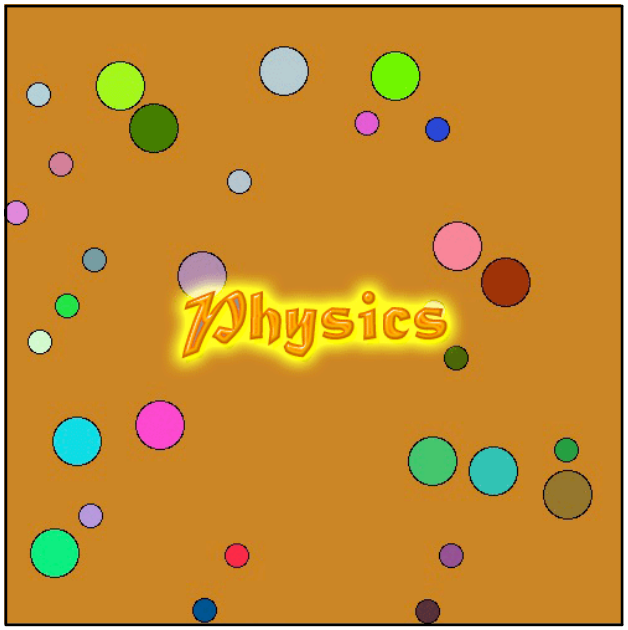


Projet réalisé par

CORLAIS Florian, JACQUEMIN Anthony

- **Préambule**

  Pour une bonne utilisation du présent slideshow...

- **Introduction**
- **Partie I : Physique du point**
- **Partie II : Quantité de mouvement et chocs**
- **Partie III : Mouvements de rotation**
- **Partie IV : Implémentation du projet**
- **Conclusion**

Préambule

Ce document existe en versions Slideshow et pdf. Vous trouverez, lors de son visionnage, ou de sa lecture :

- Des liens comme celui-ci, bleutés et soulignés, qui permettent, par une lecture via Slideshow, de profiter de l'intéractivité permise par Scheme.

- Des liens url, mauves et soulignés, qui permettent de se référer, dans les deux types de configurations, à des documents disponibles en ligne.

Préambule

Ce document est réalisé dans le cadre de l'option *Programmation fonctionnelle II* de la deuxième année de licence Mathématiques de l'université de Nice Sophia-Antipolis.

Pour en savoir plus, vous pouvez visiter le site de M. Roy, enseignant responsable du cours, à l'adresse http://deptinfo.unice.fr/~roy/. Pour toutes remarques, suggestions, ou pour nous signaler d'éventuelles erreurs, vous pouvez nous contacter par mail :

- Anthony : micheljacquemin@hotmail.fr
- Florian : amlc36@numericable.fr

Présentation

Notre projet s'est attaché à rendre physiquement cohérentes des simulations de mécanique, et nous avons ainsi pu aborder différents problèmes :

- La résolution de chocs entre particules élémentaires
  - Chocs axés (intervention de la conservation de la quantité de mouvement et de l'énergie cinétique)
  - Chocs désaxés (usage de la loi de composition des vitesses)
- La résolution de chocs entre des objets quelconques
  - Décomposition du mouvement (mouvements de translation et de rotation)
  - Traitement des chocs entre deux corps (tenant compte de la réaction normale, et de la friction lors de l'impact)

Présentation

Le traitement physique du problème passé, il a fallu implémenter les méthodes de résolution des différentes solutions. Pour ce faire, nous avons dû :

- Choisir des structures de données adéquates
- Résoudre le problème de calcul de l'enveloppe convexe d'un polygone (en O(n log(n)) par l'algorithme de Graham
- Gérer la détection de colisions entre deux polygones
- Trouver des exemples d'application pour ce slideshow

Présentation

Par ce document, nous tenterons de justifier au mieux nos choix, et de rendre compte, de la manière la plus exhaustive possible, les différentes connaissances mises en jeu dans notre projet...

- Préambule
- Introduction
- Partie I : Physique du point
- Partie II : Quantité de mouvement et chocs
- Partie III : Mouvements de rotation
- Partie IV : Implémentation du projet
- Conclusion

Introduction

"Notre Monde est en mouvement."

- Cette expression courante qui, *a priori,* ne se réfère pas à la physique, dénote néanmoins un fait remarquable : le *mouvement* est partout.
- Pour cette raison, la mécanique, a connu un engouement prononcé, et les outils nécessaires à son étude sont apparus très tôt. Pour s'en convaincre, il suffit d'isoler les deux notions utiles à la description du mouvement :
  - Les distances
  - Le temps

La mesure des longueurs

- De ces deux grandeurs fondamentales de physique, la première a été la plus facile à estimer : une fois l'unité de longueur définie, la mesure directe est possible, et facile à réaliser (du moins, pour de courtes distances).
    - Ainsi, dans l'Egypte antique, la coudée (65 cm) était l'unité en vigueur pour les travaux architecturaux et artisanaux, et des barres permettaient d'effectuer les mesures.

Barre mesurant la coudée (Egypte)

La mesure des longueurs

- De ces deux grandeurs fondamentales de physique, la première a été la plus facile à estimer : une fois l'unité de longueur définie, la mesure directe est possible, et facile à réaliser (du moins, pour de courtes distances).

  - Les mesures sont facilitées par des références à la vie quotidienne : dans la Grèce antique, les habitants parlaient en doigt (2cm), demi pied (32cm), coudée (16cm), brasse (1.86m) ou stade (185m).

Stade de Panathinaïko (Grèce)

La mesure des longueurs

- De ces deux grandeurs fondamentales de physique, la première a été la plus facile à estimer : une fois l'unité de longueur définie, la mesure directe est possible, et facile à réaliser (du moins, pour de courtes distances).
    - Aujourd'hui, les avancés technologiques permettent des mesures précises, de l'infiniment grand à l'infiniment petit (géolocalisation par satellites, microscope à balayage électronique, laser).

Prototype du mètre de 1889

La mesure du temps

- L'Homme n'est pas directement confronté au temps, il n'en ressent que ses effets. Sa quantification n'est donc pas aussi immédiate que la mesure des longueurs, mais de nombreuses solutions ont été apportées au cours de l'Histoire.

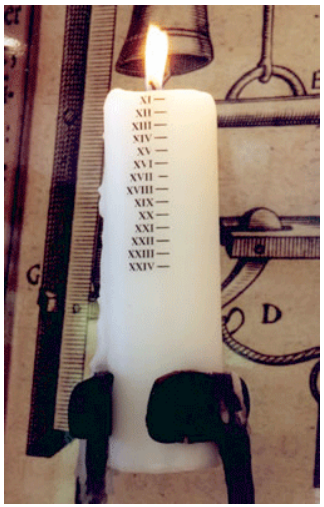
    - En Egypte, à l'instar du sablier, l'horloge à eau, permettait d'évaluer *l'écoulement* du temps, et de mesurer la durée de certains phénomènes, à l'aide de graduations gravées sur le récipient.

Clepsydre - Karnak (Egypte)

La mesure du temps

- L'Homme n'est pas directement confronté au temps, il n'en ressent que ses effets. Sa quantification n'est donc pas aussi immédiate que la mesure des longueurs, mais de nombreuses solutions ont été apportées au cours de l'Histoire.

  - A partir du neuvième siècle, la bougie est utilisée pour décompter le temps (et sera largement employée plus tard, au théâtre).

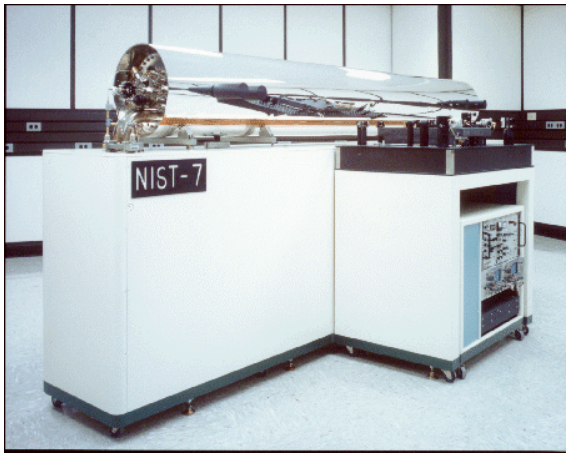


Horloge a bougie

La mesure du temps

- L'Homme n'est pas directement confronté au temps, il n'en ressent que ses effets. Sa quantification n'est donc pas aussi immédiate que la mesure des longueurs, mais de nombreuses solutions ont été apportées au cours de l'Histoire.

  - De nos jours, la mesure du temps peut se faire à la nanoseconde près, grâce à des moyens toujours plus précis : les chronomètres, les horloges à quartz, les horloges atomiques...



Une horloge atomique du NIST (USA)

Introduction

- Ces outils, indispensables à la description de phénomènes physiques, ont permis, très tôt, d'étudier des mouvements : qu'ils soient simples, comme les oscillations périodiques d'un pendule, ou plus complexes.
- L'essort de l'informatique au cours du XX° siècle a radicalement changé l'approche des problèmes de mécanique : la résolution de système d'équations par les machines, et leur rapidité d'exécution, ont rendu possible des modélisations jusqu'à lors inabordables. Pour simple exemple, prenons les dernières images, quasi-photoréalistes, d'une mer déchainée, extraites d'un jeu vidéo de la dernière génération de console de salon.

Call of Duty 4 (Playstation 3)

- **Préambule**
- **Introduction**
- **Partie I : Physique du point**

  Quantité de mouvement, théorème fondamental, lois de Newton
- **Partie II : Quantité de mouvement et chocs**
- **Partie III : Mouvements de rotation**
- **Partie IV : Implémentation du projet**
- **Conclusion**

Trajet linéaire

- Dans cette première partie, nous nous intéressons aux particules, caractérisées par leur masse, concentrée en un point.
- Lorsqu'un point effectue un trajet direct (en ligne droite) d'un point A vers un point B, il est facile de mesurer la distance qu'il parcourt. En choisissant une origine O sur (AB), nous avons la relation :

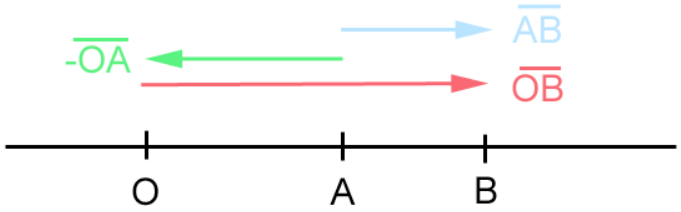
$$\overline{AB} = \overline{OB} - \overline{OA}$$

- Cette définition rend compte de l'orientation de la configuration du trajet, elle est relative au choix de l'origine, et peut ainsi être négative.

Quid des trajets quelconques ?

Trajet courbe

- Comme souvent, pour décrire un phénomène complexe, les physiciens se ramènent à un cas simple, parfaitement connu.
- Lorsqu'une particule effectue un trajet quelconque de A vers B durant un temps T, nous pouvons décomposer son trajet en N trajets élémentaires, linéaires, et appliquer la définition précédente.
- Vite, un exemple !

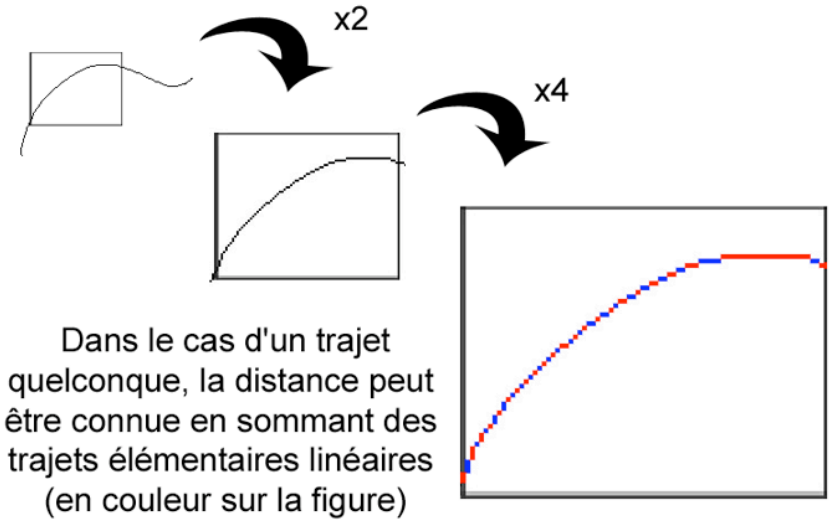


$$\overline{AB} = \overline{OB} - \overline{OA}$$

Cas du trajet linéaire

Trajet courbe

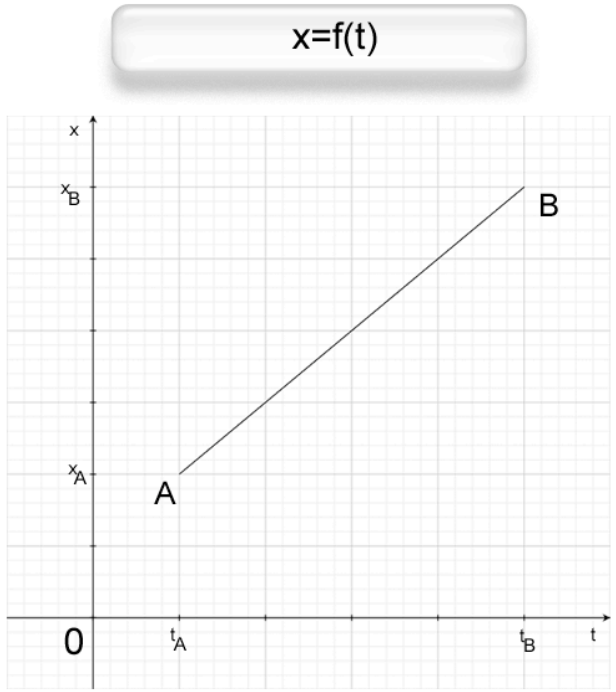
- Comme souvent, pour décrire un phénomène complexe, les physiciens se ramènent à un cas simple, parfaitement connu.
- Lorsqu'une particule effectue un trajet quelconque de A vers B durant un temps T, nous pouvons décomposer son trajet en N trajets élémentaires, linéaires, et appliquer la définition précédente.
- Vite, un exemple !



Dans le cas d'un trajet quelconque, la distance peut être connue en sommant des trajets élémentaires linéaires (en couleur sur la figure)

Description du mouvement ~ x=f(t)

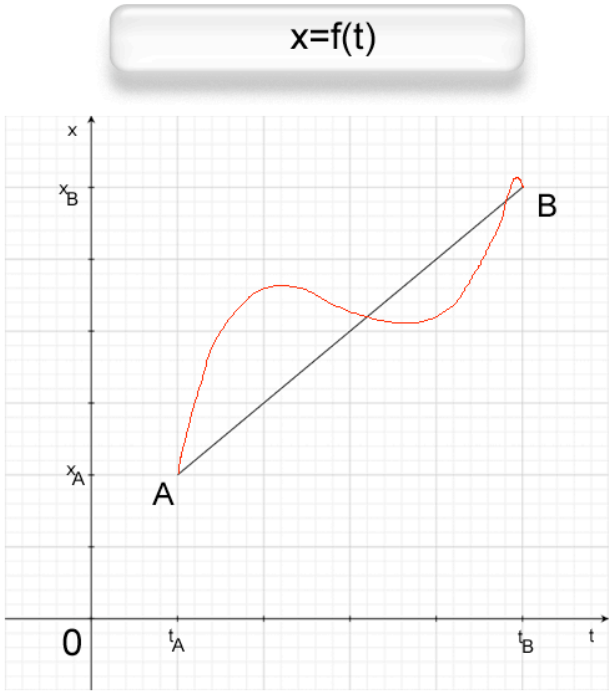
- Nous savons désormais mesurer la distance qu'effectue une particule lors d'un trajet, tout en connaissant la durée qu'elle met pour l'effectuer.
- Nous pouvons donc décrire le mouvement par une représentation graphique : dans le cas simple à une dimension (le mobile se déplaçant sur un axe (Ox)), il suffit de tracer la courbe x=f(t).



Cas d'un trajet rectiligne uniforme

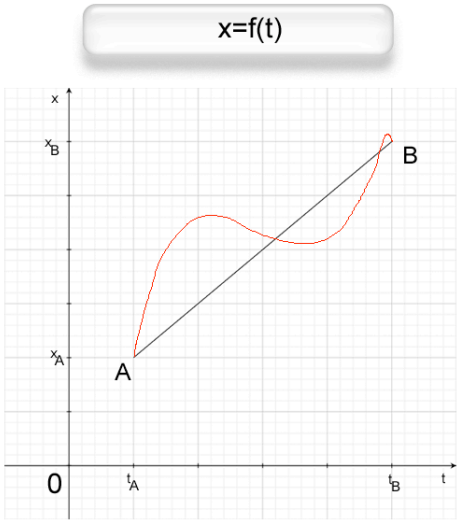
Description du mouvement ~ x=f(t)

- Nous savons désormais mesurer la distance qu'effectue une particule lors d'un trajet, tout en connaissant la durée qu'elle met pour l'effectuer.
- Nous pouvons donc décrire le mouvement par une représentation graphique : dans le cas simple à une dimension (le mobile se déplaçant sur un axe (Ox)), il suffit de tracer la courbe x=f(t).



Cas d'un trajet rectiligne quelconque

Description du mouvement ~ x=f(t)



- Vous pouvez ici vous familiariser avec ce type de graphique en contrôlant une masse se déplaçant sur une droite (AB) : des clics sur les bouton + (ou -) augmentent (ou diminuent) sa vitesse. Si la masse percute le point A, elle rebondit sans perdre d'énergie pour se diriger vers B.
- On pourra trouver une vidéo de démonstration à l'adresse : http://projetphysics.teria.org/Videos_du_slideshow/xft.html

Vitesse d'un point

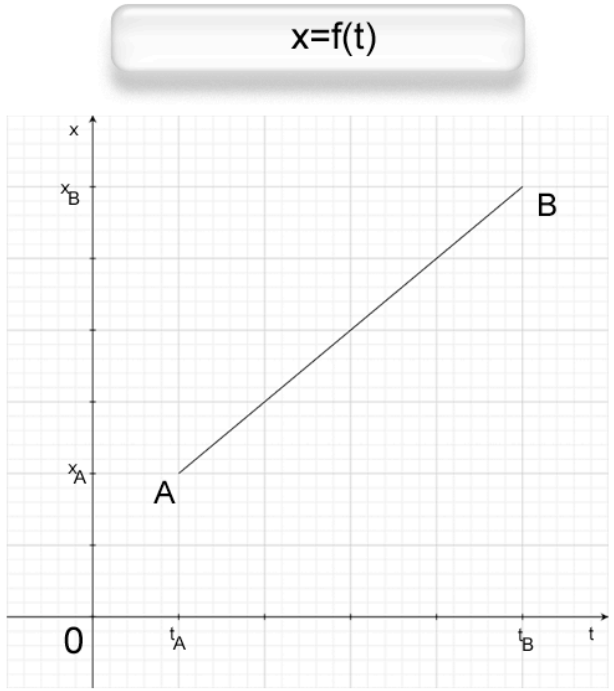
- La notion de vitesse nous est familière : il s'agit d'un rapport entre une distance et un temps. En prenant l'exemple d'une voiture partant du point A au temps tA, pour arriver au temps tB en B, nous avons coutume d'appeler vitesse moyenne :

$$v = \frac{AB}{t_B - t_A}$$

- Quelques remarques :
  - Une simple analyse dimensionnelle révèle que la vitesse n'est pas un rapport homogène : [V]=[L]/[T]. Ce fait a été, durant longtemps, une difficulté théorique pour décrire les mouvements (le même problème s'est posé en définissant des longueurs algébriques).
  - Comment interpréter sur notre précédent graphique ce terme de vitesse ?

