Institut Universitaire de Technologie
Aix*Marseille Université

Imagerie Numérique
M4102Cin - Synthèse d'images 2

7. Animation physique

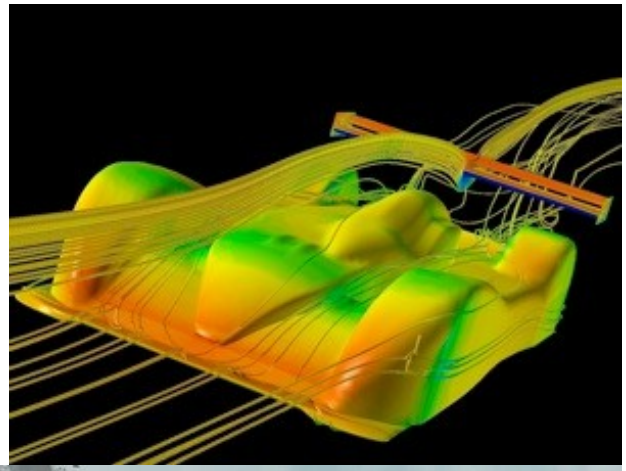
DUT Informatique 2018-2019

Sébastien THON

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

Simulations dynamiques avancées

Pour représenter de manière très réaliste certains mouvements de la réalité (de solides, liquides, tissus, etc.), on a recours à des **modèles physiques** = des équations qui décrivent ces mouvements au cours du temps en fonction des forces en jeu, selon les lois de la mécanique (des solides, des fluides, etc.)



Problème :

Les temps de calculs sont souvent très élevés.

Ça ne pose pas de problème dans des situations telles que la simulation scientifique, la résistance des matériaux, l'architecture, où on a besoin de résultats rigoureux, ou encore le cinéma où on peut se permettre de passer des heures à calculer des images.

En revanche, dans le cas d'applications 3D temps réel, cette lenteur de calcul est préjudiciable car on veut calculer et afficher la scène 3D plusieurs dizaines de fois par seconde.

Solutions :

- Faire le moins de calculs possible en simplifiant les maillages
- Approximation des équations physiques utilisées
- Utiliser le processeur de la carte graphique, le GPU, pour faire du GPGPU (general-purpose processing on graphics processing units) afin d'accélérer les calculs matriciels et vectoriels (utilisation de CUDA, de OpenCL)

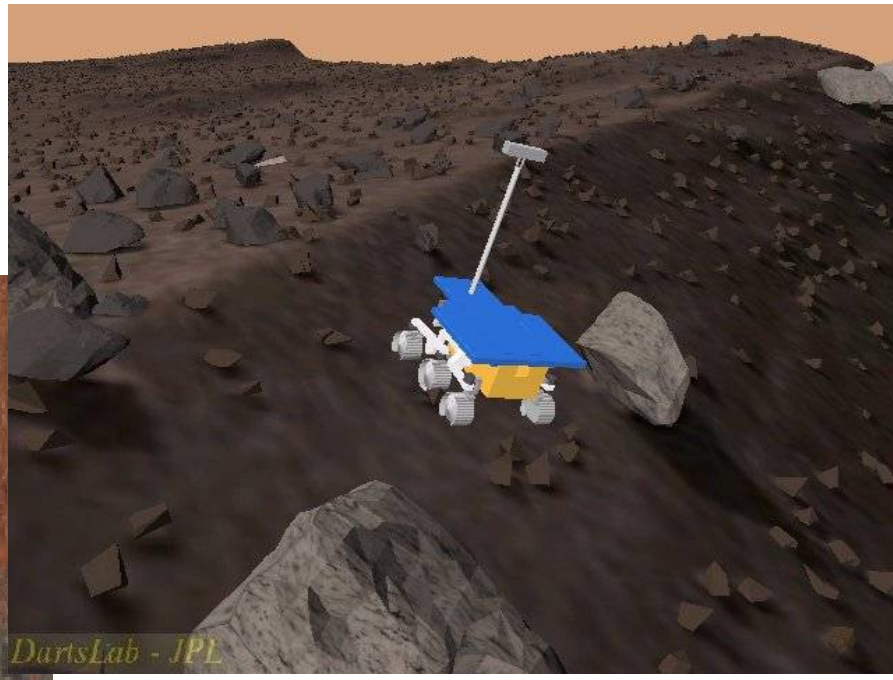
https://fr.wikipedia.org/wiki/Compute_Unified_Device_Architecture

https://fr.wikipedia.org/wiki/OpenCL

→ De telles animations physiques temps réel apparaissent dans des applications industrielles, des simulateurs, des jeux vidéos, etc.

Prototypage

Prototypage de nouveaux véhicules, de robots, de rover devant évoluer sur la Lune ou Mars, etc.

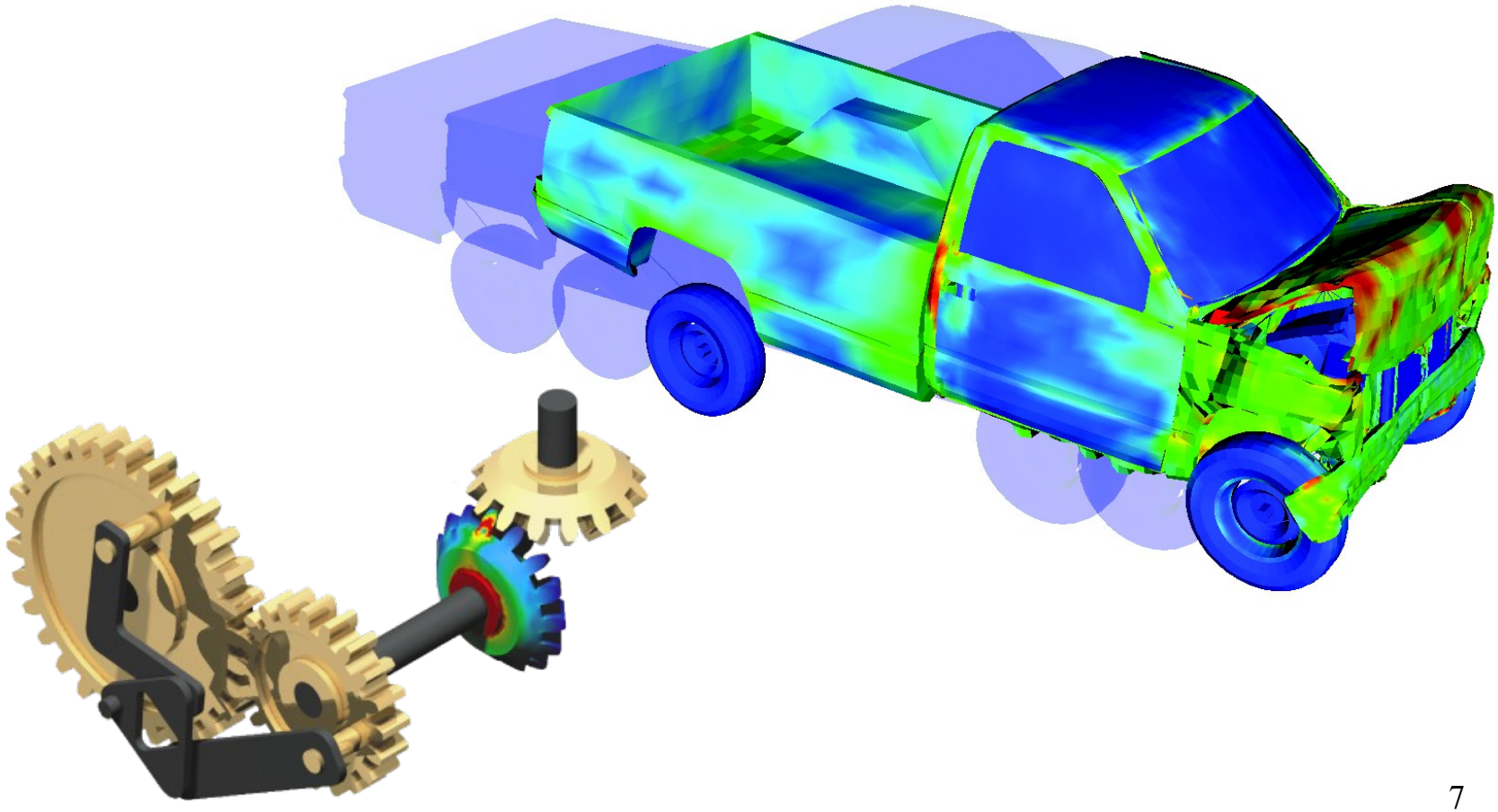


Simulations

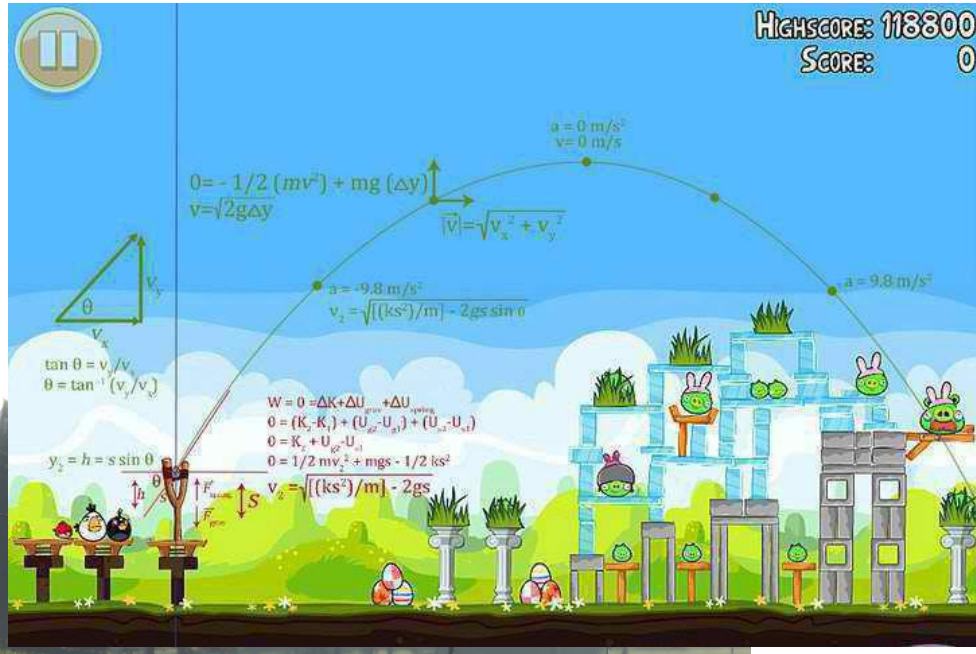
Simulation physiquement réaliste du comportement de véhicules pour que l'apprentissage de leur conduite soit valide.

