse. Mais ici, pour simplifier, nous allons supposer que cette force est proportionnelle et opposée à la vitesse :

$$\overrightarrow{F_v} = -\mu \overrightarrow{V} \tag{9}$$

**Atelier frottement de l'air**
Veillez à utiliser un pas de temps inférieur ou égal à 0.04. Toujours avec une seule particule, faire une simulation en exerçant sur cette particule la force de gravité et la résistance à l'air. Que se passe-t-il lorsqu'on choisit des valeurs élevées de $\mu$ ?

## 4.4 Le rebond

Nous allons utiliser les premières forces élastiques. Les forces élastiques sont celles dont l'intensité dépend non pas de la vitesse (comme le frottement de l'air) mais seulement de la position. On suppose que le sol se trouve au niveau $y = 0$ et on souhaite que la particule n'entre pas dans le sol. Pour cela, on applique à cette particule une force dirigée vers le haut et d'intensité proportionnelle à $y$ si celui-ci est négati.

$$\text{si } y < 0 \quad \overrightarrow{F_s} = -ky\overrightarrow{u}_y \tag{10}$$

$$\text{si } y \geq 0 \quad \overrightarrow{F_s} = \vec{0} \tag{11}$$

où $k$ est un coefficient d'élasticité qu'on appelle *raideur* et $\overrightarrow{u}_y$ est le vecteur $(0, 1, 0)$.

**Atelier rebond**
Dans un nouveau programme, compléter le programme précédent en appliquant à la particule une telle force. Là aussi, que se passe-t-il pour de grandes valeurs de la raideur $k$ ?

Lorsque ce modèle fonctionne, vous pouvez peut-être compléter le modèle en ajoutant de nombreuses particules qui rebondissent sur le sol.

## 4.5 Stabilité numérique

Vous avez du le voir, pour certaines valeurs de paramètre, la simulation s'emballe. Les positions et les vitesses tendent rapidement vers l'infini. On dit que la simulation *diverge*. C'est en particulier le cas quand les valeurs de raideur ou de viscosité dépassent une certaine limite.

**Atelier stabilité 1**
Essayer de reproduire une de ces situations. Par exemple celle de l'atelier frottement de l'air. Choisir une valeur de $\mu$ pour laquelle il y a divergence. Ensuite, baisser le pas de temps (par exemple 0.001). Est-ce que la simulation diverge toujours ? Cependant, en visualisation l'animation avec *particleView*, vous avez du voir passer des messages *Out of synch*. De fait, il est très difficile d'avoir une fréquence d'images de 1000 images par seconde.

Et d'ailleurs, est-ce que la vitesse des images justifie qu'on veuille avoir une fréquence d'images de 1000 images par seconde ? De fait, le choix du pas de temps dépend de deux critères :

1. La rapidité des images qu'on souhaite montrer
2. La stabilité de la simulation

Pour la rapidité des images, vous l'avez vu, il n'y a rien d'extrêmement rapide dans nos simulation. En tout cas rien qu'on ne puisse pas voir à la fréquence standard de 24 ou 25 images par seconde. Par contre, pour la stabilité, plus nos objets sont légers, raides et visqueux, et plus nous avons besoin de petits pas de temps. La solution est donc d'avoir deux pas de temps :

— Un pas de temps de visualisation (qui peut rester à 0.04 c'est à dire 25 images par secondes)

— Un pas de temps de simulation (qui dépendra des objets que nous devons simuler et qu'on peut choisir à 0.001)

Le rapport entre les deux fréquences est de 40. Dans le contexte d'une simulation interactive, il faudrait afficher une image tous les 40 pas de simulation. Dans notre cas, nous pouvons faire la simulation avec un pas de temps de 0.001 mais de n'enregistrer la position des points que tous les 40 pas de simulation. Par contre, le pas de temps à transmettre à *particleView* restera de 0.04.

**Atelier stabilité 2**
Avec ce schéma de calcul, reprendre l'atelier frottement de l'air et rebond avec de fortes valeurs de viscosité et de raideur. Assurez-vous que vous pouvez obtenir des frottements très visqueux et et des sols très durs.

### 4.6 Le ressort

Le cas général de la force élastique est celle qui s'exerce entre deux particules mobiles qui intéragissent véritablement. De tels phénomènes sont clairement en dehors de la portée des modèles cinématiques et analytiques. Cette force s'exprime de la façon suivante :

$$\overrightarrow{F_{12}} = k(l_0 - d)\overrightarrow{u}_{12} \quad (12)$$

$$\overrightarrow{F_{21}} = k(l_0 - d)\overrightarrow{u}_{21} = -\overrightarrow{F_{12}} \quad (13)$$

où :

$d$ est la distance entre les deux particules ;
$l_0$ est la longueur au repos du ressort ;
$\overrightarrow{F_{12}}$ est la force exercée par le point 1 sur le point 2 ;
$\overrightarrow{u_{21}}$ est le vecteur unitaire allant de 1 vers 2.

**Atelier ressort**
Dans un nouveau programme, sans gravité, créer deux particules reliées par un ressort et dont la distance initiale est différente de $l_0$. Observer les déformations du ressort. Est-ce que le ressort peut entrer en vibration. Si oui, est-ce que vous pouvez contrôler la fréquence de ces vibrations ?

**Atelier tetraedre**
Dans un nouveau programme, avec de nouveau de la gravité, créer quatre particules, chacune étant reliée à toutes les autres par un ressort, avec également des forces de rebond. Observer le tétraèdre trouver sa forme et rebondir sur le sol.

## A Format du fichier d'entrée

Voici les éléments qui doivent figurer dans le fichier d'entrée de cette application :

1. **L'entête** comporte 4 lignes portant 4 nombres. Tout ce qui est au delà du premier mot sur la ligne est ignoré
   — 1ère ligne : le *magic number* : `#PV==`

— 2ème ligne : la durée du pas de temps (en secondes)
— 3ème ligne : le nombre de points (qui doit rester constant)
— 4ème ligne : le nombre d'images (la longueur de l'animation)

2. La **visualisation** peut être décrite sur un nombre quelconque de lignes mais doit se terminer par une ligne commençant par au moins un caractère `'='`. Chaque description comporte :
   — un mot clef (de casse indifférente) spécifiant le type de primitive souhaitée :
      — `points` pour visualiser chaque particule par un point ;
      — `spheres` pour visualiser chaque particule par un cercle ;
      — `oLine` *open line* pour relier les particules par un segment de droite ;
      — `cLine` *closed line* pour relier les particules par un segment de droite, puis relier le premier et le dernier point ;
   — suivi de l'indice des points concernés par cette visualisation
      — `entier1:entier2` indique que la visualisation spécifiée par le motclef qui précédent concerne les points d'indice `entier1` jusqu'à `entier2` inclus.
      — `entier1-entier2` ne peut être utilisé que pour le motclef `oline` et indique que le point d'indice `entier1` devrait être relié au point d'indice `entier2` sans passer par les points intermédiaires.
   — pour les visus sphère, un nombre spécifiant le rayon de la sphère.

3. L'**animation** représente la position de tous les points, à raison d'un point par ligne (3 nombres x,y,z). Après le dernier point, on passe aux points de l'image suivante et ainsi de suite.