Institut Universitaire de Technologie
Aix*Marseille Université

Imagerie Numérique
M4102Cin - Synthèse d'images 2

4. Animation

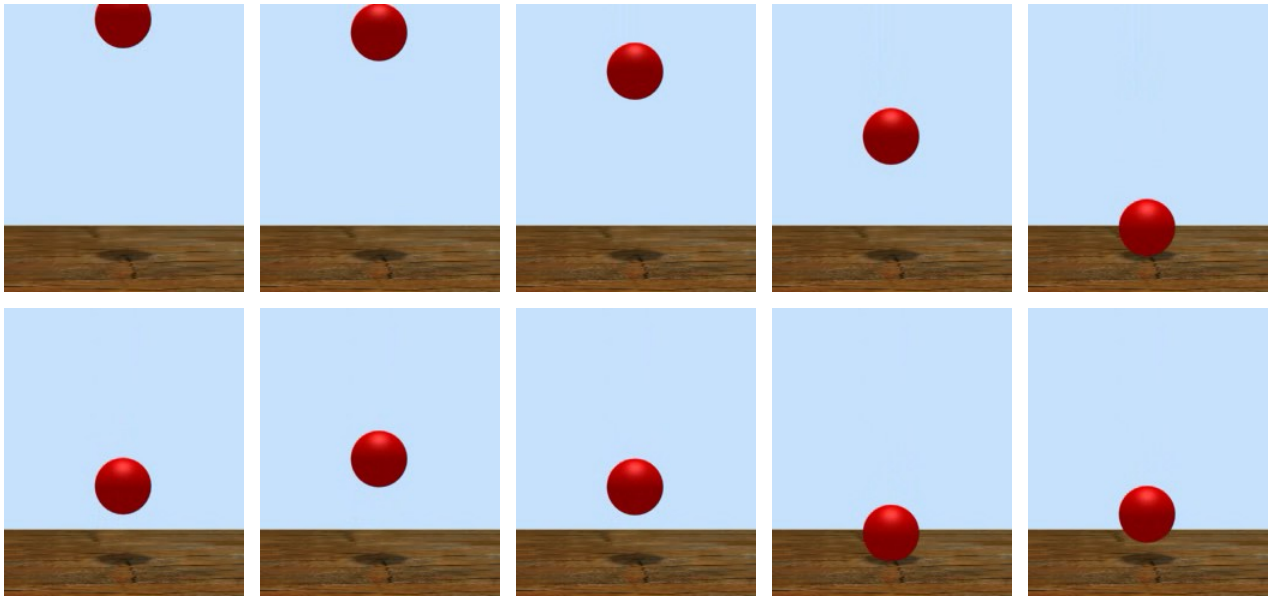
DUT Informatique 2019-2020

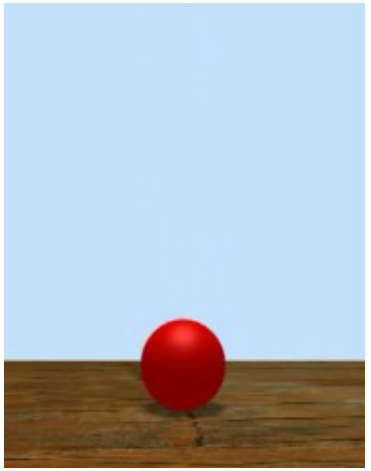
Sébastien THON

IUT d'Aix-Marseille Université, site d'Arles
Département Informatique

Introduction

Animation = succession d'images fixes, à une fréquence suffisante pour tromper l'œil (cinéma : 24 images/sec).
(FPS : « *Frames Per Second* »)





→ Pour produire une animation : modifier la position d'objets au cours du temps.

En synthèse d'images, 2 types d'animation :

- Animation 3D temps réel

Ex: simulateur, jeu, …

Produit avec une librairie graphique (OpenGL, DirectX, …).
Il faut calculer suffisamment d'images par seconde. Le nombre de FPS dépend de la machine.

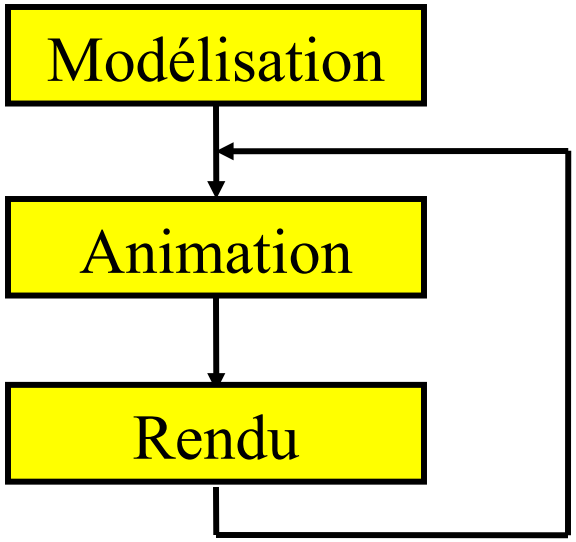
- Animation 3D précalculée

Ex: film d'animation, effets spéciaux, …

Produit avec un logiciel de synthèse d'images (3D Studio Max, Maya, Lightwave, …).
On peut passer plusieurs minutes ou heures pour calculer une seule image.

3 étapes :

- **Modélisation**
  Représenter les objets par des modèles 3D.
- **Animation**
  Associer un mouvement aux objets.
- **Rendu**
  Créer les images qui composent l'animation.



Dans le cas d'une animation temps-réel, il faut que cette boucle soit la plus rapide possible (effectuée au moins 20 fois par seconde)

Il faut que le rythme de rafraîchissement de l'image soit le plus rapide possible :
- pour donner une impression de fluidité (la différence entre une image et la suivante sera moins perceptible)
- pour être réactif aux commandes de l'utilisateur (jeu vidéo, simulateur...)

On parle de vitesse en nombre d'images par seconde (FPS : *Frames Per Second*) ou en fréquence de rafraîchissement en Hertz.

Exemples :

24 img/sec
24 FPS
24 Hz

Problème :

Si on déplace un objet d'une même distance à chaque image : il se déplacera plus vite sur une machine puissante (ex: 60 img/s) que sur une machine moins puissante (ex: 30 img/s).

2 solutions :

1) Mesurer le temps dt entre deux images et utiliser cette mesure pour calculer la position des objets étant donné leur vitesse (**glutGet(GLUT_ELAPSED_TIME)**)

2) Fixer la vitesse de rafraichissement au moyen d'un timer (**glutTimerFunc**)

1) Mesurer le temps dt

```
int old_chrono = 0;

GLvoid callback_display()
{
	int chrono = glutGet(GLUT_ELAPSED_TIME);
	float dt = (chrono – old_chrono) / 1000.0f;
	old_chrono = chrono;

	// On affiche la scène
	// ...
	glutSwapBuffers();
}

GLvoid callback_idle()
{
	glutPostRedisplay();
}

int main(int argc, char *argv[])
{	...
	glutDisplayFunc(&callback_display);	// Call-back pour l'affichage.
	glutIdleFunc(&callback_idle);	// Call-back de processeur disponible.
	glutMainLoop();	// Boucle principale de Glut.
}
```

Problème :

On appelle **callback_display()** dès que le processeur est libre

→ Utilise inutilement le processeur.

2) Fixer la vitesse de rafraichissement

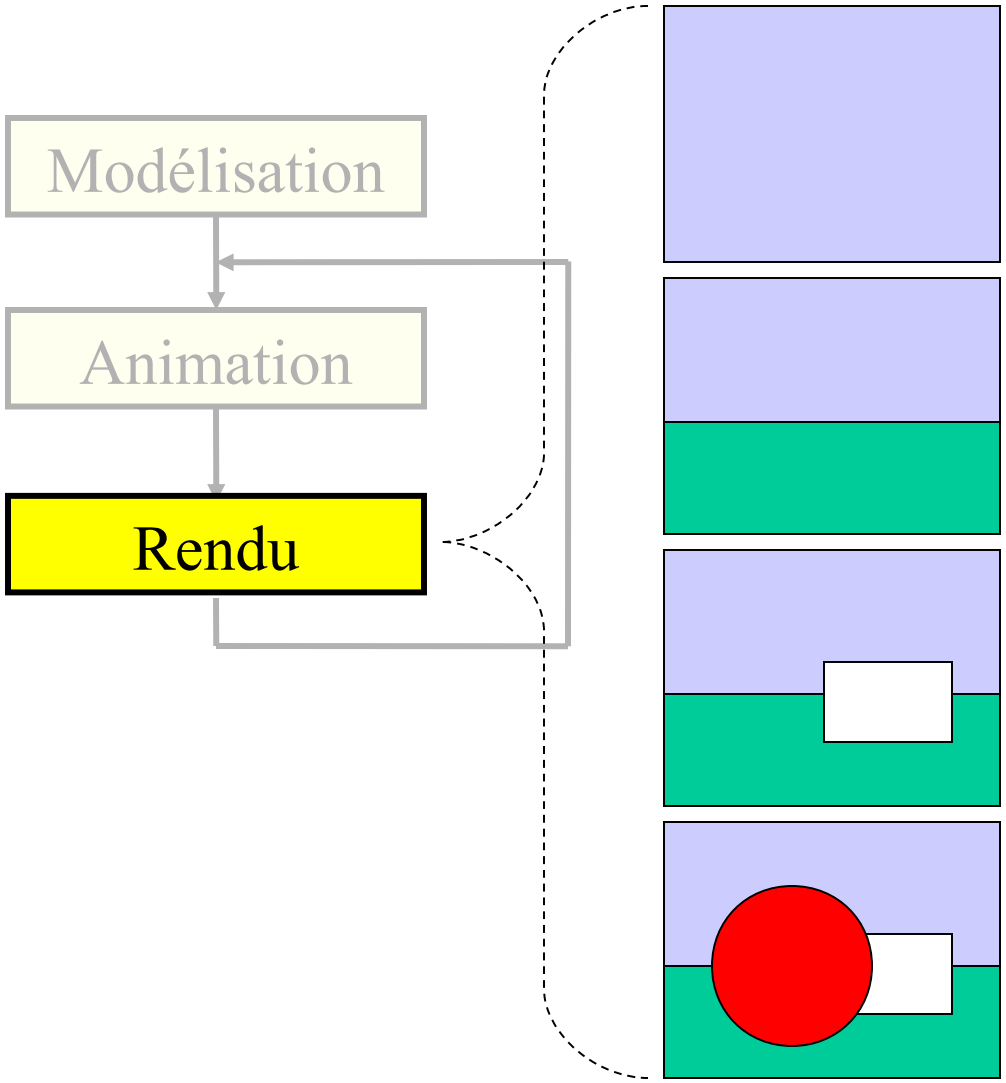
```
GLvoid callback_display()
{
	// On affiche la scène
	// ...

	glutSwapBuffers();
}

void timer(int v)
{
	glutPostRedisplay();
	glutTimerFunc(20, timer, 0);	// Le timer ne fonctionne qu'une fois,
					// il faut le relancer
}

int main(int argc, char* argv[])
{
	...
	glutDisplayFunc(display);
	glutTimerFunc(20, timer, 0);	// Appelle timer() tous les 20ms (→ 50 img/s)
					// le dernier paramètre est quelconque, passé à timer()
	glutMainLoop();
}
```