Applications industrielles

Construction automobile, aéronautique, bâtiments, resistance des matériaux.



Jeu vidéo


HIGHSCORE: 118800
SCORE: 0
a = 0 m/s²
v= 0 m/s
0= - 1/2 (mv²) + mg (Δy)
v=√2gΔy
|v|=√v_x² + v_y²
a = -9.8 m/s²
a = 9.8 m/s²
tan θ = v_y/v_x
θ = tan⁻¹(v_y/v_x)
y_2 = h = s sin θ
W = 0 =ΔK+ΔU_grav+ΔU_spring
0 = (K_2-K_1) + (U_g2-U_g1) + (U_s2-U_s1)
0 = K_2 + U_g2-U_s1
0 = 1/2 mv_2² + mgs - 1/2 ks²
v_2 =√[(ks²)/m] - 2gs

Cinéma

Brave, Disney (2012)

San Andreas (2015)

1. Utilisation de la mécanique des solides

Utilisation des lois du mouvement énoncées par Newton :

- principe d'inertie :

  S'il n'y a pas de force qui s'exerce sur un corps ou si la somme des forces s'exerçant sur lui est égale au vecteur nul, la direction et la norme de sa vitesse ne changent pas (=accélération nulle).

- principe fondamental de la dynamique :

  L'accélération subie par un corps dans un référentiel galiléen est proportionnelle à la résultante des forces qu'il subit, et inversement proportionnelle à sa masse m.

- principe des actions réciproques :

  Tout corps A exerçant une force sur un corps B subit une force d'intensité égale, de même direction mais de sens opposé, exercée par le corps B.

Prise en compte des objets, des relations de jointure entre ces objets, des forces appliquées aux objets (gravité, collisions, frictions).

→ Animation très réaliste de véhicules, de collision d'objets (billard, bowling), d'objets articulés (personnage, grue, robot), de cordes, de cheveux, etc.

1.1 Principe

1) Application de forces aux objets

On calcule la somme des vecteurs force appliqués à un objet.

La norme du vecteur est exprimée en newton.

Un newton (symbole : N) est la force capable de communiquer à une masse de 1 kilogramme une accélération de 1 m/s^2.

2) Calcul des positions des objets

On calcule la vitesse d'un objet en fonction des forces qui lui sont appliquées, selon le principe fondamental de la dynamique (deuxième loi de Newton) :

$$\Sigma F = m \,.\, a$$

ainsi que les équations de la cinématique :

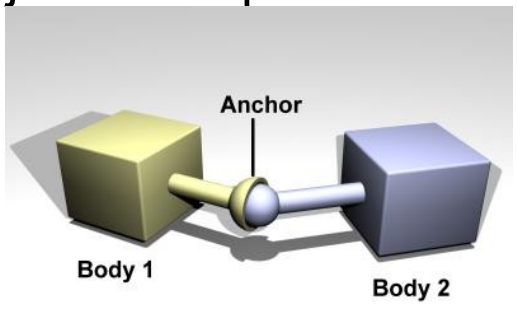
$$a = dV / dt$$
$$\Longrightarrow \quad dV = a \,.\, dt$$
$$\Longrightarrow \quad dV = (\Sigma F / m) \,.\, dt$$

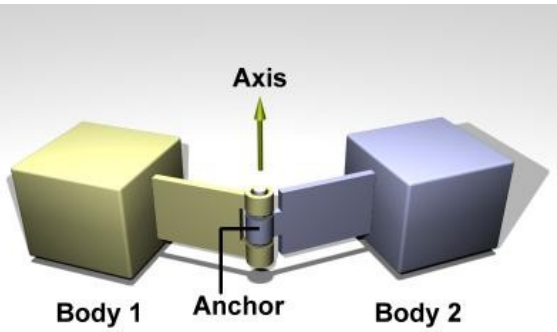
(avec **a** : accélération, **ΣF** : somme des forces, **V** : vitesse, **t** : temps, **m** : masse)

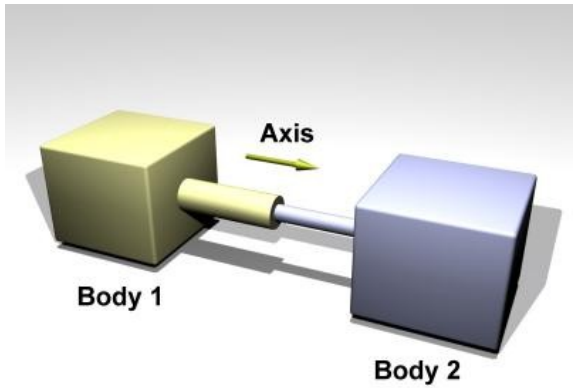
Ce qui se traduit par :
vitesse = vitesse + (somme des forces / masse) * dt
position = position + vitesse * dt

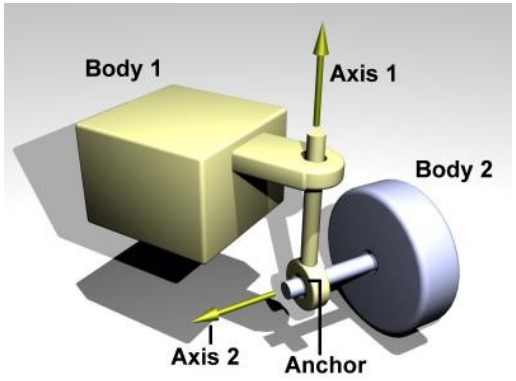
3) Application de contraintes

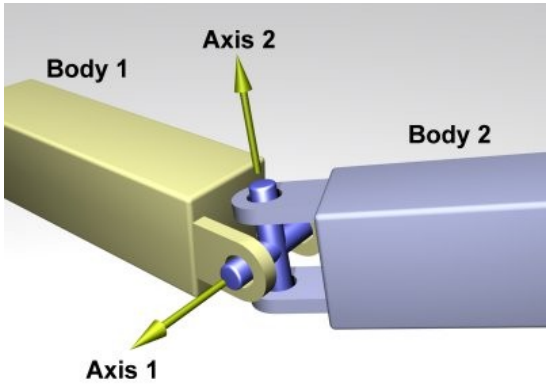
Les différents objets peuvent être reliés entre eux par différents types de jointures qui exercent des contraintes.



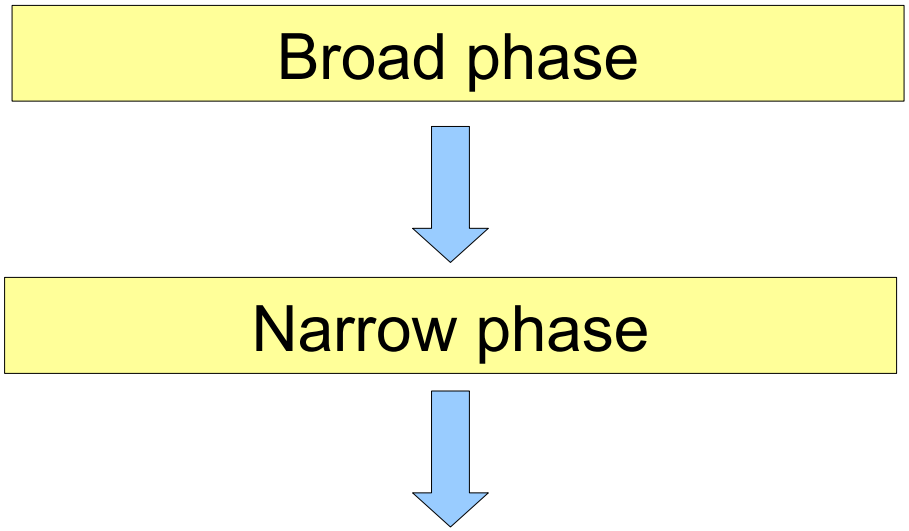








4) Gestion des collisions



Liste de points de contact (position, normale à la surface touchée, distance de pénétration)

5) Solveur

En fonction des forces appliquées et des contraintes, on calcule les nouvelles positions des objets.

