UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

INSTITUT D'ÉLECTRONIQUE ET D'INFORMATIQUE GASPARD-MONGE

M2SIS – 2014-2015
Moteur Physique
Projet

à rendre le 2 Mars 2015

Sur la base des systèmes masse-ressort *vus en TD, plusieurs développements peuvent être envisagés, en prolongement de certaines propriétés et/ou défauts de ces modèles.*

Caractéristiques des modèles de départ

- Schéma d'intégration numérique explicite d'ordre 2 Leapfrog, potentiellement instable.
- Construction totalement modulaires de type "assemblage" de briques élémentaires
- Deux types de briques (PMat et Link), déclinables en un nombre quelconque de modules physiques élémentaires (particule / point-fixe / ressort / frein / liaison-conditionnelle ...)
- Modèle construit (drapeau) : système à topologie fixe.
- Aucun modèle de gestion de collision et encore moins d'auto-collision.
- Simulateur générique "bas-niveau" :
  1. Calcul et distribution de toutes les forces circulant dans le système complet entre 2 pas de simulation
  2. Mise à jour de toutes les variables d'états (vitesses/positions) par double intégration numérique
  2. De temps en temps, appel au "moteur de rendu"
- Moteur de rendu totalement indépendant du Moteur Physique. Ils se contentent de partager certaines données (positions des particules).

Sujets

Thème 1 : Gestion des Auto-Colisions

En terme de complexité, un système masse-ressort à topologie fixe tel que le modèle de drapeau est de complexité linéaire par rapport au nombre de particules (moyennant un coefficient multiplicatif sur le nombre de liaisons, en fonction de la topologie de maillage).

Dès que l'on s'intéresse aux auto-collisions, les choses se compliquent puisque, dans l'absolu, chaque particule libre peut venir "heurter" n'importe quelle autre particule (libre ou fixe). On passe donc à un système de complexité quadratique par rapport au nombre de particule libres.

Il devient alors nécessaire de recourir à des méthodes d'optimisation. On trouve de nombreux exemples dans la littérature, mais la plupart sont basées sur des techniques de type "bounding box" permettant de déterminer les zones d'autocollisions.

La version la plus simple consiste à construire une grille (régulière ou hiérarchique) sur l'espace dans lequel évolue le modèle : à chaque instant on connait la distribution des particules dans les cellules de la grille et on effectue les calculs d'auto-collision cellule par cellule et sur des cellules voisines. On obtient donc une complelxité quadratique sur un nombre beaucoup plus réduit de particules.

Une difficulté est d'adater la résolution de la grille pour dimensionner au mieux le nombre P de cellules contenant des particules et le nombre moyen n de particules par cellule pour optimiser la complexité moyenne (de l'ordre de $P * n^2$).

Une autre difficulté est d'éviter de lancer des calculs d'auto-collision entre des particules liées topologiquement (ie. par le maillage du modèle). Une solution envisageable est de calculer, pour chaque cellule, une *signature* basée sur la *distance topologie* des particules qu'elle contient : deux particules topologiquement voisines apportent une contribution nulle à cette signature alors que 2 masses éloignées dans la structure du drapeau apportent une contribution plus importante.

Cette signature peut se calculer assez simplement lors de la phase de mise-à-jour des états (intégrations numériques), en même temps que la distribution des particules dans les cellules : il suffit pour cela de connaître (calculer ou stocker) les indices topologiques des particules dans le maillage du drapeau. Ainsi, seules les cellules possédant une signature *forte* déclencheront les calculs d'auto-collision.

Un autre méthode, plus simple, consiste à utiliser une matrice booléenne $\mathcal{B}_{(N \times N)}$, assez similaire à la matrice de connection du réseau (symétrique réelle), précisant si 2 particules sont susceptibles d'entrer en collision où non. Pour deux particules p_i et p_j se trouvant dans la même cellule et telles que $\mathcal{B}(i,j) = 0$ il faut tester la collision.

Là encore, la contrainte à respecter est que la gestion des cellules et les calculs d'auto-collision aient le moins d'impact possible sur le moteur de simulation.

Thème 2 : Collisions sur Points Fixes Géométriques

Il existe essentiellement deux méthodes pour la gestion des collisions, chacune ayant ses avantages et ses inconvénients les deux pouvant coexister dans un même modèle.