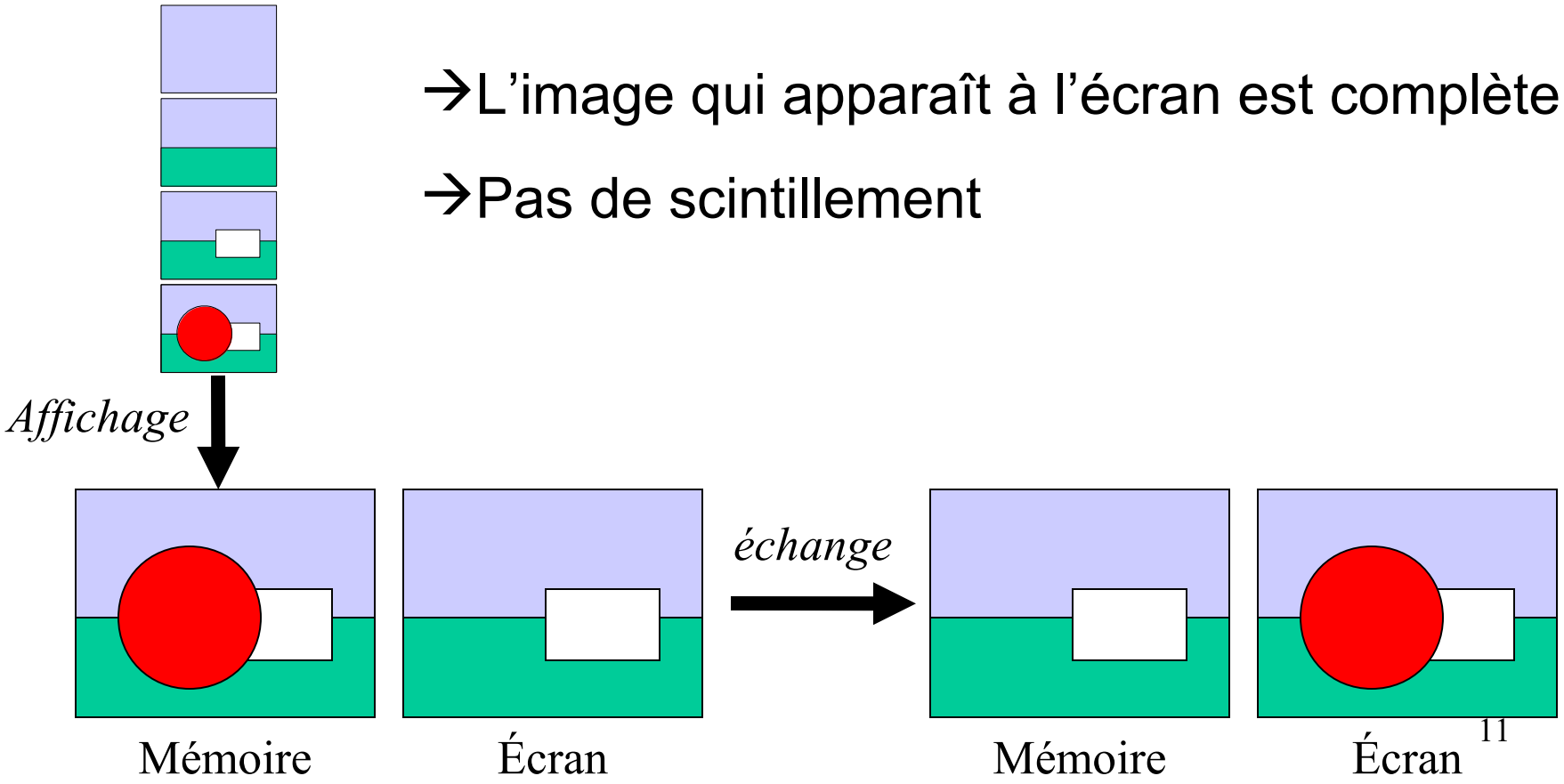
Animation temps réel



Si lors de l'étape de rendu on efface l'écran et on affiche directement les différents éléments à l'écran, alors on observe un effet de scintillement.

<u>Solution :</u> affichage en « *double buffer* »

On utilise deux buffers d'image. On affiche la scène dans un buffer en mémoire alors que l'autre buffer est affiché. Lorsque l'image est construite, on permute les deux buffers.



→L'image qui apparaît à l'écran est complète

→Pas de scintillement

affichage en « *double buffer* » avec OpenGL :

```
// Lors de l'initialisation de l'affichage Glut,on requiert un affichage
// en double buffer.
glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );


// Méthode appelée par Glut pour l'affichage
GLvoid callback_display()
{
    // On efface l'image du buffer en mémoire
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    // On affiche la scène dans le buffer en mémoire
    affiche_scene();

    // Permutation des deux buffers. Le buffer en mémoire devient visible.
    glutSwapBuffers();
}
```

Quelques techniques d'animation :

- Transformations
- Animations paramétrées
- Interpolation
- Systèmes de particules
- Animation de personnages, de foules
- Animation physique
- Détection de collision

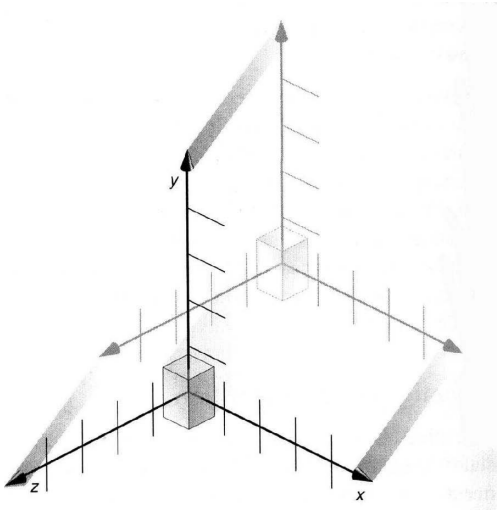
1. Transformations

Opérations sur les sommets d'un objet 3D permettant de déplacer cet objet :

- Translation (*glTranslatef*)
- Rotation (*glRotatef*)
- Mise à l'échelle (*glScalef*)

1.1 Translation

Génére une matrice M de translation puis multiplie la matrice active de modélisation.



Soit un sommet $v = (x,y,z,1)$ et un vecteur de translation $(tx,ty,tz,1)$, les nouvelles coordonnées $v' = (x',y',z',1)$ sont :

$$x' = x + tx$$

$$y' = y + ty$$

$$z' = z + tz$$

Fonction OpenGL de translation :

```
glTranslatef( GLfloat tx, GLfloat ty, GLfloat tz );
```

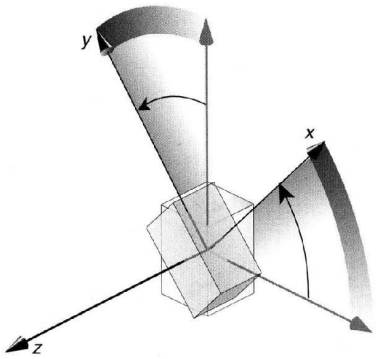
Elle génère une matrice M qui déplace l'objet selon le vecteur de translation (tx,ty,tz), puis multiplie la matrice active avec M.

$$v' = M.v$$

$$M = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.2 Rotation

Génére une matrice M de rotation puis multiplie cette matrice avec la matrice active de modélisation.



`glRotatef(GLfloat angle, rx, ry, rz);`

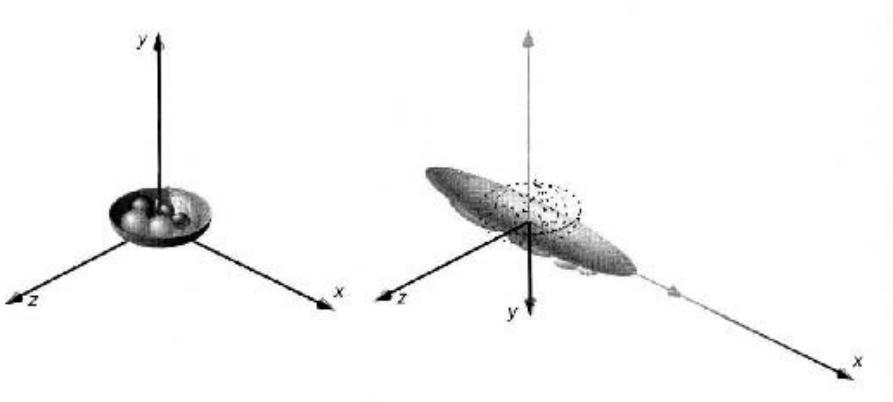
Applique sur un objet une rotation d'angle *angle* sur les axes x, y, z.