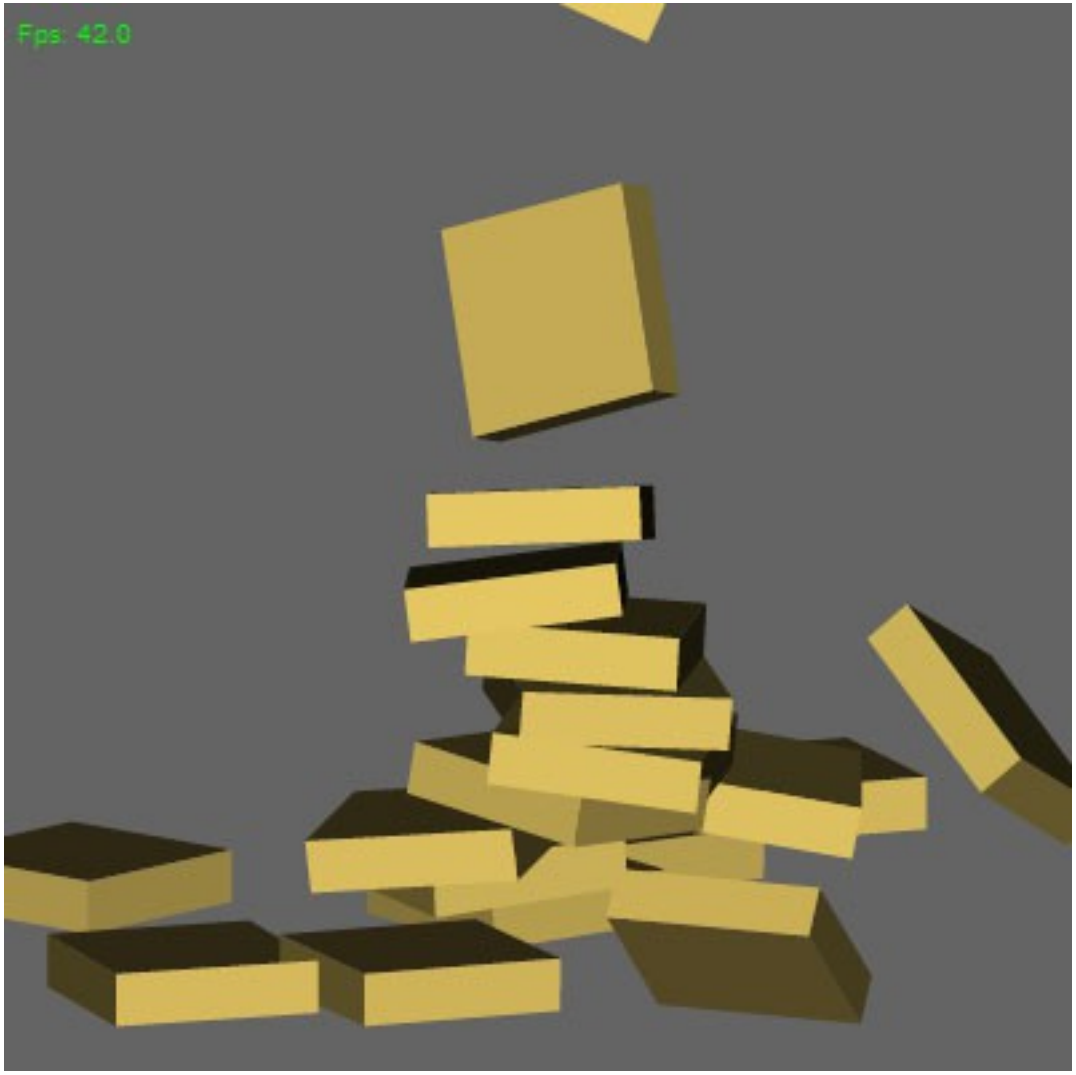
http://chamilo1.grenet.fr/ujf/main/document/document.php?curdirpath=%2FDoc_Forge&cidReq=

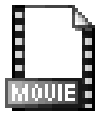
1.2 Applications

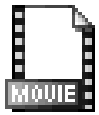
Les équations de la mécanique des solides peuvent être utilisées pour produire toutes sortes d'animations physiques :

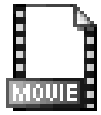
- Objets rigides
- Systèmes de particules
- Personnages
- Cordes
- Chevelures
- Tissus
- ...

1.2.1 Animation d'objets rigides





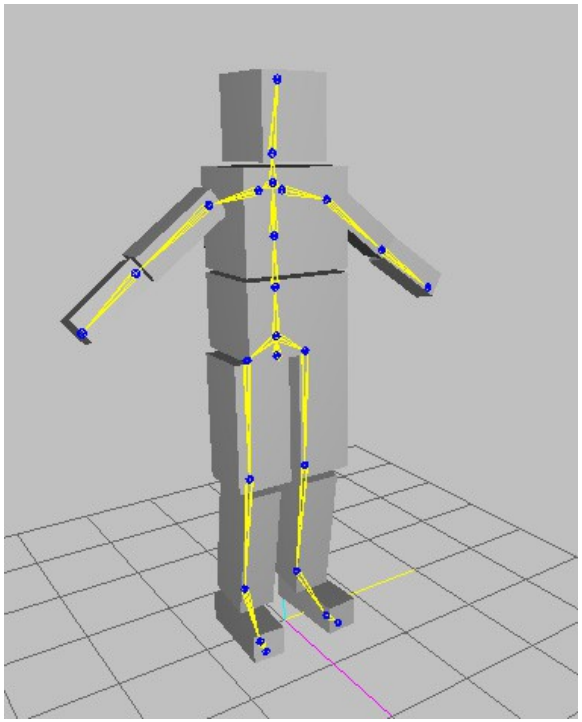




1.2.2 Ragdoll (« poupée de chiffon »)

Corps = assemblage d'éléments solides, liés par des articulations, et réagissant de manière réaliste avec son environnement (gravité, choc).

Technique très utilisée dans les jeux vidéos, le cinéma, pour simuler des chocs réalistes.

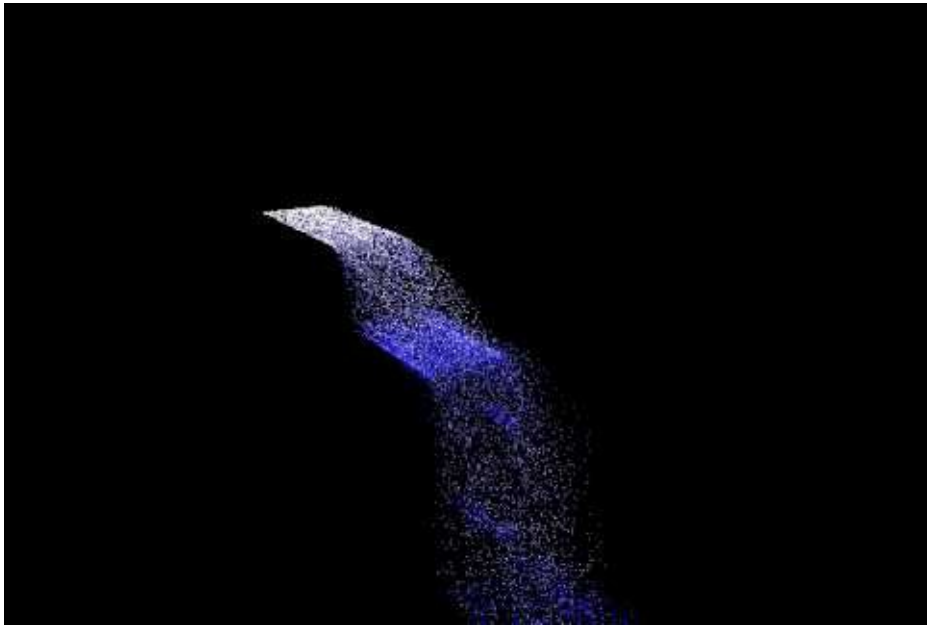


MOVIE MOVIE

1.2.3 Gestion physique de systèmes de particules

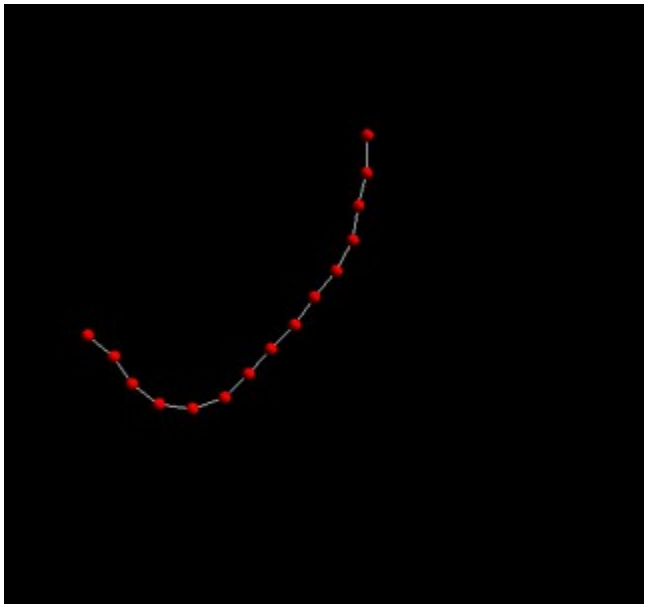
On peut animer un système de particules en prenant en compte les forces qui leur sont appliquées (force initiale, gravité, vent, collisions, frottements, etc.) plutôt qu'en donnant directement un vecteur vitesse comme on l'avait fait précédemment.

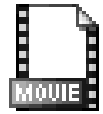
→ permet de gérer l'accélération, l'interaction avec l'environnement, offre plus de souplesse et de réalisme.



1.2.4 Animation de cordes

Principe: sommets reliés par des liaisons plus ou moins élastiques (système masses-ressorts). Des forces sont appliquées sur les sommets (gravité, etc.)