Par liaison conditionnelle

Cette méthode est simple et pratique pour gérer les auto-collisions et collisions entre deux objets mobiles (ce qui revient au même). Elle utilise des liaisons conditionnelles de type ressort-frein seuillé actives (production de force) uniquement lorsque deux particules sont proches (distance inférieure à un seuil, caractéristique de la liaison).

Avantages

Les principaux avantages de cette méthode sont sa simplicité et sa parfaite adéquation avec les principes de modularité et de généricité des systèmes *masse/ressort*. N'importe quel type de liaison peut, moyennant un seuillage sur la distance, servir à modéliser un choc entre points matériels.

Inconvénients

Les défauts sont malheureusement plus nombreux que les avantages. En premier lieu, le principe même des systèmes *masse/ressort* fait que les objets matériels sont ponctuels et que la géométrie d'une scène n'est portée que par la connexion de liaisons entre ces points et des calculs de distance. Ainsi, la distance euclidienne classique ne permet de modéliser que des objets sphériques. Heureusement on peut surmonter cette limitation (cf. plus loin).

De plus, avec des liaisons visco-élastiques, il est quasiment impossible de modéliser des objets rigides : la collision étant en fait un *choc élastique*, les objets restent en contact et se rapprochent jusqu'à ce que la force produite par le ressort les repousse au delà du seuil de collision. Ainsi, durant ce temps de contact, les deux points ont tendance à s'inter-pénétrer.

Enfin, le modèle de base de frottement étant le *frein cinétique* (force de frottement proportionnelle à la vitesse relative des points), il est très difficile de simuler des contacts prolongés. En effet, plus la vitesse relative diminue, moins le frein dissipe d'énergie et les objets ont tendance à glisser lentement l'un sur l'autre.

Par dynamique inverse

Cette méthode est beaucoup plus intéressante pour gérer les contacts prolongés et les glissements sur une surface plane, à condition que cette surface soit fixe (on peut également l'appliquer à des surfaces mobiles, mais c'est beaucoup plus délicat). Elle permet donc de définir des obstacles de forme quelconque du moment qu'ils sont facétisés.

Principe de fonctionnement

Une facette est caractérisée par ses sommets (A, B, C) qui délimitent une zone du plan et par la normale $\overrightarrow{N}$ à ce plan.

① *Détection* *(détermination du point d'impact)* :

Se fait par un algorithme très simple de type *intersection rayon-facette*, classique en infographie, adapté pour la circonstance. Le rayon est ici donné par la position courante $P(t)$ de la masse libre et son vecteur vitesse courant $\overrightarrow{V}(t)$.

L'algorithme commence par calculer la position qu'aurait le point à l'instant suivant en l'absence de collision et de toute autre force active, soit $Q(t) = P(t) + h\times\overrightarrow{V}(t)$.

On teste ensuite si le segment $[P(t), Q(t)]$ intersecte la facette. Pour qu'une collision entre la masse libre et la facette ait lieu entre les instants (t) et $(t+h)$, il faut tout d'abord vérifier que la masse est bien *au-dessus de la facette* et que sa direction de déplacement est bonne. Pour cela, si on note $\overrightarrow{N}$ la normale (constante) à la facette, il suffit de vérifier que la double inégalité suivante :

$$\begin{cases} \overrightarrow{N} \bullet \overrightarrow{AP(t)} > 0. & \text{la masse est au dessus du plan de la facette} \quad (a) \\ \overrightarrow{N} \bullet \overrightarrow{PQ(t)} < 0. & \text{la masse se dirige vers la facette facette} \quad (b) \end{cases}$$

En pratique, pour l'inéquation (a), on vérifiera plutôt : $(\overrightarrow{N} \bullet \overrightarrow{AP(t)} > \epsilon)$, où ϵ est un valeur strictement positive faible (10^{-6} par exemple). Cela permet d'éviter la plupart des cas d'inter-pénétration.

Si l'une des deux inéquations (a) et (b) n'est pas vérifiée, l'algorithme s'arrête. Sinon cela signifie que la demi-droite $(\overrightarrow{PQ(t)})$ traverse le plan de la facette en un point M à déterminer.