Exemple : rotation de 45 degrés sur l'axe des z :

`glRotatef(45,0,0,1);`

1.3 Mise à l'échelle

Génére une matrice M de mise à l'échelle puis multiplie cette matrice avec la matrice active de modélisation.



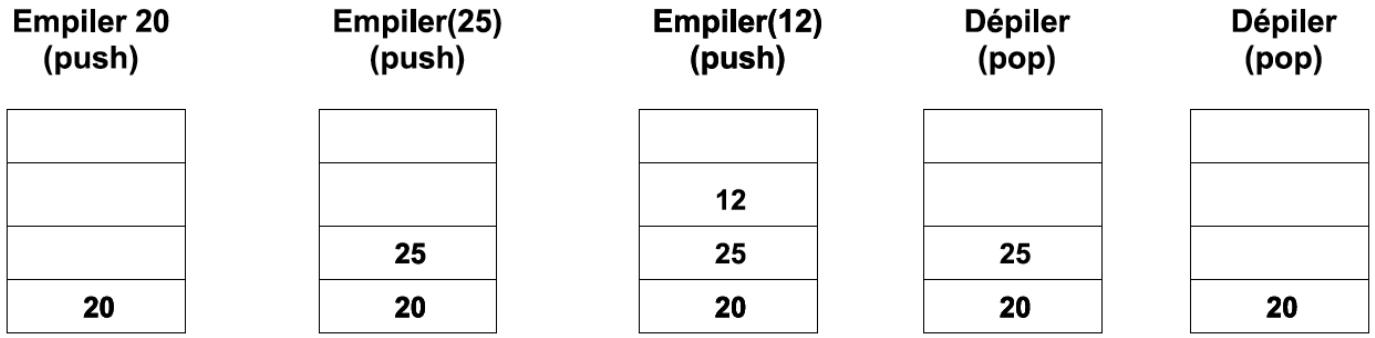
- **`glScalef(GLfloat sx, GLfloat sy, GLfloat sz);`**

Elargit (si $sx, sy, sz > 1$),

rétrécit (si $sx, sy, sz < 1$),

réfléchit (si $sx, sy, sz = -1$) un objet sur les axes x, y, z.

La matrice active peut être sauvegardée pour être restaurée plus tard grâce à une pile de matrices (de modélisation/visualisation ou de projection).



Fonctions :

glMatrixMode : choisit la matrice active (de modélisation/visualisation avec *GL_MODELVIEW*, ou de projection avec *GL_PROJECTION*).

glLoadIdentity : initialise la matrice active.

glPushMatrix : empile la matrice active courante.

glPopMatrix : dépile et écrase la matrice active.

1.4 Transformation de visualisation

Transformation qui consiste à définir la position et l'orientation de la caméra :

```
gluLookAt( ex,ey,ez, rx,ry,rz, ox,oy,oz);
```

ex,ey,ez : position de la caméra

rx,ry,rz : point visé par la caméra

ox,oy,oz : orientation de la caméra vers le haut

gluLookAt génère une matrice M appliquant un positionnement de la caméra, puis multiplie la matrice active avec M.

On peut réaliser une animation en déplaçant au cours du temps la position de la caméra, et/ou le point qu'elle vise, et/ou son orientation.

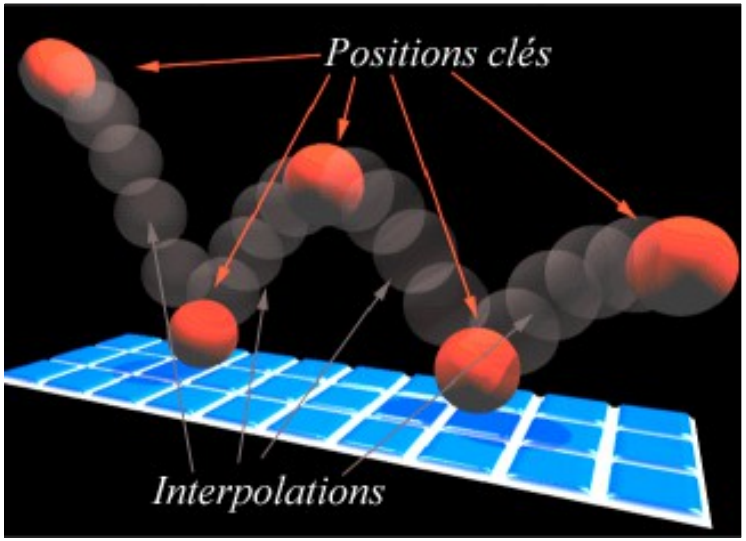
→ c'est la caméra qui bouge

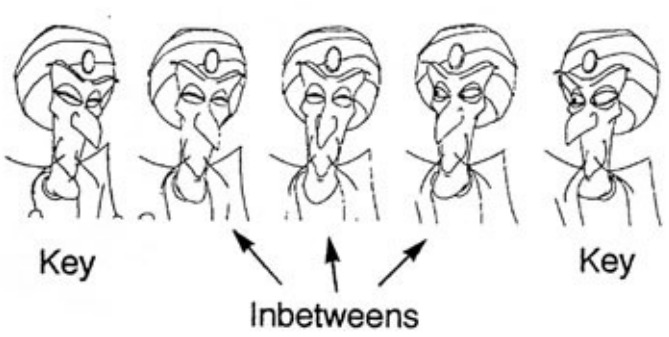
→ on peut aussi pendant ce temps déplacer les objets de la scène

2. Interpolation de positions clés

Permet à l'utilisateur de ne pas avoir à définir toutes les étapes de l'animation :

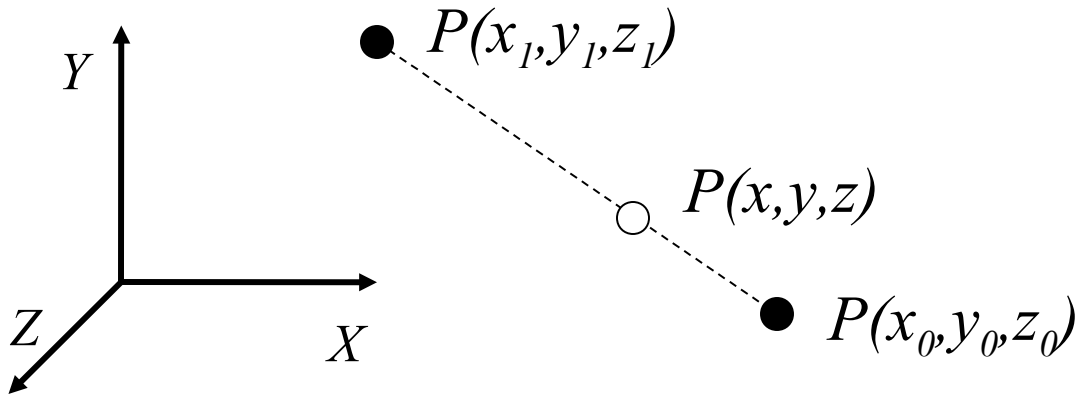
- L'utilisateur définit des **positions clés** (« *key frame* »)
- L'ordinateur se charge de calculer les positions intermédiaires par **interpolation** des positions clés (équivalent du *in-betweening* dans l'animation traditionnelle).





2.1 Interpolation linéaire

Soit un point $P(x,y,z)$ en mouvement. On veut l'amener de (x_0,y_0,z_0) au temps t_0 au point (x_1,y_1,z_1) au temps t_1. On peut utiliser une interpolation linéaire :



Si on considère que $t_0=0$ et $t_1=1$, alors :

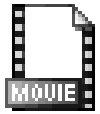
$$x(t) = (1-t).x_0 + t.x_1 \quad \text{avec } t \in [0,1]$$
$$= x_0 + t.(x_1-x_0)$$

Autres interpolations possibles : cosine, cubique, hermite, etc.

2.2 Exemples d'application

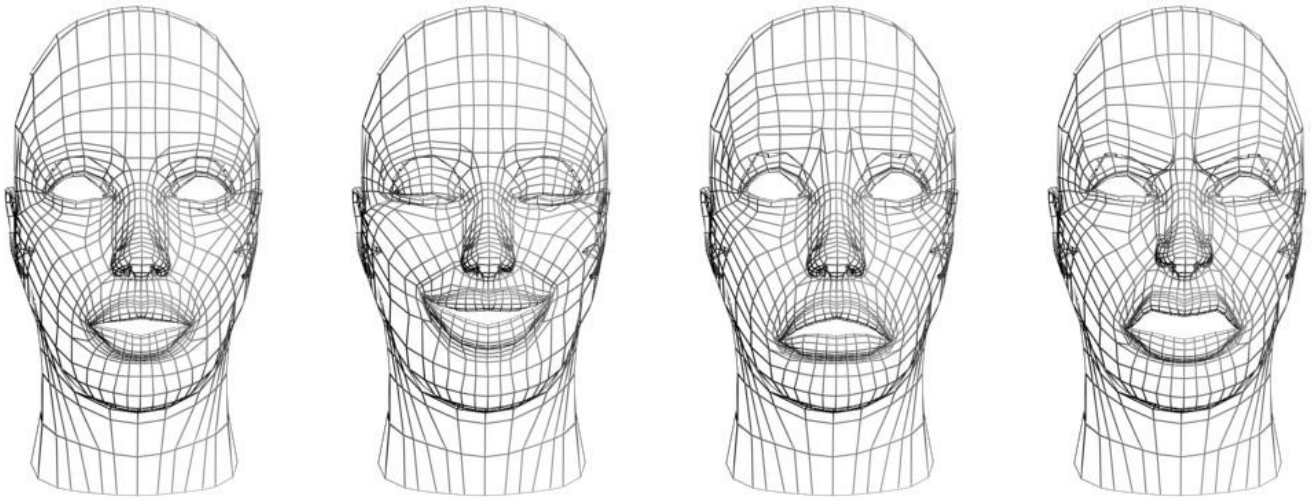
Animation de personnages

Technique ancienne utilisée dans les jeux vidéo (Quake2, Quake3, etc.)