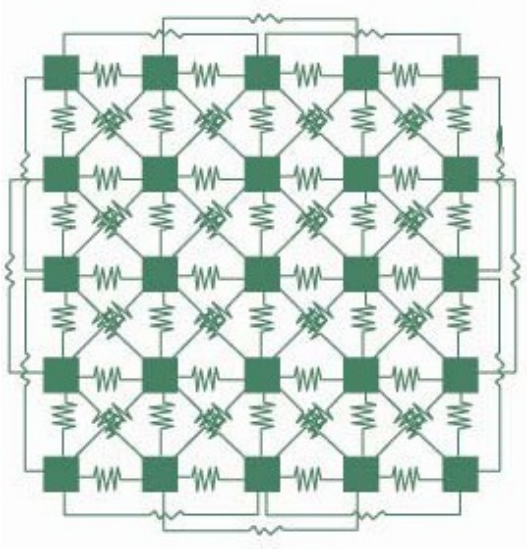


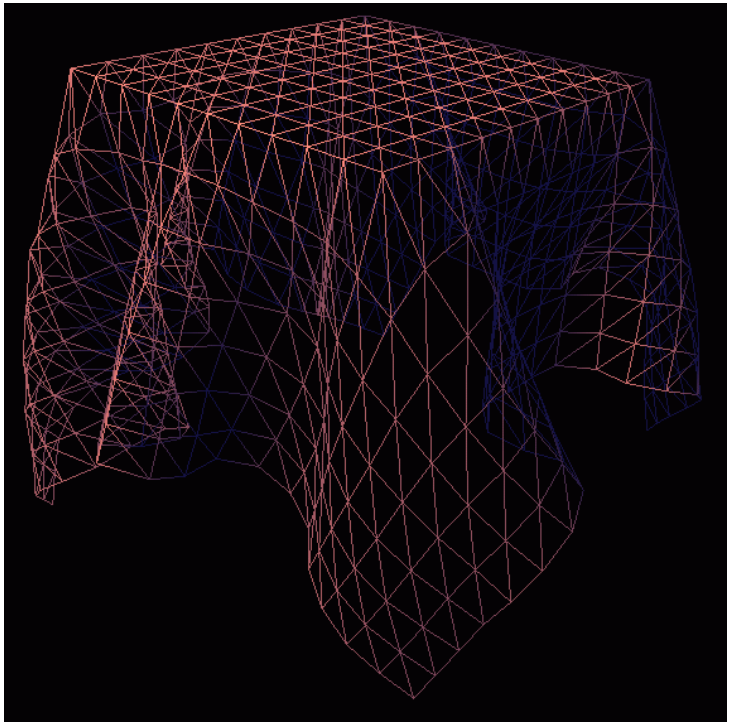


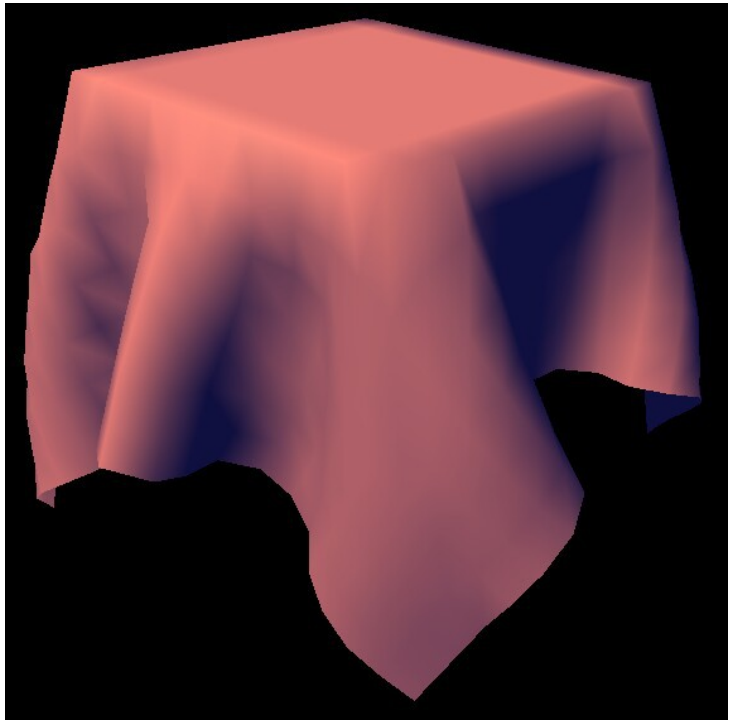
http://nehe.gamedev.net/tutorial/rope_physics/17006/

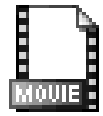
1.2.5 Animation de tissus

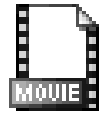
Principe: maillage 2D de sommets reliés par des liaisons plus ou moins élastiques (système masses-ressorts). Des forces sont appliquées sur les sommets (gravité, etc.)
(“Soft body dynamics”)



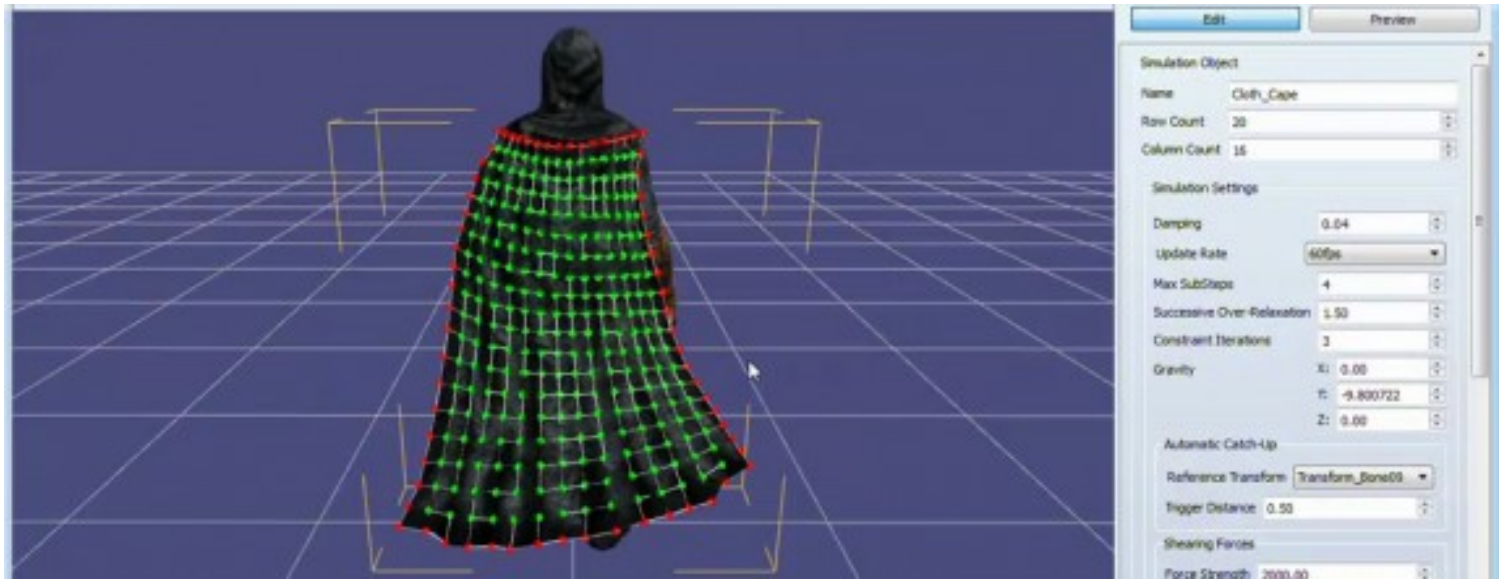




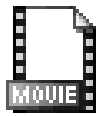




Très courants dans les films d'animation, l'augmentation de la puissance des CPU et l'utilisation du GPU permettent d'animer des tissus en temps réel pour le jeu vidéo.



Shroud – Cloth Simulation Engine
Permet de transformer en tissu certaines parties d'un modèle 3D, ou de créer un tissu et de l'attacher à un objet importé
http://www.cloak-works.com/



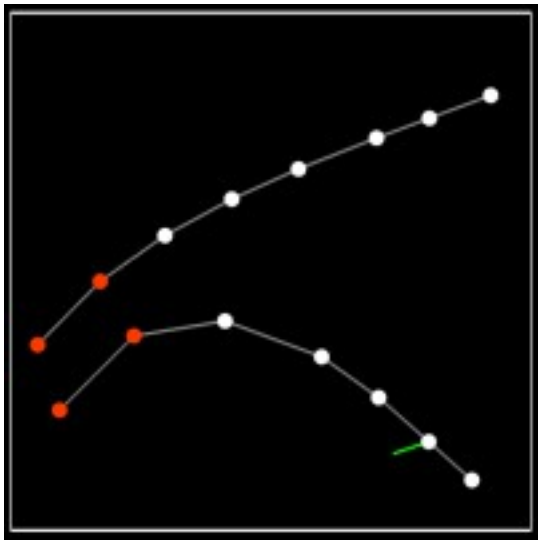
Apex Clothing (Nvidia)
https://developer.nvidia.com/clothing

Havok Cloth (Havok)
http://www.havok.com/?page=havok-cloth

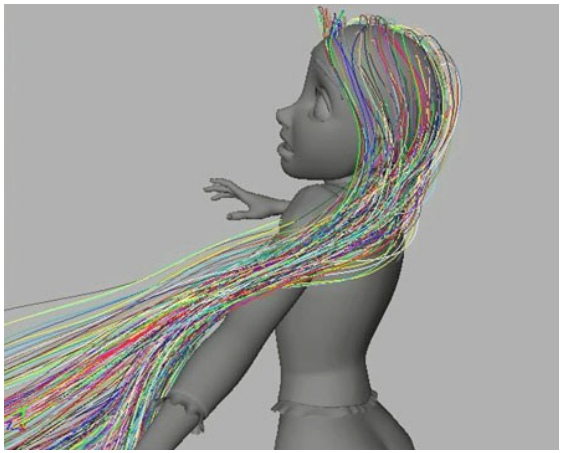
Carbon (Numerion Software)
http://www.numerion-software.com/

1.2.6 Animation de cheveux

On peut représenter une chevelure (ou de la fourrure) par des milliers de cheveux constitués d'une chaîne d'une dizaine de liens réagissant à différentes forces physiques (gravité, inertie, vent, mouvements de la tête du personnage, ...)



