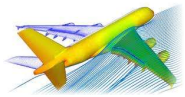
ANIMATION PHYSIQUE EN SYNTHESE D'IMAGES

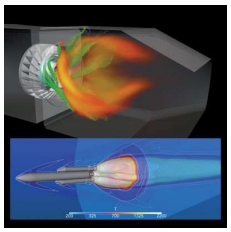
DÉRIVÉES ET MÉCANIQUE DU POINT
SYSTÈME DE PARTICULES
COLLISIONS
MASSE-RESSORT

Alexandre Meyer
Equipe SAARA, Laboratoire LIRIS
Université Lyon 1

Math/info et la physique



- Les physiciens programment
  - Offre de formation Lyon 1
    - Mention physique, parcours Physique
    - 2e année, semestre 3
    - ➔ UE Programmation C/C++



S3-UE1 [UE Obligatoire] (6 Crédits) :
Electromagnétisme (6 cts)

S3-UE2 [UE Obligatoire] (6 Crédits) :
Mathématiques 3 (Mécanique, Physique, SPI) (6 cts)

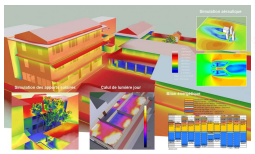
S3-UE3 [UE Obligatoire] (6 Crédits) :
Programmation C/C++ (6 cts)

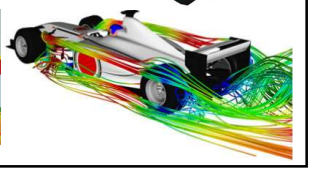
S3-UE4 [UE Obligatoire] (6 Crédits) :
Mécanique des systèmes de solides et de points matériels (6 cts)

Math/info et la physique

La physique utilise la simulation numérique tout le temps

- Mécanique
- Aérodynamique
- Simulation de fluide
- Thermique
- Etc.





Math/info et la physique

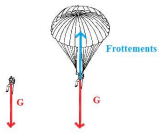
- Les équations en physique
  - Souvent des équations différentielles
    = relation entre une fonction et ses dérivées

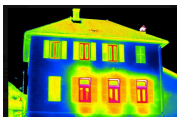
$$af''+bf'+cf+d=0$$

  - Par exemple, une très simple : Position = Vitesse x temps :

$$x(t)=v.t=\frac{dx(t)}{dt}.t$$

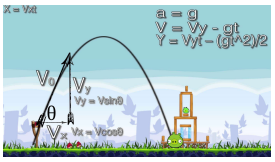
  - Mais aussi
    - Equation de la chaleur
    - Dynamique : F=m.x"
    - Etc.
  - Rarement de solution analytique
    - Sauf dans des cas simples ou théorique
      - en général ce que vous avez étudiez aux lycée

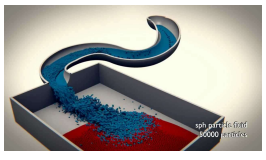




Physique en informatique graphique

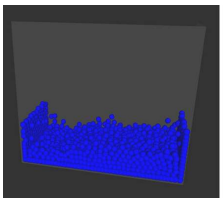
- Jeux vidéo
- Films
- …

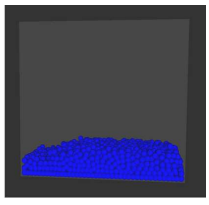




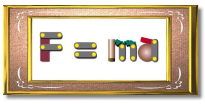
Système de particules

- On voudrait faire des particules en mouvement … pour commencer …
  - Pas de fluide, uniquement en collisions avec le sol

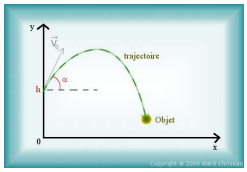




Besoin de Newton



- Accélération = masse * Force
  - Dérivée de la vitesse
    - Dérivée de la position
- Mais en informatique pas de calcul analytique
  - Toujours cette histoire de discret / continue
  - Plutôt un calcul itératif = une simulation



ANIMATION PHYSIQUE EN SYNTHESE D'IMAGES

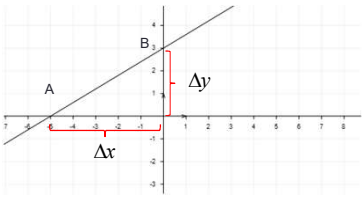
Dérivée et intégrale en informatique (sous entendu avec des données discrètes), ca change quoi ?

Pente d'une droite

$A(-5,0)$ $B(0,3)$

Entre x et x+1, on monte de 0.6

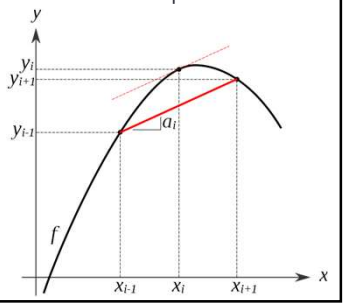
$$p_{AB} = \frac{\Delta y}{\Delta x} = \frac{f(b)-f(a)}{b-a} = \frac{3-0}{0-(-5)} = \frac{3}{5} = 0.6$$