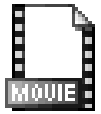
Morph targets

Animation de visages 3D par interpolation entre des **expressions clés** (*Morph targets*).



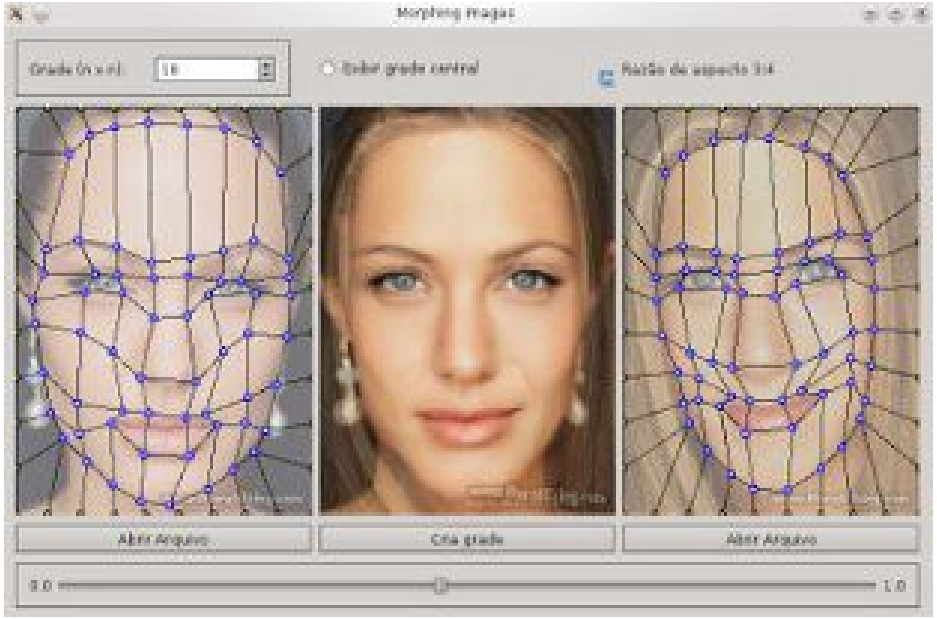
Au repos *Joie* *Peur* *Colère*

Technique aussi appelée « shape targets », « per vertex animation », « blend shapes » ou « shape interpolation ».



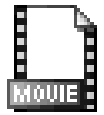
Morphing

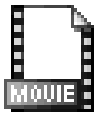
Transformation progressive d'une image 2D (par exemple un visage) ou d'un objet 3D en un autre.



Applications :

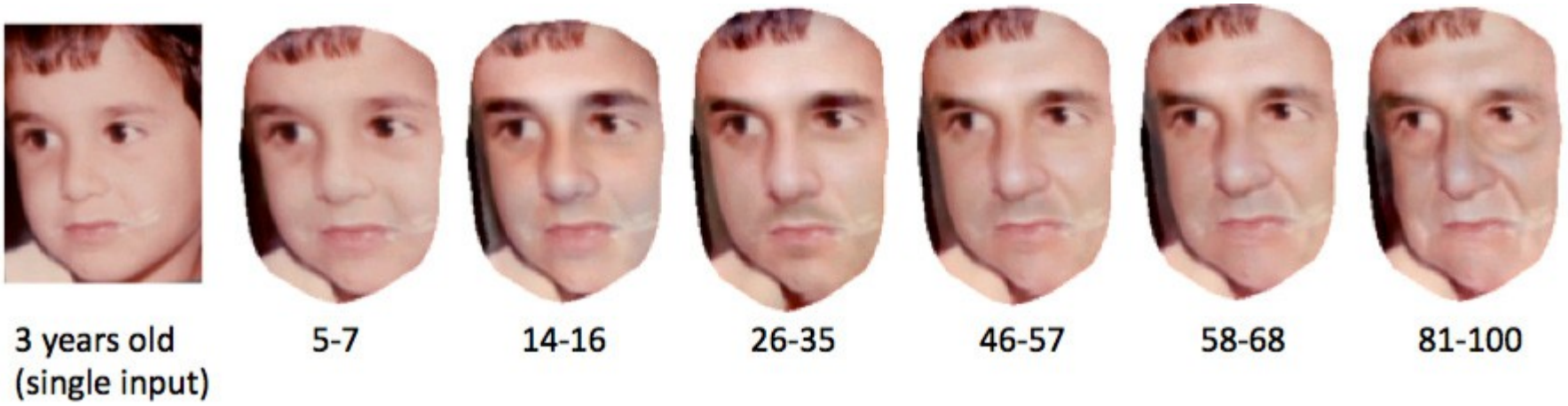
- Cinéma (Terminator 2, Underworld, …)
- Publicité
- Clips vidéo (« Black or white » de Michael Jackson, …)
- Animation entre des positions clés (« *in-betweening* »)



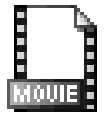


– ***Vieillissement de visages (« computer age progression »)***

La police utilise des logiciels de morphing permettant de calculer l'apparence du visage d'un enfant disparu, plusieurs années plus tard.



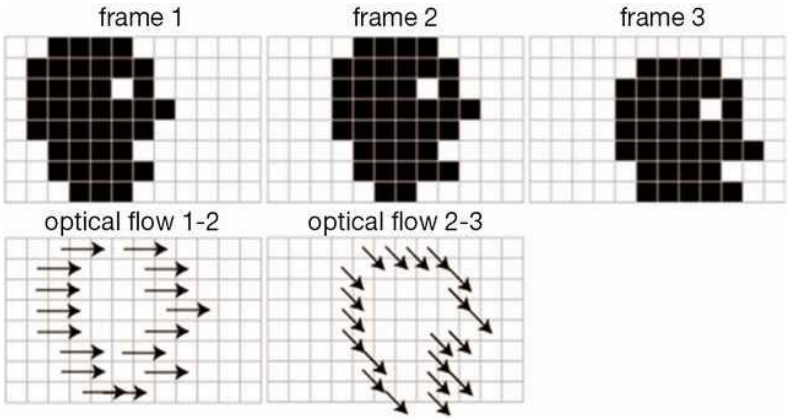
Illumination-Aware Age Progression
Ira Kemelmacher-Shlizerman, Supasorn Suwajanakorn, Steven M. Seitz
CVPR 2014
http://grail.cs.washington.edu/aging/

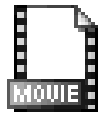


– ***Interpolation entre images successives d'une animation***

Des effets de ralentis ou d'accélération peuvent être obtenus en calculant de nouvelles images par interpolation entre des images successives d'une animation.

Pour cela, on analyse le **flux optique**, c'est-à-dire le mouvement des pixels dans une séquence d'images, lorsqu'ils se déplacent d'une image à la suivante.

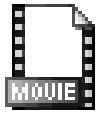


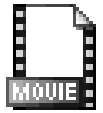


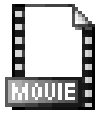
– ***Interpolation entre plusieurs photos***

Les effets de ralentis (appelés « *Bullet time* ») du film *Matrix* ont été obtenus au moyen d'un ensemble de photos prises de manière quasi-simultanée par 120 appareils photo.

Pour rallonger la séquence (passer de 5 sec. à 10 sec.), de nouvelles images ont été calculées par interpolation entre les images prises par les appareils photo.





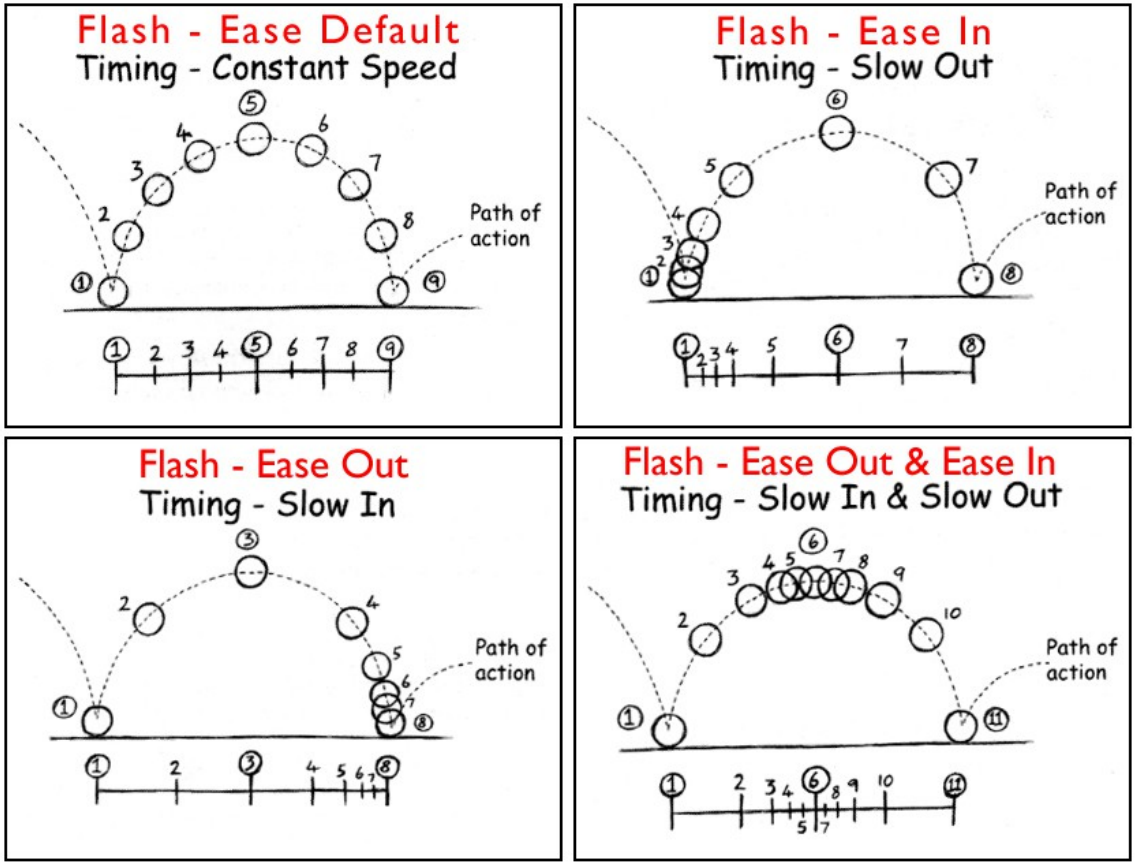


2.3 Vitesse

La vitesse peut être modifiée au cours de l'interpolation, pour simuler une accélération, un amortissement, etc.