Vitesse lors d'un mouvement rectiligne uniforme

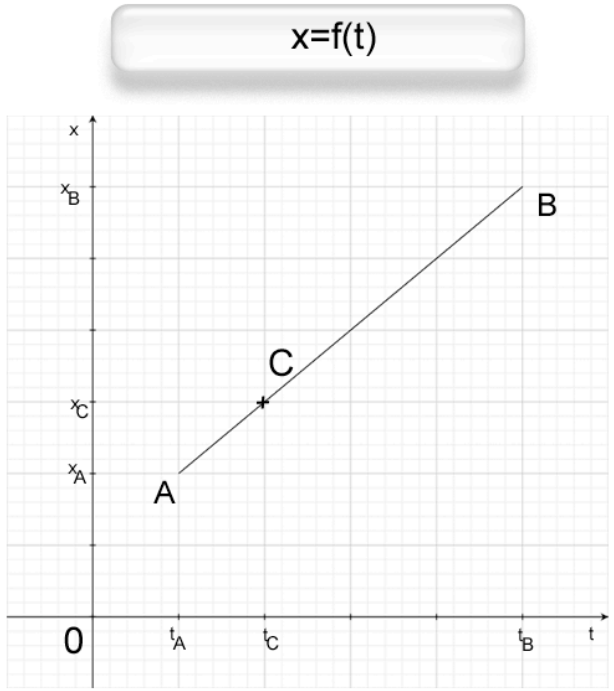
- Reprenons notre graphique x=f(t) dans le cas d'un mouvement rectiligne uniforme :



Mouvement rectiligne uniforme

Vitesse lors d'un mouvement rectiligne uniforme

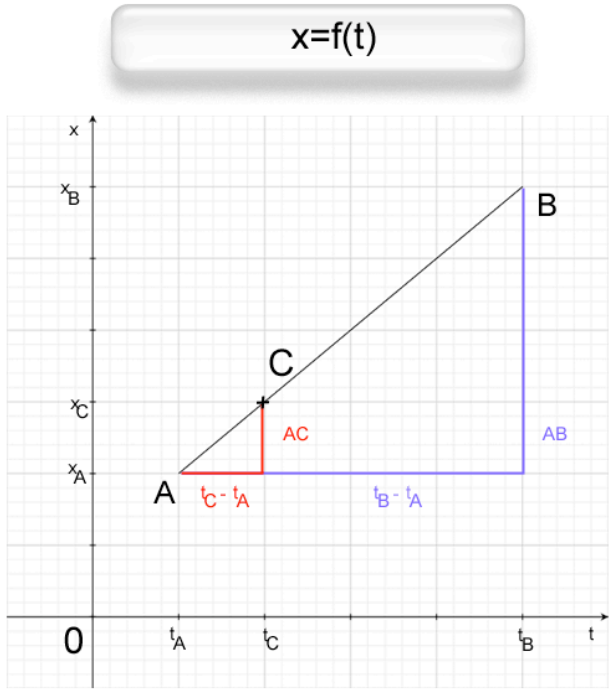
- Reprenons notre graphique x=f(t) dans le cas d'un mouvement rectiligne uniforme :



Soit C un point sur le trajet AB : d'après le théorème de Thales...

Vitesse lors d'un mouvement rectiligne uniforme

- Reprenons notre graphique x=f(t) dans le cas d'un mouvement rectiligne uniforme :



L'égalité des rapports $\frac{AC}{t_C - t_A} = \frac{AB}{t_B - t_A}$ nous permet de déduire :

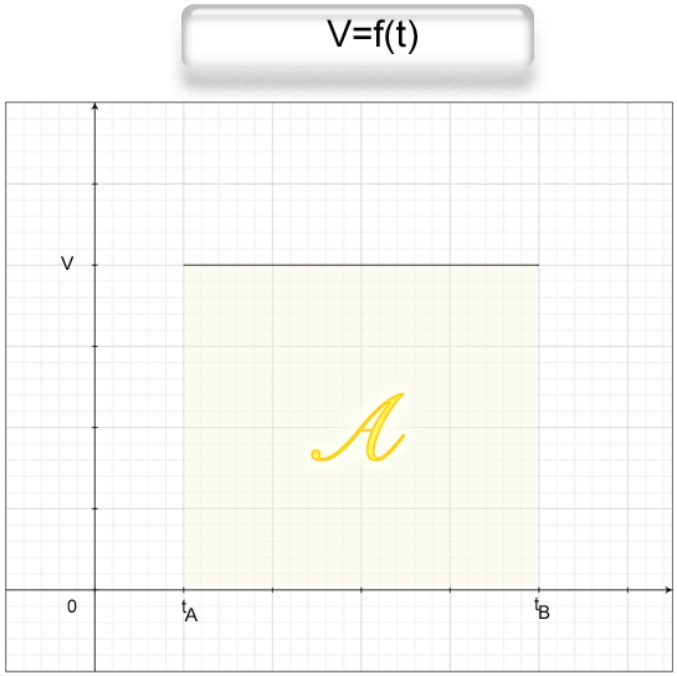
Vitesse lors d'un mouvement rectiligne uniforme

Au cours d'un trajet rectiligne uniforme, la vitesse d'une masse ponctuelle est constante.

Une masse, dont la représentation graphique x=f(t) est une droite, est en mouvement rectiligne uniforme, et réciproquement.

Interprétation graphique du terme de vitesse

- A l'instar de la représentation précédente x=f(t), nous pouvons tracer la courbe v=f(t).
- Nous l'avons vu, la vitesse d'un point sur le trajet AB en mouvement rectiligne uniforme est constante :



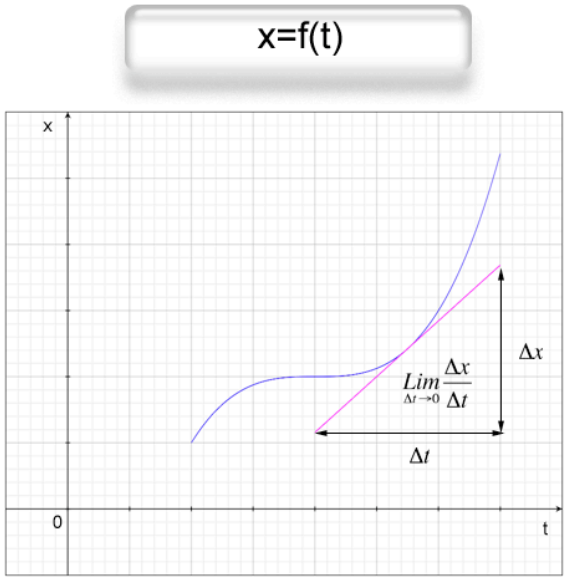
- De notre définition de la vitesse, on en déduit : AB = v*(tB-tA). Ainsi, graphiquement, l'aire colorée est la longueur AB.

Vitesse instantanée

- Nous avons caractérisé les mouvements de translation, uniformes, et n'avons abordé que la notion de vitesse moyenne. Introduisons celle de vitesse instantanée.
- Beaucoup plus récente (1690), elle est définie comme suit :

$$v(t) = \frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t}$$

- Graphiquement, elle s'interprète comme la pente de la tangente au point (t,x(t)) de la représentation graphique x=f(t).



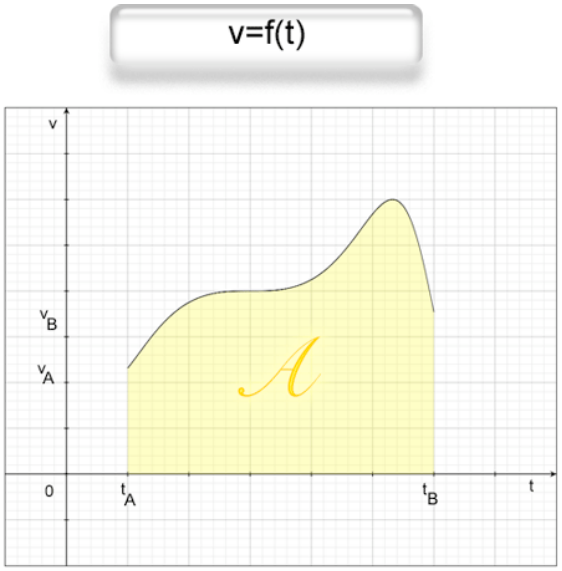
Interprétation graphique du terme de vitesse

- Comme précédemment, cherchons à interpréter graphiquement le terme de vitesse à partir de la courbe v=f(t).

$$v(t) = \frac{dx}{dt} \Rightarrow \int_{t_A}^{t_B} v(t)dt = \int_{t_A}^{t_B} \frac{dx}{dt}dt = \int_{t_A}^{t_B} dx = x(t_B) - x(t_A)$$

$$\text{Finalement : } \int_{t_A}^{t_B} v(t)dt = \overline{AB}$$

- Ainsi, l'aire (algébrique) colorée ci-dessous correspond à la distance (algébrique) du trajet de A à B parcouru par la masse.



Loi de composition des vitesses

Les vitesses ont une propriété remarquable :

Soit $\vec{v}_R(M)$ la vitesse (absolue) de M dans un référentiel R.
Soit $\vec{v}_{R'}(M)$ sa vitesse (relative) dans un référentiel R'.
En supposant R' en mouvement de translation rectiligne uniforme par rapport à R, avec une vitesse d'entraînement $\vec{v}_e$.
Alors $\vec{v}_R(M) = \vec{v}_{R'}(M) + \vec{v}_e$

Démonstration :

Soient O l'origine du référentiel R, et O' celui de R'.

Comme $\vec{v}_e$ est la vitesse de R' par rapport à R : $\vec{v}_e = \frac{d\overrightarrow{OO'}}{dt}$.

Ainsi : $\vec{v}_R(M) = \frac{d\overrightarrow{OM}}{dt} = \frac{d}{dt}\left(\overrightarrow{OO'} + \overrightarrow{O'M}\right) = \frac{d\overrightarrow{OO'}}{dt} + \frac{d\overrightarrow{O'M}}{dt}$

C'est - à - dire : $\vec{v}_R(M) = \vec{v}_e + \vec{v}_{R'}(M)$

Vite, un exemple !

Loi de composition des vitesses

- Pour ne pas rater le prochain métro, vous décidez d'emprunter un tapis roulant, ayant une vitesse de 9km/h (vitesse du tapis de la station *Montparnasse*), en marchant d'un pas décidé à 5km/h.
- Votre vitesse est, par rapport au référentiel terrestre, de 14km/h.
- Ce théorème de composition formalise donc notre intuition naturelle concernant l' *addition* des effets des vitesses, lorsqu'elles restent raisonnablement faibles (pour ne pas tomber dans le cas relativiste).

Quantité de mouvement

Pour poursuivre notre étude, nous devons introduire une nouvelle grandeur vectorielle, la quantité de mouvement :

- Le vecteur quantité de mouvement d'une masse ponctuelle m, se déplaçant à une vitesse v, est le vecteur :

$$\vec{p} = m\vec{v}$$

- Cette définition se généralise à un ensemble S de N masses en sommant les N vecteurs quantité de mouvement associés à chacun des points du système :

Si $S = \{m_1 ... m_N\}$ et si $\forall i \in \{1...N\}$, $\vec{p}_i$ est la quantité de mouvement de la masse m_i,

alors : $\vec{p}_S = \vec{p}_1 + ... + \vec{p}_i = m_1\vec{v}_1 + ... + m_N\vec{v}_N = \sum_{i\in\{1...N\}} m_i\vec{v}_i$

Théorème fondamental

Utilisons ce nouveau vecteur pour énoncer le théorème fondamental de la mécanique : le mouvement est conservatif.

$$\overrightarrow{\mathbf{p}_{\mathbf{Univers}}} = \overrightarrow{\mathbf{constant}}$$

Première loi de Newton

- Newton dans ses Principia de 1687, n'évoque pas ouvertement les travaux de Descartes, à l'origine du précédent théorème, mais il va introduire les notions de force, et de système isolé, indispensables à l'étude de la mécanique.
- Un système S est dit isolé si : $\overrightarrow{\mathbf{p}_S} = \overrightarrow{\mathbf{constant}}$
- Première loi de Newton :

> Si S est un système isolé, alors $\vec{v}_S$ est constant

- L'application de la définition d'un système isolé conduit au résultat :

$$S \text{ isolé} \Rightarrow \vec{p}_S = \vec{K} \Rightarrow m_S \vec{v}_S = \vec{K} \quad \text{or comme } m_S \text{ est constant :}$$
$$\Rightarrow \vec{v}_S = \vec{k}$$

Deuxième loi de Newton

- Par définition, une force F caractérise une variation de mouvement.
  - Il s'agit donc d'une grandeur vectorielle
  - Elle est la dérivée temporelle de la quantité de mouvement
- En supposant la masse m de la particule M constante, on définit la force s'exerçant sur M par :

$$\vec{F} = \frac{d\vec{p}}{dt} = \frac{dm\vec{v}}{dt} = m\frac{d\vec{v}}{dt}$$

- Deuxième loi de Newton :

Si S est un système isolé, alors $\sum \vec{F}_{\text{Extérieures s'exerçant sur S}} = \frac{d\vec{p}_S}{dt}$

Troisième loi de Newton

- Enoncé de la troisième loi de Newton :

> Si A et B sont deux points en interaction, et forment un système isolé, alors la force qu'exerce B sur A est l'opposée de la force exercée sur B par A

- La démonstration se base encore sur la définition d'un système isolé :

Si $S = \{A,B\}$ est un système isolé, alors :

$$\vec{p}_S = \vec{K} \Rightarrow \vec{p}_A + \vec{p}_B = \vec{K} \Rightarrow \frac{d}{dt}\left(\vec{p}_A + \vec{p}_B\right) = \frac{d}{dt}\left(\vec{K}\right)$$

$$\Rightarrow \frac{d\vec{p}_A}{dt} + \frac{d\vec{p}_B}{dt} = \vec{0} \Rightarrow \frac{d\vec{p}_A}{dt} = -\frac{d\vec{p}_B}{dt}$$

$$\Rightarrow \vec{F}_{B/A} = -\vec{F}_{B/A}$$

Chute libre

Appliquons la seconde loi de Newton à une masse en chute libre pour décrire son mouvement.

- Un corps est dit en chute libre s'il n'est soumis qu'à son poids, *i.e.* la seule force extérieure s'exerçant sur lui est la force :

$$\boxed{\vec{F} = m\vec{g}}$$

- Le poids est une force verticale, dirigée vers le bas, proportionnelle à la masse, résultant de l'action de la Terre sur l'objet. Le coefficient de proportionnalité g vaut environ 10 N/kg.
- Soit une masse M en chute libre dans un repère (Oxy) ayant une vitesse initiale v(t0) au temps t0.

Chute libre

- Comme M est en chute libre, d'après la seconde loi de Newton :

$$\boxed{m\vec{g} = m\frac{d\vec{v}}{dt} \Leftrightarrow \vec{g} = \frac{d\vec{v}}{dt}}$$

- Ainsi, en envisageant des intervalles de temps suffisamment petits pour approximer le mouvement de M, durant ces intervalles, en un mouvement rectiligne uniforme, sa vitesse varie suivant la relation :

$$\boxed{\vec{v}(t+\Delta t) = \vec{v}(t) + \vec{g}}$$

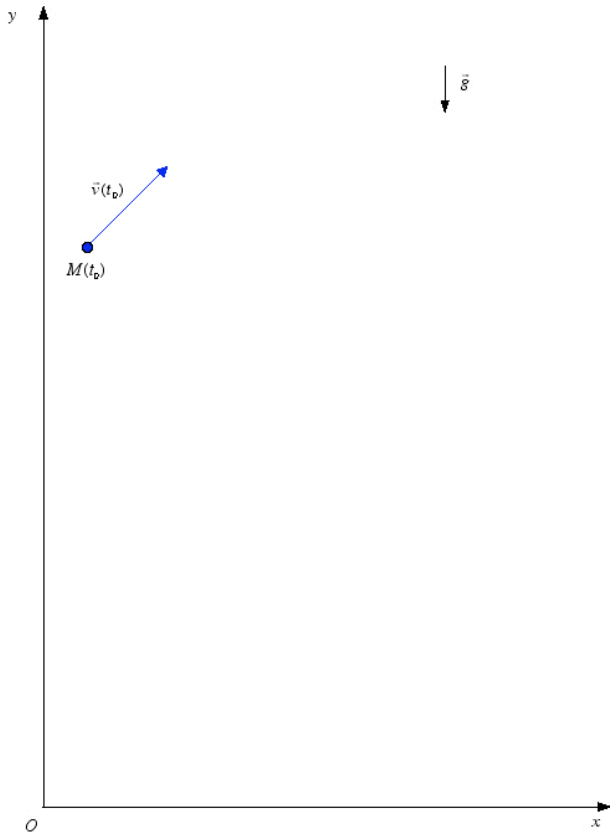
- En exploitant finalement le fait que, pour un mouvement rectiligne uniforme, nous avons la relation :