Dérivée

La dérivée en un point de la fonction est la pente de la tangente à la courbe

$$f'(x_i) \simeq \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}}$$



Dérivée

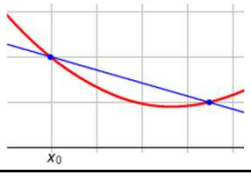
- On peut approcher une fonction f par développement limité

$$f(x+h) = f(x) + f'(x)\cdot h + \frac{f''(x)}{2}\cdot h^2 + \mathrm{O}(h^2)$$

Une approximation

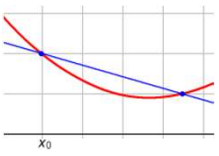
$$f'(x) \simeq \frac{f(x+h)-f(x)}{h} - \frac{f''(x)}{2}\cdot h \simeq \frac{f(x+h)-f(x)}{h}$$

$$f'(x) \cong \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$$



Dérivée

$$f'(x) = \frac{df(x)}{dx} \cong \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$$



- Dérivée de f par rapport à x
  - Notation f'(x) ou $\frac{df(x)}{dx}$
  - Signifie : variation de f quand on fait varier un petit peu x
  - Ou : **d**ifférence de f divisé par **d**ifférence de x
  - Ou : de combien augmente f quand on augmente un petit peu x

Dérivée : question 1

Soit *f* la fonction définie par $f(x)=3x^2.+2$

La fonction analytique dérivée *f* ' est définie par ?

a. $f'(x)=3$ **b.** $f'(x)=3x$ **c.** $f(x)=6x$

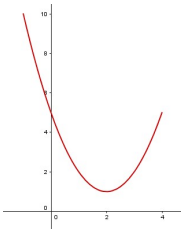
On peut aussi faire $\frac{df(x)}{dx} \cong \frac{f(x+h)-f(x)}{h}$

$= \frac{3(x+h)^2-3x^2-2}{h}$

=... Dérivée numérique

Dérivée : question 2

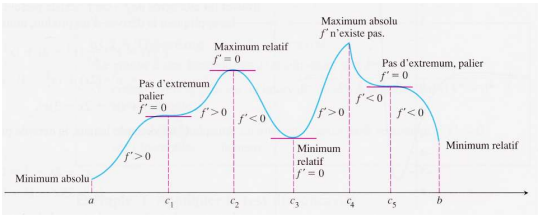
Soit *f* la fonction définie sur [–1 ; 4] représentée ci-contre.



Sur [–1 ; 4], la fonction dérivée *f* ' :
a. Est positive.
b. Est négative.
c. Change de signe.

Dérivée : croissante/nulle/décroissante

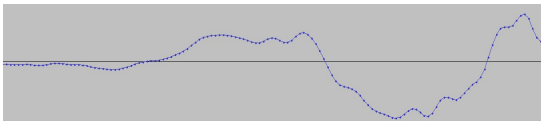
- Soit une fonction f définie sur [a , b]



- Remarque : a et b, les bornes, sont automatiquement des nombres critiques car la dérivée n'y existe pas.

Dérivée discrète

- En math, souvent calcul de dérivées analytique
  - cos'(x) = sin(s)
  - $(x^2+3x+5)' = 2x+3$
- En informatique
  - Souvent une fonction est représentée par une version discrète
  - Tableau de valeurs : float f[10000];
  - La dérivée est alors ?



Exemple

- **f(t)** : fonction donnant la hauteur de la Saône chaque jour en centimètres

Jour	...	105	106	107	108	109	110	111	112	113	114	115
Hauteur en cm		233	230	234	240	245	247	248	241	237	243	246

- f(109) = 245
  - au 109e jour la Saône a une hauteur de 245cm
- **Dérivée de la fonction f au jour 109 est donc de +2cm/jour**

$$\frac{df(t)}{dt} \cong \frac{f(t+dt)-f(t)}{dt} = \frac{f(109+1)-f(109)}{1} = 247-245 = 2$$

Question

- F(t) : fonction donnant la hauteur de la Saône chaque jour en centimètres

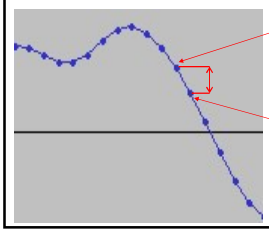
Jour	...	105	106	107	108	109	110	111	112	113	114	115
Hauteur en cm		233	230	234	240	245	247	248	241	237	243	246

- f(112) = 241
  - au 112e jour la Saône a une hauteur de 241cm
- **Dérivée de cette fonction f au jour 112 est donc de ?**

$$\frac{df(t)}{dt} \cong \frac{f(t+dt)-f(t)}{dt} = \frac{f(112+1)-f(112)}{1} = ?$$

Dérivée discrète

- En informatique
  - Souvent une fonction est représentée par une version discrète
  - Tableau de valeurs : float f[10000];
  - La dérivée est alors la différence entre 2 cases du tableau



f[x]

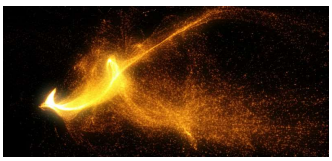
f[x+1]

$$\frac{df(x)}{dx} \cong \frac{f(x+h)-f(x)}{h}$$

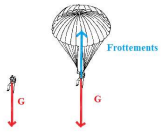
$$\cong \frac{f[x+1]-f[x]}{1}$$

ANIMATION PHYSIQUE EN SYNTHESE D'IMAGES

Mécanique du point / système de particules



Seconde loi de Newton



$$F = ma$$

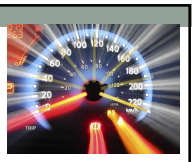
N kg m/s^2

- F : les forces en N
- m : la masse en kg
- a : accélération en $m.s^{-2}$

Seconde loi de Newton

L'accélération c'est quoi ?

