GMIN 317 – Moteur de Jeux

Animation temps réel

Rémi Ronfard

Remi.ronfard@inria.fr

https://team.inria.fr/imagine/remi-ronfard/

Ce cours est largement inspiré des cours de ***Marc Moulis et Benoit Lange***.

Le but de cette présentation est de fournir une vue globale des dix séances de cours et TP et de présenter les choix d'études documentaires et projets.

Plan du cours

Chemins et trajectoires

Animation de modèles

Animation réaliste

Animation faciale

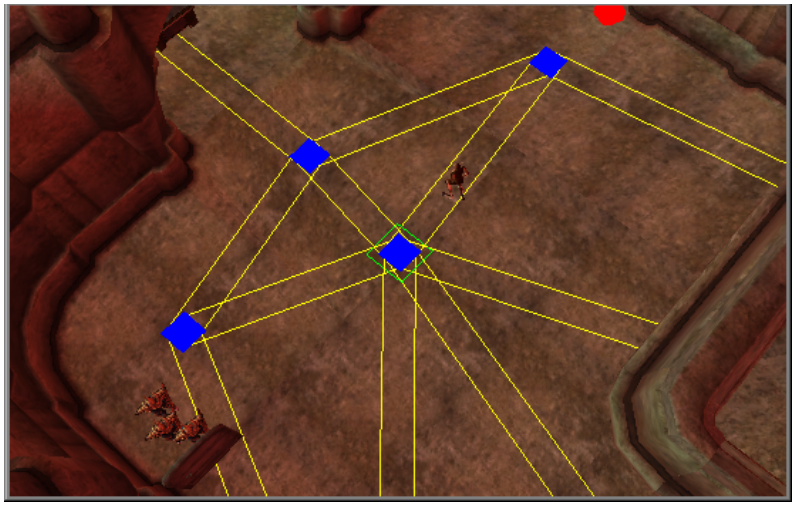
Animation de caméras

Chemins et trajectoires

La manière la plus simple de représenter un chemin de déplacement en 3D est de définir une série de points (ou nœuds) dans l'espace, reliés 2 à 2 par des segments. On produit ainsi une ligne brisée, qui représente notre chemin. Selon les besoins, les segments peuvent être orientés, et ainsi définir un point de départ et un point d'arrivée.

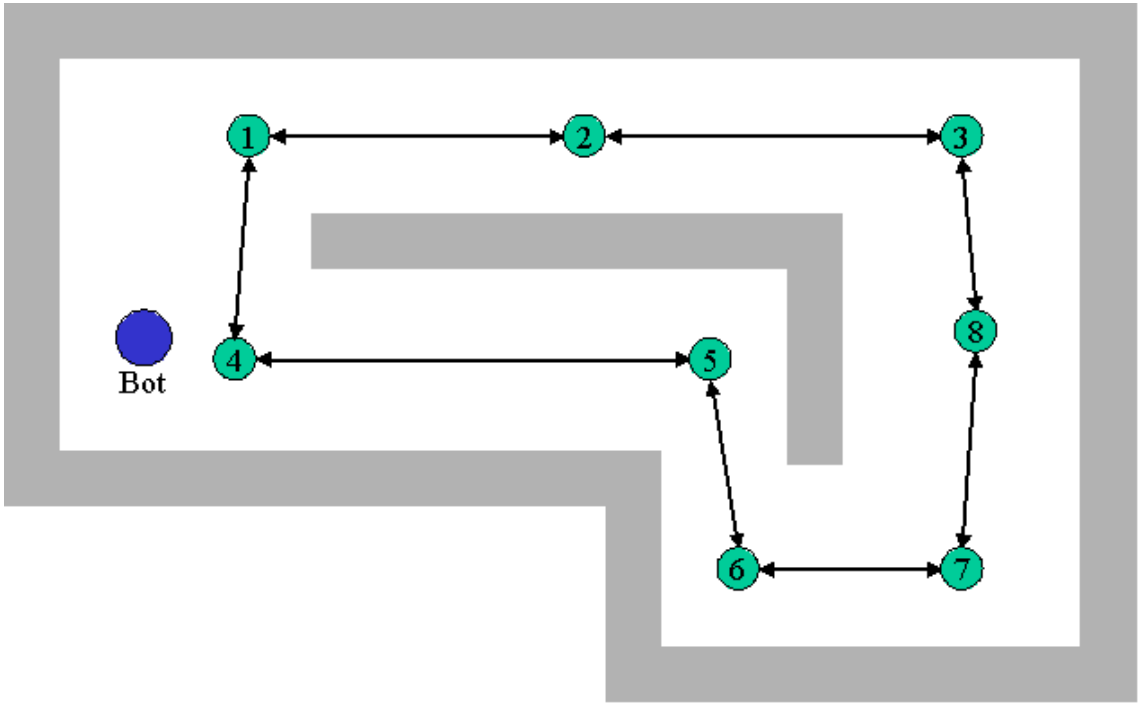
Lorsque un nœud est connecté à plus de 2 voisins, on obtient un graphe de déplacement.