Ease in : débute lentement et prend de la vitesse

Ease out : débute rapidement puis décélère.



3. Animation procédurale

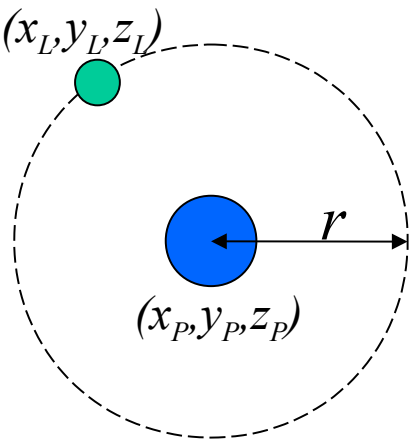
La position des objets 3D en mouvement peut être donnée par une expression mathématique quelconque dépendante du temps :

$$x = f_1(t)$$
$$y = f_2(t)$$
$$z = f_3(t)$$

→Courbes **paramétriques** (paramétrées par le temps t)

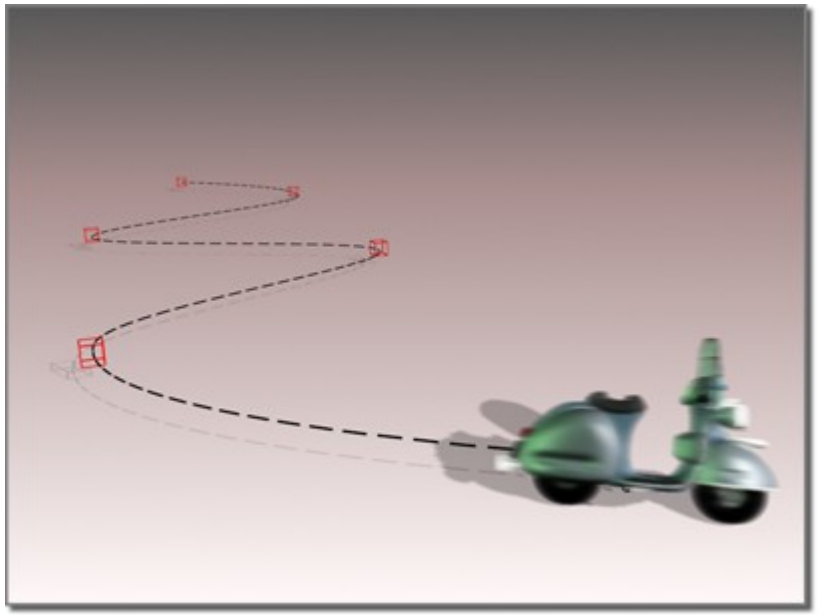
Ex: Trajectoire circulaire d'une lune autour d'une planète :

$$x_L = x_P + r \,.\, cos(2.\pi.f.t)$$
$$y_L = y_P + r \,.\, sin(2.\pi.f.t)$$
$$z_L = z_P$$



On peut utiliser n'importe quelle équation mathématique pour définir une trajectoire.

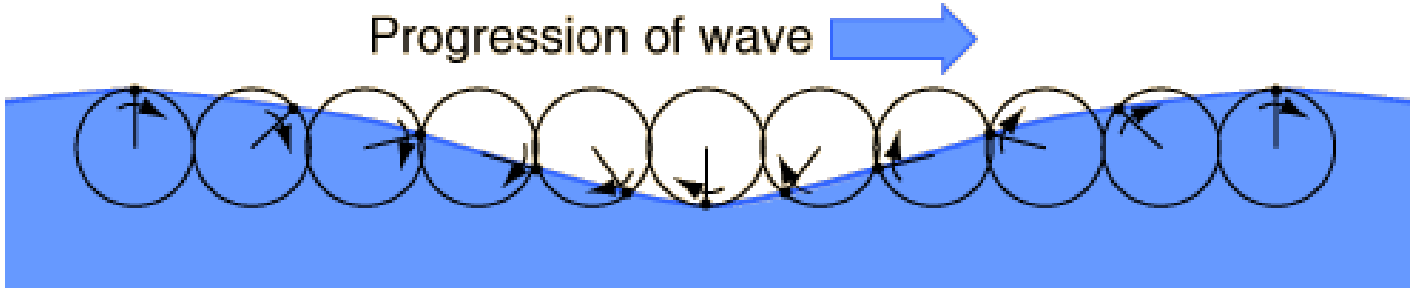
Les splines et courbes de Bézier présentent pour avantage de pouvoir définir une trajectoire définie par des points de contrôle, ce qui facilite l'édition de la trajectoire.



Animation de vagues

Profil d'une vague

Gerstner (1804) : le profil d'une vague est une *trochoïde*.

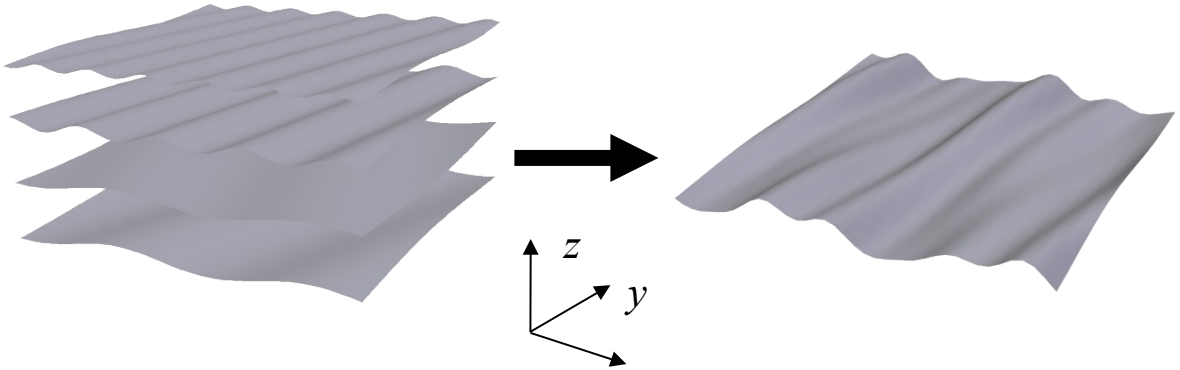


Chaque particule de l'eau a un mouvement circulaire de rayon r autour d'un point fixe (x_0, z_0)

$$x = x_0 + r.sin(k.x_0 - \omega.t)$$

$$z = z_0 - r.cos(k.x_0 - \omega.t)$$

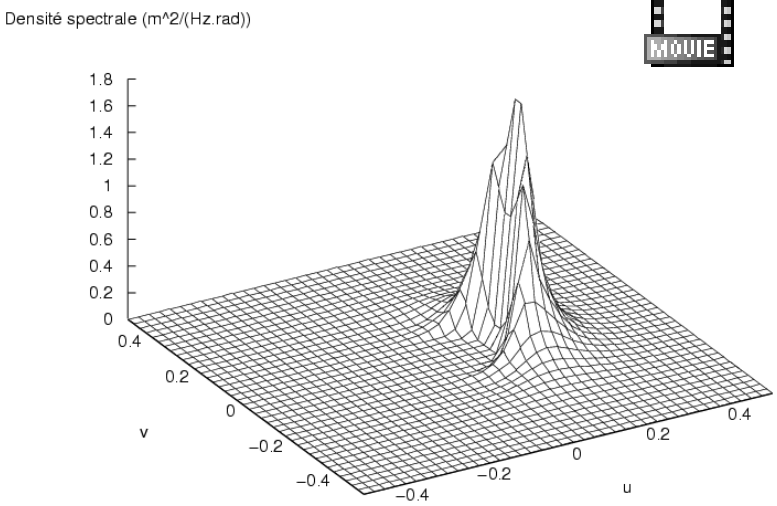
Modèle 3D de vagues d'océan : superposition de plusieurs trochoïdes 3D



$$h(x,y,t) = \sum_{i=1}^{n} A_i.trochoide\left(k_i\left(x.\cos\theta_i + y.\sin\theta_i\right) - \omega_i.t\right)$$