Seconde loi de Newton



L'accélération c'est quoi ?

- C'est le fait d'aller plus vite ?
- En physique c'est aussi le fait d'aller moins vite
- L'accélération est une valeur, qui change au cours du temps
- Si l'accélération est nulle, la vitesse ne change pas
- Si l'accélération est positive, la vitesse augmente

- ~L'accélération est la variation de vitesse entre deux moments du temps

L'accélération est la variation de vitesse en 1 seconde

- Comme la vitesse est en m/s

→ l'accélération est en m/s^2

Seconde loi de Newton

Comment se calcule l'accélération ?

- Vitesse de pointe du sprinter Usain Bolt pendant son record du monde du 100 m
  - 12.42 m/s = 44km/h = qu'il attend en environ 7s
  - Après sa vitesse se stabilise

t=0 s
v=0 m/s

t=7 s
v=12.42 m/s

$$a = \frac{v_2 - v_1}{t_2 - t_1} = \frac{12.42 - 0}{7 - 0} = 1.77 m/s^2$$

Chaque seconde, sa vitesse augmente de 1.77m/s donc de 6.3 km/h

Seconde loi de Newton

Comment se calcule l'accélération ?

- Voiture qui accélère le plus fort
  - De 0 à 300km en 13.6s

t=0 s
v=0 m/s

t=13.6 s
v=300 km/h=83.33 m/s

$$a = \frac{v_2 - v_1}{t_2 - t_1} = \frac{83.33 - 0}{13.6 - 0} = 6.127 m/s^2$$

Chaque seconde, sa vitesse augmente de 6.127 m/s donc de 22 km/h

Seconde loi de Newton

Comment se calcule l'accélération ?

- L'accélération entre t_1 et t_2 est la différence de vitesse divisée par la différence de temps

$$a = \frac{v_2 - v_1}{t_2 - t_1}$$

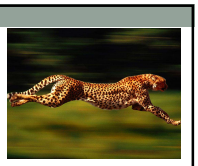
- L'accélération est la dérivée de la vitesse par rapport au temps

$$a(t) = \frac{dv(t)}{dt} \cong \frac{v(t+dt) - v(t)}{dt}$$

Seconde loi de Newton

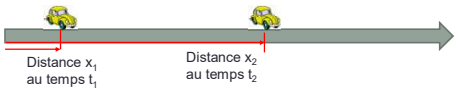
La vitesse c'est quoi ?

Seconde loi de Newton



La vitesse c'est quoi ?

- Distance parcourue durant un intervalle de temps

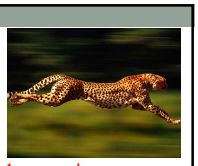


Distance x_1 au temps t_1

Distance x_2 au temps t_2

$$v = \frac{x_2 - x_1}{t_2 - t_1}$$

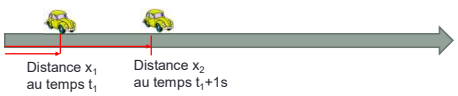
Seconde loi de Newton



La vitesse c'est quoi ?

La vitesse est la variation de position en 1 seconde

→en m/s



Distance x_1 au temps t_1

Distance x_2 au temps t_1+1s

La vitesse est la dérivée de la position par rapport au temps

$$v(t) = \frac{dp(t)}{dt} \cong \frac{p(t+dt) - p(t)}{dt}$$

Seconde loi de Newton

On a donc

- L'accélération est la dérivée de la vitesse par rapport au temps

$$a(t) = \frac{dv(t)}{dt} \cong \frac{v(t+dt) - v(t)}{dt}$$

- La vitesse est la dérivée de la position par rapport au temps

$$v(t) = \frac{dp(t)}{dt} \cong \frac{p(t+dt) - p(t)}{dt}$$

Seconde loi de Newton

On peut donc mettre à jour

- la vitesse en fonction de l'accélération et de la vitesse au pas de temps précédent

$$v(t+dt) = v(t) + a(t).dt$$

- la position en fonction de la vitesse et de la position au pas de temps précédent

$$p(t+dt) = p(t) + v(t).dt$$

Seconde loi de Newton

Comme $F = m.a(t) \Leftrightarrow a(t) = \frac{F}{m}$

- la vitesse en fonction de l'accélération et de la vitesse au pas de temps précédent

$$v(t+dt) = v(t) + \frac{F}{m}.dt$$

- la position en fonction de la vitesse et de la position au pas de temps précédent

$$p(t+dt) = p(t) + v(t).dt$$

Seconde loi de Newton

- Particule = 1 point animé
  - P = position = (X(t), Y(t), Z(t)) = (X_t , Y_t , Z_t) au temps précédent ($X_{t-\Delta t}$, $Y_{t-\Delta t}$, $Z_{t-\Delta t}$) sous entendu temps_suivant = t+Δt
  - V = vitesse = dP/dt

$$V_t = \frac{dP}{dt} = \lim_{\Delta t \to 0} \frac{\Delta P}{\Delta t} \approx \frac{P_t - P_{t-\Delta t}}{\Delta t}$$