Si $\vec{r}$ est le vecteur position de M dans le repère d'étude,

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t)$$

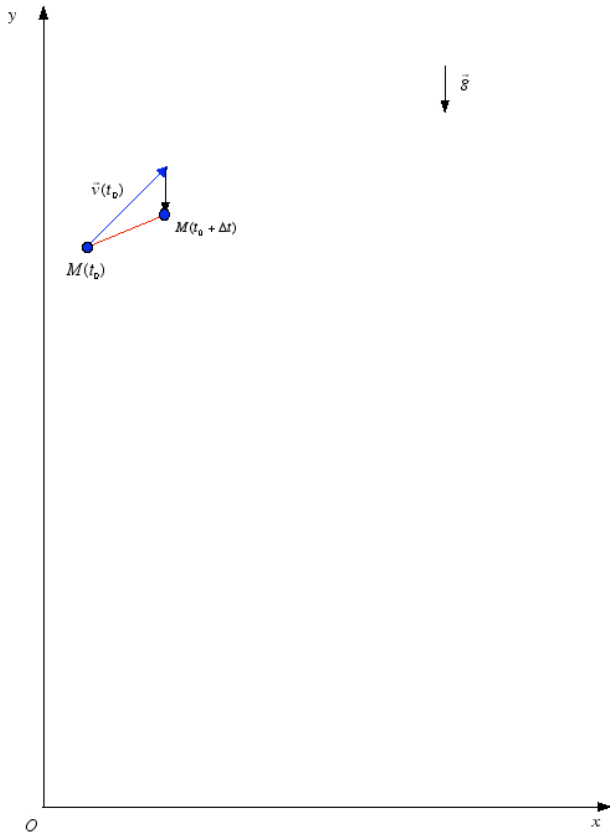
On aboutit à la construction suivante

Chute libre



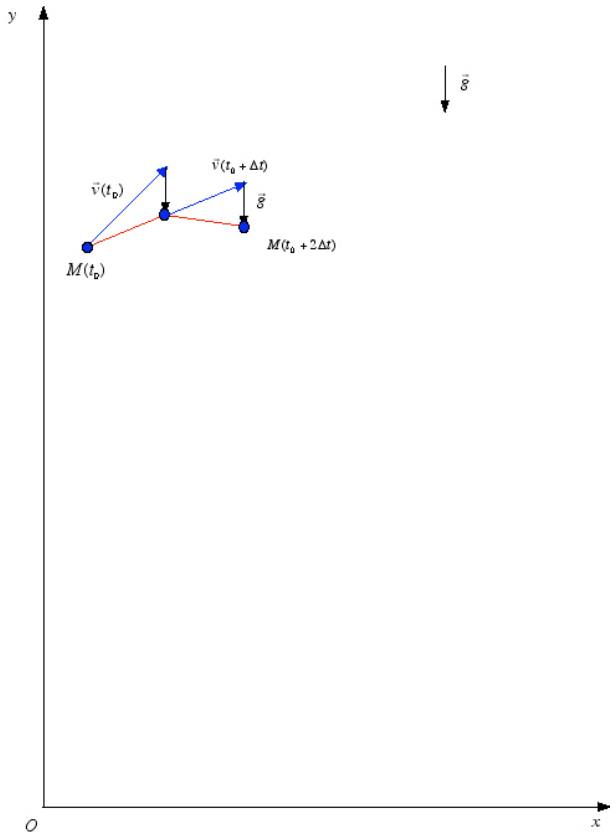
Mouvement d'une masse M en chute libre

Chute libre



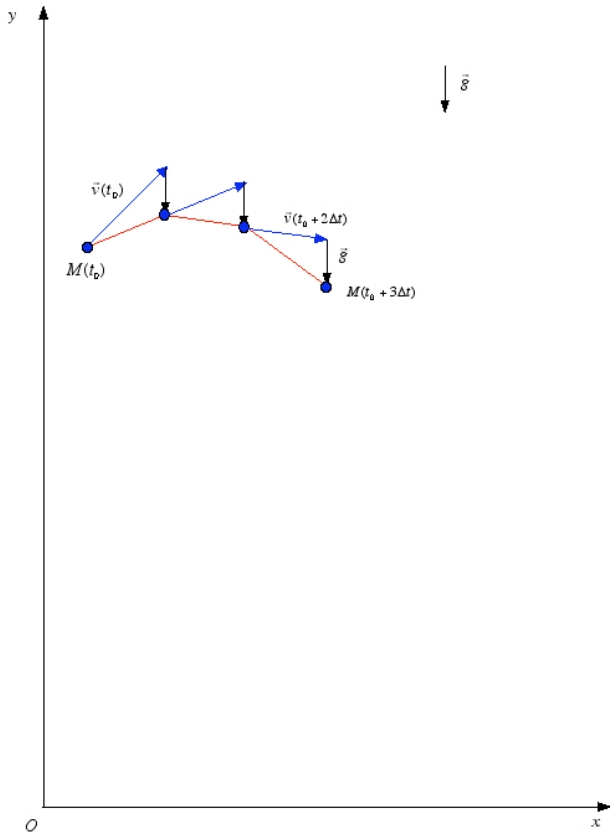
Mouvement d'une masse M en chute libre

Chute libre



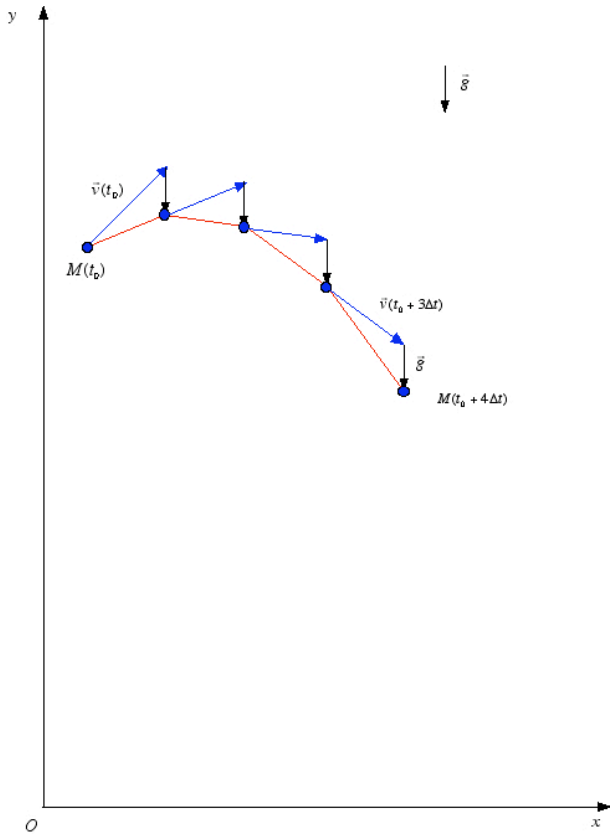
Mouvement d'une masse M en chute libre

Chute libre



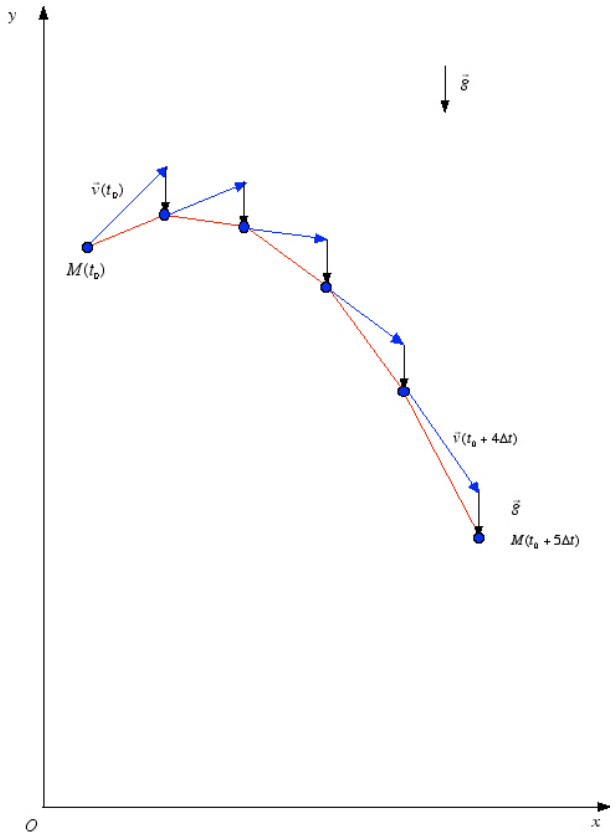
Mouvement d'une masse M en chute libre

Chute libre



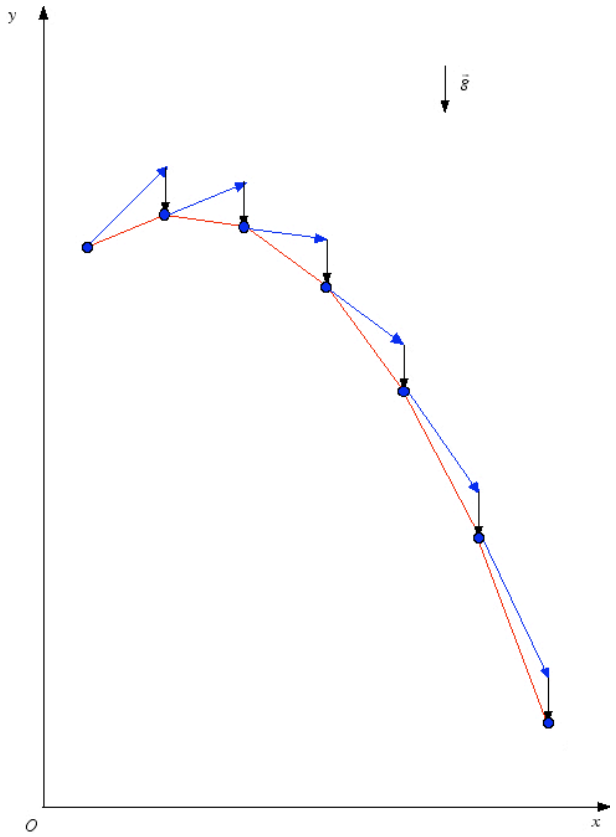
Mouvement d'une masse M en chute libre

Chute libre



Mouvement d'une masse M en chute libre

Chute libre



La masse semble suivre une trajectoire parabolique.

Peut-on trouver ce résultat par le calcul ?

Chute libre

La seconde loi de Newton nous donne :

$$\boxed{m\vec{g} = m\frac{d\vec{v}}{dt} \Leftrightarrow \vec{g} = \frac{d\vec{v}}{dt}}$$

On en déduit, par intégration, la vitesse de la masse :

$$\boxed{\begin{cases} \vec{v}(t) = \vec{g}t + \vec{k} \\ \vec{v}(t_0) = \vec{g}t_0 + \vec{k} \end{cases} \Rightarrow \vec{v}(t) = \vec{g}t + \left(\vec{v}(t_0) - \vec{g}t_0\right)}$$

Puis sa position en fonction du temps :

$$\boxed{\begin{aligned} &\frac{d\vec{r}}{dt} = \vec{v} = \vec{g}t + \left(\vec{v}(t_0) - \vec{g}t_0\right) \Rightarrow \begin{cases} \vec{r}(t) = \frac{1}{2}\vec{g}t^2 + \left(\vec{v}(t_0) - \vec{g}t_0\right)t + \vec{l} \\ \vec{r}(t_0) = \frac{1}{2}\vec{g}{t_0}^2 + \left(\vec{v}(t_0) - \vec{g}t_0\right)t_0 + \vec{l} \end{cases} \\ &\Rightarrow \vec{r}(t) = \frac{1}{2}\vec{g}t^2 + \left(\vec{v}(t_0) - \vec{g}t_0\right)t + \left(\vec{r}(t_0) - \frac{1}{2}\vec{g}{t_0}^2 + \left(\vec{v}(t_0) - \vec{g}t_0\right)t_0\right) \end{aligned}}$$

On retrouve bien l'équation d'une parabole.

Chute libre

- Une petite visualisation devrait finir par vous convaincre ! Vous pouvez modifier la masse de la boule en cliquant sur "Change B" (B comme bleu), modifier la composante horizontale de sa vitesse de manière aléatoire via "Change VB", ou l'annuler par "VB=0". Nous pouvons vérifier deux choses :
  - La variation de la composante verticale de la vitesse est indépendante de la masse de l'objet
  - La masse suit bien une trajectoire parabolique
- Une vidéo reprenant ces points est visionnable à cette adresse : http://projetphysics.teria.org/Videos_du_slideshow/parabole.html

- **Préambule**
- **Introduction**
- **Partie I : Physique du point**
- **Partie II : Quantité de mouvement et chocs**
  - Applications aux chocs entre des masses circulaires
- **Partie III : Mouvements de rotation**
- **Partie IV : Implémentation du projet**
- **Conclusion**

Travail d'une force

Nous allons, pour traiter le problème des chocs entre particules, préalablement introduire la notion de travail d'une force, et d'énergie.

- De manière qualitative, le travail est *l'effort* à fournir pour mettre en mouvement un objet : il quantifie la dépense énergétique associée à un tel déplacement.
- Si M est un point matériel et dl la distance infinitésimale parcourue par M au cours d'un intervalle de temps dt, le travail de la force F est par définition :

$$\boxed{\delta W = \vec{F}.\vec{dl}}$$

- Calcul du travail du poids :

$$W = \int_A^B \vec{P}.\vec{dl} = m\int_A^B \vec{g}.\vec{dl} = m\vec{g}.\int_A^B \vec{dl} = m\vec{g}.\vec{AB} = m\begin{pmatrix}0\\-g\end{pmatrix}.\begin{pmatrix}x_B - x_A\\y_B - y_A\end{pmatrix} = mg(y_A - y_B)$$

Energie cinétique

- Qualitativement, l'énergie est la ressource nécessaire au travail d'une force.
  - Il existe une énergie potentiellement délivrable par un système physique (ressort, altitude, potentiel chimique)
  - Mais une énergie, liée au mouvement, est plus universelle : il s'agit de l'énergie cinétique
- L'expression de l'énergie cinétique d'une masse M est :

$$E_C = \frac{1}{2} m v^2$$

- Le dernier théorème dont nous aurons besoin dans cette partie est le théorème de l'énergie cinétique :

La variation d'énergie cinétique d'un point matériel M est égale au travail total de toutes les forces s'exerçant sur M au cours du déplacement.

Théorème de l'énergie cinétique

Calcul pour un degré de liberté

$$\begin{cases} \dfrac{dx}{dt} = v \\ m\dfrac{dv}{dt} = F \end{cases} \Rightarrow \frac{dx}{mdv} = \frac{v}{F} \Rightarrow F\,dx = mvdv$$

$$\Rightarrow \int_A^B F\,dx = \int_A^B mvdv$$

$$\Rightarrow W = \frac{1}{2}m\left(v_B^2 - v_A^2\right)$$

Calcul pour deux degrés de liberté

$$\begin{cases} \displaystyle\int_A^B F_x\,dx = \frac{1}{2}m\left(v_{xB}^{\ 2} - v_{xA}^{\ 2}\right) \\ \displaystyle\int_A^B F_y\,dy = \frac{1}{2}m\left(v_{yB}^{\ 2} - v_{yA}^{\ 2}\right) \end{cases} \Rightarrow \int_A^B F_x\,dx + \int_A^B F_y\,dy = \frac{1}{2}m\left(\left(v_{xB}^{\ 2} - v_{xA}^{\ 2}\right) + \left(v_{yB}^{\ 2} - v_{yA}^{\ 2}\right)\right)$$

$$\Rightarrow \int_A^B \begin{pmatrix} F_x \\ F_y \end{pmatrix} \cdot \begin{pmatrix} dx \\ dy \end{pmatrix} = \frac{1}{2}m\left(v_B^2 - v_A^2\right)$$

$$\Rightarrow W = \frac{1}{2}m\left(v_B^2 - v_A^2\right)$$

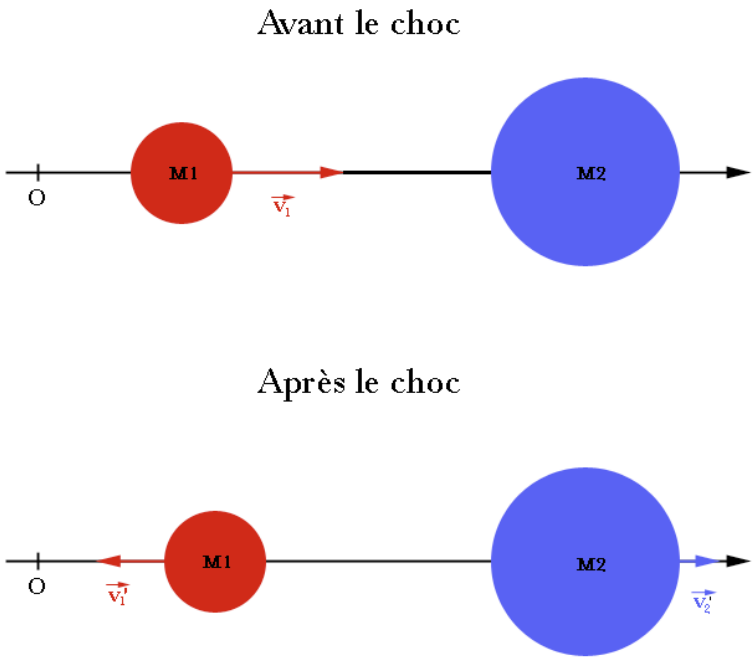
Choc élastique frontal

- Considérons le problème suivant :
  - Soit S un système isolé constitué de deux masses M1 et M2
  - Soit v1 la vitesse de M1, M2 étant immobile
  - On suppose v1 dirigé selon la direction des centres de masse
  - M1 et M2 entrent en collision au cours d'un choc parfaitement élastique (il y a conservation de l'énergie mécanique du système)
  - Soient v1' et v2' les vitesses des masses après le choc

Choc élastique frontal

- Il s'agit donc de comprendre ce phénomène. Les masses B et V (bleu et vert) peuvent être modifiées ainsi que leur vitesse.
- Nous pouvons déjà remarquer que si les deux objets ont la même masse, et que l'un des deux est à l'arrêt, alors celui-ci, après le choc, se déplace à la même vitesse que l'autre masse, qui s'arrête net.
- Le tout est consultable sur le site internet du projet à l'adresse http://projetphysics.teria.org/Videos_du_slideshow/axe.html

Choc élastique frontal



Quelles sont les vitesses des masses après le choc ?

Choc élastique frontal

- Bilan énergétique :

$$E_S = E_{C1} + E_{C2} = \tfrac{1}{2} m_1 v_1^{\,2} \qquad \text{(énergie avant le choc)}$$

$$E'_S = E'_{C1} + E'_{C2} = \tfrac{1}{2}\left(m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2\right) \qquad \text{(énergie après le choc)}$$

- Bilan de la quantité de mouvement :

$$P_S = P_1 + P_2 = m_1 v_1 \qquad \text{(qdm avant le choc)}$$

$$P'_S = P'_1 + P'_2 = m_1 v'_1 + m_2 v'_2 \qquad \text{(qdm après le choc)}$$

- Applications des lois de conservation de la quantité de mouvement, et de l'énergie :

$$\begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1 v_1 = m_1 v'_1 + m_2 v'_2 \end{cases}$$

Choc élastique frontal

- Résolution du système :

$$\begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1 v_1 = m_1 v'_1 + m_2 v'_2 \end{cases} \Leftrightarrow \begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1^{\,2} v_1^{\,2} = m_1^{\,2} v'^{\,2}_1 + m_2^{\,2} v'^{\,2}_2 + 2 m_1 m_2 v'_2 v'_1 \end{cases}$$

$$\Leftrightarrow \begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + \dfrac{m_2^{\,2} v'^{\,2}_2}{m_1} + 2 m_2 v'_2 v'_1 \end{cases}$$

$$\Leftrightarrow \begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ \dfrac{m_2^{\,2} v'^{\,2}_2}{m_1} - m_2 v'^{\,2}_2 + 2 m_2 v'_2 v'_1 = 0 \end{cases}$$

Choc élastique frontal

- Résolution du système :

$$\begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1 v_1 = m_1 v'_1 + m_2 v'_2 \end{cases} \Leftrightarrow \begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1^{\,2} v_1^{\,2} = m_1^{\,2} v'^{\,2}_1 + m_2^{\,2} v'^{\,2}_2 + 2 m_1 m_2 v'_2 v'_1 \end{cases}$$

$$\Leftrightarrow \begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ v'_2 \left(\dfrac{m_2^{\,2} v'_2}{m_1} - m_2 v'_2 + 2 m_2 v'_1 \right) = 0 \end{cases}$$

$$\Leftrightarrow \begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ v'_1 = v'_2 \dfrac{\left(-\dfrac{m_2}{m_1} + 1 \right)}{2} \end{cases}$$

Choc élastique frontal

- Résolution du système :

$$\begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1 v_1 = m_1 v'_1 + m_2 v'_2 \end{cases} \Leftrightarrow \begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1^{\,2} v_1^{\,2} = m_1^{\,2} v'^{\,2}_1 + m_2^{\,2} v'^{\,2}_2 + 2 m_1 m_2 v'_2 v'_1 \end{cases}$$

$$\Leftrightarrow \begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_2 \left(\dfrac{m_1 - m_2}{2m_1} \right)^2 + m_2 v'^{\,2}_2 \\ v'_1 = v'_2 \left(\dfrac{m_1 - m_2}{2m_1} \right) \end{cases}$$

$$\Leftrightarrow \begin{cases} m_1 v_1^{\,2} = v'^{\,2}_2 \left(m_1 \left(\dfrac{m_1 - m_2}{2m_1} \right)^2 + m_2 \right) \\ v'_1 = v'_2 \left(\dfrac{m_1 - m_2}{2m_1} \right) \end{cases}$$

Choc élastique frontal

- Résolution du système :

$$\begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1 v_1 = m_1 v'_1 + m_2 v'_2 \end{cases} \Leftrightarrow \begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1^{\,2} v_1^{\,2} = m_1^{\,2} v'^{\,2}_1 + m_2^{\,2} v'^{\,2}_2 + 2 m_1 m_2 v'_2 v'_1 \end{cases}$$

$$\Leftrightarrow \begin{cases} m_1 v_1^{\,2} = v'^{\,2}_2 \left(\dfrac{m_1^{\,2} + 2 m_1 m_2 + m_2^{\,2}}{4 m_1} \right) \\ v'_1 = v'_2 \left(\dfrac{m_1 - m_2}{2 m_1} \right) \end{cases}$$

$$\Leftrightarrow \begin{cases} v_1^{\,2} = v'^{\,2}_2 \left(\dfrac{(m_1 + m_2)^2}{4 m_1^{\,2}} \right) \\ v'_1 = v'_2 \left(\dfrac{m_1 - m_2}{2 m_1} \right) \end{cases}$$

Choc élastique frontal

- Résolution du système :

$$\begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1 v_1 = m_1 v'_1 + m_2 v'_2 \end{cases} \Leftrightarrow \begin{cases} m_1 v_1^{\,2} = m_1 v'^{\,2}_1 + m_2 v'^{\,2}_2 \\ m_1^{\,2} v_1^{\,2} = m_1^{\,2} v'^{\,2}_1 + m_2^{\,2} v'^{\,2}_2 + 2 m_1 m_2 v'_2 v'_1 \end{cases}$$

$$\Leftrightarrow \begin{cases} v'_2 = \dfrac{2m_1}{m_1 + m_2} v_1 \\ v'_1 = \dfrac{2m_1}{m_1 + m_2} \left(\dfrac{m_1 - m_2}{2m_1} \right) v_1 \end{cases}$$

$$\Leftrightarrow \begin{cases} v'_2 = \dfrac{2m_1}{m_1 + m_2} v_1 \\ v'_1 = \dfrac{m_1 - m_2}{m_1 + m_2} v_1 \end{cases}$$

Ouf !!!