Ligne brisée représentant un chemin



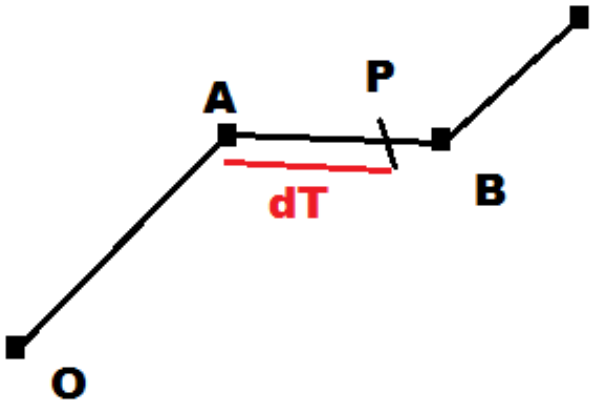
Exemple de graphe de déplacement : chaque nœud est connecté à un ou plusieurs voisins

Ce type de représentation peut par exemple être utilisé en recherche de chemin, en explorant de proche en proche tous les nœuds d'un graphe, et ainsi déterminer une trajectoire à suivre dans l'environnement (waypoint).



Exemple de chemin placé dans un environnement. Les nœuds du chemin servent de repères à l'entité lors de la planification de ses déplacements.

Lignes polygonales



Lignes polygonales

- Le gros problème de l'utilisation d'une simple ligne brisée est que la trajectoire associée est souvent non-naturelle, avec des changements de direction brutaux au niveau des nœuds.
- On pourrait imaginer raffiner énormément la ligne afin d'avoir un grand nombre de nœuds et ainsi l'adoucir, mais le travail d'édition serait fastidieux et le stockage mémoire important.
- On choisira donc de préférence une méthode plus adaptée de génération d'un chemin courbe, en utilisant les splines.

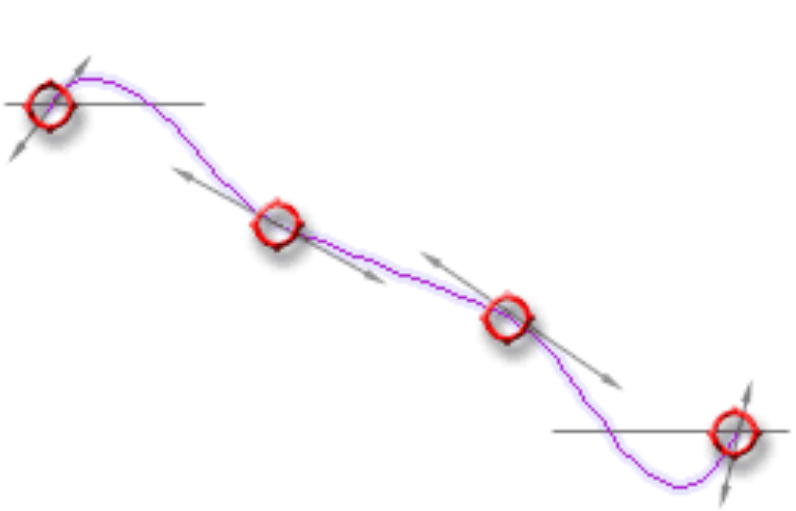
Splines

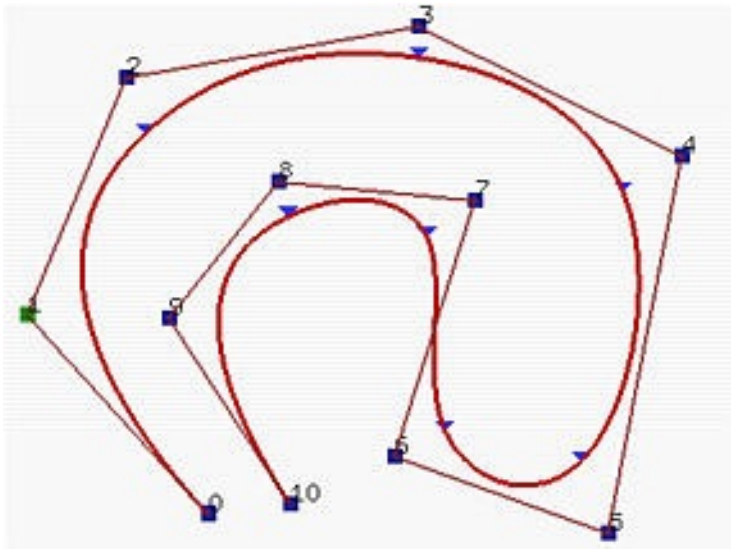
- Qu'est ce qu'une spline ? C'est une fonction définie par morceaux, chaque morceau étant décrit par un polynôme.
- Par ce moyen on représente une courbe, en spécifiant une série de points (appelés points de contrôle) à des intervalles réguliers le long de la courbe, et en définissant une fonction qui permet de calculer tous les points entre les points de contrôle.
- Il existe plusieurs types de splines, avec des propriétés différentes:
  - Courbes de Bézier, B-Splines, Catmull-Rom, Cardinal, Hermite
  - NURBS (Non-Uniform Rational B-Splines)