Seconde loi de Newton

- Particule = 1 point animé
  - A = accélération = dV/dt

$$A_t = \frac{dV}{dt} = \lim_{\Delta t \to 0} \frac{\Delta V}{\Delta t}$$
$$\approx \frac{V_t - V_{t-\Delta t}}{\Delta t}$$
$$\approx \frac{P_t - P_{t-\Delta t} - (P_{t-\Delta t} - P_{t-2\Delta t})}{\Delta t^2}$$
$$\approx \frac{P_t - 2P_{t-\Delta t} + P_{t-2\Delta t}}{\Delta t^2}$$

Euler explicite

- En connaissant la position au 2 temps précédents,

$$\sum F = F = mA$$
$$A_t \approx \frac{P_t - 2P_{t-\Delta t} + P_{t-2\Delta t}}{\Delta t^2}$$

$$donc \Rightarrow P_t = \Delta t^2 \times \frac{F}{m} + 2P_{t-\Delta t} - P_{t-2\Delta t}$$

Les forces ?

Les forces indiquent comment change l'accélération (F=ma)

- **Sur terre**, poids P

$P = m.G$ avec $G = (0 \quad g)$

avec g = –9.81

Les forces ? + demo du code

Les forces indiquent comment change l'accélération (F=ma)

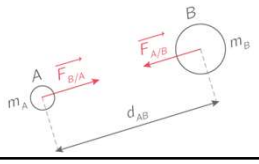
- **Sur terre**, poids P

$$P = m.G \quad \text{avec} \quad G = (0 \quad g)$$

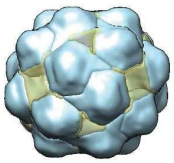
avec $g = -9.81$

- **Les astres.** Deux astres de position A et B et de masse m_A et m_B s'attirent avec une force

$$F_{A/B} = \frac{g.m_A.m_B}{d^2} AB$$



Une particule en 3D



- Une particule à l'écran est en 3D
  - Masse
  - Position
  - Vitesse
- Qu'est-ce que ca change pour les calculs ?
  - Vitesse : (v_x,v_y) en m/s
  - Position : (p_x,p_y) en m
  - Masse : 1 scalaire en kg
  - F : (f_x,f_y) en N=Newton
- Vitesse et position sont des Vecteurs à 2 dimensions
  - Comme des complexes vus au cours 1

Et le code ?

- On a besoin de Vecteur 3D

```
class Vec3
{
public:
        Vec3(float x, float y);
        float x, y;
        float norm();
};

Vec3 operator+(const Vec3& a, const Vec3& b);
Vec3 operator-(const Vec3& a, const Vec3& b);
Vec3 operator*(float a, const Vec3& b);
```

Ceux sont les mêmes fonctions que pour les complexes

Et le code ?

- On a besoin d'une particule

```
class Particle
{
        Vec2 p;         // position
        Vec2 v;         // vitesse
        Vec2 f;         // force
        float m;        // masse
        …
};
```

Tant qu'on y est codons plusieurs particules

```
class World
{
public:
        vector<Particle> part;
        float dt;
        …
};
```

Et le code ?

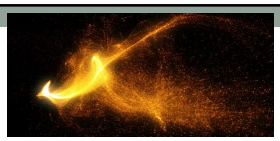
```
void World::updateParticle()          // advect
{
        int i;
        for (i = 0; i < d.part.size(); ++i)
        {
                if (d.part[i].m>0)
                {
                        part[i].p = … VOUS FEREZ EN TD/TP …
                        part[i].v = … VOUS FEREZ EN TD/TP …
                        part[i].f = Vec2(0,0);
                }
        }
```

$$p(t+dt) = p(t) + v(t).dt$$

$$v(t+dt) = v(t) + \frac{F}{m}.dt$$

Et pour dt ?

- dt petit c'est mieux mais allonge le temps de calcul
- dt trop grand peut être instable (cf. démo masses-ressorts)
- En pratique dt=0.01 ou 0.001



ANIMATION PHYSIQUE EN SYNTHESE D'IMAGES

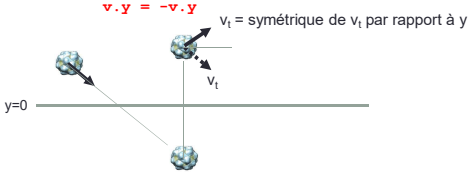
Mécanique du point / système de particules
Une première simulation physique
Finalement pas si compliqué que ca !

- Collisions
- Masse-Ressorts
- Différents schéma d'intégrations

Collisions avec le sol

- Collisions avec 4 murs : horizontaux et verticaux
  - 4 cas simples, par exemple avec sol
  - Remettre la particule au dessus du sol
  - Changer V=vecteur vitesse de manière symétrique

```
si p.y < 0 alors
        p.y = -p.y
        v.y = -v.y
```



Contraintes, collisions

- Exemple : sol en Y=0 y<0 ?