Pour conclure qu'il y a collision, il faut encore vérifier deux points :

1. Le point M doit se trouver entre $P(t)$ et $Q(t)$.
   Pour cela, on forme le rapport $u = -\frac{\overrightarrow{N}\bullet\overrightarrow{AP(t)}}{\overrightarrow{N}\bullet\overrightarrow{PQ(t)}}$ et on obtient le point $M = P(t) + u\times\overrightarrow{PQ(t)}$.
   Puisque l'on sait déjà que $(u > 0)$, il suffit de vérifier que $(u \leq 1)$. Si ce n'est pas le cas, c'est que $Q(t)$ est aussi au-dessus de la facette. Il n'y donc pas de collision dans cet intervalle de temps et l'algorithme s'arrête.

2. Si le point précédent est validé, il reste à vérifier que le point M trouvé appartient bien à la facette associée au point fixe. Il suffit de vérifier que les tous les produits vectoriels $(\overrightarrow{MA} \wedge \overrightarrow{MB})$, $(\overrightarrow{MB} \wedge \overrightarrow{MC})$ et $(\overrightarrow{MC} \wedge \overrightarrow{MA})$ ont même orientation.
   Si ce n'est pas le cas, alors M est à l'extérieur de la facette et l'algorithme s'arrête.

Si tous les tests précédents ont été passés avec succés, on peut alors conclure qu'un collision est prévisible. On met alors en place la réponse.

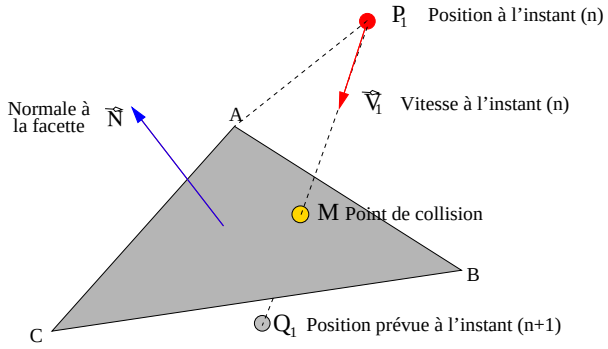
② *Traitement par dynamique inverse* *(calcul des forces de réaction)* :

Dans cette version, lorsque l'on a détecté une future collision, on traite la réponse non pas avec la position M du point d'impact trouvé, mais avec la position courante de la masse libre (là encore c'est pour limiter les inter-pénétrations).

1. On commence par récupérer la force $\overrightarrow{F(t)}$ stockée dans le buffer de force de la masse libre, et on la décompose, par rapport au plan de la facette, en une composante normale $\overrightarrow{F_N} = (\overrightarrow{F(t)} \bullet \overrightarrow{N})\overrightarrow{N}$ et une composante tangentielle $\overrightarrow{F_T} = \overrightarrow{F(t)} - \overrightarrow{F_N}$.
2. On fait de même pour la vitesse de la masse libre : $\overrightarrow{V_N} = (\overrightarrow{V(t)} \bullet \overrightarrow{N})\overrightarrow{N}$ et $\overrightarrow{V_T} = \overrightarrow{V(t)} - \overrightarrow{V_N}$.
3. Ici interviennent deux paramètres *physique* associés à la facette : un coefficient de *dissipation* α et un coefficient de *frottement* β. Le premier exprime la quantité d'énergie perdue lors du choc, c'est-à-dire la *dureté* de l'obstacle, le second exprime la résistance au glissement, c'est-à-dire la *rugosité* de l'obstacle ($(\alpha, \beta) \in [0,1]^2$).
   La réaction à la collision s'exprime alors de la manière suivante :

$$\overrightarrow{F'(t)} = \begin{cases} -\alpha\overrightarrow{F_N} & \text{si } ||\overrightarrow{F_T}|| < \beta||\overrightarrow{F_N}|| \quad \text{(rebond sans glissement)} \\ -\alpha\overrightarrow{F_N} + \left(1 - \beta\frac{||\overrightarrow{F_N}||}{||\overrightarrow{F_T}||}\right)\overrightarrow{F_T} & \text{sinon} \quad \text{(rebond avec glissement)} \end{cases}$$

   Et de même pour la vitesse $\overrightarrow{F'(t)}$.



Avantages

L'avantage principal est bien sûr la possibilité de modéliser des obstacles facétisés de forme quelconque qui se comporteront comme de vrais solides indéformables (il n'y a pas d'inter-pénétration).