Choc élastique frontal

Avant de tirer une quelconque conclusion, nous pouvons appliquer ce calcul à trois exemples :

Le pendule de Newton

http://projetphysics.teria.org/Videos_du_slideshow/pendule.html

La chute d'une balle sur un plan incliné

http://projetphysics.teria.org/Videos_du_slideshow/plan.html

Collisions entre deux boules dans une enceinte circulaire

http://projetphysics.teria.org/Videos_du_slideshow/circulaire.html

(Ce dernier lien scheme est une animation qui prend fin automatiquement)

Choc élastique frontal

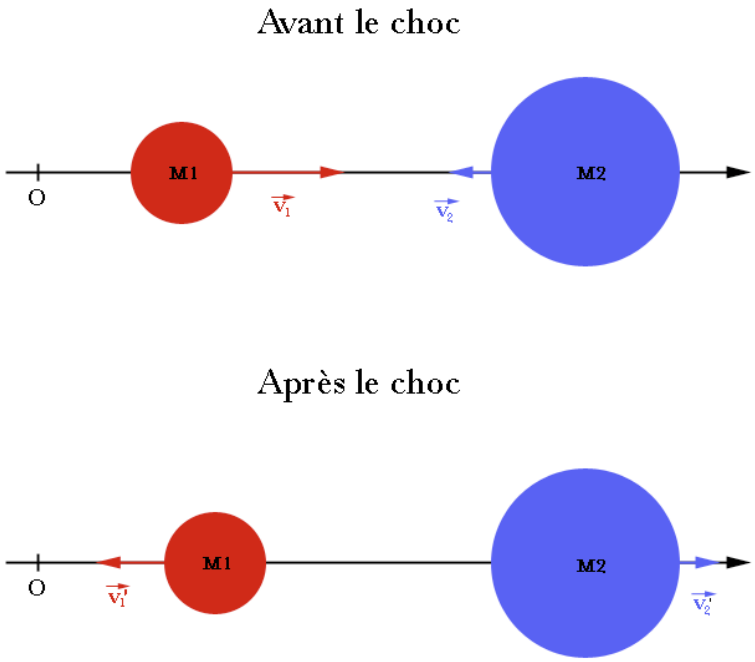
- Finalement, les vitesses après le choc sont données par :

$$\begin{cases} v'_1 = \dfrac{m_1 - m_2}{m_1 + m_2} v_1 \\ v'_2 = \dfrac{2m_1}{m_1 + m_2} v_1 \end{cases}$$

- Quelques remarques :
  - On retrouve l'effet constaté sur la simulation précédente : si M1 et M2 ont la même masse, alors M1 après le choc est à l'arrêt, et M2 a la même vitesse que M1 avant le choc
  - Si M2 est de masse infinie, par passage à la limite, nous avons v1' = -v1
  - Ce modèle nous sert pour modéliser les chocs entre les masses et les parois de la fenêtre d'affichage : la composante de la vitesse dirigée vers la paroi change de signe lors d'un tel choc

Et lorsque M2 est en mouvement ?

Choc élastique frontal



Le système devient :

$$\begin{cases} m_1 v_1 + m_2 v_2 = m_1 v_1' + m_2 v_2' \\ m_1 v_1^{\,2} + m_2 v_2^{\,2} = m_1 v_1'^{2} + m_2 v_2'^{2} \end{cases}$$

Choc élastique frontal

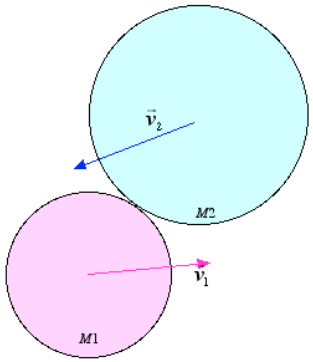
Sa résolution se fait en divisant la première ligne avec la seconde après avoir mis en facteur m1 et m2 à gauche et à droite des égalités, puis par substitution. Sa solution est :

$$\begin{cases} v'_1 = \dfrac{m_1 - m_2}{m_1 + m_2} v_1 + \dfrac{2m_2}{m_1 + m_2} v_2 \\ v'_2 = \dfrac{2m_1}{m_1 + m_2} v_1 + \dfrac{m_2 - m_1}{m_1 + m_2} v_2 \end{cases}$$

Le cas du choc frontal étant traité, il ne reste plus qu'à généraliser aux chocs désaxés.

Choc élastique quelconque

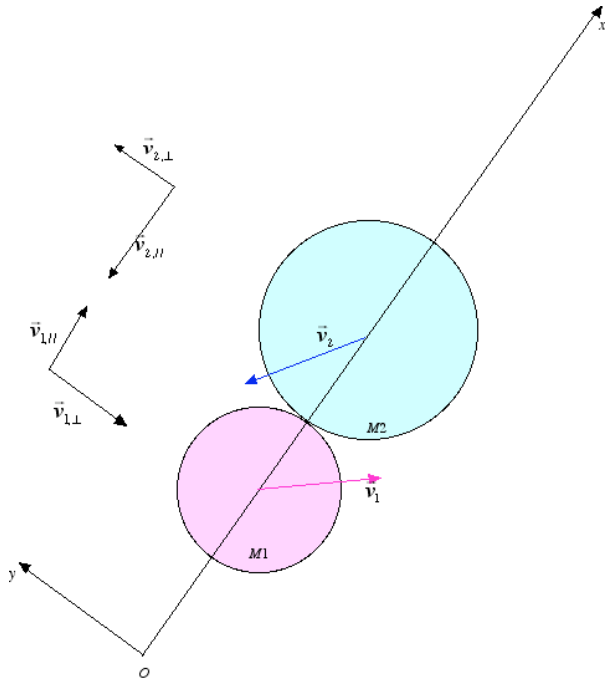
- Ici, c'est la loi de composition des vitesses qui va être mis en jeu.
- Partons de la configuration initiale suivante :



Avant le choc

Choc élastique quelconque

- Ici, c'est la loi de composition des vitesses qui va être mis en jeu.
- Partons de la configuration initiale suivante :

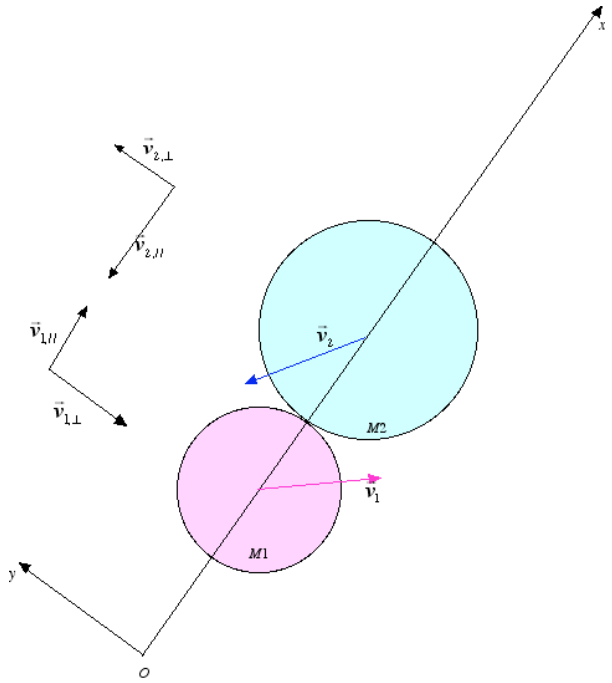


Avant le choc

Décomposons les vitesses selon deux axes orthogonaux

Choc élastique quelconque

- Ici, c'est la loi de composition des vitesses qui va être mis en jeu.
- Partons de la configuration initiale suivante :

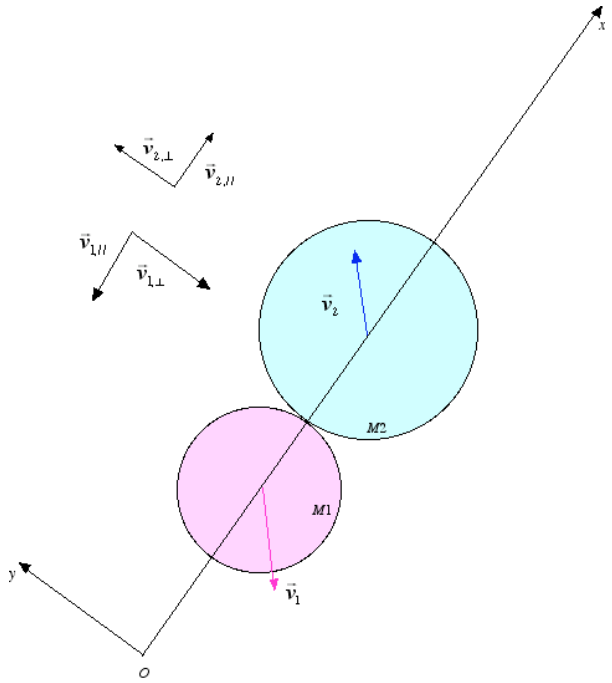


Avant le choc

Suivant (Ox), nous nous retrouvons dans le précédent cas d'un choc frontal, et suivant (Oy), il n'y a pas de choc...

Choc élastique quelconque

- Ici, c'est la loi de composition des vitesses qui va être mis en jeu.
- Partons de la configuration initiale suivante :



Après le choc

Tout a été préalablement traité !

Choc élastique frontal

Nous pouvons donc reprendre la simulation précédente, dans le cas de deux boules n'entrant pas en collision axée.

Si les deux objets ont la même masse, et que l'un est à l'arrêt, nous remarquons que les deux masses évoluent après le choc suivant des directions orthogonales : en fait, l'intégralité de la composante de vitesse parallèle au mouvement a été transmise à la masse au repos (cas d'un choc axé entre deux masses identiques dont l'une est à l'arrêt), et seule la composante orthogonale est restée invariante pour la masse en mouvement.

Cliquez ici

ou

http://projetphysics.teria.org/Videos_du_slideshow/desaxe.html

- **Préambule**
- **Introduction**
- **Partie I : Physique du point**
- **Partie II : Quantité de mouvement et chocs**
- **Partie III : Mouvements de rotation**
  - Applications aux chocs entre des polygones
- **Partie IV : Implémentation du projet**
- **Conclusion**

Système à plusieurs masses

Maintenant que nous savons comment traiter le problème des collisions entre des corps simples, nous allons tenter de comprendre comment des objets de formes plus générales réagissent lorsqu'ils entrent en contact.

- Rappelons la définition de la quantité de mouvement d'un système :
  - Lorsque le système est réduit à une masse :

$$\vec{p} = m\vec{v}$$

  - Pour un système à n masses :

$$\vec{p} = \sum_{i \in \{1 \ldots n\}} m_i \vec{v}_i$$

- On appelle centre de masse le barycentre des masses du système

$$\vec{\Omega} = \sum_{i \in \{1 \ldots n\}} m_i \vec{r}_i$$

Système à plusieurs masses

L'intérêt d'étudier le centre de masse d'un système tient dans le théorème suivant :

Le centre de masse d'un système isolé a une accélération nulle

- La définition d'un système isolé conduit au résultat escompté :

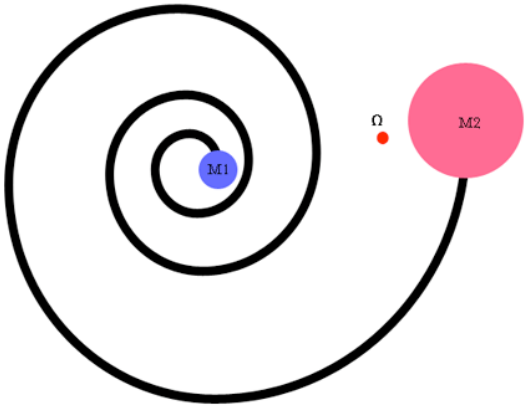
$$\vec{p}_S = \vec{K} \Rightarrow \frac{d\vec{p}_S}{dt} = \vec{0} \Rightarrow \frac{d}{dt}\sum_{i\in\{1\ldots n\}} m_i\vec{v}_i = \vec{0}$$

$$\Rightarrow \sum_{i\in\{1\ldots n\}} m_i \frac{d\vec{v}_i}{dt} = \vec{0} \Rightarrow \sum_{i\in\{1\ldots n\}} m_i \frac{d^2\vec{r}_i}{dt^2} = \vec{0}$$

$$\Rightarrow \frac{d^2}{dt^2}\sum_{i\in\{1\ldots n\}} m_i\vec{r}_i = \vec{0} \Rightarrow \frac{d^2}{dt^2}\vec{r}_\Omega = \vec{0}$$

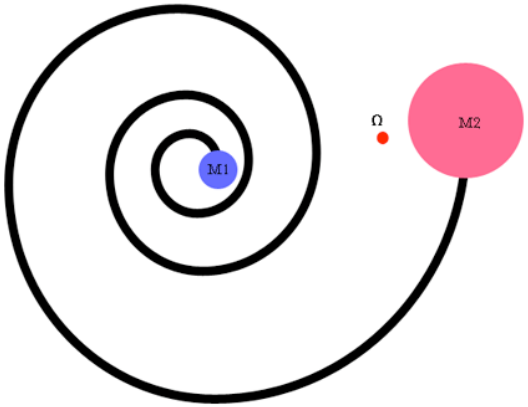
Comment appliquer ce résultat ?

Système de deux masses reliées par un ressort



- On considère le système isolé S constitué de deux masses reliées par un ressort spiral de masse négligeable.

Système de deux masses reliées par un ressort



- Sans rien supposer des forces s'exerçant sur M1 et M2, on peut affirmer que le centre de masse du système a un mouvement rectiligne uniforme. En effet :

Par conservation du mouvement :

$$\frac{d\vec{p}_S}{dt} = \frac{d}{dt}\left(\vec{p}_1 + \vec{p}_2\right) = \vec{0} \Rightarrow \frac{d}{dt}\left(m_1\vec{v}_1 + m_2\vec{v}_2\right) = \vec{0}$$

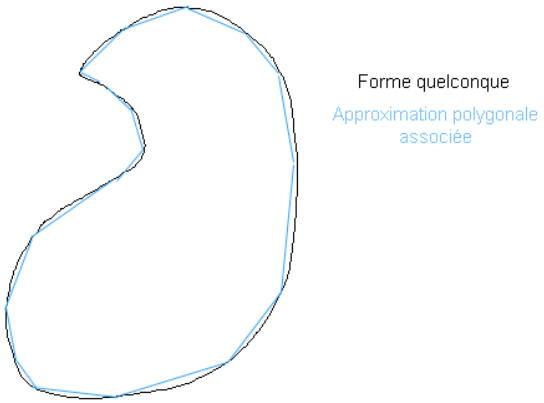
$$\Rightarrow \frac{d}{dt}\left(\frac{m_1\vec{v}_1 + m_2\vec{v}_2}{m_1 + m_2}\right) = \vec{0} \Rightarrow \frac{d\vec{v}_\Omega}{dt} = \vec{0} \Rightarrow \vec{v}_\Omega = \vec{k}$$

Modélisation des objets polygonaux

Nous allons aborder le dernier point de notre étude : le choc entre deux objets de forme quelconque.

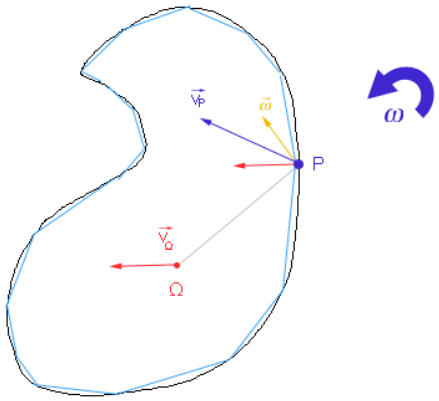
- De la même manière que nous l'avons fait pour mesurer des courbes, on supposera qu'il est possible d'approcher n'importe quelle forme par une suite de segments : tous les objets qui suivront pourront donc être modélisés par des polygones.

- Par exemple :



Modélisation des objets polygonaux

- Nous supposerons les objets indéformables, ainsi tout mouvement pourra être décomposé comme :
  - Un mouvement de translation : celui du centre de masse
  - Un mouvement de rotation autour de ce centre de masse
- ω est la vitesse angulaire, en rad/s
- La vitesse de tout point délimitant la forme considérée aura donc les deux composantes suivantes :

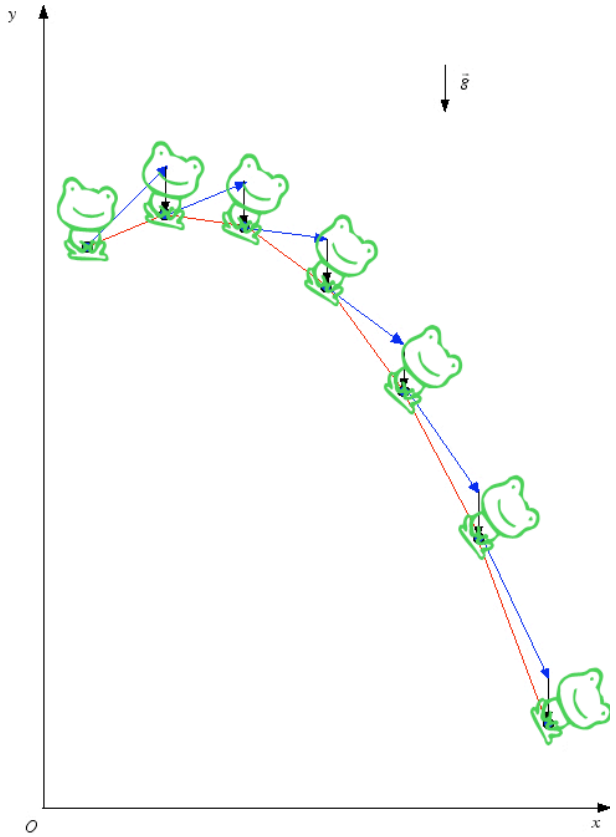


Chute libre de polygones

Reprenons l'exemple de la chute libre d'un corps.

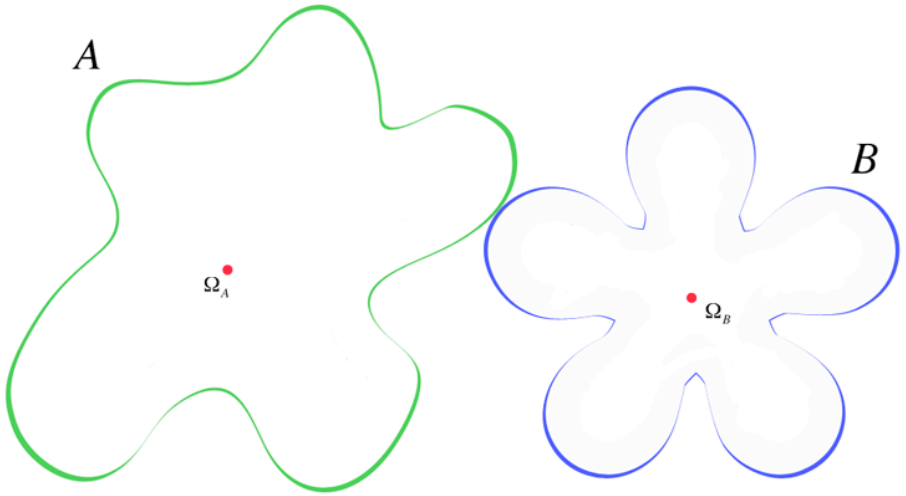
- Comme le centre de masse suit la trajectoire qu'aurait une masse ponctuelle où serait concentrée toute la masse du corps, le centre de masse de l'objet en chute libre suit une trajectoire parabolique.
- De plus, en supposant que l'objet est en rotation, il suffit, pour décrire sa trajectoire, qu'à chaque intervalle de temps élémentaire dt, l'objet subisse une rotation élémentaire autour du centre de masse.
- A partir du tracé de la parabole de la partie précédente, nous obtenons le graphique suivant...