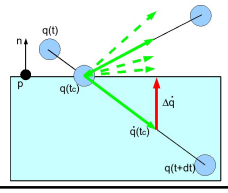
➔ Différentes solutions
- Appliquer la force F_{sol} de réaction du sol proportionnelle à la distance sous le sol
- Autorise la pénétration sous le sol mais la force va corriger le tissus au pas de temps suivant
- OU
  - …

Avec ∑F=ma on a équivalence entre F et déplacement

Collisions avec retour arrière

- Détection d'une collision puis
  - Revenir en arrière au moment de la collision
  - Puis calcul de la mise à jour de V : $\Delta V = -(V.n)n$
  - Applique un coefficient r de rebond : $V \mathrel{+}= (1+r).\Delta V$

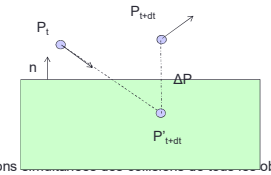
- Problème : assez lourd de revenir en arrière quand les 2 objets sont en mouvements



Sur la figure P=q et V= $\dot{q}$

Collisions

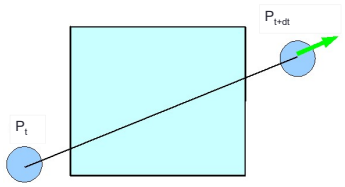
- Détection d'une collision puis
  - Mise à jour de P : ΔP = 2 x vecteur entre P' et le sol (symétrique de P' par rapport au sol)
  - Applique un coefficient r de rebond : $P \mathrel{+}= (1+r)\,\Delta P$
  - Mise à jour de V : symétrique par rapport au sol



+ Résolutions [illegible] objets en même temps

Collisions

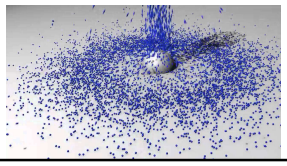
- Cas limite de détection de collision avec pas de temps discret



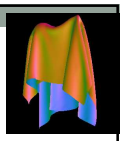
Voir le cours en M2

ANIMATION PHYSIQUE EN SYNTHESE D'IMAGES

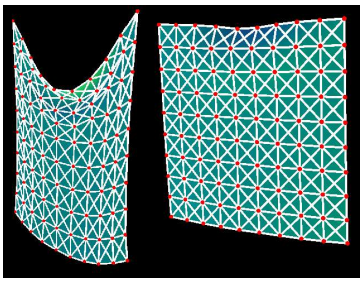
MASSES-RESSORTS



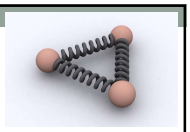
Masses-ressorts



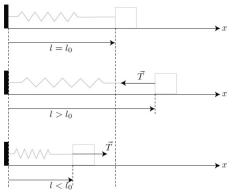
- Maillage de masse-ressort

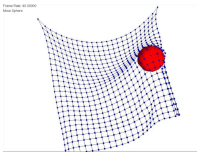


Masses-ressorts



- Loi des ressorts

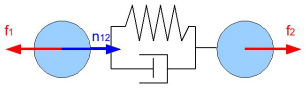




$$F_{ressort} = k.(l - l_{repos})$$

Système masse-ressort

stiffness k , rest length l_0



damping ν

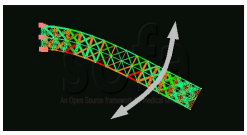
- Force viscoelastique (k=raideur du ressort)
  - Force de rappel : F = -k . (l - l_0)
  - F Viscosité

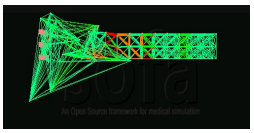
$$f_1 = \left(k \frac{\|\mathbf{q}_2 - \mathbf{q}_1\| - l_0}{l_0} + \nu(\dot{\mathbf{q}}_2 - \dot{\mathbf{q}}_1).\mathbf{n}_{12} \right) \mathbf{n}_{12}$$

$$\mathbf{n}_{12} = \frac{\mathbf{q}_2 - \mathbf{q}_1}{\|\mathbf{q}_2 - \mathbf{q}_1\|}$$

Système masse-ressort

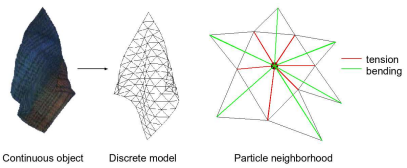
- Raideur du ressort : k
  - F = -k . (l - l_0)
- K petit (objet « mou »)
- K élevé (objet rigide) : stabilité demande dt petit sinon le système explose





Système masse-ressort

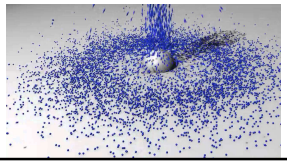
- Sommet = particules, arêtes = ressort



- Maillage avec des configurations diiférentes

ANIMATION PHYSIQUE EN SYNTHESE D'IMAGES

Pour aller plus loin …



Position Based Dynamics

Développeur de NVIDIA Physics
http://matthias-mueller-fischer.ch/

- Mise à jour de V :

$$mA_t = F$$
$$m\frac{dV}{dt} = F$$
$$m\frac{V_t - V_{t-\Delta t}}{\Delta t} = F$$
$$V_t = V_{t-\Delta t} + \Delta t \frac{F}{m}$$

- Puis mise à jour de P :

$$\frac{dP}{dt} = V_t$$
$$\frac{P_t - P_{t-\Delta t}}{\Delta t} = V_t$$
$$P_t = P_{t-\Delta t} + \Delta t \times V_t$$