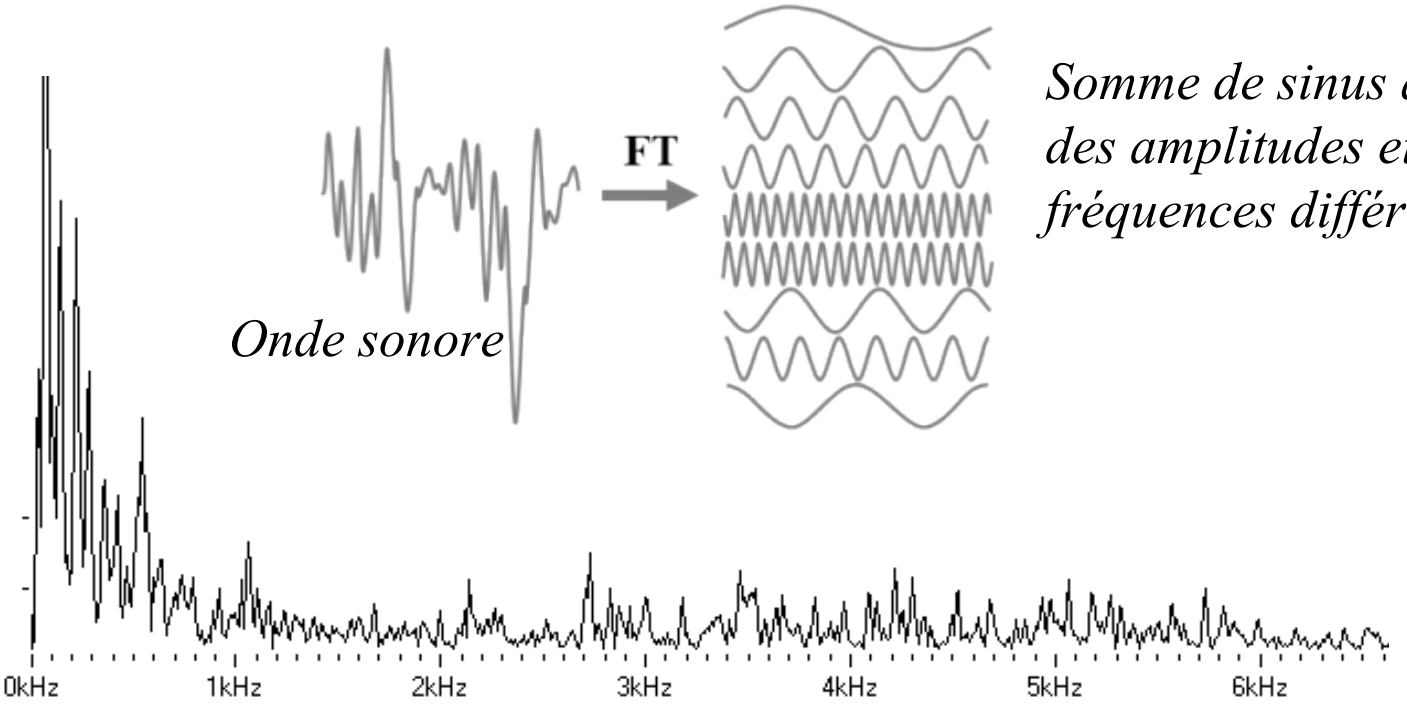
Chaque trochoïde est caractérisée par son amplitude A_i , sa direction θ_i et sa fréquence ω_i.

Ces paramètres peuvent être déterminés à partir d'un spectre de vague (ex: Pierson-Moskowitz)



Animation générée par du son

Une animation peut être générée par une expression paramétrée par le temps, mais aussi par d'autres paramètres comme par exemple la décomposition frequentielle d'un son obtenue par transformée de Fourier (FFT : *Fast Fourier Transform*).
→ permet de visualiser un son, une musique.



Somme de sinus ayant des amplitudes et des fréquences différentes

Représentation fréquentielle de l'onde sonore

On a vu plusieurs méthodes d'animation (transformations, interpolations, animation procédurale)

Mais...

Certaines animations seraient trop complexes à régler manuellement (cascade, feu d'artifice, ...)

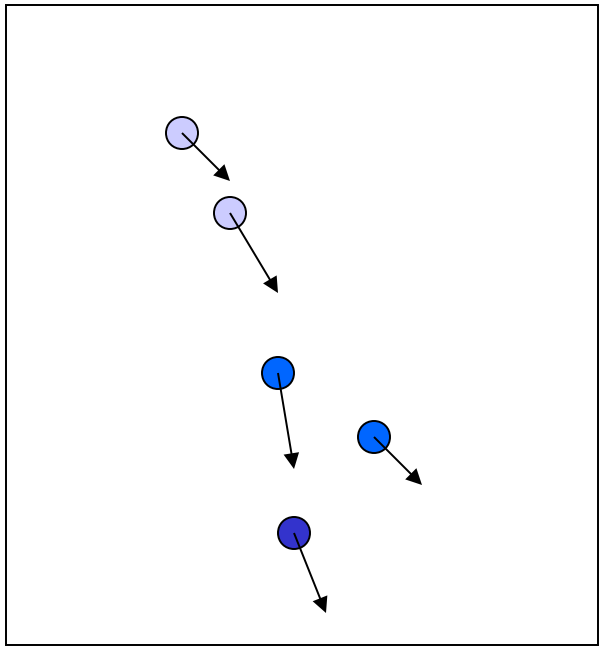
Certaines formes dans la nature peuvent difficilement être modélisées en 3D car leur surface n'est pas bien définie.

→ Utilisation d'une simulation dynamique en utilisant des lois de la physique.

4 Systèmes de particules

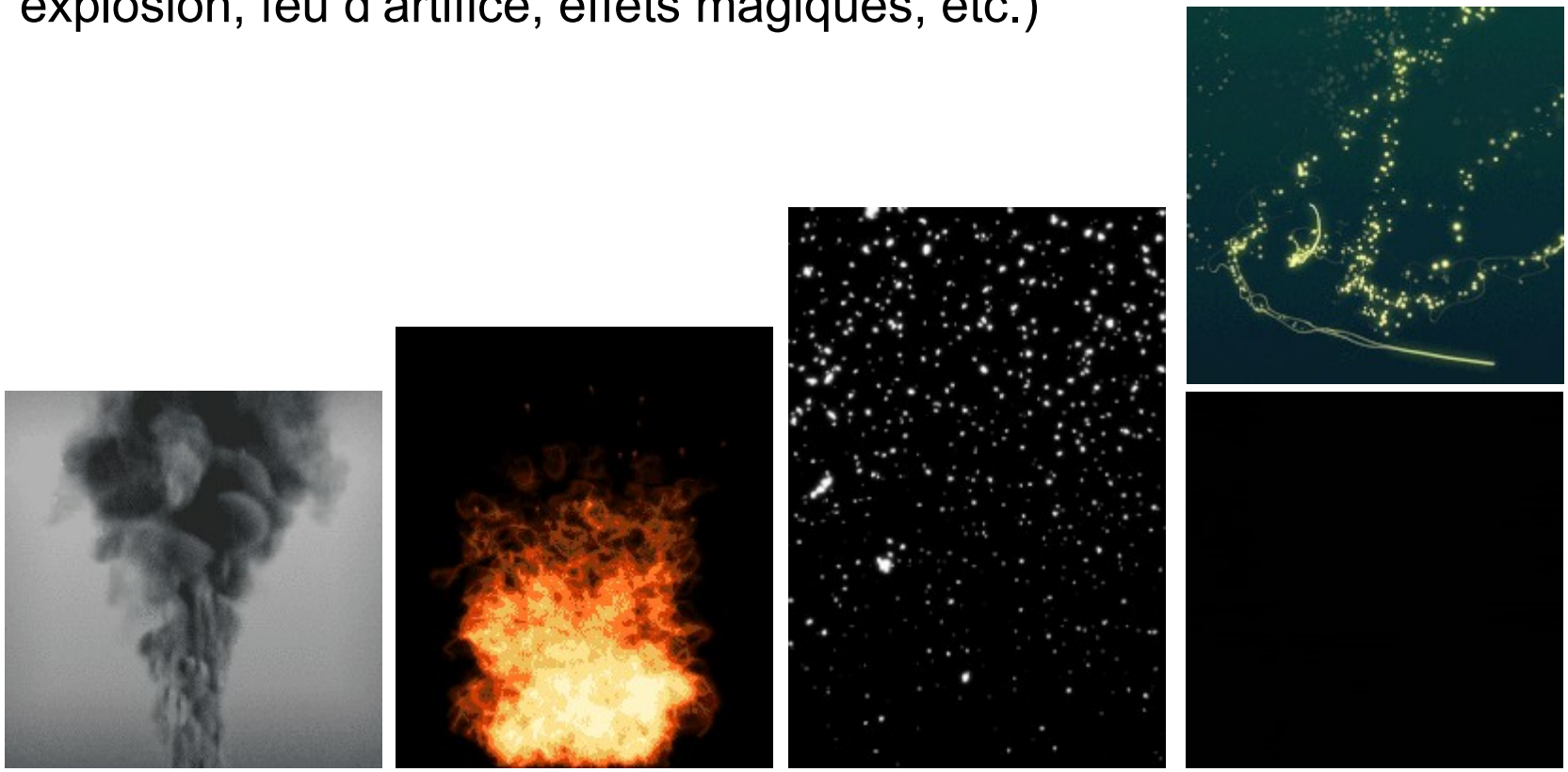
4.1 Présentation

Ensemble d'éléments dont les mouvements sont régis par des lois physiques (gravité, forces de frottement, collisions, etc.)



Ensemble de particules dotées de vitesses propres

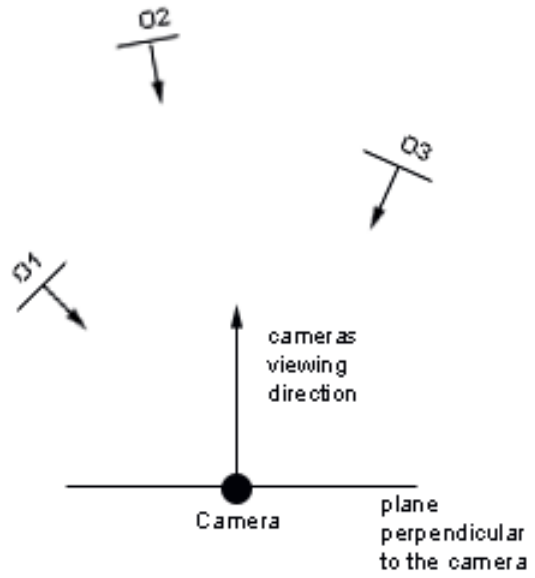
Les système de particules permettent de représenter des objets qui n'ont pas forme bien définie (pluie, neige, feu, fumée, chute d'eau, explosion, feu d'artifice, effets magiques, etc.)