On différenciera tout particulièrement 2 catégories de splines:

- Splines interpolantes (la courbe passe par les points de contrôle)
- Splines approximantes (la courbe ne passe pas par les points de contrôle)

Remarque: certaines splines sont quasi-interpolantes (la courbe passe par certains des points de contrôle)





Exemple de spline interpolante (gauche) et de spline approximante (droite).

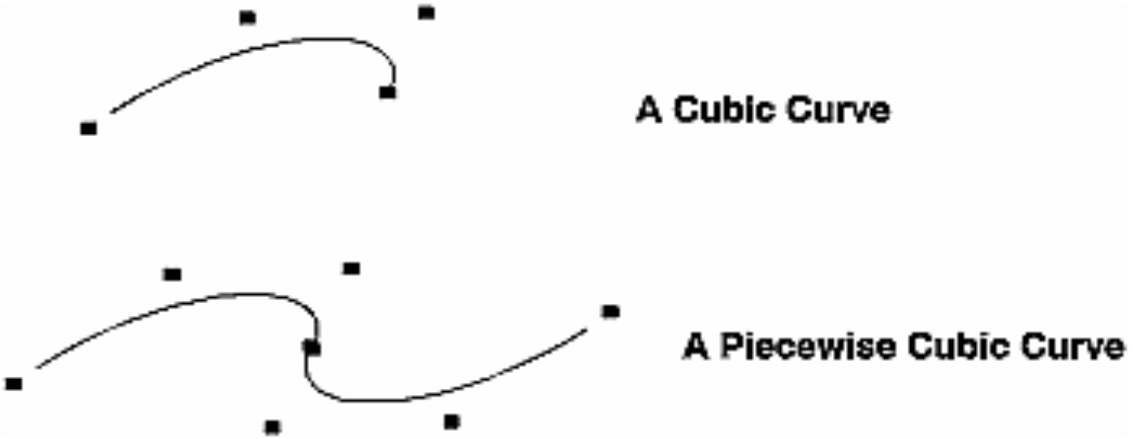
Points de contrôle et degrés

Le degré d'une courbe détermine le nombre de points de contrôle nécessaires pour définir la courbe:

degré 2 (quadratique) : 3 points de contrôle

degré 3 (cubique) : 4 points de contrôle.

On peut construire une courbe plus longue de même ordre en rattachant des morceaux de courbe.

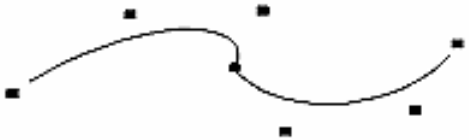


Le degré d'une courbe affecte également le comportement de la courbe lorsqu'un des points de contrôle est déplacé. En général, les courbes de trop haut degré sont sujettes à des effets oscillatoires indésirables.

Continuité d'une courbe

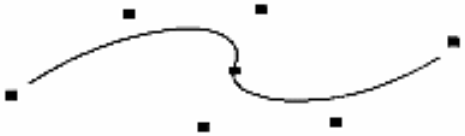
No Continuity

On parle de continuité Cn d'une courbe pour exprimer le fait que les n premières dérivées des morceaux de courbe consécutifs sont égales au niveau des points de jonction entre les morceaux.



C0 Continuity (positional)

Continuité C0 : la courbe est continue uniquement pour les positions



C1 Continuity (tangential)

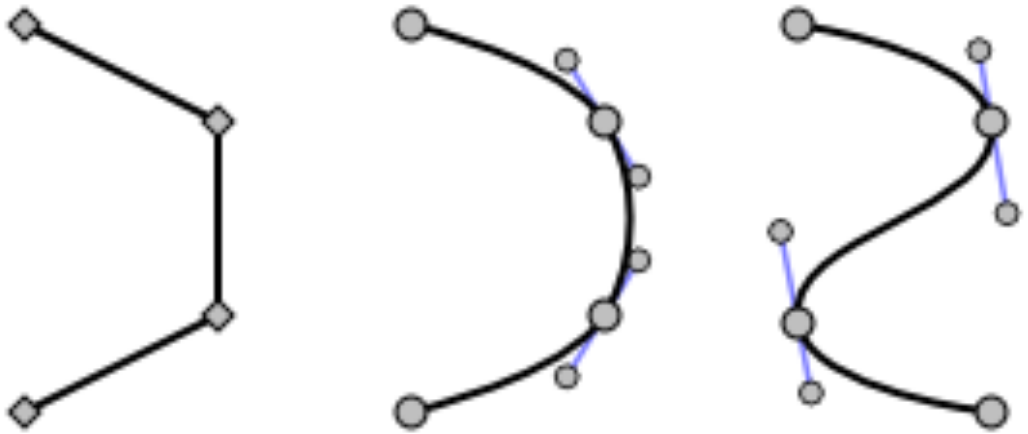
Continuité C1 : les tangentes à la courbe (dérivées premières) sont constantes au niveau des points de jonction