Chute libre d'une grenouille



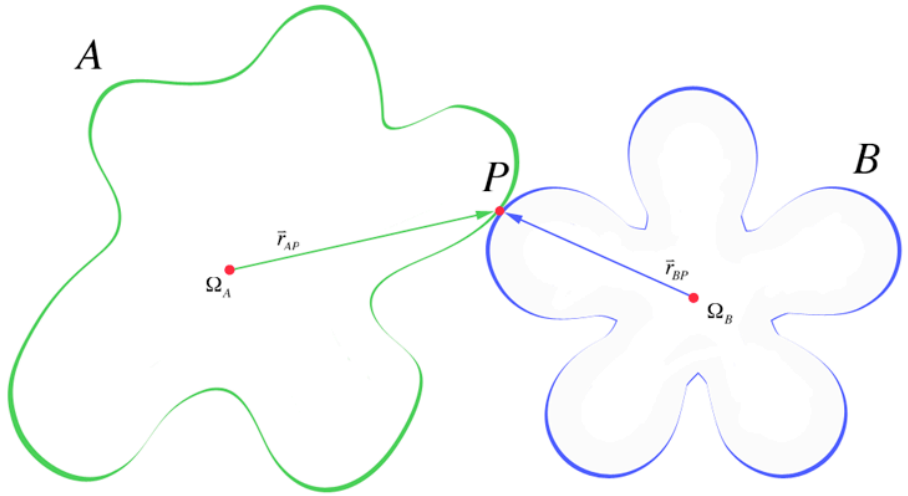
Choc entre deux polygones

On considère le choc suivant entre deux objets A et B :



Choc entre deux polygones

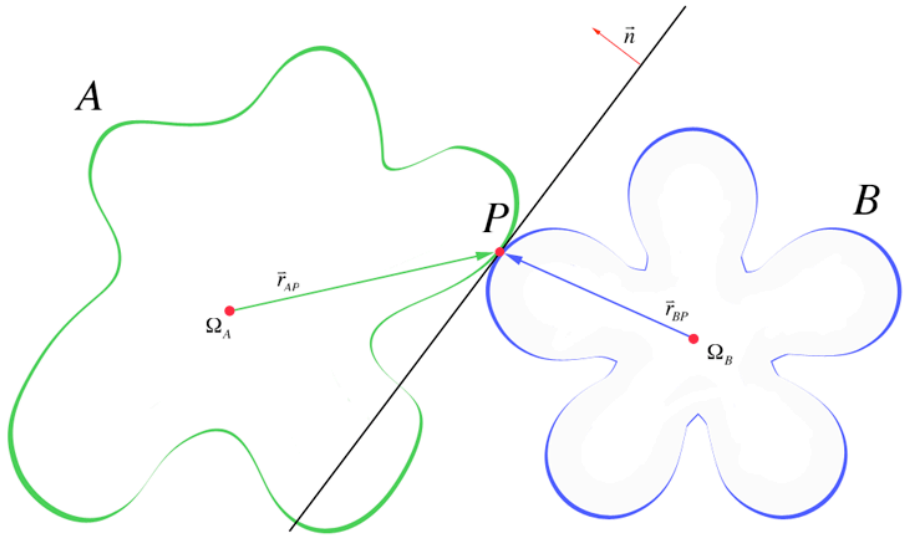
On considère le choc suivant entre deux objets A et B :



Soit P le point de contact entre A et B

Choc entre deux polygones

On considère le choc suivant entre deux objets A et B :

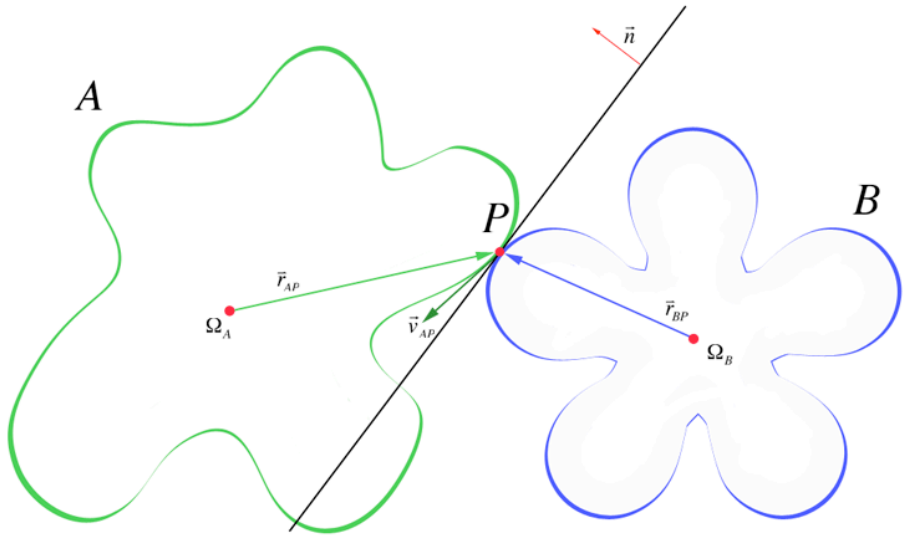


On définit un vecteur normal à la surface de contact,

normé, pointant vers A

Choc entre deux polygones

On considère le choc suivant entre deux objets A et B :

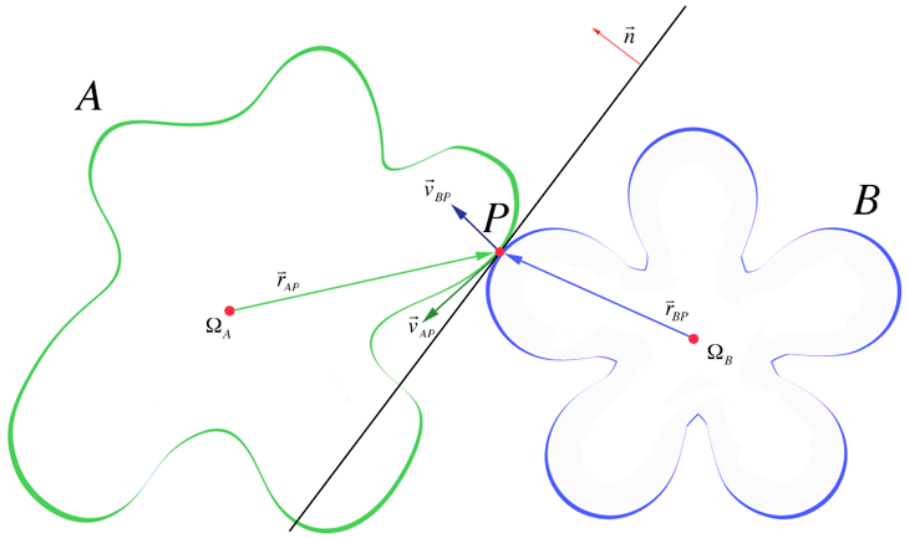


P appartient à A, soit vAP sa vitesse

(composée de la vitesse de translation, et de rotation)

Choc entre deux polygones

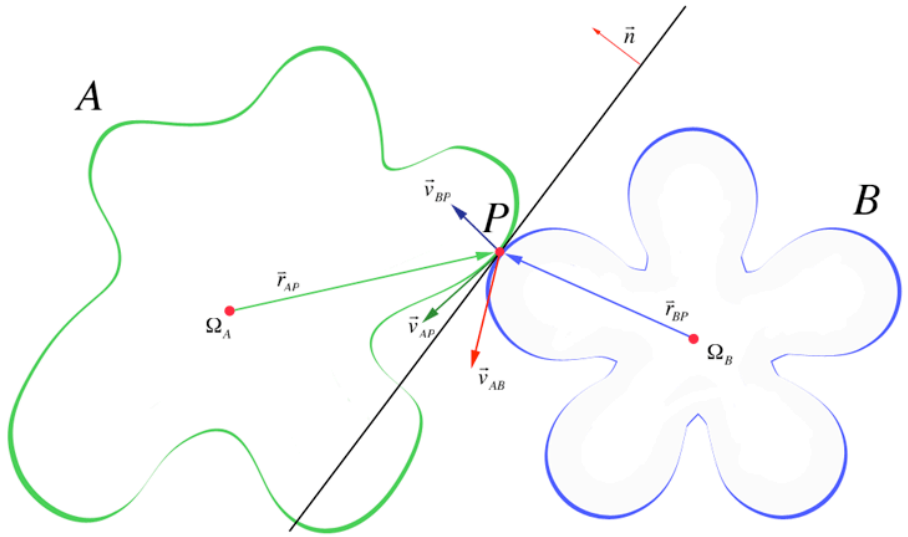
On considère le choc suivant entre deux objets A et B :



P appartenant également à B, on note vBP la vitesse de P considéré comme point de B

Choc entre deux polygones

On considère le choc suivant entre deux objets A et B :



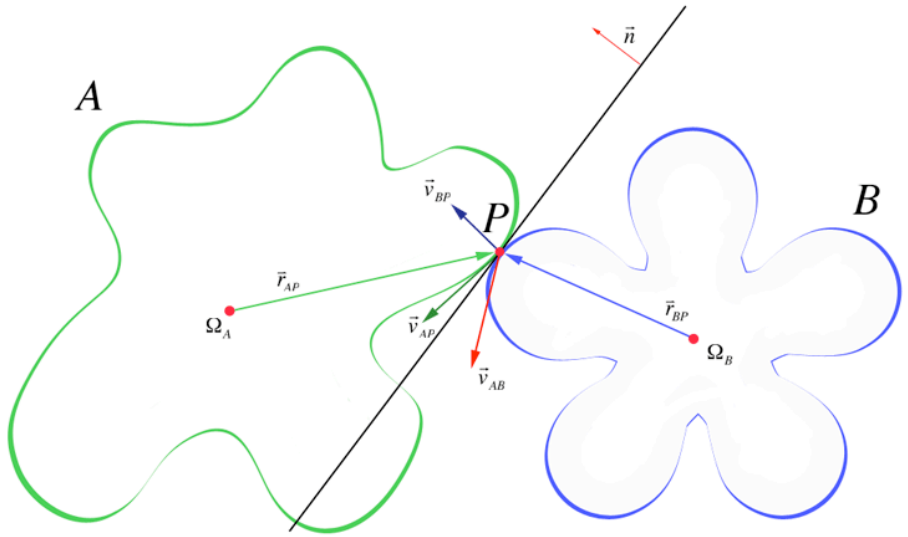
Dans le repère du système des deux objets,

on note vAB la vitesse relative de P

On a alors : $\vec{v}_{AB} = \vec{v}_{AP} - \vec{v}_{BP}$

Choc entre deux polygones

On considère le choc suivant entre deux objets A et B :



En omettant le mouvement de rotation des objets,

nous nous retrouvons dans la situation de la partie précédente :

seule la composante normale de la vitesse est affectée par le choc.

Mais...

Choc entre deux polygones

- Un choc peut être vu à plusieurs échelles :
  - A l'échelle macroscopique : nous observons une impulsion affectant les vitesses
  - A l'échelle atomique, des interactions entre les particules constituant les objets surviennent : ces interactions, à l'origine d'une dissipation d'énergie, expliquent pourquoi, dans la réalité, les chocs ne sont pas parfaitement élastiques (on observe une *perte de mouvement* au cours du temps)
- Pour modéliser cette perte d'énergie, introduisons le coefficient d'élasticité e :

$$e = -\frac{v_{AB,après,\perp}}{v_{AB,avant,\perp}}$$

- e quantifie la vitesse restituée par le choc suivant la direction normale au choc.

Choc entre deux polygones

- Soit j l'impulsion dûe au choc ressentie par A, MA sa masse :

$$\vec{v}_{A,après} = \vec{v}_{A,avant} + \frac{j}{M_A}\vec{n}$$

- D'après la troisième loi de Newton, -j est l'impulsion de B :

$$\vec{v}_{B,après} = \vec{v}_{B,avant} - \frac{j}{M_B}\vec{n}$$

- Avec le coefficient d'élasticité, nous aboutissons au système :

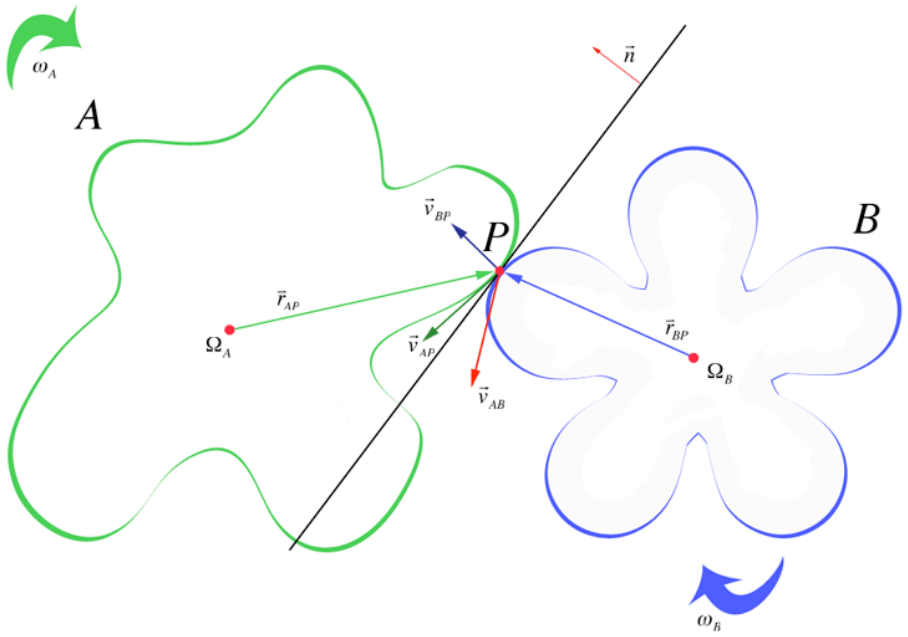
$$\begin{cases} \vec{v}_{A,après} = \vec{v}_{A,avant} + \dfrac{j}{M_A}\vec{n} \\ \vec{v}_{B,après} = \vec{v}_{B,avant} - \dfrac{j}{M_B}\vec{n} \\ \left(\vec{v}_{A,après} - \vec{v}_{B,après}\right).\vec{n} = -e\left(\vec{v}_{A,avant} - \vec{v}_{B,avant}\right).\vec{n} \end{cases}$$

Choc entre deux polygones

- La résolution est immédiate, et nous livre l'impulsion :

$$j = \frac{-(1+e) \times \vec{v}_{AB,avant} . \vec{n}}{\frac{1}{M_A} + \frac{1}{M_B}}$$

Et en cas de rotation ?



Moment d'inertie

- La masse d'un corps tend à s'opposer à la mise en mouvement (par translation) de celui-ci. En effet, par la seconde loi de Newton :

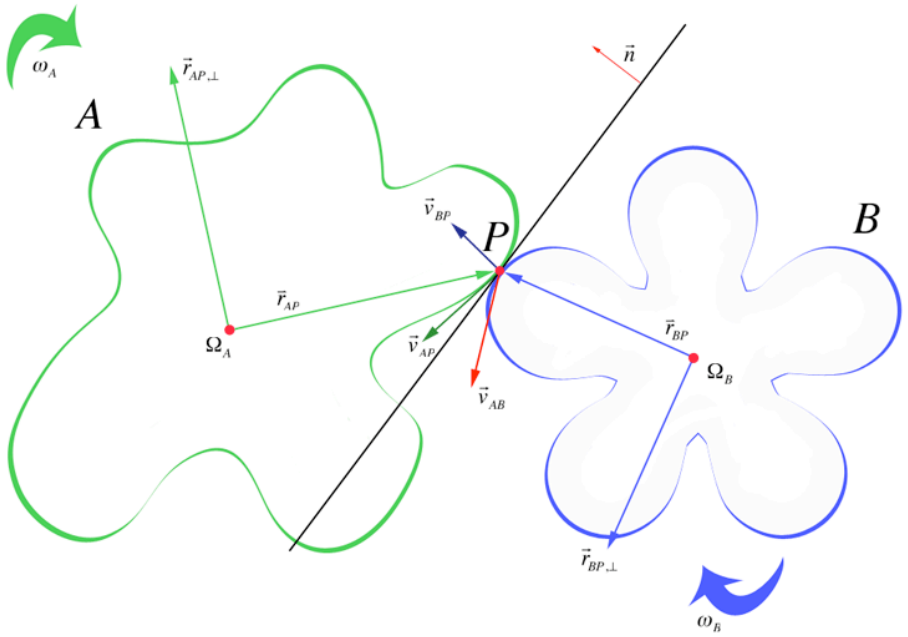
$$\frac{d\vec{v}_M}{dt} = \frac{\vec{F}}{m_M} \Rightarrow \text{Plus } m_M \text{ est grand, moins la variation de vitesse est importante}$$

- Dans le cas d'un système à n masses, il existe également un facteur s'opposant à la mise en rotation du système autour de son centre de masse : il s'agit du moment d'inertie.

- Dans notre étude, nous adopterons pour définition du moment d'inertie :

$$I = \sum_{i \in \{1 \ldots n\}} m_i r_i^2, \text{ avec } r_i \text{ la distance du ième point au centre de masse}$$

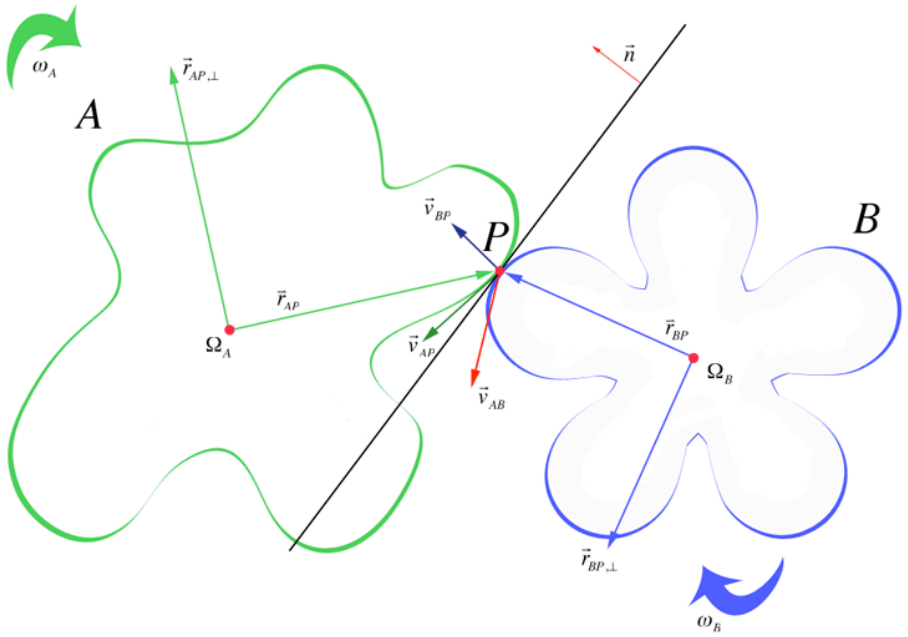
Choc entre deux polygones



Maintenant, nous avons :

$$\begin{cases} \vec{v}_{AP,après} = \vec{v}_{A,après} + \omega_{A,après}\vec{r}_{AP,\perp} \\ \vec{v}_{BP,après} = \vec{v}_{B,après} + \omega_{B,après}\vec{r}_{BP,\perp} \end{cases}$$