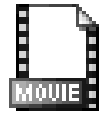
Cheveux modélisé par un système masses-ressorts



Laboratoire GRAVIR, Grenoble

Pour les cheveux très bouclés de Merida dans *Brave*, utilisation de 1500 B-Splines comme cheveux clefs, servant à interpoler 111000 cheveux.

https://www.fxguide.com/featured/brave-new-hair/



L'utilisation du GPU permet l'animation de chevelures en temps réel dans les jeux vidéo (Tomb Raider, The Witcher 3, etc.) avec illumination, auto-ombrage, antialiasing, transparence.

AMD TressFX

http://fr.slideshare.net/DevCentralAMD/gs4147-billbilodeau/

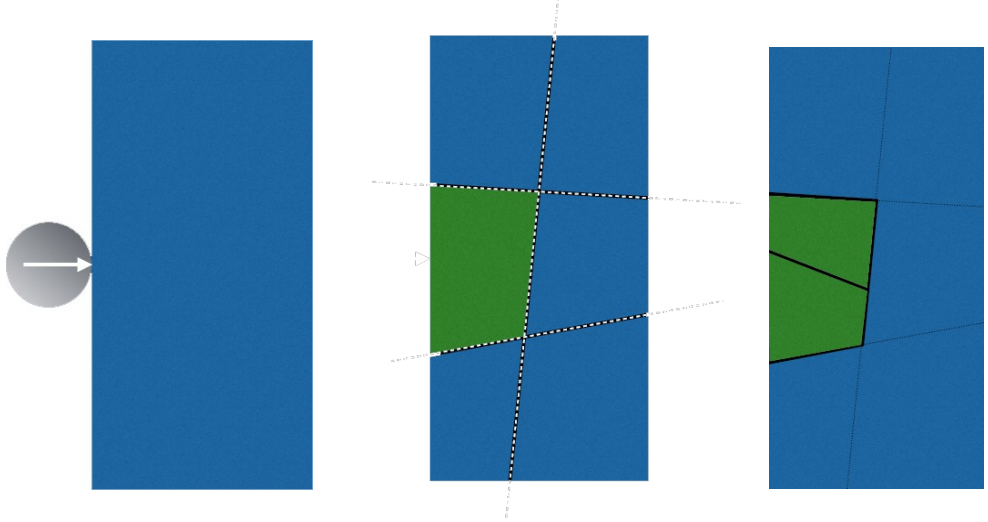
http://www.amd.com/en-us/innovations/software-technologies/technologies-gaming/tressfx

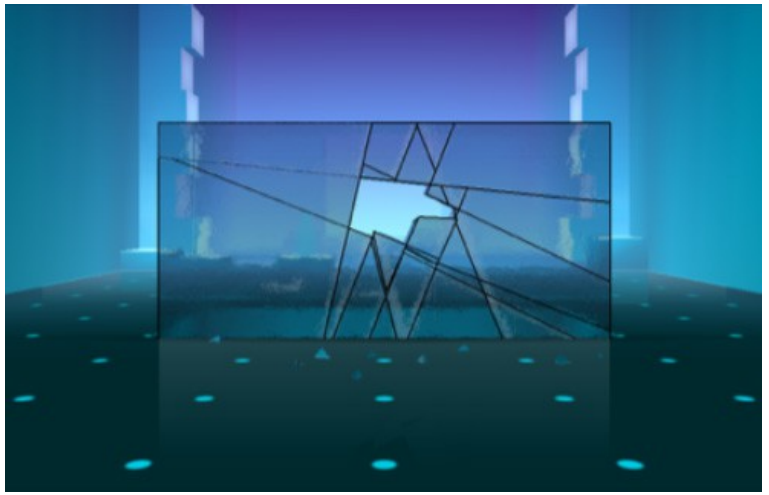
NVIDIA HairWorks

https://developer.nvidia.com/hairworks

1.2.7 Destruction d'objets

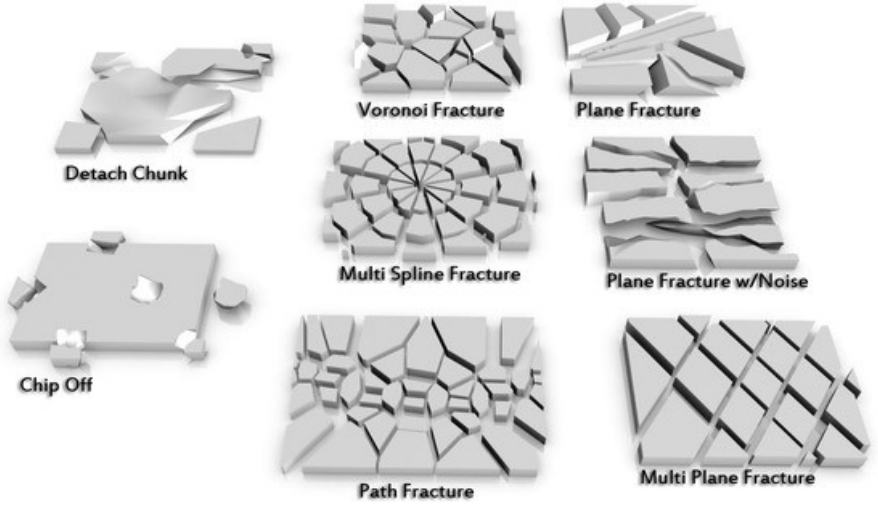
Création de lignes de fracture suite à la collision entre deux objets, détachement et subdivision en éclats de certains des polygones formés qui sont ensuite soumis à la gravité et aux forces de collision.

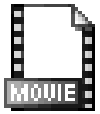




http://www.benicourt.com/blender/wp-content/uploads/2015/03/DennisGustafsson_Destruction.pdf

On peut choisir parmi plusieurs patterns de fracturation.

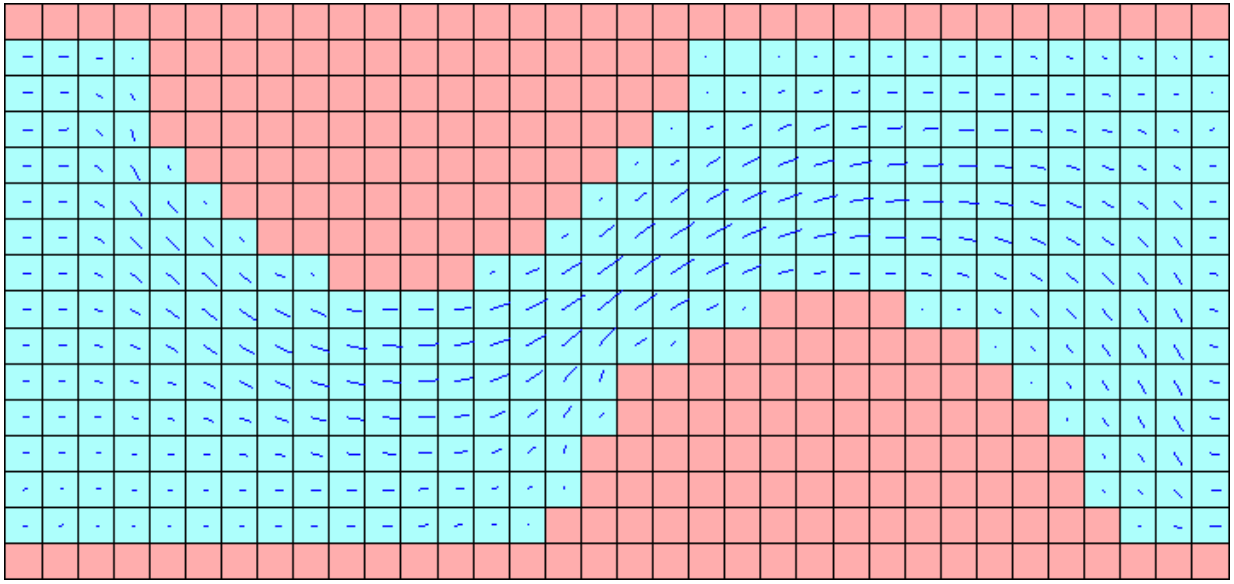




2. Utilisation de la mécanique des fluides

Utilisation des équations de Navier et Stokes, qui décrivent le comportement de fluides (liquides, gaz) en fonction de leur viscosité, de la pression, de la température, du temps, des forces en présence.