Variant : leap-frog, Stoermer-Verlet

Position Based Dynamics

http://matthias-mueller-fischer.ch/

m_i	mass
$\mathbf{x}_i$	position
$\mathbf{v}_i$	velocity

Algorithm 1 Position-based dynamics

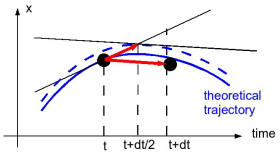
1: **for all** vertices i **do**
2: initialize $\mathbf{x}_i = \mathbf{x}_i^0$, $\mathbf{v}_i = \mathbf{v}_i^0$, $w_i = 1/m_i$
3: **end for**
4: **loop**
5: **for all** vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
6: **for all** vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
7: **for all** vertices i **do** genCollConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
8: **loop** solverIteration **times**
9: projectConstraints($C_1, \ldots, C_{M+M_{Coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
10: **end loop**
11: **for all** vertices i **do**
12: $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
13: $\mathbf{x}_i \leftarrow \mathbf{p}_i$
14: **end for**
15: velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
16: **end loop**

Schéma d'intégration

- Runge-Kutta
  - Calcul un pas de temps dt/2 fictif avec Euler
  - Calcul la dérivée
  - Utilise cette dérivée pour un pas de temps

➔erreur proportionnelle à dt^2 au lieu de dt

- Plus sta



D'autres méthodes d'intégrations existent

Schéma d'intégration

- Verlet

$$P_{t+\Delta t} = 2P_t - P_{t-\Delta t} + \Delta t^2 \times A_t$$

D'autres méthodes d'intégrations existent
Voir le cours en M2

Solide rigide

Voir le cours en M2

- Idem point
  - Physique du point : ∑F=ma
- + orientation (rotation)
  - ∑L=I.ω
  - Somme des moments des forces extérieurs = moment d'inertie du solide * Accélération angulaire

Remarque : Un mouvement au sens le plus général peut être considéré à chaque instant comme la superposition d'une translation et d'une rotation autour d'un axe.(par exemple le mouvement d'une bille sur un plan incliné) Pour résoudre les équations du mouvement, les 2 équations si dessus sont nécessaires

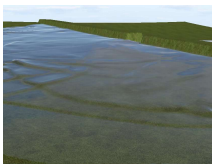
ANIMATION PHYSIQUE D'UNE SURFACE D'EAU

Equation de Saint-Venant ou Shallow Water Equation
TP : OpenGL

Equation de Saint-Venant

Equations de Saint-Venant ou Shallow Water equations

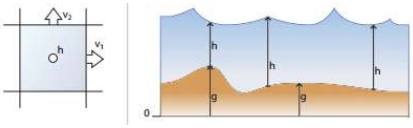
- Dérivée des eq. de Navier-Stokes
- Real Time Physics, Siggraph 2008 class.
- décrit les écoulements naturels : surface libre et eau peu profonde
- Grille 2D Approche Eulerienne (!= particule, approche Lagrangienne)

Equation de Saint-Venant


Champ de vecteur v

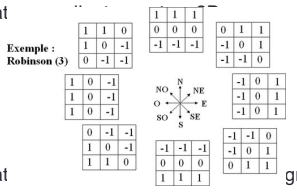
- Grille 2D
- h hauteur du fluide
- g hauteur du sol
- n hauteur du fluide sur le sol : n = h−g.
- v(v_1,v_2) vitesse du fluide dans le plan horizontal



Préambule

- Opérat

$$\vec{\nabla} = \left(\frac{\partial}{\partial x} \quad \frac{\partial}{\partial y}\right)$$



- Operat gradient

$$\nabla \cdot \boldsymbol{A} \equiv \operatorname{div} \boldsymbol{A} = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y}$$

Equation de Saint-Venant : intuitif

- Equations de Saint-Venant (Shallow Water)

$$\frac{\partial \eta}{\partial t} + (\nabla \eta)\mathbf{v} = -\eta \nabla \cdot \mathbf{v} \quad (1) \text{ avec } \nabla \cdot \boldsymbol{A} \equiv \operatorname{div} \boldsymbol{A} = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y}$$

$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla \mathbf{v})\mathbf{v} = a_n \nabla h\,, \quad (2)$$

- a_n : accélération verticale du fluide (gravité)=9.81
- (1) s'occupe de la variation de quantité d'eau
- (2) s'occupe de la variation de la vitesse

- Partie gauche : calculer par advection

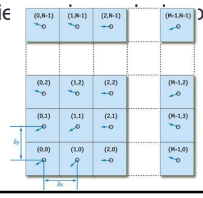
L'advection correspond au transport d'une quantité (scalaire ou vectorielle) par un champ vectoriel.

Equation de Saint-Venant : advection

- Equation de Saint-Venant (Shallow Water)

$$\begin{aligned} \partial\eta/\partial t + (\nabla\eta)\mathbf{v} &= -\eta\nabla\cdot\mathbf{v} && (1)\\ \partial v_1/\partial t + (\nabla v_1)\mathbf{v} &= a_n \nabla h && (2)\\ \partial v_2/\partial t + (\nabla v_2)\mathbf{v} &= a_n \nabla h\,. && (2') \end{aligned}$$

- Partie par advection



L'advection correspond au transport d'une quantité (scalaire ou vectorielle) par un champ vectoriel.