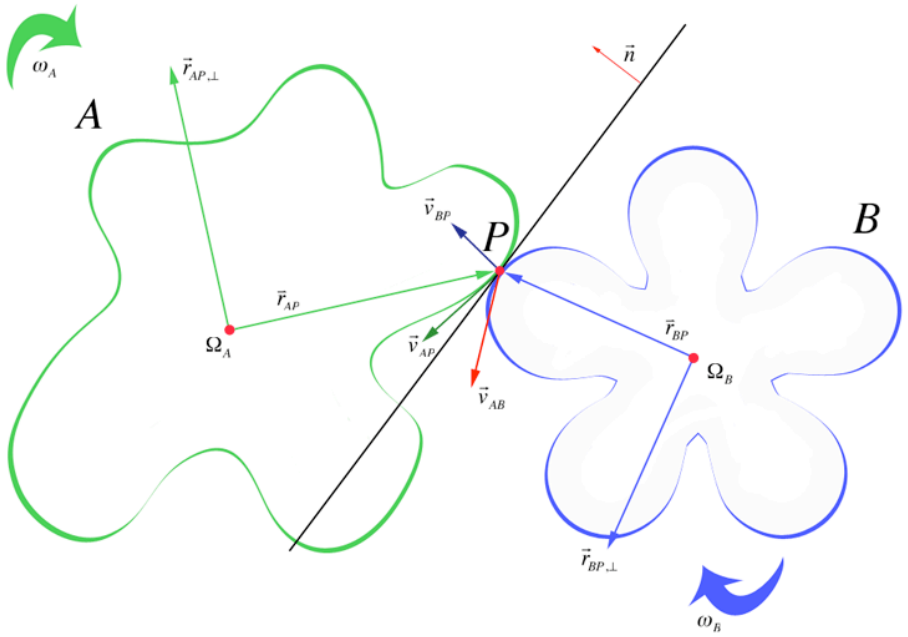
Choc entre deux polygones



La modification de la vitesse angulaire est donnée par la loi des moments :

$$\begin{cases} \omega_{A,après} = \omega_{A,avant} + \dfrac{\vec{r}_{AP,\perp}.j\vec{n}}{I_A} \\ \omega_{B,après} = \omega_{B,avant} - \dfrac{\vec{r}_{BP,\perp}.j\vec{n}}{I_B} \end{cases}$$

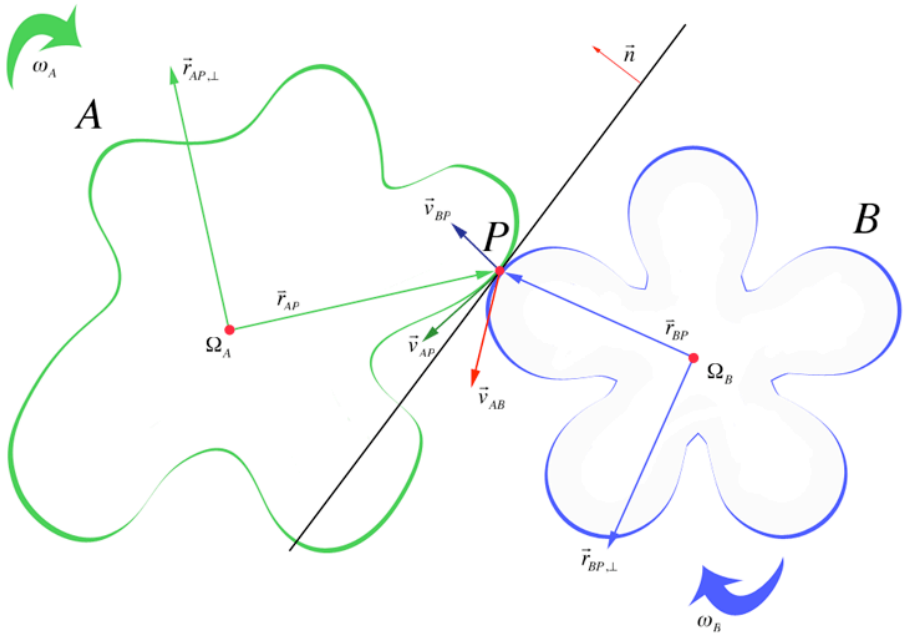
Choc entre deux polygones



En résolvant le système, nous trouvons :

$$j = \frac{-(1+e)\vec{v}_{AB,avant}.\vec{n}}{\frac{1}{M_A} + \frac{1}{M_B} + \frac{(\vec{r}_{AP,\perp}.\vec{n})^2}{I_A} + \frac{(\vec{r}_{BP,\perp}.\vec{n})^2}{I_B}}$$

Choc entre deux polygones



Pour conclure, connaissant le point d'impact, la normale au choc, et les caractéristiques du système, nous sommes capables de prévoir l'évolution du système, avant et après un choc.

- **Préambule**
- **Introduction**
- **Partie I : Physique du point**
- **Partie II : Quantité de mouvement et chocs**
- **Partie III : Mouvements de rotation**
- **Partie IV : Implémentation du projet**

  Structures, détection de colision, enveloppe convexe, traitement des chocs
- **Conclusion**

Implémentation du projet

- Notre projet a été développé, de manière régulière, durant les trois derniers mois, en parallèle des cours de l'option *Programmation fonctionnelle II* (2 heures hebdomadaires).
  - En premier lieu, dès la présentation de la librairie `(lib "world.ss" "htdp")`, nous avons programmé des exemples simples de collisions entre deux masses : certains d'entre eux se retrouvent dans ce slideshow.
  - Une fois l'introduction à la programmation orientée objet faite, nous nous sommes attaqués au problème plus général de la collision au sein d'un système composé de nombreuses masses.
  - Finalement, durant les dernières semaines qui nous étaient accordées, nous avons implémenté le problème des chocs entre polygones (en privilégiant l'utilisation de structures).

Implémentation du projet

- L'objectif de cette partie n'est évidemment pas de détailler exhaustivement l'intégralité du code produit durant un trimestre, mais, en n'en prenant que quelques extraits, de décrire les algorithmes mis en oeuvre dans la résolution de la dernière partie :
  - Le problème de l'enveloppe convexe
  - Le traitement des chocs entre deux polygones
  - La détection des chocs

```
(when
    lecteur-is-ready
  (play-the-end))
```

Structure de polygone

- Dans cette partie, nous nous intéresserons uniquement aux chocs entre polygones. Nous définirons un polygone comme une structure de la forme :

```
(define-struct polygone
  (; taille du polygone
   size
   ; vecteur contenant les size masses formant le polygone
   poly
   ; centre de masse du polygone (masse calculée à la création)
   cm
   ; vitesse angulaire (type float)
   vang
   ; vitesse de translation (type vecteur)
   vtrans
   ; moment d'inertie
   i
   ; liste des indices de poly constituant l'enveloppe convexe
   ev
   ; rayon du plus petit cercle de centre le centre de masse du polygone contenant le polygone
   rayon))
```

- Un vecteur sera une structure à deux champs x et y (on dispose des opérations vectorielles usuelles), et une masse correspondra à la donnée d'un vecteur position r, d'une masse m, et d'un angle alpha compris entre (Ox), et la droite passant par la masse et le centre de masse du polygone.

Evolution libre d'un polygone

Si un polygone ne heurte pas une paroi de la fenêtre d'affichage, ou un autre polygone, alors sa position change entre t et t+dt de la manière suivante :

- Translation de l'ensemble des points

```
; La fonction a un effet de bords
(define (translation-polygone poly)
  ; Quand la vitesse de translation n'est pas nulle
  (when (not (and (zero? (vect-x (polygone-vtrans poly))) (zero? (vect-y (polygone-vtrans poly)))))
    ; On translate chacune des masses du polygone
    (for i from 0 to (- (polygone-size poly) 1)
         (set-masse-r! (vector-ref (polygone-poly poly) i)
                       (+vect (masse-r (vector-ref (polygone-poly poly) i)) (polygone-vtrans poly))))
    ; Et l'on met à jour le centre de masse
    (set-masse-r! (polygone-cm poly)
                  (+vect (masse-r (polygone-cm poly)) (polygone-vtrans poly)))))
```

- Complexité : O((polygone-size poly))

Evolution libre d'un polygone

Si un polygone ne heurte pas une paroi de la fenêtre d'affichage, ou un autre polygone, alors sa position change entre t et t+dt de la manière suivante :

- Rotation de l'ensemble des points

```
; La fonction a un effet de bords
(define (rotation-polygone poly)
  ; Quand la vitesse angulaire est non nulle
  (when (not (zero? (polygone-vang poly)))
    (for i from 0 to (- (polygone-size poly) 1)
         ; On met à jour la position de chaque masse
         (set-masse-r! (vector-ref (polygone-poly poly) i)
                       ; En faisant la rotation vectorielle de centre CM, d'angle vang
                       (rotation (masse-r (polygone-cm poly))
                                 (masse-r (vector-ref (polygone-poly poly) i))
                                 (polygone-vang poly)))
         ; Reste à mettre à jour l'angle alpha formé entre la masse et le centre de masse
         (set-masse-alpha! (vector-ref (polygone-poly poly) i)
                           (mod2pi (- (masse-alpha (vector-ref (polygone-poly poly) i))
                                      (polygone-vang poly)))))))
```

- Complexité : O((polygone-size poly))

Evolution libre d'un polygone

Si un polygone ne heurte pas une paroi de la fenêtre d'affichage, ou un autre polygone, alors sa position change entre t et t+dt de la manière suivante :

- Effet de la gravité

```
; La fonction a un effet de bords
(define (gravite poly)
  ; GRAVITE est une variable globale, de type vecteur
  (set-polygone-vtrans! poly (+vect (polygone-vtrans poly) GRAVITE)))
```

- Complexité : O(1)

Evolution libre d'un polygone

Ainsi, lorsque le polygone n'est en contact avec aucun objet,
il suffit, pour le faire passer à l'état suivant,
de lui faire subir séquentiellement une translation,
puis une rotation, puis finalement l'effet de la gravité.

Détection des chocs contre les parois

- Pour savoir si un polygone touche l'une des parois de la fenêtre d'affichage, il suffit de savoir si les points extrema (ceux dont l'une des coordonnées majore celles de tous les autres points) se trouvent ou non dans la fenêtre.
- Par exemple, pour rechercher l'indice du point d'abscisse minimale, nous pouvons utiliser la fonction :

```
; minx : polygone -> int
(define (minx poly)
  ; iter : int x int -> int
  (define (iter i indice-min)
    ; Si i dépasse la taille du polygone
    (if (<= (polygone-size poly) i)
        ; On renvoit l'indice du point de plus petit abscisse
        indice-min
        ; Sinon, si l'abscisse du point courant est plus petit que celui de indice-min
        (if (< (vect-x (masse-r (vector-ref (polygone-poly poly) (car L))))
               (vect-x (masse-r (vector-ref (polygone-poly poly) indice-
                                            min))))
            ; On itére en modifiant l'indice
            (iter (add1 i) i)
            (iter (add1 i) indice-min))))
  (iter 1 0))
```

- Cette méthode, en O((polygone-size poly)), est extrêmement naïve, car elle parcourt l'ensemble des points de tout le polygone...

Détection des chocs contre les parois

- Nous pouvons, pour améliorer cette recherche, suivre deux pistes :
  - Ne pas parcourir tout le polygone, mais seulement les points de son enveloppe convexe : en effet, les extrema sont bien situés sur cette dernière, et si le polygone n'est pas convexe, l'algorithme précédent sera en O(length (polygone-ev poly)).
  - Utiliser le principe de dichotomie pour parcourir plus efficacement le nuage de point (la complexité devient alors logarithmique).

Détection des chocs contre les parois

- Nous nous contenterons, devant les faibles tailles de données (au plus quelques centaines de points), d'un parcours linéaire. En revanche l'application de la première remarque est immédiate.

```
(define (minx-ev poly)
  ; iter : list x int -> int
  (define (iterev L indice-min)
    ; Si la liste est vide
    (if (null? L)
        ; On renvoit l'indice du point de plus petit abscisse
        indice-min
        ; Sinon, si l'abscisse du (car L) est plus petit que celui de indice-min
        (if (< (vect-x (masse-r (vector-ref (polygone-poly poly) (car L))))
               (vect-x (masse-r (vector-ref (polygone-poly poly) indice-
                                            min))))
            ; On itére en modifiant l'indice
            (iter (cdr L) (car L))
            (iter (cdr L) indice-min))))
  (iter (polygone-ev poly) 0))
```

Détection des chocs contre les parois

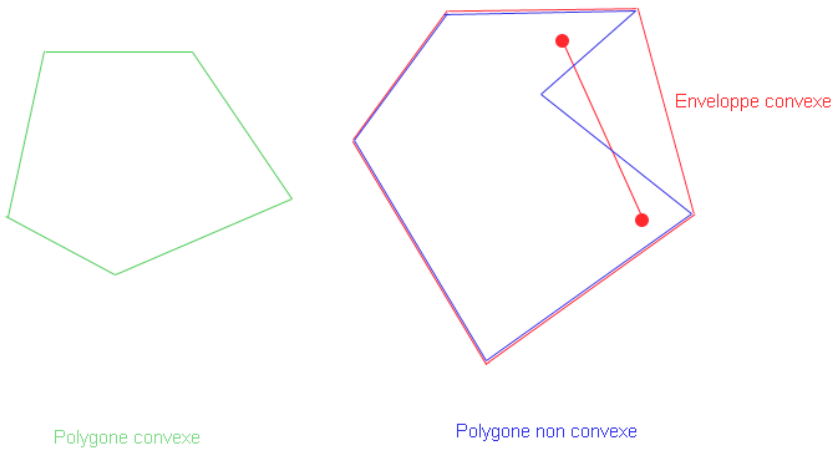
- Nous nous contenterons, devant les faibles tailles de données (au plus quelques centaines de points), d'un parcours linéaire. En revanche l'application de la première remarque est immédiate.
- Comme le montre cet exemple (également visionnable à l'adresse http://projetphysics.teria.org/Videos_du_slideshow/ev.html), le gain en terme de nombre de points parcourus peut être sensible si l'objet considéré n'est pas convexe...

Reste à calculer l'enveloppe convexe

d'un polygone

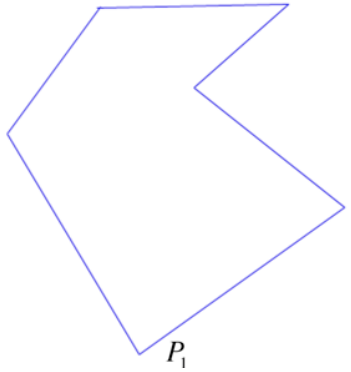
Enveloppe convexe

- Le problème de recherche de l'enveloppe convexe d'un polygone P revient à trouver un polygone convexe C (polygone tel que pour tout couple de point à l'intérieur de celui-ci, le segment les liant est inclus dans ce polygone) dont les sommets appartiennent à P, et tel que P soit à l'intérieur de C.



Présentation de l'algorithme de Graham

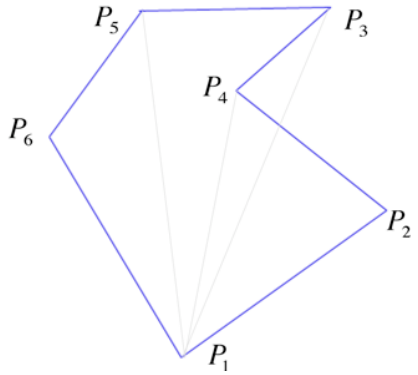
- Nous allons résoudre ce problème par l'algorithme de Graham, optimal dans le pire des cas, en O(nlog(n))



Soit P1 le point d'ordonnée minimale le plus à droite.

Présentation de l'algorithme de Graham

- Nous allons résoudre ce problème par l'algorithme de Graham, optimal dans le pire des cas, en O(nlog(n))

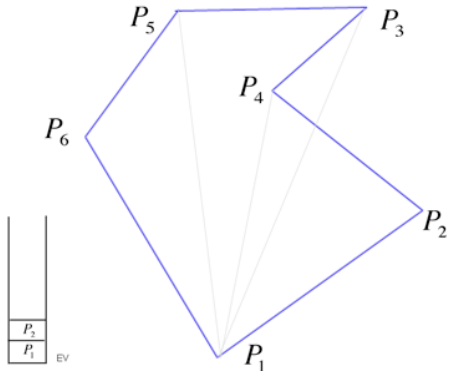


On classe les points par ordre d'angles croissants

par rapport à P1

Présentation de l'algorithme de Graham

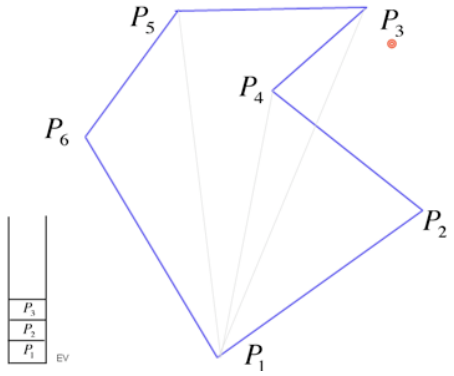
- Nous allons résoudre ce problème par l'algorithme de Graham, optimal dans le pire des cas, en O(nlog(n))



On initialise une pile EV par les points P1 et P2.

Présentation de l'algorithme de Graham

- Nous allons résoudre ce problème par l'algorithme de Graham, optimal dans le pire des cas, en O(nlog(n))

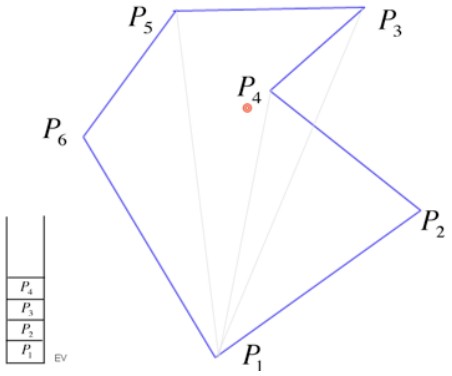


On avance dans la liste :

Tant que la ligne est convexe...

Présentation de l'algorithme de Graham

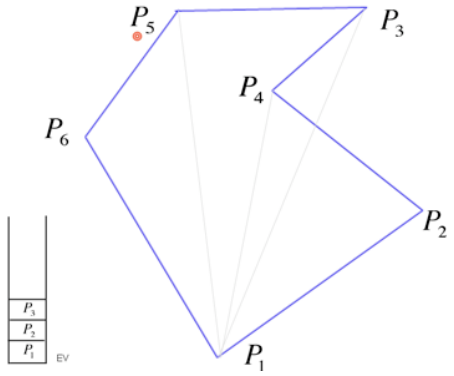
- Nous allons résoudre ce problème par l'algorithme de Graham, optimal dans le pire des cas, en O(nlog(n))



On empile le point courant dans la EV.

Présentation de l'algorithme de Graham

- Nous allons résoudre ce problème par l'algorithme de Graham, optimal dans le pire des cas, en O(nlog(n))

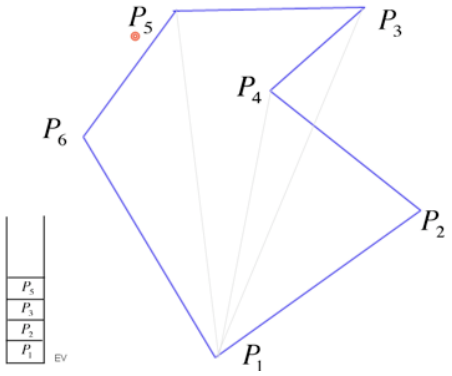


Si la ligne ne l'est plus :

On dépile le dernier élément inséré dans EV...

Présentation de l'algorithme de Graham

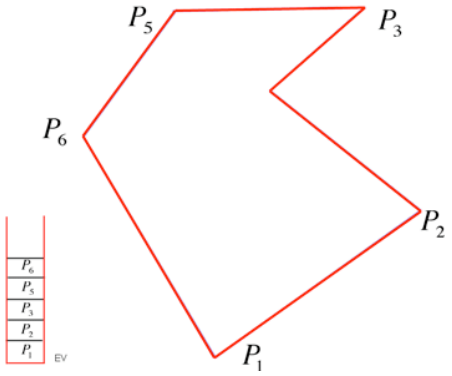
- Nous allons résoudre ce problème par l'algorithme de Graham, optimal dans le pire des cas, en O(nlog(n))



Puis on empile le point courant

Présentation de l'algorithme de Graham

- Nous allons résoudre ce problème par l'algorithme de Graham, optimal dans le pire des cas, en O(nlog(n))



On s'arrête quand le dernier point est traité !