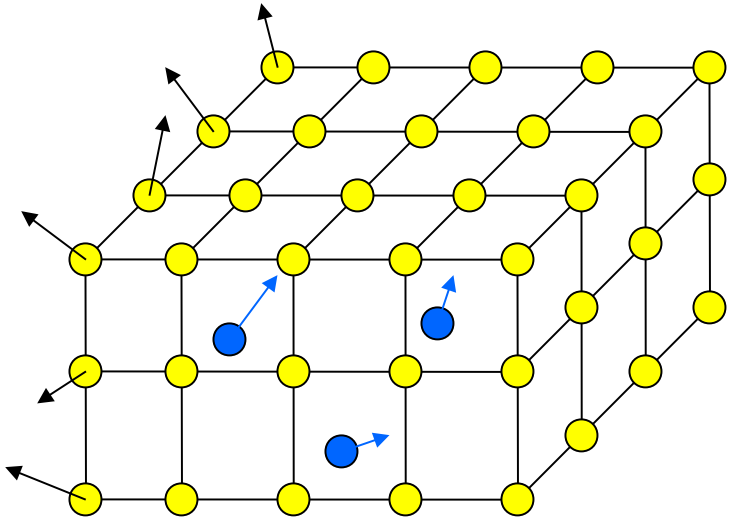
L'espace qui sera rempli par l'eau est discrétisé en un maillage 3D de sommets. Au cours du temps, des vecteurs vitesse de fluide sont calculés en chaque sommet du maillage grâce aux équations de Navier-Stokes.



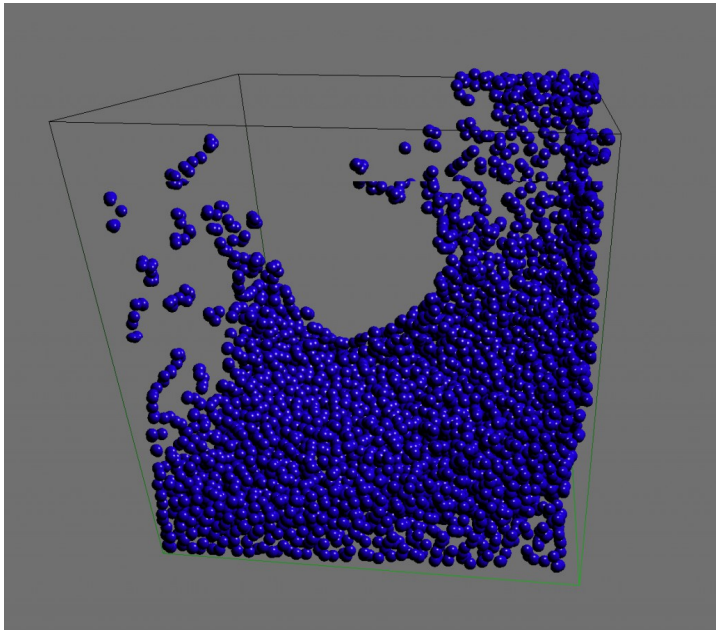
SPH : Smoothed Particle Hydrodynamics

https://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-15

Des particules de fluide sont placées dans cette matrice 3D, soumises aux vecteurs vitesse, et leurs trajectoires sont suivies au cours du temps.
On gère les collisions des particules avec leur environnement, ce qui permet d'obtenir des eclaboussures.

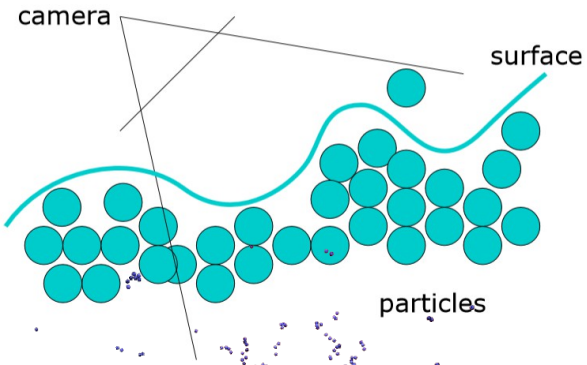


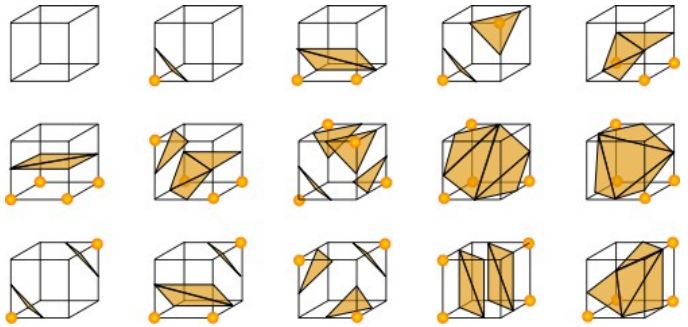
Suivi de particules au cours du temps dans un maillage 3D de vecteurs vitesse

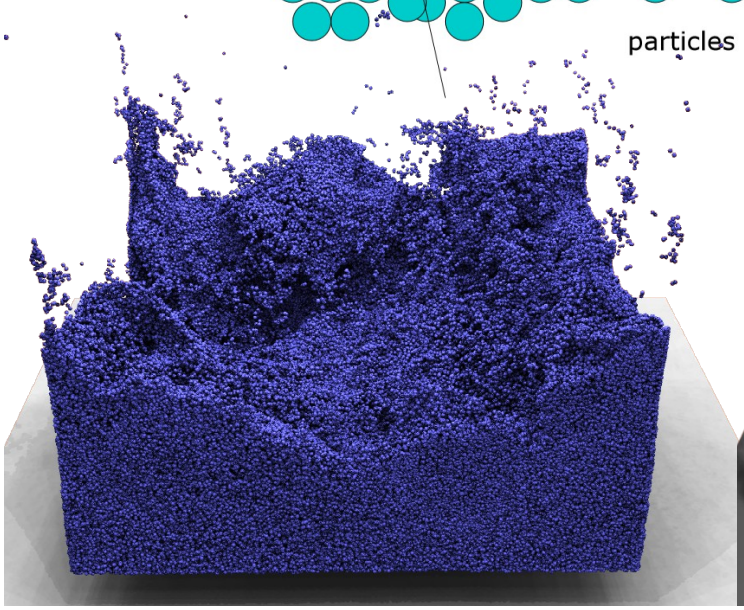


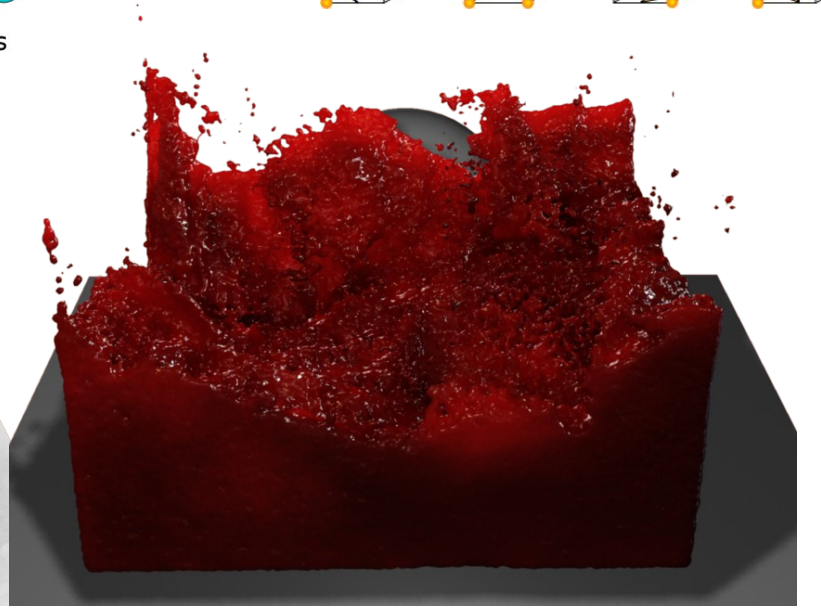
2.1 Animation de liquides

Une surface englobant les particules est rendue (surface implicite, marching cubes, point sprites, ….)





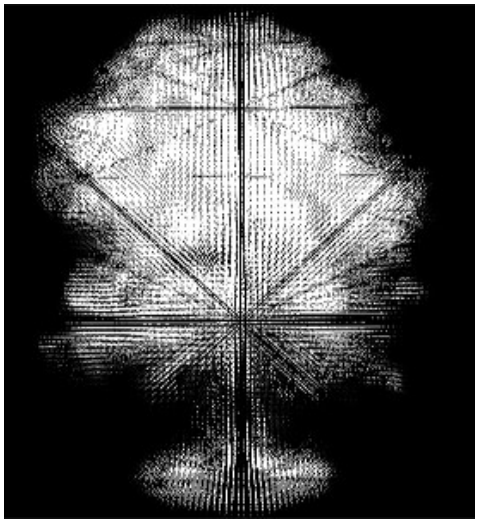




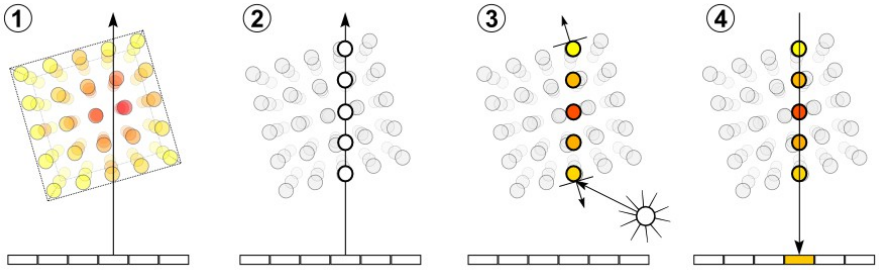
https://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-1/
https://developer.nvidia.com/content/fluid-simulation-alice-madness-returns

2.2 Animation de gaz

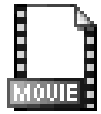
L'ensemble des particules peut être rendu avec des billboards, des textures volumiques, du raymarching, etc.