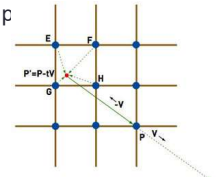
Equation de Saint-Venant : advection

- Equation de Saint-Venant (Shallow Water)

$$\begin{aligned} \partial\eta/\partial t + (\nabla\eta)\mathbf{v} &= -\eta\nabla\cdot\mathbf{v} && (1)\\ \partial v_1/\partial t + (\nabla v_1)\mathbf{v} &= a_n \nabla h && (2)\\ \partial v_2/\partial t + (\nabla v_2)\mathbf{v} &= a_n \nabla h\,. && (2') \end{aligned}$$

- p

```
Advect(s, v)
(1) for j = 1 to n2 − 1
(2)   for i = 1 to n1 − 1
(3)     x = (i · Δx, j · Δx)
(4)     x' = x − Δt · v(x)
(5)     s'(i, j) = interpolate(s, x')
(6)   endfor
(7) endfor
(8) return(s')
```



Equation de Saint-Venant : intuitif

- Equations de Saint-Venant (Shallow Water)

$$\frac{\partial \eta}{\partial t} + (\nabla \eta)\mathbf{v} = -\eta \nabla \cdot \mathbf{v} \qquad (1)$$
$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla \mathbf{v})\mathbf{v} = a_n \nabla h\,, \qquad (2)$$

- (1) correspond à la variation de quantité d'eau
  - Variation de la quantité d'eau = dn/dt = -n∇.v = en fonction du champ de vitesse, on fait le bilan des arrivées des départs en eau
  - Si plus d'eau part, que d'eau arrive (∇.v>0), ca descend.
  - Et inversement.

Eq. Saint-Venant : variation quantité d'eau

- Equation de Saint-Venant (Shallow Water)

$$\partial \eta/\partial t + (\nabla \eta)\mathbf{v} = -\eta \nabla \cdot \mathbf{v} \quad (1)$$
$$\partial v_1/\partial t + (\nabla v_1)\mathbf{v} = a_n \nabla h \quad (2)$$
$$\partial v_2/\partial t + (\nabla v_2)\mathbf{v} = a_n \nabla h\,. \quad (2')$$

- Partie gauche : calculer par advection

Update-height$(\eta, \mathbf{v})$
(1) **for** $j = 1$ **to** $n_2 - 1$
(2) **for** $i = 1$ **to** $n_1 - 1$
(3) $\eta(i,j) -= \eta(i,j) \cdot \left(\frac{(v_1(i+1,j) - v_1(i,j))}{\Delta x} + \frac{(v_2(i,j+1) - v_2(i,j))}{\Delta x} \right) \Delta t$
(5) **endfor**
(6) **endfor**
(7) **return**(η')

Equation de Saint-Venant : intuitif

- Equations de Saint-Venant (Shallow Water)

$$\frac{\partial \eta}{\partial t} + (\nabla \eta)\mathbf{v} = -\eta \nabla \cdot \mathbf{v} \qquad (1)$$
$$\frac{\partial \mathbf{v}}{\partial t} + (\nabla \mathbf{v})\mathbf{v} = a_n \nabla h\,, \qquad (2)$$

- (2) correspond à : m.a = m.dv/dt = ∑F
  - Variation de vitesse
  - Ici F=a∇h = gravité dans la direction « vers moins d'eau »
  - le fluide subit une force dans la direction : de plus d'eau vers moins d'eau

Eq. de Saint-Venant : variation de v

- Equation de Saint-Venant (Shallow Water)

$$\partial \eta/\partial t + (\nabla \eta)\mathbf{v} = -\eta \nabla \cdot \mathbf{v} \quad (1)$$
$$\partial v_1/\partial t + (\nabla v_1)\mathbf{v} = a_n \nabla h \quad (2)$$
$$\partial v_2/\partial t + (\nabla v_2)\mathbf{v} = a_n \nabla h\,. \quad (2')$$

- (2)(2') donnent variation de v donc
  - v1 = v1 + $a_n \nabla h_1$ avec ∇h_1 = dh/dx
  - v2 = v2 + $a_n \nabla h_2$ avec ∇h_2 = dh/dy

Update-velocities(h, v_1, v_2, a)
(1) **for** $j = 1$ **to** $n_2 - 1$
(2) **for** $i = 2$ **to** $n_1 - 1$
(3) $v_1(i,j) += a(\frac{h(i-1,j) - h(i,j)}{\Delta x})\Delta t$
(5) **endfor**
(6) **endfor**
(7) **for** $j = 2$ **to** $n_2 - 1$
(8) **for** $i = 1$ **to** $n_1 - 1$
(9) $v_2(i,j) += a(\frac{h(i,j-1) - h(i,j)}{\Delta x})\Delta t$
(10) **endfor**
(11) **endfor**

Saint-Venant : intégration

- Shallow-water-step(h,v,g)
  - (1) n = Advect(n,v)
  - (2) v1 = Advect(v1,v)
  - (3) v2 = Advect(v2,v)
  - (4) n' = Update-height(n,v)
  - (5) h = n'+g
  - (6) Update-velocities(h,v1,v2)
  - (7) n=n'

➔ TP en C/C++

SHALLOW WATER OPENCL

OpenCL (wikipedia)

- **OpenCL** (**Open** **C**omputing **L**anguage) est la combinaison d'une API et d'un langage de programmation dérivé du C, proposé comme un standard ouvert par le Khronos Group.
- OpenCL est conçu pour programmer des systèmes parallèles hétérogènes comprenant par exemple à la fois un CPU multi-cœur et un GPU.
- OpenCL propose donc un modèle de programmation se situant à l'intersection naissante entre le monde des CPU et des GPU, les premiers étant de plus en plus parallèles, les seconds étant de plus en plus programmables.