Implémentation de l'algorithme de Graham

- On se munit d'une structure abstraite de pile :

```
; Une pile est une structure à deux champs :
; sa taille et la liste qui la compose
(define-struct pile (size l))
; Une pile est vide si elle est de taille nulle
(define (vide? pile)
  (zero? (pile-size pile)))
; Une pile initiale est vide
(define (init-pile)
  (make-pile 0 '()))
```

Implémentation de l'algorithme de Graham

- On se munit d'une structure abstraite de pile :

```
; Ajout d'un élément à une pile - effet de bords
(define (push! x pile)
  (set-pile-size! pile (add1 (pile-size pile)))
  (set-pile-l! pile (cons x (pile-l pile))))
; pop! : pile* -> int
(define (pop! pile)
  (if (vide? pile)
      ; La liste ne doit pas être vide
      (error "Pile vide !")
      (begin
        ; On décrémente la taille de la pile
        (set-pile-size! pile (- (pile-size pile) 1))
        ; On effectue une sauvegarde du premier élément...
        (let ((copie (car (pile-l pile))))
          ; ...avant de le supprimer
          (set-pile-l! pile (cdr (pile-l pile)))
          ; Puis on le restitue
          copie))))
```

Implémentation de l'algorithme de Graham

- On se munit d'une structure abstraite de pile :

```
; elmt1 : pile* -> int
; Renvoit le premier élément de la pile
(define (elmt1 pile)
  (if (vide? pile)
      (error "Pile vide !")
      (car (pile-1 pile))))
; elmt2 : pile* -> int
; Renvoit le second élément de la pile
(define (elmt2 pile)
  (if (<= (pile-size pile) 1)
      (error "Pas assez d'elements dans la pile !")
      (cadr (pile-1 pile))))
```

Implémentation de l'algorithme de Graham

- On crée la fonction indice-depart retournant l'indice du point d'ordonnée minimale le plus à droite possible :

```
; list-min : poly -> list
; Renvoit la liste du (ou des) point(s) dont l'ordonnée est minimale
(define (liste-min poly)
  ; Itérativement, on balaye tous les points du polygone
  (define (iter i liste-indices-min)
    ; Si on les a tous traités
    (if (<= (polygone-size poly) i)
        ; On renvoit la liste des indices
        liste-indices-min
        ; Sinon, on compare l'ordonnée du point courant avec l'ordonnée minimale trouvée
        (let ((a (vect-y (masse-r (vector-ref (polygone-poly poly) i))))
              (b (vect-y (masse-r (vector-ref (polygone-poly poly) (car liste-indices-min))))))
          (cond
            ; Si le point courant est plus bas, on remplace la liste ((Oy) dirigé vers le bas)
            ((> a b) (iter (add1 i) (list i)))
            ; S'il est à la même hauteur, on ajoute l'indice à la liste
            ((= a b) (iter (add1 i) (cons i liste-indices-min)))
            ; S'il est plus haut, on ne modifie pas la liste...
            (else (iter (add1 i) liste-indices-min))))))
  (iter 1 (list 0)))
```

- Complexité en O((polygone-size poly)).

Implémentation de l'algorithme de Graham

- On crée la fonction indice-depart retournant l'indice du point d'ordonnée minimale le plus à droite possible :

```
; max : poly -> int
; A partir de la liste liste-indice-min obtenue précédemment,
; max renvoit l'indice du point initial de l'algorithme
(define (max poly)
  (define (iter L acc)
    (if (null? L)
        acc
        ; On compare désormais les abscisses des points
        (if (< (vect-x (masse-r (vector-ref (polygone-poly poly) acc)))
               (vect-x (masse-r (vector-ref (polygone-poly poly) (car L)))))
            (iter (cdr L) (car L))
            (iter (cdr L) acc))))
  (let ((liste-indice-min (liste-min poly)))
    (iter  (cdr liste-indice-min) (car liste-indice-min))))
```

- On en déduit la fonction cherchée, de complexité linéaire :

```
(define (indice-depart poly)
  (max poly))
```

Implémentation de l'algorithme de Graham

- On crée la fonction create-liste-angles associant à chaque indice de point l'angle qu'il fait avec pinit, le point initial de l'algorithme, d'indice init.

```
; create-liste-angles : polygone x int x vecteur -> list
(define (create-liste-angles poly init pinit)
  ; On travaille sur une liste L
  (let* ((L '()))
    ; Il faut veiller à ne pas intégrer le point initial dans L
    (for i from 0 to (- init 1)
         ; On calcule les coordonnées du vecteur Pinit Pi
         ; En veillant à l'orientation de l'axe (Oy) en Scheme
         (let ((vecteur (make-vect (- (vect-x (masse-r (vector-ref (polygone-poly poly) i))) (vect-x pinit))
                                   (- (vect-y pinit) (vect-y (masse-r (vector-ref (polygone-poly poly) i)))))))
           (if (zero? (vect-y vecteur))
               ; Si les points sont à même altitude, l'angle est pi
               ; car pinit est le plus à droite des points les plus bas
               (set! L (cons (cons i pi) L))
               (set! L (cons (cons i (vect-angle vecteur)) L)))))
    ; On procède de même pour les points d'indices supérieurs à init
    (for i from (add1 init) to (- (polygone-size poly) 1)
         (let ((vecteur (make-vect (- (vect-x (masse-r (vector-ref (polygone-poly poly) i))) (vect-x pinit))
                                   (- (vect-y pinit) (vect-y (masse-r (vector-ref (polygone-poly poly) i)))))))
           (if (zero? (vect-y vecteur))
               (set! L (cons (cons i pi) L))
               (set! L (cons (cons i (vect-angle vecteur)) L)))))
    ; On retourne L
    L))
```

- La complexité de l'algorithme est clairement en O((polygone-size poly))

Implémentation de l'algorithme de Graham

- On développe un prédicat renvoyant #t si le point d'indice P est à gauche de la droite passant par les points d'indices A et B

```
(define (a-gauche P A B)
  (let* ((PA (-vect (masse-r (vector-ref (polygone-poly poly) A))
                    (masse-r (vector-ref (polygone-poly poly) P))))
         (PB (-vect (masse-r (vector-ref (polygone-poly poly) B))
                    (masse-r (vector-ref (polygone-poly poly) P))))
         (d (- (* (vect-x PA) (vect-y PB)) (* (vect-x PB) (vect-y PA)))))
    (> 0 d)))
```

- On suppose de plus que l'on dispose d'une fonction **`(trier-selon-angles poly L pinit)`** de tri (implémenter dans notre projet par le tri fusion, en O(n log(n))) renvoyant la liste des indices des points du polygone triés par ordre croissant d'angles (avec, dans le cas de deux points ayant le même angle formé avec Pinit, pour seul indice apparaissant dans la liste l'indice du point le plus éloigné du point initial).

Implémentation de l'algorithme de Graham

On déduit finalement l'algorithme de l'enveloppe convexe d'un polygone en O(n log(n)) :

```
; enveloppe-convexe : polygone -> int list
(define (enveloppe-convexe poly)
  ; Si le polygone est de taille 1, l'enveloppe est réduite à ce seul point
  (if (= (polygone-size poly) 1)
      (list 0)
      (let* ((init (indice-depart poly))
             (pinit (masse-r (vector-ref (polygone-poly poly) init)))
             ; La pile qui fournira la solution
             (EV (init-pile))
             (lpts (trier-selon-angles poly (create-liste-angles poly init pinit) pinit))
             (m (length lpts))
             ; On stocke la liste des points triés dans une pile
             (ppts (make-pile m lpts)))
        ; On initialise EV avec l'élément initial
        (push! init EV)
        ; Ainsi que le point formant avec lui un angle minimal
        (push! (pop! ppts) EV)
        ; Tant qu'il reste des points à traiter
        (while (not (vide? ppts))
               (let ((A (elmt1 EV)) (B (elmt2 EV)))
                 ; Si le point courant ne perturbe pas la convexité de la ligne polygonale
                 (if (a-gauche (elmt1 ppts) A B)
                     ; On l'ajoute à EV (et on le supprime de ppts)
                     (push! (pop! ppts) EV)
                     ; Sinon... on dépile EV
                     (pop! EV))))
        (pile-l EV))))
```

Evolution d'un polygone dans la fenêtre d'affichage

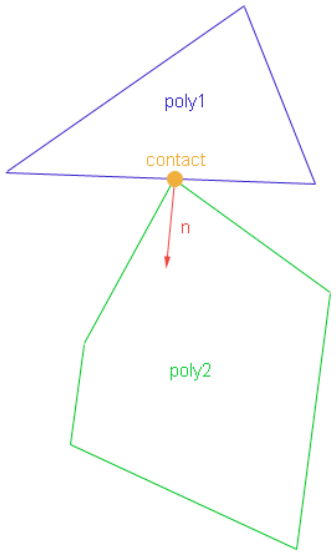
- Maintenant que nous savons parfaitement calculer (lors de leur création) l'enveloppe convexe de nos polygones, nous sommes en mesure de laisser évoluer, sous l'effet de la gravité, n'importe quel polygone.
- Nous procèderons selon l'algorithme rédigé en pseudo-language :

```
(define (evolution polygone)
  (A ,B ,C ,D <- indices des points extrema de l 'enveloppe convexe)
  (si ((pA est dans la fenêtre)
       et (pB est dans la fenêtre)
       et (pC est dans la fenêtre)
       et (pD est dans la fenêtre))
      alors
      (debut (translation polygone)
             (rotation polygone)
             (gravite polygone))
      sinon
      (choc polygone paroi)))
```

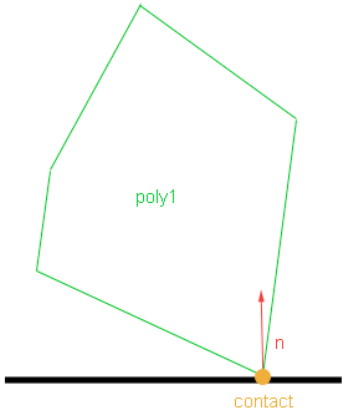
Evolution d'un polygone dans la fenêtre d'affichage

- Nous ne détaillerons pas l'écriture des prédicats "pX est dans la fenêtre", en revanche, intéressons nous au développement de la fonction choc.
- En réalité, nous n'allons pas programmer la fonction `(choc polygone paroi)` mais implémenter directement la fonction `(choc poly1 poly2 contact normale e)` qui modifiera les polygones 1 et 2 se percutant au point *contact* (un vecteur passé en argument) au cours d'un choc dont le coefficient d'élasticité est e, et ayant une normale n pointant vers le poly1.
- Cette dernière fonction permettra bien de traiter, en plus du choc entre deux polygones, le choc entre un polygone et une paroi : il suffit de prendre comme poly2 un polygone réduit à une masse infinie positionnée à l'endroit du point (extrema) le plus proche de la paroi, comme le montre l'illustration suivante.

Evolution d'un polygone dans la fenêtre d'affichage



Choc entre deux polygones



Rebond d'un polygone sur un mur

Evolution d'un polygone dans la fenêtre d'affichage

La fonction choc sera donc définie de la manière suivante, conformément aux formules vues dans les parties précédentes :

```
(define (choc poly1 poly2 contact normale e)
  (let* ((pi/2 (/ pi 2.0))
         ; Vecteur r1 : centre de masse de poly1 vers le point de contact
         (r1 (-vect contact (masse-r (polygone-cm poly1))))
         ; Vecteur r1 : centre de masse de poly2 vers le point de contact
         (r2 (-vect contact (masse-r (polygone-cm poly2))))
         ; Définition des vecteurs orthogonaux à ceux calculés juste avant
         (r1ortho (rotation (make-vect 0 0) r1 (+ pi/2)))
         (r2ortho (rotation (make-vect 0 0) r2 (+ pi/2)))
         ; Calcul de j à partir de la dernière formule de la partie précédente
         (j (/ (- (* (+ 1 e)
                     (.scal normale
                            (-vect (+vect (polygone-vtrans poly1) (*vect (polygone-vang poly1) r1ortho))
                                   (+vect (polygone-vtrans poly2) (*vect (polygone-vang poly2) r2ortho))))))
               (+ (/ (masse-m (polygone-cm poly1)))
                  (/ (masse-m (polygone-cm poly2)))
                  (/ (sqr (.scal normale r1ortho)) (polygone-i poly1))
                  (/ (sqr (.scal normale r2ortho)) (polygone-i poly2))))))
    ; Modification de la vitesse angulaire du poly1
    (set-polygone-vang! poly1 (+ (polygone-vang poly1) (/ (* j (.scal normale r1ortho)) (polygone-i poly1))))
    ; Modification de la vitesse angulaire du poly2
    (set-polygone-vang! poly2 (+ (polygone-vang poly2) (/ (* (- j) (.scal normale r2ortho)) (polygone-i poly2))))
    ; Modification de la vitesse de translation du poly1
    (set-polygone-vtrans! poly1 (*vect 1 (+vect (polygone-vtrans poly1) (*vect (/ j (masse-m (polygone-cm poly1))) normale))))
    ; Modification de la vitesse de translation du poly2
    (set-polygone-vtrans! poly2 (+vect (polygone-vtrans poly2) (*vect (/ (- j) (masse-m (polygone-cm poly2))) normale)))))
```

Que faire si l'on est en présence de deux polygones ?

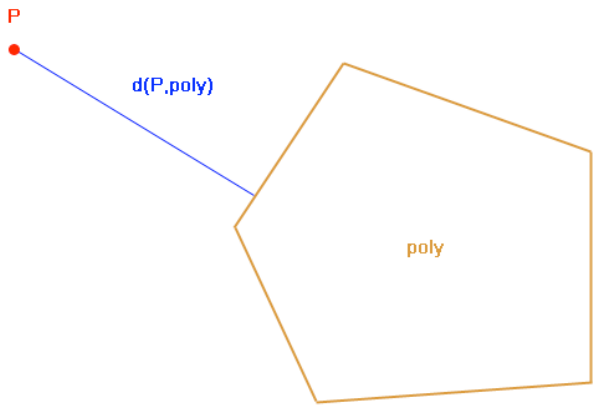
Détection de collisions entre deux polygones convexes

- Pour faire s'entrechoquer plusieurs polygones entre eux, nous allons d'abord poser quelques restrictions à notre modèle :
  - Les polygones seront des convexes
  - Deux polygones au plus seront dans la fenêtre d'affichage
  - Les polygones auront des vitesse faibles : en particulier, le champ de gravité ne sera pas important
- Modulo ces quelques limitations pratiques, nous allons être capable de :
  - Déterminer à quel moment deux polygones entrent en collision
  - Déterminer le point d'impact
  - Connaître la normale au choc

Grâce à la fonction précédente, nous saurons faire interagir deux polygones entre eux.

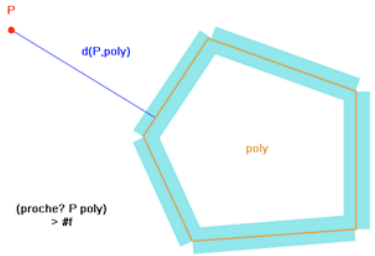
Présentation de l'algorithme de détection

- Pour savoir si deux polygones sont en collision, nous allons nous demander, pour le polygone1 d'abord, puis pour le polygone 2, s'il existe un point pour lequel la distance qui le sépare d'un coté de l'autre polygone est faible ou pas.
- On définit la distance entre un point et un polygone comme étant la plus courte distance entre ce point et l'un des cotés du polygone :

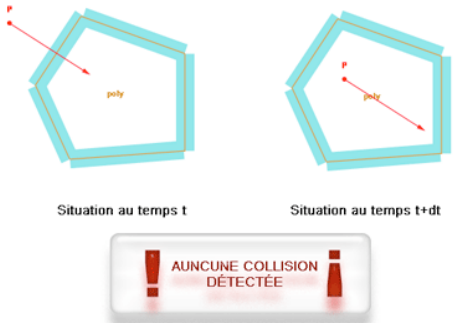


Présentation de l'algorithme de détection

- L'hypothèse des faibles vitesses est dictée par la technique de reconnaissance de la collision choisie :
- En fait, nous détecterons si un point appartient ou non à la bande bleue ci-dessous :



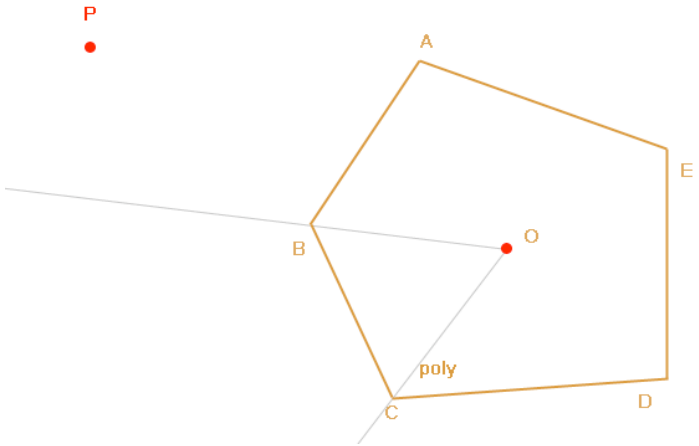
- Le risque avec les vitesses élevées étant d'être dans cette configuration :



Présentation de l'algorithme de détection

- En fait, on peut définir la distance entre un point et un polygone comme étant la distance de ce point au segment [AB], de telle sorte que si O est l'isobarycentre des points constituant le polygone, le point appartient au secteur défini par AOB.

Vite un schéma !

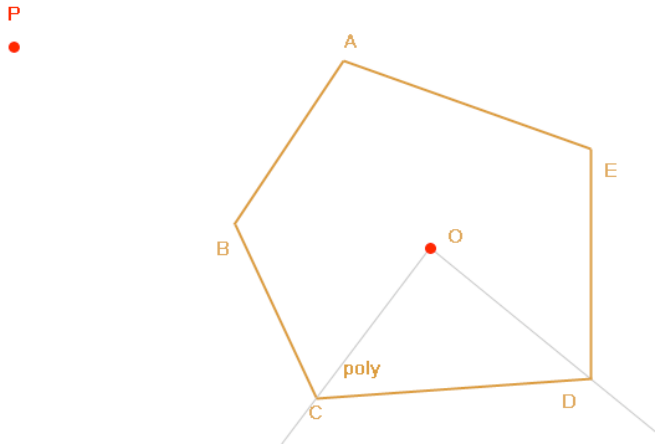


P n'appartient pas au secteur BOC

Présentation de l'algorithme de détection

- En fait, on peut définir la distance entre un point et un polygone comme étant la distance de ce point au segment [AB], de telle sorte que si O est l'isobarycentre des points constituant le polygone, le point appartient au secteur défini par AOB.

`Vite un schéma !`

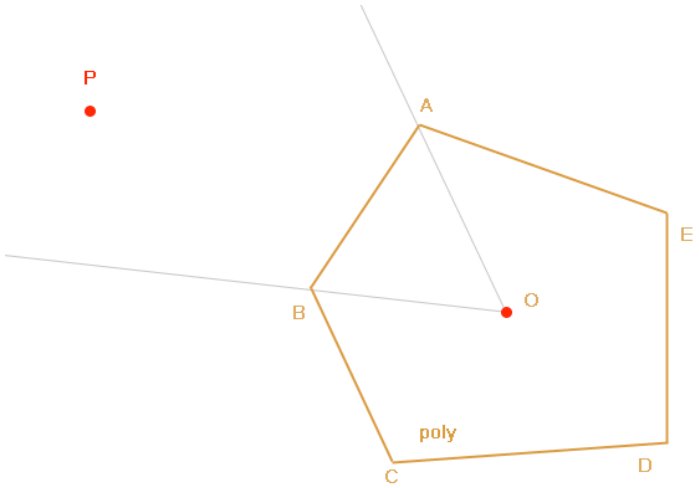


Ni au secteur COD

Présentation de l'algorithme de détection

- En fait, on peut définir la distance entre un point et un polygone comme étant la distance de ce point au segment [AB], de telle sorte que si O est l'isobarycentre des points constituant le polygone, le point appartient au secteur défini par AOB.

Vite un schéma !