C2 Continuity (curvature)

Continuité C2 : la courbure des courbes (dérivées secondes) est constante au niveau des points de jonction

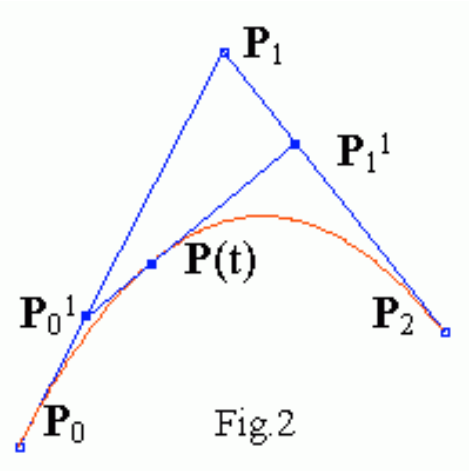
Le principe d'édition d'une spline est d'associer à chacun des points de contrôle une tangente à la courbe, qui déterminera donc de quelle manière la courbe doit s'orienter dans le voisinage local au point de contrôle.

Techniquement, la tangente à un point de contrôle est souvent calculée en utilisant la position des points de contrôle précédent et suivant.

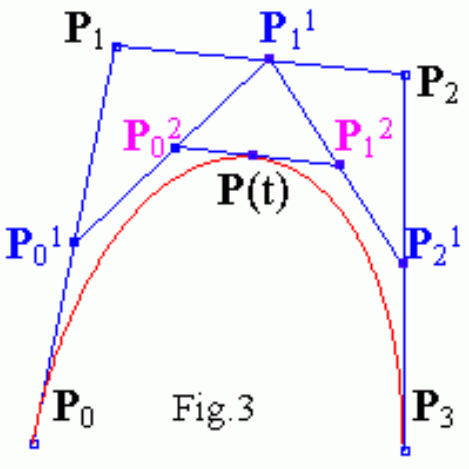


Génération d'une spline (interpolante) à partir d'un maillage de points de contrôle et des tangentes associées.

Splines de Bézier quadratiques



Bézier quadratique



Bézier cubique

On les obtient par l'algorithme de De Casteljau, en réalisant une interpolation linéaire des interpolations linéaires entre les points de contrôle P0, P1 et P2.

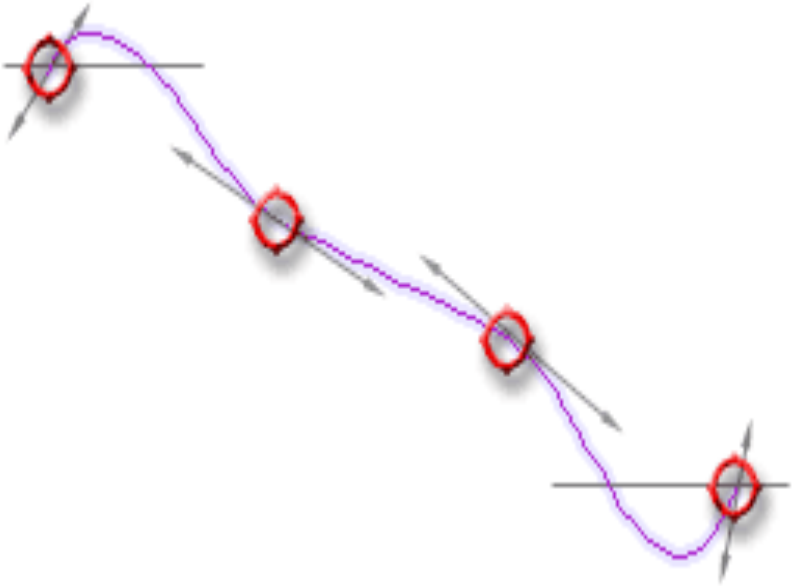
$$P01 = (1-t)P0 + tP1$$
$$P11 = (1-t)P1 + tP2$$
$$P(t) = (1-t)P01 + tP11$$