Attention cependant pour les objets composés de plusieurs facettes : des fuites peuvent se produire aux jointures. Pour éviter celà il faut parfois tricher un peu...

Inconvénients

Un gros défaut ce cette méthode est qu'elle n'est pas très *physique* puisque la détection se fait sur des critères purement géométriques et le traitement (totalement déconnecté de la collision) consiste à décrire *l'effet* de la collision et non à *intégrer* une description des *causes*.

Mais c'est surtout à un niveau purement algorithmique que se pose le principal problème. Avec la méthode par liaison conditionnelle, une liaison de contact est traitée comme n'importe quelle autre, de manière totalement transparente pour le moteur de simulation. Ce n'est plus le cas ici : le fait que le traitement de la collision consiste à *symétriser* le vecteur de force stocké dans la particule fait que ce calcul doit *impérativement* être fait *après* que toutes les liaisons classiques ont apporté leur contribution à ce vecteur et *avant* que la particule ne mette à jour sa position en intégrant l'équation de Newton (avec le nouveau vecteur de force).

Avec cette méthode, le moteur de simulation doit donc fonctionner en trois étapes :

ⓐ Calcul des forces produites par les liaisons physiques classiques.
ⓑ *Gestion des collisions* sur objets facétisés.
ⓒ Mise-à-jour des états des particules.

Liaison conditionnelle sur distance non euclidienne

Jusqu'à présent, lorsqu'il fallait calculer des distances, il s'agissait toujours de la distance associée à la *norme euclidienne* $\mathsf{d(A,B)} = \sqrt{(\mathsf{x_B} - \mathsf{x_A})^2 + (\mathsf{y_B} - \mathsf{y_A})^2 + (\mathsf{z_B} - \mathsf{z_A})^2}$.

Une liaison conditionnelle de seuil r branchée sur un point A délimite une *sphére* de rayon r autour de A. Il suffira donc de tracer cette sphere pour avoir l'illusion qu'elle existe physiquement en tant qu'objet.

1^o exemple : la distance cubique et transformations géométriques

Mais la norme euclidienne n'est pas la seule envisageable. On connait par exemple la *norme du max* définissant la distance $\mathsf{d(A,B)} = \mathsf{Max}(|\mathsf{x_B} - \mathsf{x_A}| + |\mathsf{y_B} - \mathsf{y_A}| + |\mathsf{z_B} - \mathsf{z_A}|)$.

Dans une liaison conditionnelle classique, si on remplace la distance euclidienne par celle-ci, le volume délimité devient un *cube* de demi-côté r.

Le seul problème qui apparait est que, si la distance euclidienne est isotrope (une sphère n'a pas d'orientation), ce n'est plus le cas ici : le cube définit par cette distance est *parallèle aux axes* du repère.

Pour en faire quelque chose d'intéressant, il faut au moins associer à cet objet la possibilité de l'orienter. Pour celà il faut pouvoir lui appliquer des rotations et donc lui associer des matrices de transformation directes et inverses. Dès lors on peut aussi lui appliquer les autres transformations classiques : homothéties et translation.

Avec ces transformation, la distance euclidienne classique permet de construire n'importe quel type d'ellipsoïde et la distance cubique n'importe quel type de parallélépipèdes et autres *losanges*.

Un nouvel objet matériel et une nouvelle liaison

Ainsi, on peut définir un nouvel objet dans notre environnement masse/ressort (ou particule) : le *point fixe géométrique*. C'est un point fixe comme les autres mais possédant en plus deux matrices de transformation qui lui confèrent un repère local et surtout une *fonction de distance* qui lui est propre et qui n'est pas nécessairement la distance euclidienne. L'utilisation de matrices de transformation permet en plus d'utiliser des *seuil canoniques* de valeurs 1.

Bien sûr il faut lui associer une liaison un peu spéciale et prendre quelques précautions : cette liaison ne peut être utilisée qu'entre un *point fixe géométrique* et une mase libre, elle n'est pas symétrique (chaque point de connexion doit respecter un type précis), et elle devra exécuter son algorithme (du *ressort/frein seuillé* classique) en utilisant la fonction de distance et les matrices de transformation propres au point fixe.