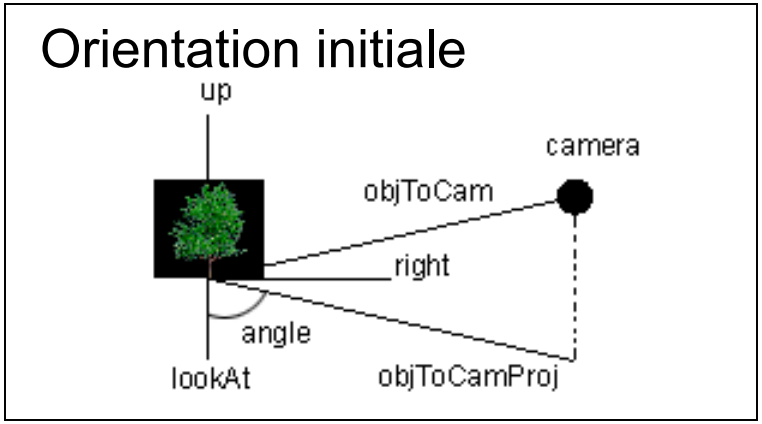


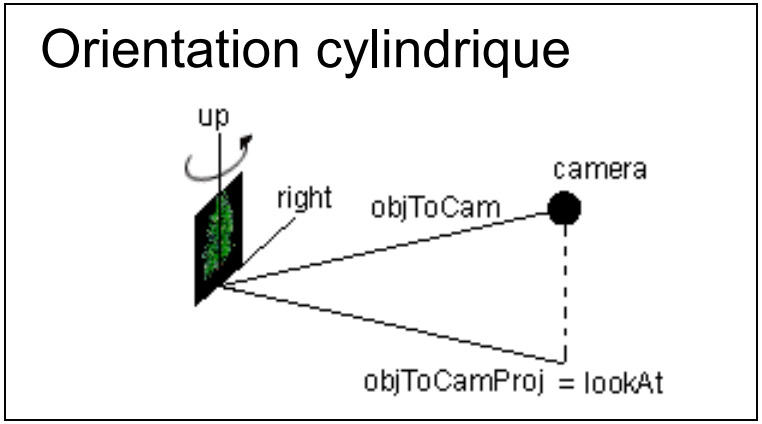
Les particules peuvent être représentées par de simples points, des polygones ou des objets 3D (sphères, blobs, etc.). On utilise souvent des quadrilatères texturés avec une texture RGBA, orientés face à l'utilisateur (« *billboards* »).

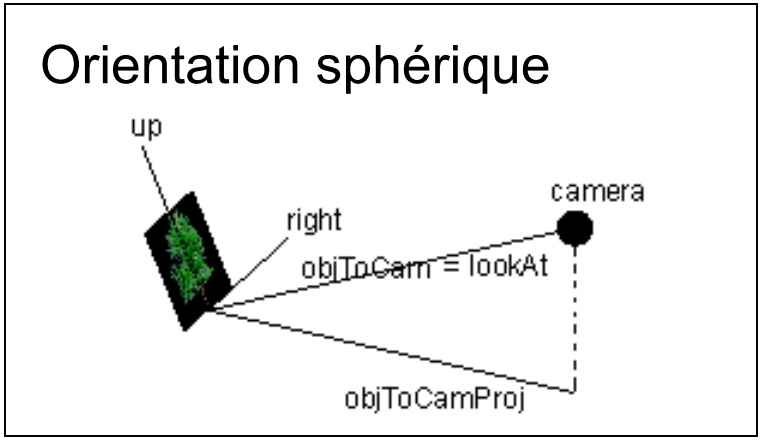
Billboard

C'est un quadrilatère texturé (texture RGBA) qui est toujours orienté face à l'observateur.



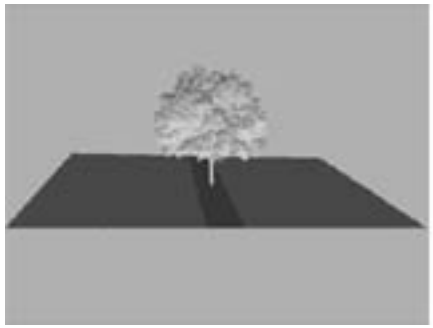


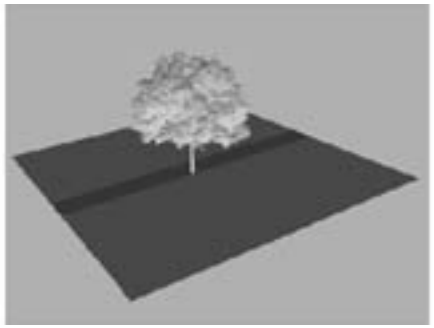


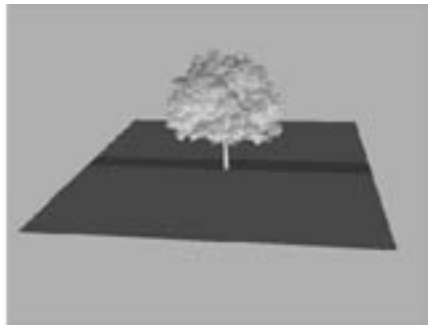


http://www.lighthouse3d.com/opengl/billboarding/


RP
Visit www.esrb.org
for updated rating
information.



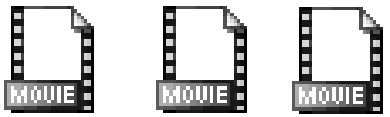




4.2 Attributs des particules

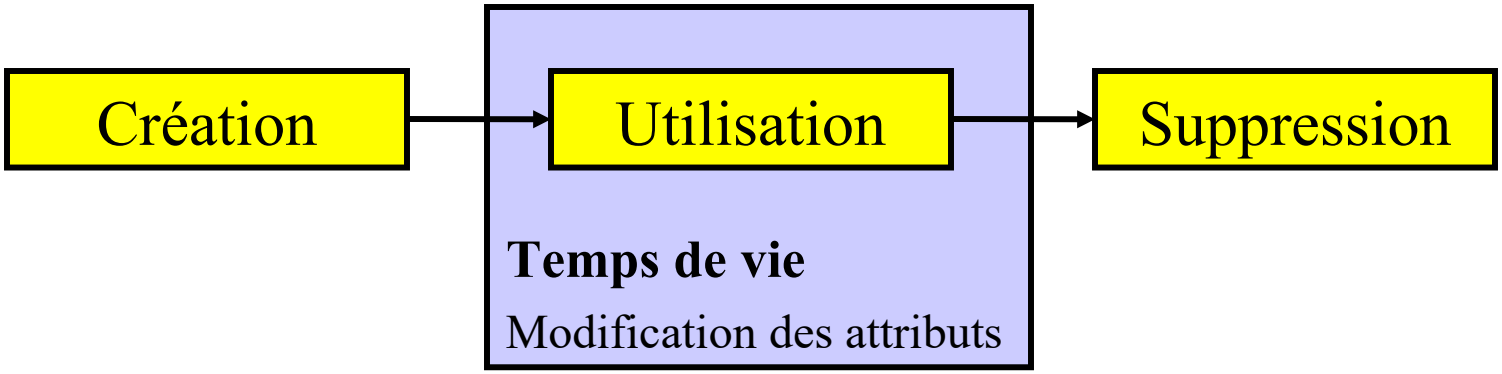
Chaque particule du système est caractérisée par un ensemble d'attributs, comme par exemple :

- Position
- Vitesse
- Masse
- Taille
- Couleur
- Transparence
- Durée de vie
- etc.



4.3 Gestion des particules

Système dynamique : lorsque les particules sont crées, elles ont un temps de vie au cours duquel leurs attributs peuvent être modifiés, puis elles disparaissent.



Gestion d'une particule au cours du temps

Gestion d'un ensemble de particules

A chaque nouvelle image de l'animation, on effectue les étapes suivantes :

1. Génération de nouvelles particules
2. Réglage des attributs des nouvelles particules
3. Les particules en fin de vie sont détruites
4. Les particules restantes sont déplacées et transformées en fonction de leurs attributs
5. Les particules restantes sont affichées

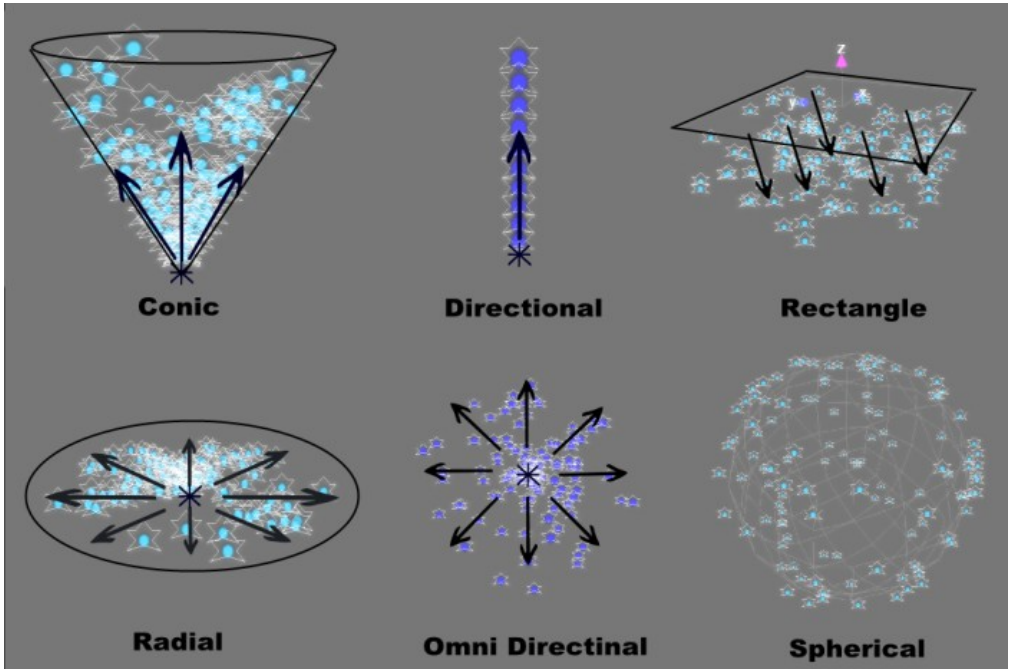
4.4 Structures de données

```
class Particule
{
    float  x, y, z;
    float  masse;
    float  duree_de_vie;
};

class Systeme
{
    list <Particule>  liste;
};
```

Emitter

La source des particules peut être un point, une surface, un volume, avec différentes directions d'emission.