Splines de Bézier cubiques

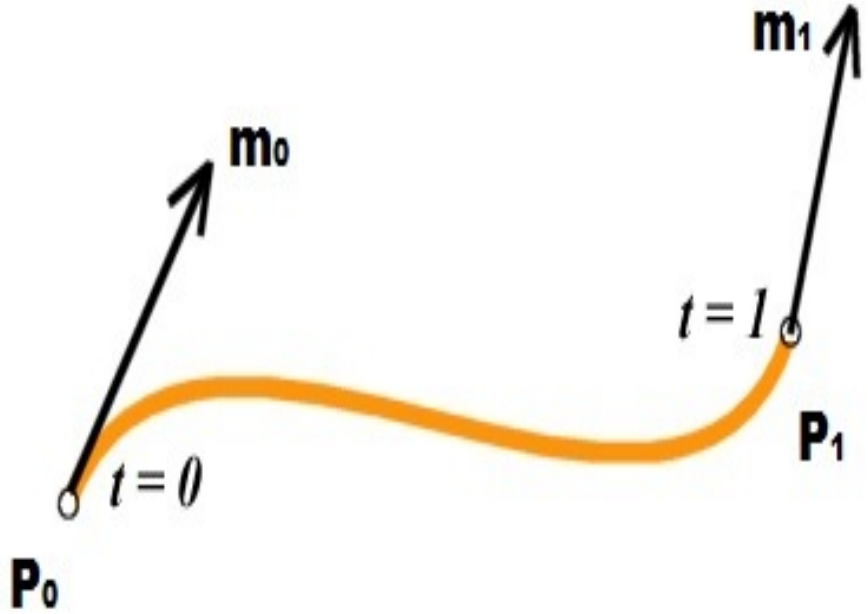
De la même manière, il est possible d'interpoler une courbe entre (n+1) points de contrôle. Si n=3, on obtient une spline de Bézier cubique.

Les splines de Bézier ont une continuité C1.