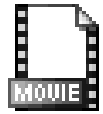


Nuage de particules



Raymarching

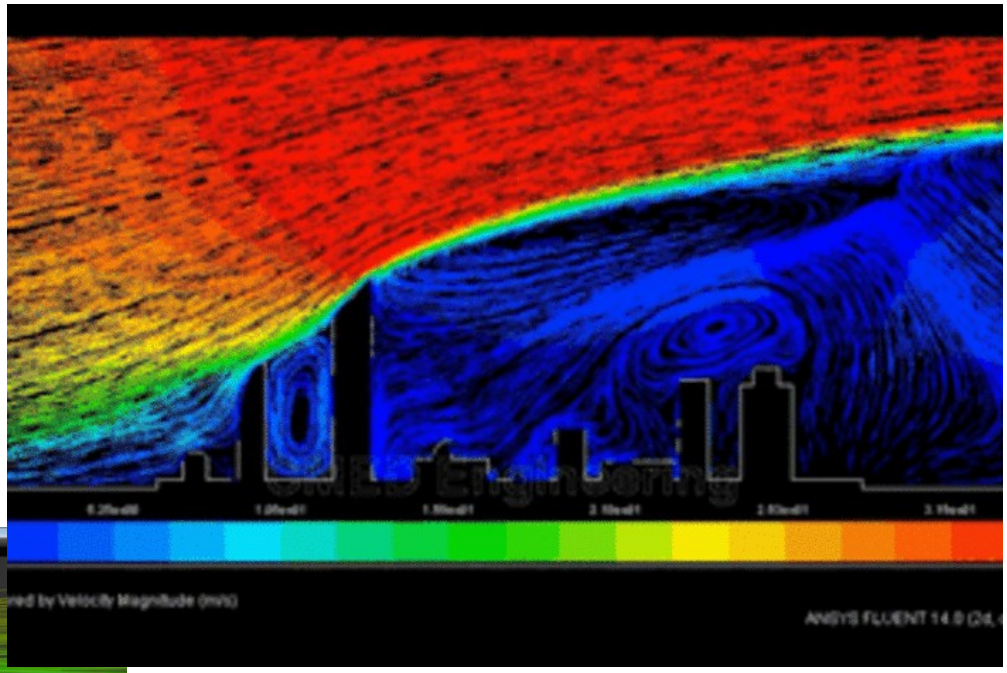


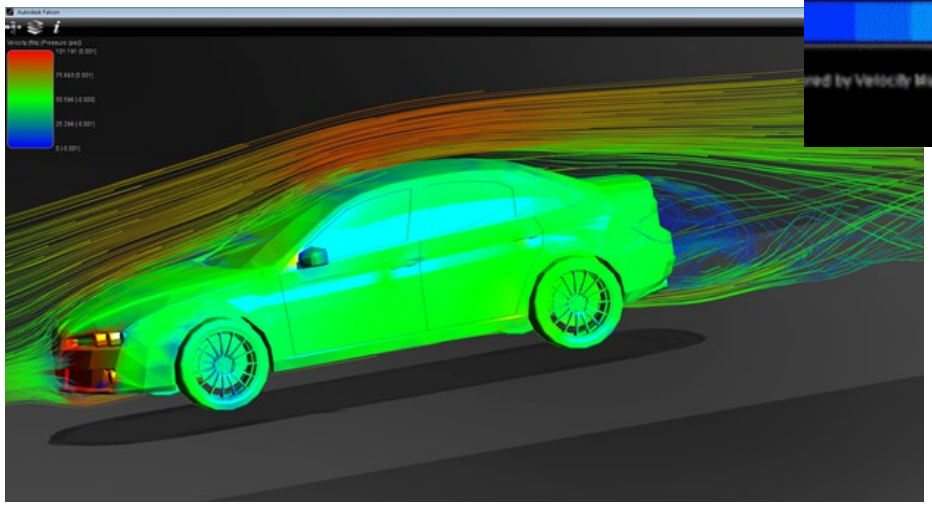


http://physbam.stanford.edu/~fedkiw/

Simulation d'écoulement

Simulation d'aérodynamisme de véhicules, de résistance au vent, de propagation de polluants, etc.





3. API physiques

API (Application Programming Interface) = bibliothèque de code.

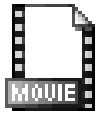
Il existe de nombreux moteurs physiques sous la forme d'API dans différents langages (C++, Java, C#, Python, JavaScript, ...), pouvant être utilisés dans des programmes pour calculer des animations de manière physique en 2D ou 3D.

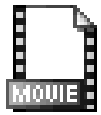
http://physxinfo.com/index.php?p=wrp

HAVOK (Microsoft)
http://www.havok.com

Simulation de solides, de tissus, de destructions.

Utilisé dans plus de 600 jeux sur PC, PS4, PS3, PSP, Wii, Switch, XBOX, XBOX 360, XBOX One.
(The Legend of Zelda: Breath of the Wild, Assassin's Creed, Bioshock, Company of Heroes, Fable 2, Fallout 3, FEAR, Halo 3, Spore…)
https://en.wikipedia.org/wiki/List_of_games_using_Havok





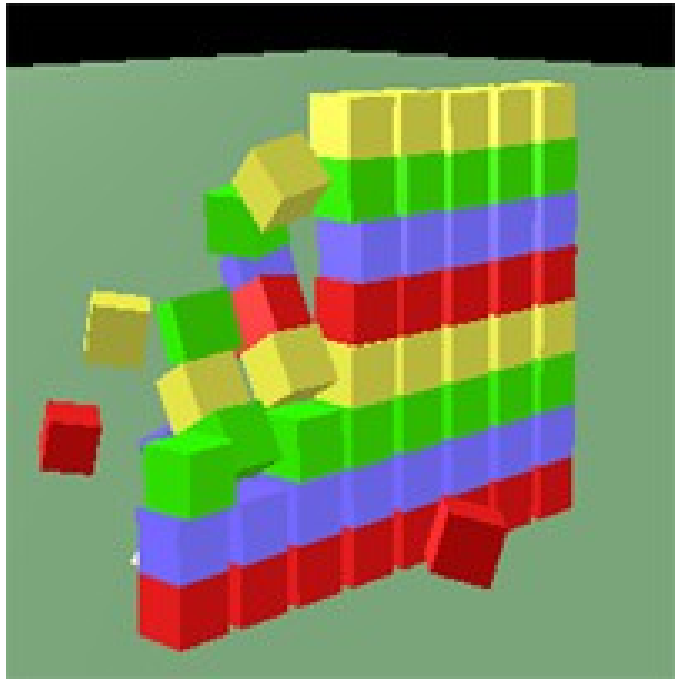
Unity
https://unity3d.com/fr

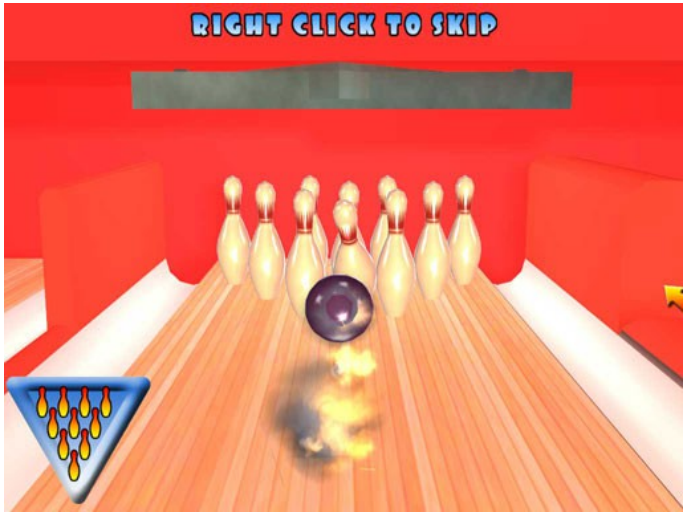
NVIDIA PhysX
http://www.nvidia.com/object/physx_new.html