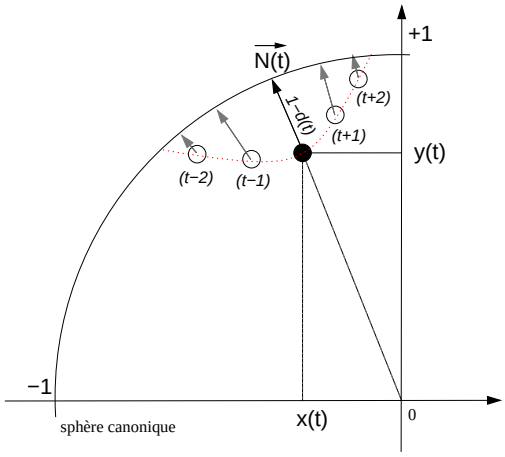
Structure de données

Ces nouveaux objets sont, du point de vue du simulateur, assimilables à des points matériels. Ils ont néanmoins une structure de donnée plus complexe.

- En tant que points fixes, ils n'ont pas besoin de vitesse ni d'accumulateur de force.
- En tant qu'objets volumiques il ont une position et une orientation : ces données sont gérées par des matrices de transformation classique (directe Md, inverse Mi)
- Et surtout ils ont une forme (i.e. un *in*terieur et un *ex*terieur), définie par une référence canonique et une *fonction de distance* qui renvoie, sous la forme d'un vecteur, la distance et la direction de pénétration d'un point P à l'intérieur de l'objet canonique

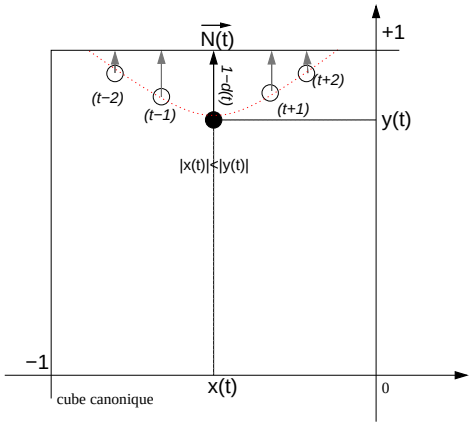
Par exemple, pour l'objet canonique <sphere>, la fonction de distance est définie par :

```
bool dist_sphere(Point P, Vecteur *N)
{
  double d = SQR(P.x)+SQR(P.y)+SQR(P.z);
           /* norme euclidienne */
  if (d>1.)
    return FALSE; /* le point P est dehors */
  d = 1./sqrt(d)-1.;
  N->x = P.x*d;
  N->y = P.y*d;
  N->z = P.z*d;
  return TRUE;
}
```



De même, pour l'objet canonique <cube>, la fonction de distance se définie par :

```
bool dist_cube(Point P, Vecteur *N)
{
  double d = MAX3(|P.x|,|P.y|,|P.z|);
         /* norme du max. */
  if (d>1.)
    return FALSE; /* le point P est dehors */
  N->x=N->y=N->z=0.;
  if (d==|P.x|)
    { N->x = P.x*(1./d-1.); return TRUE; }
  if (d==|P.y|)
    { N->y = P.y*(1./d-1.); return TRUE; }
  N->z = P.z*(1./d-1.);
  return TRUE;
}
```