Splines de Catmull-Rom et Hermite

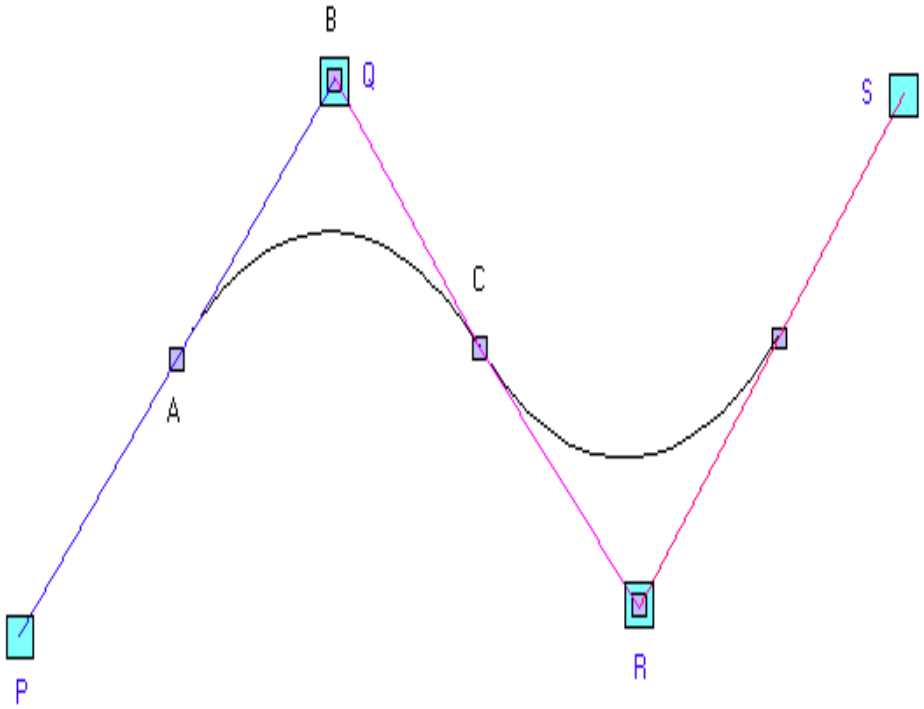


Catmull-Rom

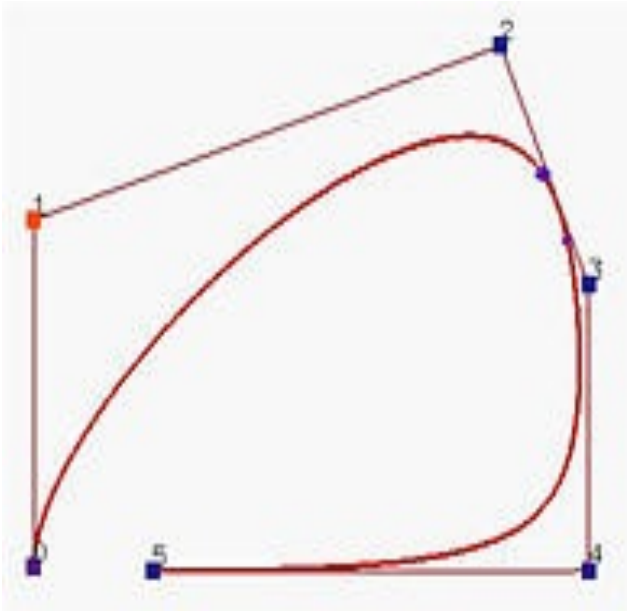


Hermite

B-Splines et NURBS



B-spline

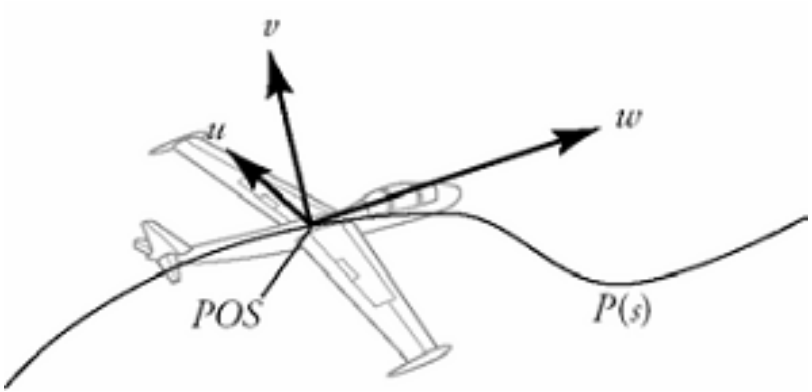


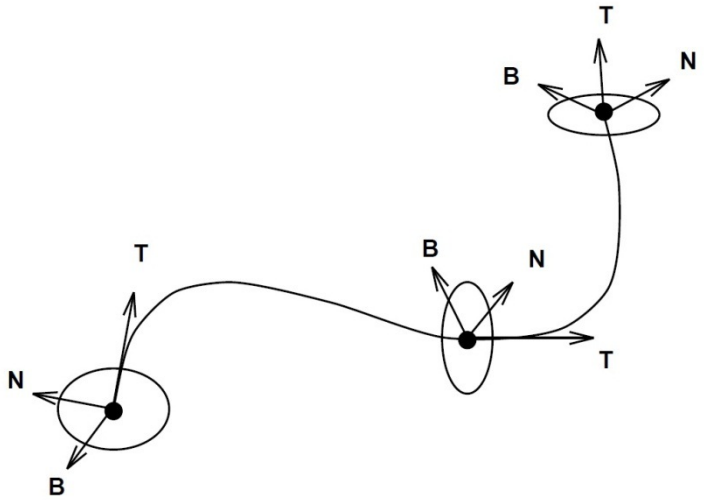
NURBS

Repère de Frenet

Une fois le chemin déterminé, il faut être capable d'orienter l'objet qui va se déplacer le long de ce chemin.

Une méthode permettant de déterminer automatiquement une telle orientation est le calcul du repère de Frenet.





Visualisation du repère de Frenet le long de la courbe

Calcul du repère de Frenet

$$\begin{aligned}
\vec{\mathbf{T}}(s) &= \frac{\vec{\mathbf{x}}'(s)}{\|\vec{\mathbf{x}}'(s)\|} \\
\vec{\mathbf{B}}(s) &= \frac{\vec{\mathbf{x}}'(s) \times \vec{\mathbf{x}}''(s)}{\|\vec{\mathbf{x}}'(s) \times \vec{\mathbf{x}}''(s)\|} \\
\vec{\mathbf{N}}(s) &= \vec{\mathbf{B}}(s) \times \vec{\mathbf{T}}(s) \ .
\end{aligned}$$

En pratique, on calcule rarement le véritable repère de Frenet lorsque l'on connaît l'orientation globale de l'un des axes B ou N, ce qui peut simplifier les calculs (on évite le calcul de la dérivée seconde).

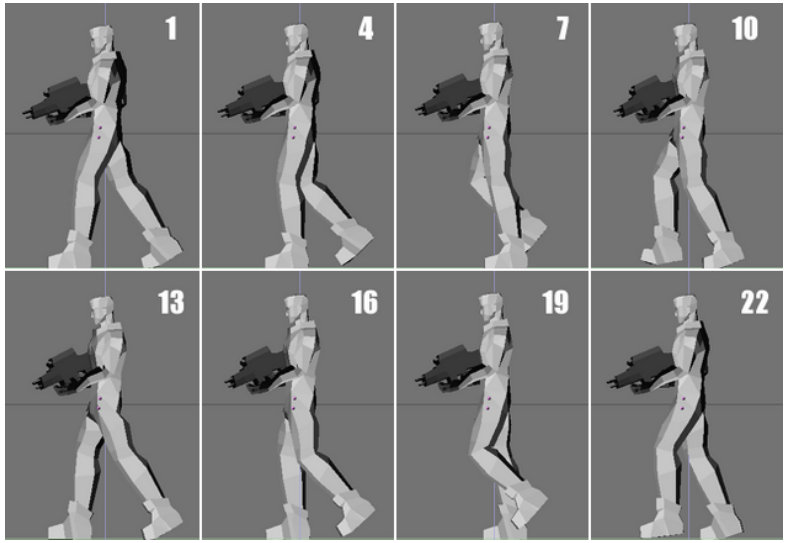
Exemples typiques : caméra en mode « poursuite », connaissance de l'orientation de la « verticale » de l'objet, etc…

Dans ce cas là, on détermine l'axe T (soit à partir du vecteur déplacement le long de la courbe (identique au repère de Frenet), soit en orientant vers une position cible), on fixe l'orientation d'un des deux autres axes (par exemple B dans le sens de la verticale du monde), et on déduit N par simple produit vectoriel entre B et T.