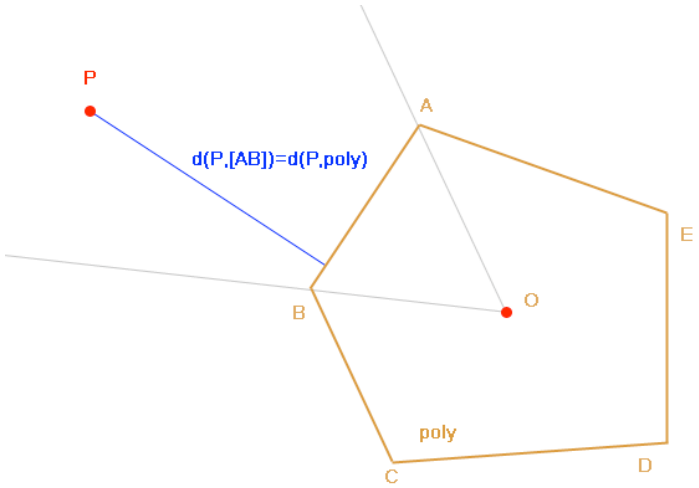


En revanche, il appartient au secteur AOB

Présentation de l'algorithme de détection

- En fait, on peut définir la distance entre un point et un polygone comme étant la distance de ce point au segment [AB], de telle sorte que si O est l'isobarycentre des points constituant le polygone, le point appartient au secteur défini par AOB.

Vite un schéma !



La distance du point P à poly

est bien la distance d(P,[AB])

Présentation de l'algorithme de détection

- Le traitement de la détection de colision de deux convexes C1 et C2 de tailles n1 et n2, de centres O1 et O2 peut se faire comme suit :

```
(define (collision C1 C2)
  (pour (tous les sommets S de C1)
        ; Si S appartient au secteur angulaire (AO2B)
        ((A.B) <- (secteur angulaire auquel appartient S))
        (d <- d (S ,(A ,B)))
        (si (d < epsilon)
            (traitement de la collision entre C1 et C2)))
  (pour (tous les sommets S de C2)
        ; Si S appartient au secteur angulaire (AO1B)
        ((A.B) <- (secteur angulaire auquel appartient S))
        (d <- d (S ,(A ,B)))
        (si (d < epsilon)
            (traitement de la collision entre C1 et C2))))
```

L'algorithme de détection

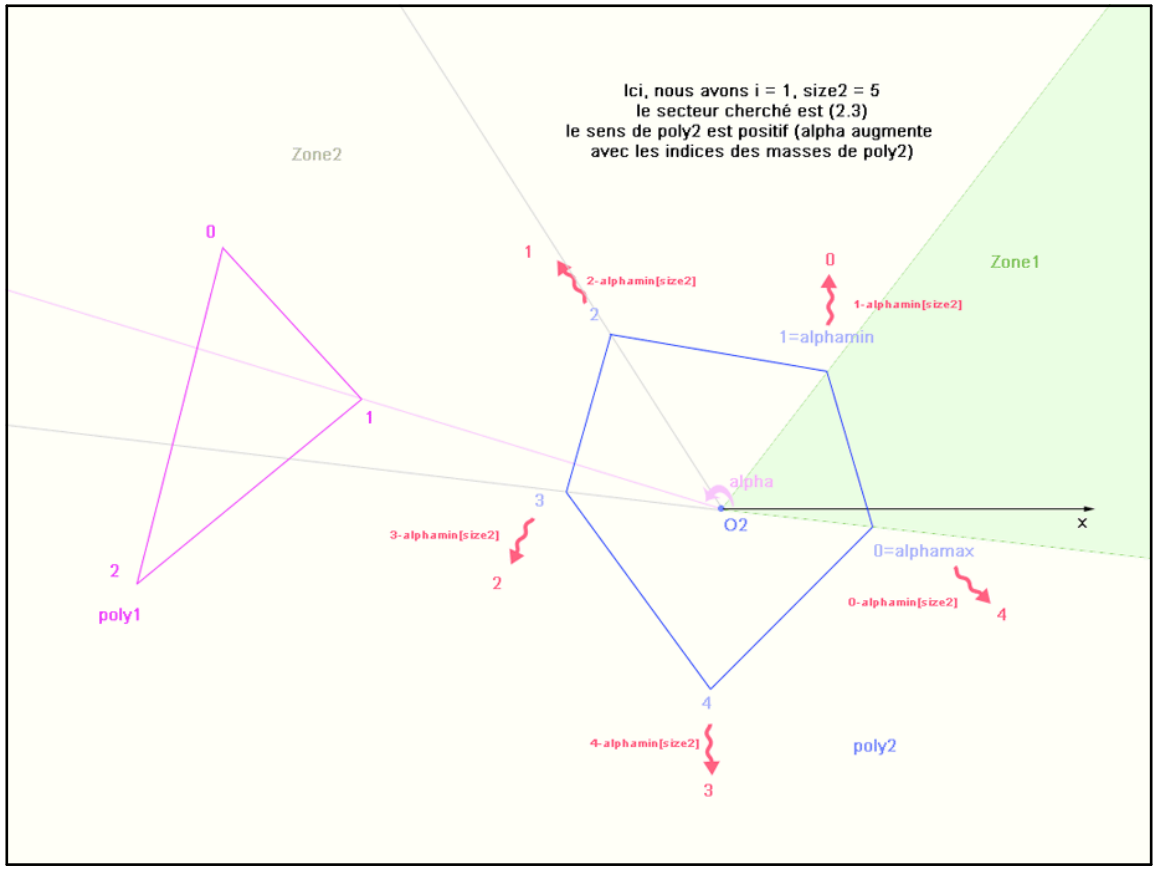
- Définissons d'abord deux fonctions utiles :

```
; mod2pi : float -> float
; Renvoit le paramètre modulo 2pi
(define (mod2pi x)
  (let ((2pi (* 2 pi)))
    (cond ((< x 0) (mod2pi (+ x 2pi)))
          ((<= 2pi x) (mod2pi (- x 2pi)))
          (else x))))
```

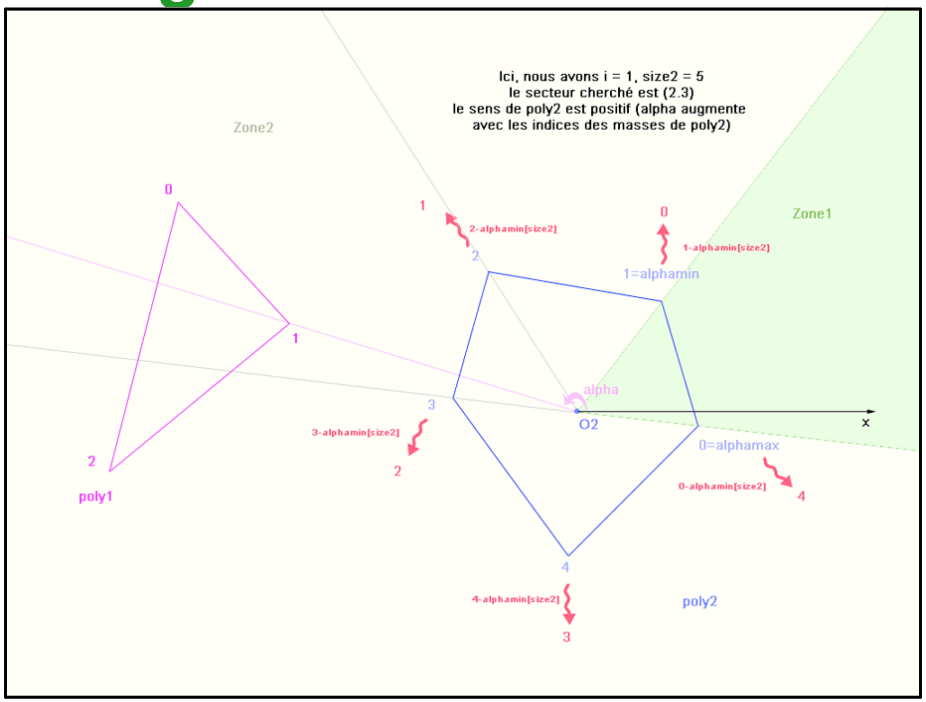
```
; alphaext : polygone -> pair
; Renvoit les indices des masses possédant le plus petit et le plus grand alpha
(define (alphaext poly)
  ; Renvoit l'indice du point de plus petit alpha (parmis les deux points passés en argument)
  (define (min i j)
    (if (< (masse-alpha (vector-ref (polygone-poly poly) i))
           (masse-alpha (vector-ref (polygone-poly poly) j)))
        i
        j))
  ; Renvoit l'indice du point de plus grand alpha (parmis les deux points passés en argument)
  (define (max i j)
    (if (> (masse-alpha (vector-ref (polygone-poly poly) i))
           (masse-alpha (vector-ref (polygone-poly poly) j)))
        i
        j))
  (do ((k 0 (add1 k)) (mini 0 (min mini k)) (maxi 0 (max maxi k)))
    ((= k (polygone-size poly)) (cons mini maxi))))
```

L'algorithme de détection

- Vient ensuite la fonction `(secteur i poly1 poly2 alphaext2)` qui renvoit le secteur `(A.B)`, comportant les indices des masses du poly2 tel que le ième point du poly1 appartienne au secteur AO2B, connaissant les indices des masses du poly2 ayant les alpha extrema.



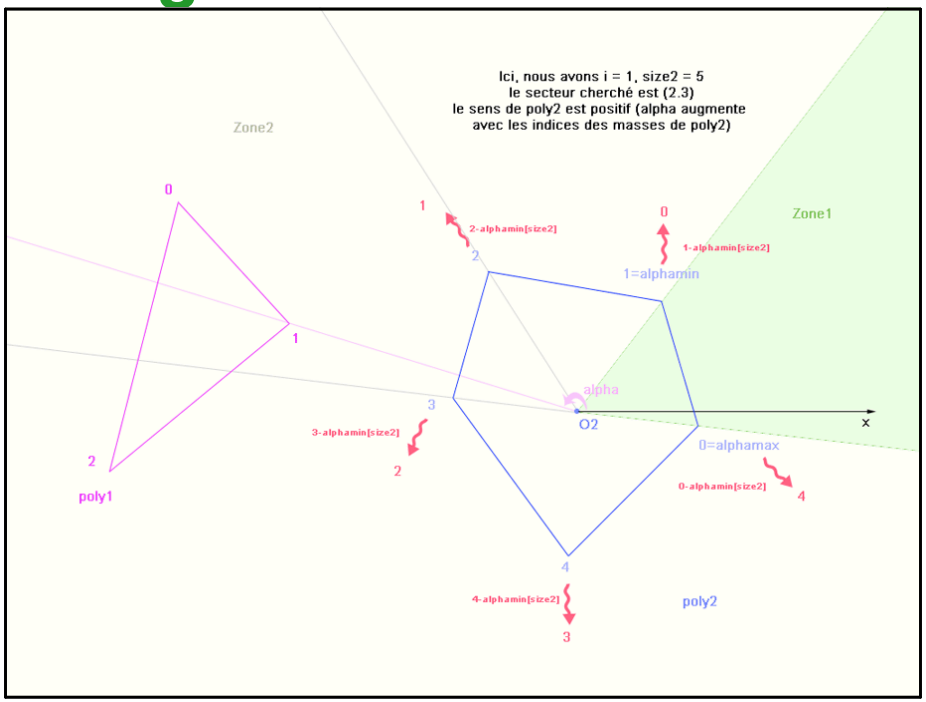
L'algorithme de détection



- On définit les constantes suivantes :

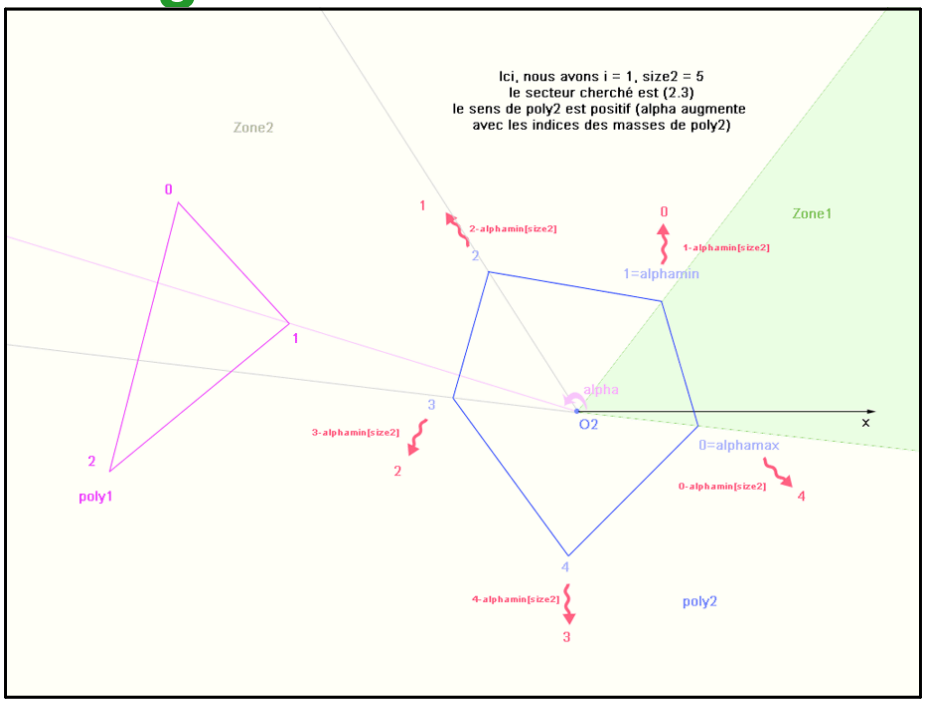
```
(alpha (mod2pi (vect-angle (make-vect (- (vect-x (masse-r (vector-ref (polygone-poly poly1) i)))
                                         (vect-x (masse-r (polygone-cm poly2))))
                                      (- (vect-y (masse-r (polygone-cm poly2)))
                                         (vect-y (masse-r (vector-ref (polygone-poly poly1) i))))))))
(size2 (polygone-size poly2))
(alphamin (car alphaext2))
(alphamax (cdr alphaext2))
(sens (if (= (modulo (add1 alphamax) size2) alphamin) 'positif 'negatif))
```

L'algorithme de détection



- Evidemment, nous pourrions définir **`secteur`** en parcourant tous les points de poly2 à partir d'alphamin, et en nous arrêtant à la première masse dont l'angle dépasserait alpha...
- Dans le pire des cas, il faudrait parcourir les size2 points de poly2. En faisant le test pour les size1 points de poly1, la complexité de l'algorithme de détection serait en O(size1*size2) (puisque les calculs de distances sont O(1)). Pour des polygones comportant beaucoup de points (une centaine), cet algorithme serait impraticable...

L'algorithme de détection



- En fait, nous pouvons optimiser l'algorithme naïf et lui donner une complexité logarithmique. De deux choses l'une :
  - Soit le point appartient à la zone 1, et il n'y a rien à faire.
  - Sinon, il est dans la zone deux, et l'on peut par dichotomie, en réindexant de 0 à size2-1 les points de poly2 (par la transformation x->x-alphamin[size2] dans le cas du sens positif), définir le secteur auquel il appartient.

L'algorithme de détection

- Définissons la fonction **`(dicho i j)`** tenant compte de l'observation suivante :

```
; On utilisera dicho avec les conditions initiales
; (0,size2-1), et l'on aura toujours i<j
(define (dicho i j)
  ; Si le sens est positif
  (if (equal? sens 'positif)
      (cond
        ; Si l'on considère deux points consécutifs
        ((<= (- j i) 1)
         ; Si le point de poly1 est dans ce secteur
         (if (and (<= alpha (masse-alpha (vector-ref (polygone-poly poly2) (modulo (+ j alphamin) size2))))
                  (>= alpha (masse-alpha (vector-ref (polygone-poly poly2) (modulo (+ i alphamin) size2)))))
             ; On retourne ce secteur
             (cons (modulo (+ i alphamin) size2) (modulo (+ j alphamin) size2))
             ; Sinon, le point n'appartient au secteur
             #f))
        ; Si le point appartient au secteur (i,j) d'amplitude >= 2
        ((and (<= alpha (masse-alpha (vector-ref (polygone-poly poly2) (modulo (+ j alphamin) size2))))
              (>= alpha (masse-alpha (vector-ref (polygone-poly poly2) (modulo (+ i alphamin) size2)))))
         ; On détermine le milieu du secteur
         (let ((new (quotient (+ i j) 2)))
           ; Le secteur cherché est soit dans le secteur (i,new), soit dans (new,j)
           (or (dicho i new) (dicho new j))))
        (else #f))
      ; Si le sens est négatif, la transformation de réindexation est x->x-alphamax[size2]
      (cond ((<= (- j i) 1)
             (if (and (>= alpha (masse-alpha (vector-ref (polygone-poly poly2) (modulo (+ j alphamax) size2))))
                      (<= alpha (masse-alpha (vector-ref (polygone-poly poly2) (modulo (+ i alphamax) size2)))))
                 (cons (modulo (+ i alphamax) size2) (modulo (+ j alphamax) size2))
                 #f))
            ((and (>= alpha (masse-alpha (vector-ref (polygone-poly poly2) (modulo (+ j alphamax) size2))))
                  (<= alpha (masse-alpha (vector-ref (polygone-poly poly2) (modulo (+ i alphamax) size2)))))
             (let ((new (quotient (+ i j) 2)))
               (or (dicho i new) (dicho new j))))
            (else #f))))
```

L'algorithme de détection

- On en déduit la fonction secteur :

```
(define (secteur i poly1 poly2 alphaext2)
  (if
   ; Si le point est dans la zone 1
   (or (<= alpha (masse-alpha (vector-ref (polygone-poly poly2) alphamin)))
       (>= alpha (masse-alpha (vector-ref (polygone-poly poly2) alphamax))))
   ; (alphamin . alphamax) est le secteur auquel appartient le point
   (cons alphamin alphamax)
   ; Sinon, on le détermine avec dicho
   (dicho 0 (- size2 1))))
```

- La fonction secteur est bien en O(log(size2)) : à chaque itération, la longueur de l'intervalle passé en paramètre de dicho (j-i) est divisée par deux.

- Pour traiter la collision entre poly1 et poly2, il faut définir les secteurs auxquels appartiennent les size1 points de poly1 (avec pour chacun d'eux, O(log(size2)) opérations), puis les secteurs des size2 points de poly2. Finalement :

L'algorithme de détection est quasi-linéaire, en O(size1 log(size2) + size2 log(size1))

- Préambule
- Introduction
- Partie I : Physique du point
- Partie II : Quantité de mouvement et chocs
- Partie III : Mouvements de rotation
- Partie IV : Implémentation du projet
- Conclusion

Conclusion

- Malgrès ses limites, notre projet permet d'apréhender de manière un peu plus concrète les phénomènes physiques auxquels nous sommes confrontés au quotidien.
- Le fait qu'il use d'exemples académiques (directement tirés de cours), ou plus ludiques (le dessin à main levée) est suceptible d'intéresser le plus grand nombre de personnes. Et la présente documentation, relativement détaillée, permet à chacun de (re)découvrir les principes de bases de la mécanique, en abordant certains points de manière moins conventionnelle qu'on pourrait le penser (cf la définition d'une force, ou le postulat de Descartes comme fondement de la mécanique).
- La dernière partie peut également pousser les lecteurs les plus inspirés à réfléchir aux solutions des limitations imposées par notre implémentation : envisager la triangularisation des polygones pour traiter le cas des non convexes, généraliser le problème à N corps, gérer la friction (comme cela est fait dans la partie objet du projet)...

Conclusion

- Pour ce slideshow, ainsi que le reste du projet, nous avons utilisés : DrScheme, Pages, Appleworks, Slideshow, Equation editor, Adobe Photoshop, Jing, iWeb, Cyberduck.
- Ce projet a été réalisé à partir :
  - De notes prises lors du cours de *Mécanique I* dispensé en première année de licence MP par M. COULLET
  - Du polycopié du cours de complexité de M. AVNAIM
  - Et, bien évidemment, du cours de l'option *Programmation Fonctionnelle II*
- Nous tenons à remercier ces enseignants pour l'aide qu'ils nous ont apportée au cours du projet...
- et espèrons vivement vous retrouver sur notre site web.

```
(the-end)
```