OpenCL

- OpenCL can accelerate code by a factor 10 or more
- OpenCL is an open standard
- OpenCL can help save power
- OpenCL can save you hardware cost
- OpenCL adoption is ramping up rapidly
- OpenCL may be used as the basis for generating custom hardware
- The OpenCL C99 language is based on C
- OpenCL can be used from a variety of host languages
- It is easy to start with OpenCL
- OpenCL is platform independent

http://www.amdahlsoftware.com/ten-reasons-why-we-love-opencl-and-why-you-might-too/

OpenGL / OpenCL : le TP

- Le code de départ
  - Carte de hauteur dans une texture 2D (GL)
  - Affichage GL avec un vertex shader
    - Vertex.glsl
  - kernel OpenCL modifie la texture
    - Texture 2D = vu comme une 'image2d'
    - CLWater.cl
    - Kernel = fonction appelée sur chaque case du tableau

OpenCL : un kernel

```
__kernel void shallowWaterInit( const int dimD,
                                __global __write_only image2d_t DD0 )
{
        const int x = get_global_id(0);
        const int y = get_global_id(1);

        if (x>=dimD) return; if (y>=dimD) return;
        const sampler_t sampler =
CLK_NORMALIZED_COORDS_FALSE | CLK_ADDRESS_CLAMP |
CLK_FILTER_NEAREST;

        float4 pixel={0,0,0,0};
        write_imagef( DD0, (int2)(x,y), pixel);
}
```

OpenCL : l'appel

```
clSetKernelArg( m_kernelShallowWater, 0, sizeof(Dim), (void*)&Dim);
clSetKernelArg( m_kernelShallowWater, 1, sizeof(D0), (void*)&D0);

const size_t localWorkSize[] = { LocalWorkSize, LocalWorkSize };
const size_t globalWorkSize[] = { Dim, Dim };

cl_uint workDim = 2;
clEnqueueNDRangeKernel ( m_queue, m_kernelShallowWaterInit,
                         workDim, NULL,
                         globalWorkSize, localWorkSize,
                         0, NULL, NULL);
```

TP ANIMATION PHYSIQUE DE PERSONNAGE AVEC BULLETPHYSICS

TP

- Combiner MoCap et Ragdoll
  - Utiliser Bullet pour la physique
- Regardez le fichier Ragdoll.h/.cpp
  - Ce ragdoll doit avoir la même configuration que le squelette de la MoCap
  - Pour l'instant, c'est un squelette avec 2 membres (bras, avant-bras) qui sont entrés en dur.

TP : Lib BulletPhysics

- Le monde physique gérés par la lib
  - btDynamicsWorld* m_dynamicsWorld;

➔Les objets physiques doivent y être ajoutés

- Dans le TP, il y a une class CAPhysics qui s'occupe du monde physique de Bullet avec
  - computePhysics() : calcul la physique depuis le dernier appel
  - renderPhysics() : affiche les objets physiques, appuyer sur 'P' dans le viewer pour les afficher
  - createRigidBody(…) : ajoute un objet rigide dans le monde physique et renvoie un pointeur dessus

TP : Lib BulletPhysics

- Ragdoll = ensemble de
  - vector<btRigidBody*> m_bodies;
    - Solides rigides
  - vector<btCollisionShape*> m_shapes;
    - Les formes pour les collisions (optionnel)
  - vector<btTypedConstraint* > m_jointsConstraint;
    - Les articulations

TP : Lib BulletPhysics

- Bullet gère les transformations
  - btTransform transform;
    - Rotation + translation
  - Construit avec un quaternion + un vecteur
    - btTransform t(q, v)
  - Construit avec une matrice 3x3 + vecteur*

  …
  - Toutes les fonctions dont vous avez besoin sont disponibles …

TP : Lib BulletPhysics

- Créer un corps solide btRigidBody
  - Utiliser la fonction de CAPhysics

```
btRigidBody* CAPhysics::createRigidBody(
        float mass,
        const btTransform& startTransform,
        btCollisionShape* shapeForCollision)
```

- Créer une forme pour les collisions

```
new btCapsuleShape( rayon, hauteur );
```

TP : Lib BulletPhysics

- Créer une articulation

coneC = new btConeTwistConstraint(A, B, localA, localB);

  - A et B sont des btRigidBody
  - localA et localB sont des btTransform relatif à au repère local du btRigidBody

- Articulation avec des contraintes en forme de Cone

- Il existe d'autres types d'articulation
  - btHingeConstraint : mouvement dans le plan type doude
  - …

- Ne pas oublier d'ajouter les articulations au monde physique
  - m_physics.getDynamicsWorld()->addConstraint(coneC, true);
  - true pour ne pas calculer de colision entre 2 btRigidBody relié par l'articulation

TP : Lib BulletPhysics

- Regardez les 2 exemples dans CARagdoll.h