Animation de modèles : vertex tweening

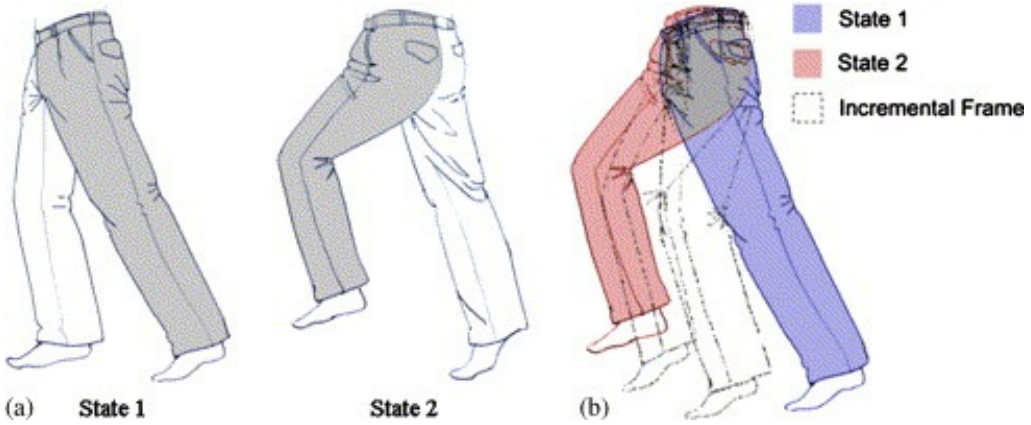
La méthode d'animation la plus basique s'inspire directement des méthodes d'animation 2D traditionnelles.

Pour chacune des images de l'animation, on crée un maillage 3D qui représente l'étape de l'animation.



Vertex tweening

Pour produire l'animation, les positions des vertices de la géométrie sont linéairement interpolés 2 à 2 (vertex tweening).



Cette technique est simple à mettre en œuvre, mais présente de nombreux défauts:

- Un maillage complet doit exister pour chacune des poses de l'animation, ce qui est coûteux en mémoire
- Si les étapes de l'animation sont trop éloignées les unes des autres, des déformations du maillage peuvent devenir visibles, dues à l'interpolation linéaire des positions des vertices
- Les maillages d'une animation doivent être strictement identiques en termes d'organisation des vertices/faces
- Si différents modèles géométriques ont la même animation, les maillages doivent pourtant être dupliqués et animés séparément.