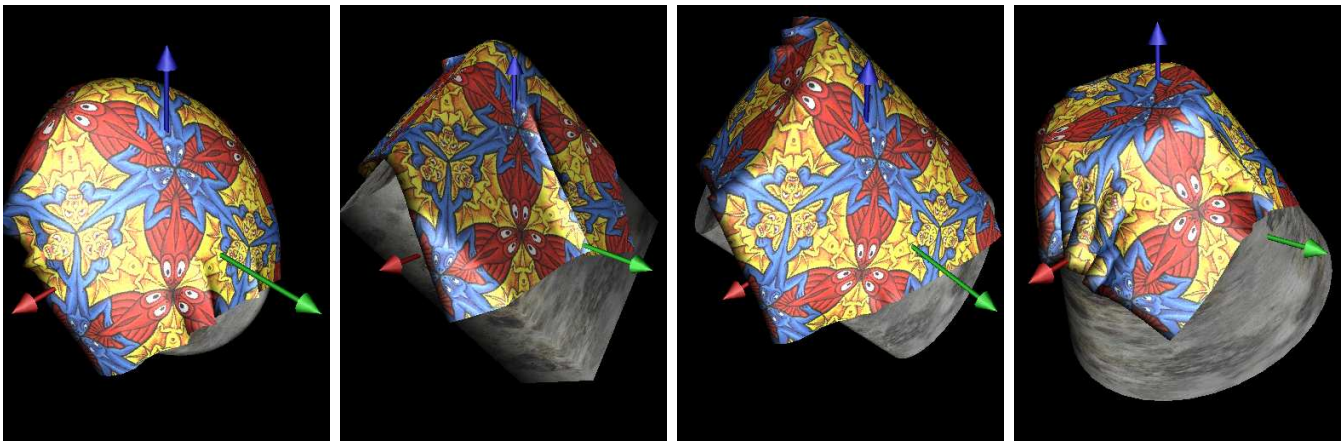
Ainsi, dans ce modèle, la collision est gérée une pénétration temporaire de la particule dans l'obstacle et non pas de manière instantanée comme dans le modèle par dynamique inverse. La particule "entre" par un endroit, et "sort" par un autre. A chaque instant passé à l'intérieur, un ressort-frein virutel produit une force de répulsion proporionnelle, en intensité et direction, au *vecteur de pénétration canonique* $\overrightarrow{N}(t)$

Mise en œuvre

Le principe d'utilisation de ce nouveau *point matériel* est le suivant :

1. Modélisation : on choisit la nature de la distance (sphere, cube...), les transformations qui définiront forme, position et orientation de l'objet géométrique souhaité (i.e. on construit ses deux matrices), on choisit la nature des interactions (ressort, frein...) de cet objet avec les particules libres.
2. Simulation : l'algoritme associé aux liaisons spéciales est le suivant :
   - a. envoyer la particule (position et vitesse) dans le repère canonique du point fixe (via sa matrice de transformation inverse).
   - b. tester la position de ce point 'transformé' par rapport à l'objet canonique (avec la fonction de distance propre à l'objet)
   - c. Si le point a pénétré l'objet (collision détectée), la fonction renvoie le *vecteur de pénétration*.
     - i. calculer la *force de répulsion canonique* appliquée à la particule transformée en fonction des la nature et des paramètres de la liaison.
     - ii. renvoyer cette force dans le repère global (via la matrice directe) et la sommer dans le *buffer de force* de la particule.

Avec ce principe on peut envisager de construire d'autres objets tels que cylindre, tronc de cône, tore...



Thème 3 : Rendu par Surfaces Implicites et GPU

Un autre sujet envisageable est l'utilisation d'un Modèle Physique limité (donc très dynamique, mais pas très beau) compensé par un moteur de rendu utilisant des surfaces implicites (Splines, NURBS...) s'appuyant sur le maillage physique pour interpoller les positions de l'objet entre les particules.

Ce dernier thème se marrie très bien avec un portage total ou partiel sur GPU.

Travail à réaliser

Il s'agit, a priori, d'un travail individuel. Chaque étudiant doit choisir un des thèmes proposés (ou en en proposer de nouveaux, ça n'est pas interdit...) et fournir un programme avec un maximum d'exemples d'illustration.

Ces programmes sont censés tourner sur les machines de l'université. Evitez donc si possible l'accumulation de bibliothèques exotiques et d'environnements spécialisés.

Pour le Thème.1 et la mise en œuvre de la méthode d'*Euler Implicite* des outils d'algèbre linéaire pourront néanmoins s'avérer utiles (lib. Boost ou Eigen par exemple).

Les Thèmes 1, 2 et 4 étant assez ouverts et méritant des tests comparatifs et quelques statistiques, un rapport devra accompagner le code.

Vous pouvez vous associer en binôme pour réaliser ces projets à condition bien sûr que deux thèmes soient traîtés. Chacun se "spécialisera" sur un des thèmes et sera évalué individuellement.

Les projets sont à rendre à la rentrée de Janvier 2014. Une séance de soutenance sera organisée à ce moment là.

Les principaux critères d'évaluation concerneront les aspects "dynamiques" des simulations proposées (réalisme, performances, modularité). Le "rendu" est un problème différent qui ne nous intéresse pas réellement ici. Un affichage simple suffira. Pensez plutôt "visualistion scientifique" que "rendu".