4.5 Calcul des positions des particules au cours du temps

Discrétisation du temps, car affiche un certain nombre d'images par seconde → dt = intervalle de temps écoulé entre deux images.

On calcule la position d'un objet au cours du temps en utilisant un shéma d'intégration :

- Euler
- Midpoint
- Runge-Kutta à l'ordre 4
- ...

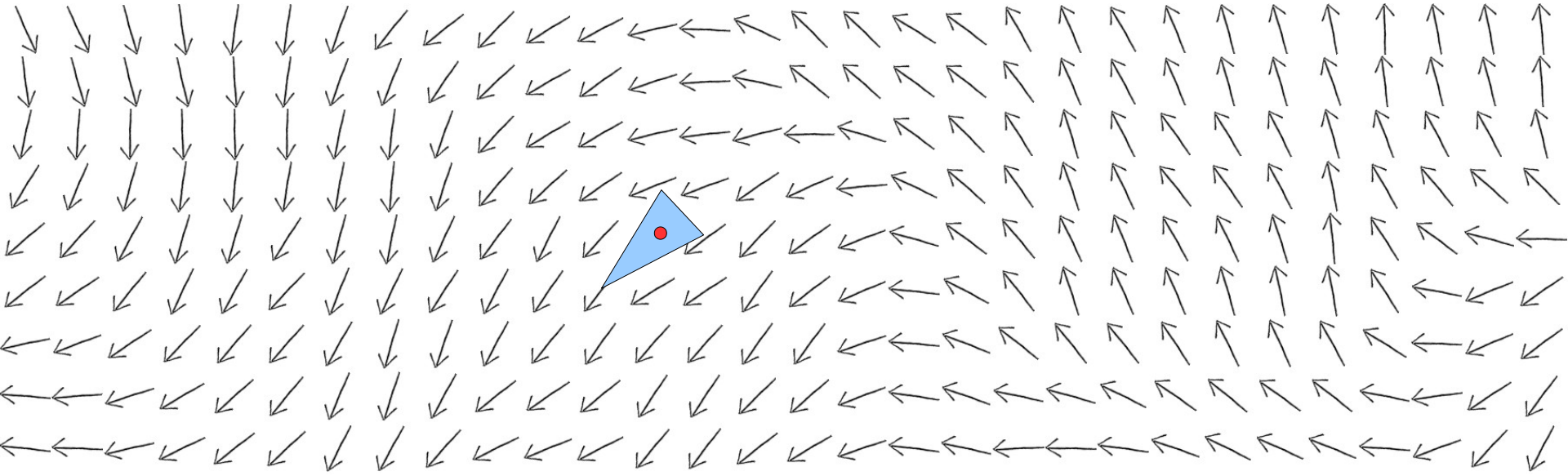
Schéma d'intégration d'Euler

Calcule la distance parcourue pendant un intervalle de temps dt et l'ajoute à la dernière position connue.

$$P(t+dt) = P(t) + V(t).dt$$

4.6 Animation par champ de vecteur

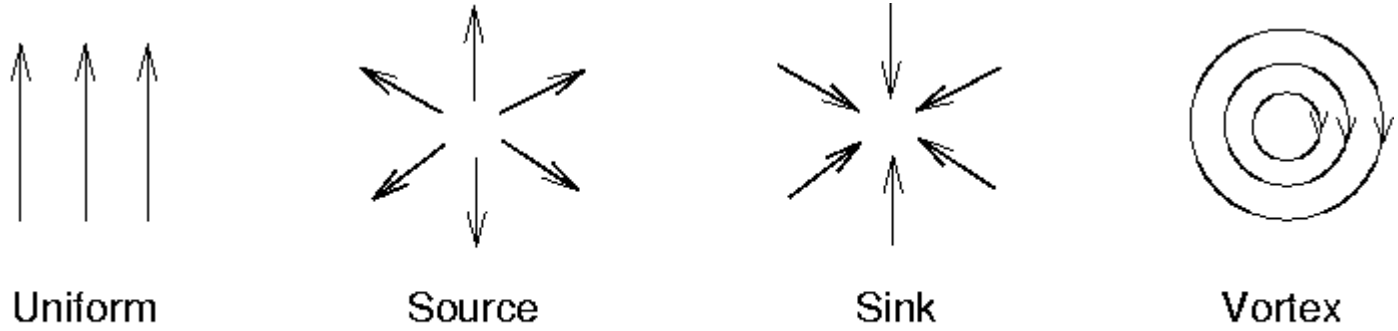
Le vecteur déplacement d'un objet est donné par sa position dans un champ de vecteurs.



Ce champ de vecteurs peut être obtenu de différentes manières (mécanique des fluides, bruit de Perlin, ...)

Champ de vecteurs obtenu par somme de primitives de flux

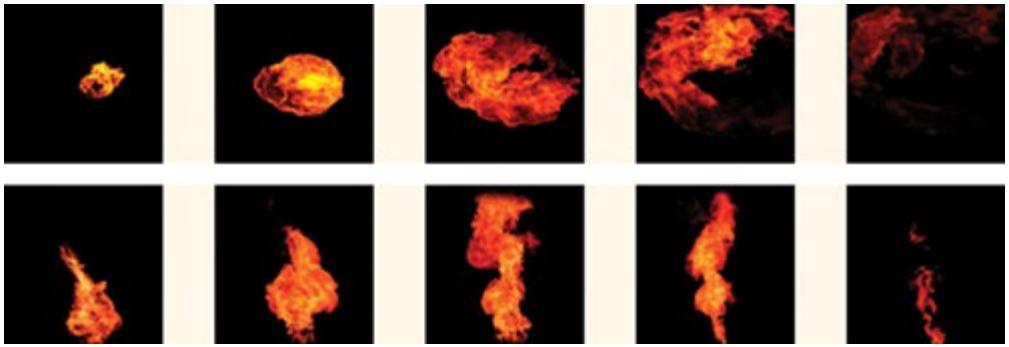
Vent complexe généré par une superposition de plusieurs motifs de vecteurs (*flow primitives*).



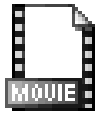
Dynamic Simulation of Grass Field Swaying in Wind
Qiu, Chen, Chen, Liu, Journal of software, 2012

Particules animées

Il faut parfois beaucoup de particules et mettre au point des équations très complexes pour parvenir à obtenir certains effets, comme par exemple des flammes.
Il est parfois plus simple d'utiliser très peu de particules (voire un seul billboard) sur lequel on plaque différentes textures au cours du temps, provenant d'une séquence d'animation.



Séquence de textures de flammes



http://http.developer.nvidia.com/GPUGems/gpugems_ch06.html

Logiciels

Des logiciels permettent de configurer facilement des systèmes de particules et de les sauver dans des fichiers à des formats (souvent en XML) qui peuvent être lus par des moteurs de jeu (Corona, Cocos2D, etc.)

ParticleDesigner
http://particledesigner.71squared.com


```
<particleEmitterConfig>
  <texture name="texture.png"/>
  <sourcePosition x="300.00" y="300.00"/>
  <sourcePositionVariance x="0.00" y="0.00"/>
  <speed value="100.00"/>
  <speedVariance value="30.00"/>
  <particleLifeSpan value="2.0000"/>
  <particleLifespanVariance value="1.9000"/>
  <angle value="270.00"/>
  <angleVariance value="2.00"/>
  <gravity x="0.00" y="0.00"/>
  <radialAcceleration value="0.00"/>
  <tangentialAcceleration value="0.00"/>
  <radialAccelVariance value="0.00"/>
  <tangentialAccelVariance value="0.00"/>
  <startColor red="1.00" green="0.31" blue="0.00" alpha="0.62"/>
  <startColorVariance red="0.00" green="0.00" blue="0.00" alpha="0.00"/>
  <finishColor red="1.00" green="0.31" blue="0.00" alpha="0.00"/>
  <finishColorVariance red="0.00" green="0.00" blue="0.00" alpha="0.00"/>
  <maxParticles value="500"/>
  <startParticleSize value="70.00"/>
  <startParticleSizeVariance value="49.53"/>
  <finishParticleSize value="10.00"/>
  <FinishParticleSizeVariance value="5.00"/>
  <duration value="-1.00"/>
  <emitterType value="0"/>
  <maxRadius value="100.00"/>
  <maxRadiusVariance value="0.00"/>
  <minRadius value="0.00"/>
  <minRadiusVariance value="0.00"/>
  <rotatePerSecond value="0.00"/>
  <rotatePerSecondVariance value="0.00"/>
  <blendFuncSource value="770"/>
  <blendFuncDestination value="1"/>
  <rotationStart value="0.00"/>
  <rotationStartVariance value="0.00"/>
  <rotationEnd value="0.00"/>
  <rotationEndVariance value="0.00"/>
</particleEmitterConfig>
```

Remarque

Des systèmes de particules statiques peuvent aussi être utilisés pour représenter des cheveux, de la fourrure, de l'herbe, etc. en calculant plusieurs positions des particules au cours du temps et en les reliant. La durée de vie donne la longueur.