TOKAMAK (Gratuit)
http://www.tokamakphysics.com/

ODE (Open Dynamics Engine) (Gratuit)
http://www.ode.org

Bullet Physics (Gratuit)
http://bulletphysics.org

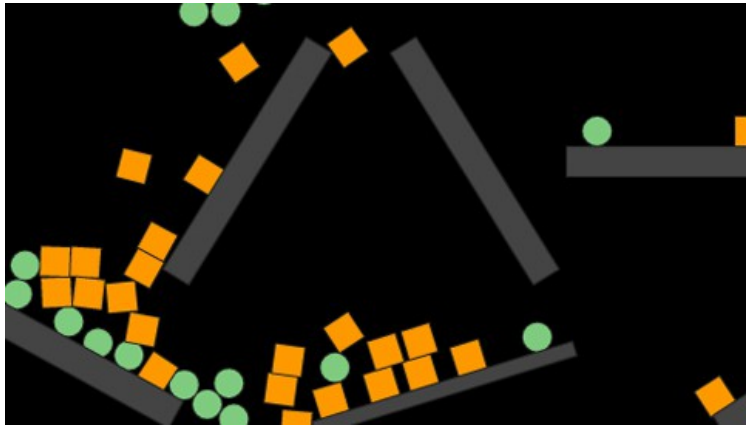


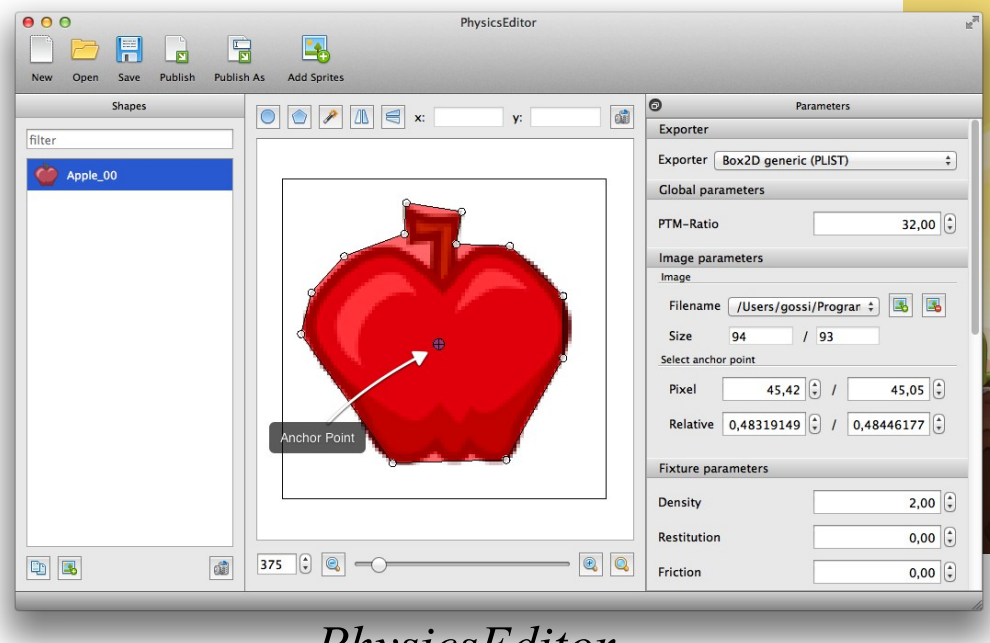


Box2D

http://box2d.org

Moteur physique open source (C++, javascript, java, C#, ...) permettant de simuler la mécanique des solides rigides en 2D.





PhysicsEditor

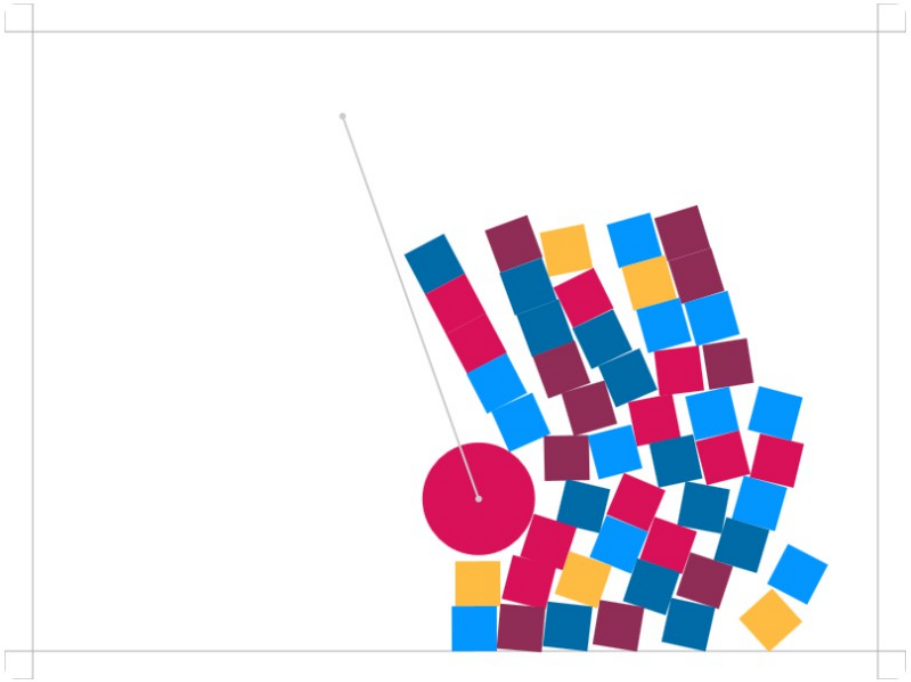
Angry Birds

MOVIE

Animation physique dans un navigateur web

PhysicsJS

http://wellcaffeinated.net/PhysicsJS
Moteur physique 2D en JavaScript.



Matter.js

http://brm.io/matter-js/
Moteur physique 2D en JavaScript.

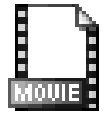
Ammo.js

https://github.com/kripken/ammo.js/
Moteur physique 3D en JavaScript,
portage de Bullet Physics.
https://threejs.org/examples/?q=phys#webgl_physics_rope

Euphoria

https://en.wikipedia.org/wiki/Euphoria_(software)

Module complémentaire au moteur physique d'un jeu, permettant de générer des animations biomécaniques 3D (personnage, animaux) par Synthèse de Mouvement Dynamique (*Dynamic Motion Synthesis*) en tenant compte des lois appliquées par le moteur physique ainsi que de l'I.A et de l'environnement du jeu afin de générer des animations réalistes et cohérentes en toutes circonstances.



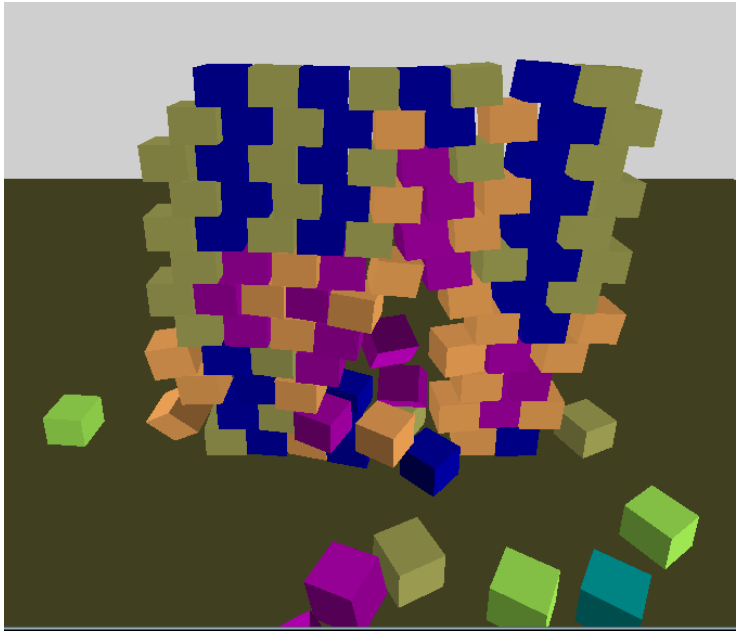
Red Dead Redemption 2 (Rockstar, 2018)

Bullet Physics

http://bulletphysics.org

Moteur physique open source, libre pour une utilisation commerciale.

Utilisé dans plusieurs jeux sur PS4, PS3, Xbox One, Xbox 360, Nintendo Wii, PC, Android, iPhone. Elle est aussi utilisée dans plusieurs logiciels de synthèse d'images comme Maya, Houdini, Cinema 4D, Lightwave, Blender.



Bibliothèque pouvant être utilisée dans un programme en C++.

Moteur phyique permettant de simuler la dynamique de solides rigides, mais aussi de solides déformables tels que des tissus ou des cordes.

La détection de collisions utilise des primitives basiques (sphères, boites, cylindres, etc.) mais aussi des enveloppes concaves et convexes.

Bullet ne se charge que du **calcul** du positionnement d'objets par simulation physique, c'est à vous ensuite d'**afficher** ces objets en 3D, par exemple avec OpenGL.

Utilisation de Bullet

- Création d'un monde dynamique (**btDiscreteDynamicsWorld**)

- Définition d'une gravité (**setGravity**)

- Définition d'enveloppes de collision (**btCollisionShape**) servant pour la détection de collision (la véritable forme 3D de l'objet doit être affichée par vous, par exemple avec OpenGL) : btSphereShape, btBoxShape, btCylinderShape, btCapsuleShape, btMultiSphereShape, btStaticPlaneShape, btConvexHullShape, btHeightfieldTerrainShape, etc.

- Définition de solides rigides (**btRigidBody**) auxquels on associe une enveloppe de collision, une position, une masse (une masse de 0 correspond à un objet fixe, ne pouvant être déplacé), des propriétés de matériau (friction, restitution), …

- Ajout d'un solide rigide au monde dynamique (**addRigidBody**)

- Broadphase : utilisation de structure (btDbvtBroadphase, btAxisSweep3, btCudaBroadphase, …) permettant d'accélérer la détection de collision en éliminant rapidement des paires d'objets en fonction de leur volume englobant.

```
btBroadphaseInterface* broadphase = new btDbvtBroadphase();
```

- Mise à jour de la simulation à chaque frame (**stepSimulation**)

- Récupération du positionnement des objets donnés par la simulation, pour le répercuter sur les objets 3D correspondants pour l'affichage (**getMotionState**, **getWorldTransform**).

Pour récupérer à la fois la position et l'orientation sous la forme d'une matrice OpenGL, on peut procéder comme suit :

```
btTransform trans;
monRigidBody->getMotionState()->getWorldTransform(trans);
btScalar m[16];
trans.getOpenGLMatrix(m);

glPushMatrix();
glMultMatrixf((GLfloat*)m);

// Afficher ici l'objet 3D
// ...

glPopMatrix();
```

http://bulletphysics.org/mediawiki-1.5.8/index.php/Hello_World