Rigging et skinning

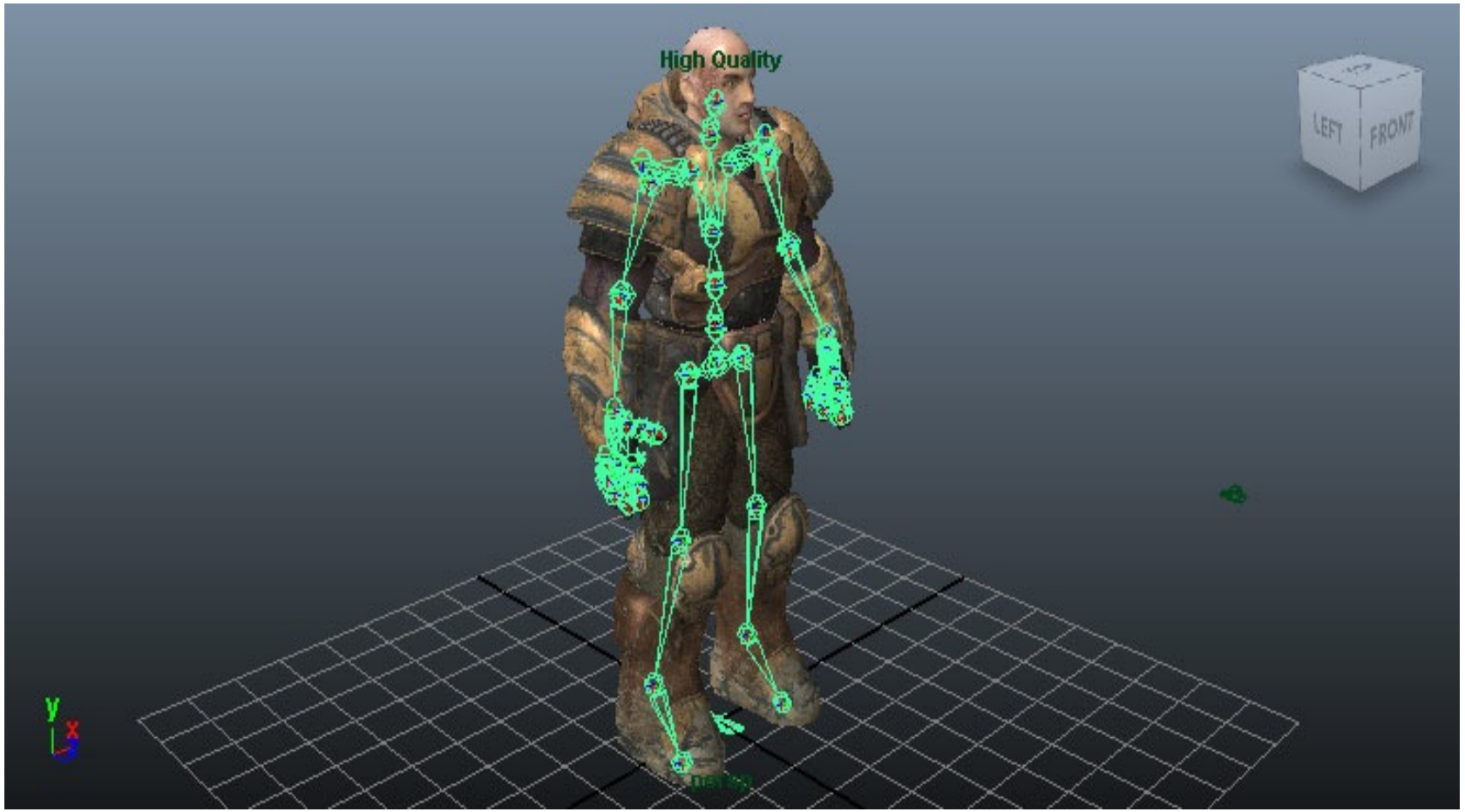
- Pour supprimer les inconvénients du vertex tweening, on utilise une structure de données permettant de séparer les données d'animation du maillage 3D animé.

  L'animation sera appliquée sur une hiérarchie de transformations, qui composent le squelette de l'objet à animer

  Un modèle géométrique séparé, la skin, servira d'enveloppe au squelette et sera déformé par celui-ci

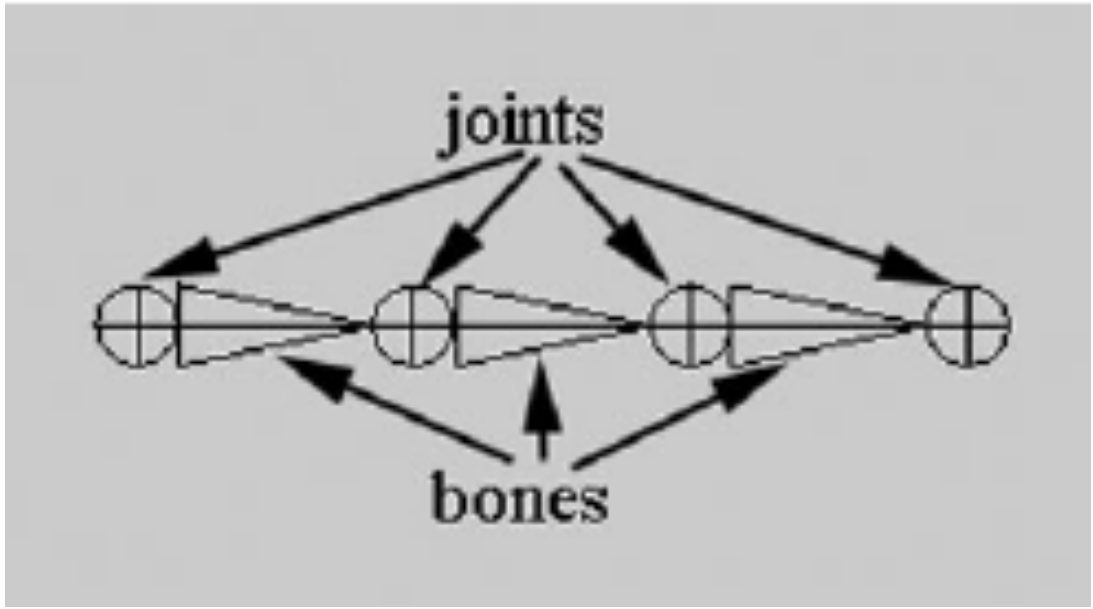
Rigging & skinning

Le squelette d'animation est composé d'une hiérarchie de transformations (matrices 4x4). Chaque nœud de la hiérarchie représente une articulation. Le squelette d'animation est également appelé « rig ». Sa racine est habituellement située au niveau des hanches.



Armatures

Chaque nœud (ou paire de nœuds) de transformation de la hiérarchie est traditionnellement appelé « bone », par analogie avec les os constituant un squelette.



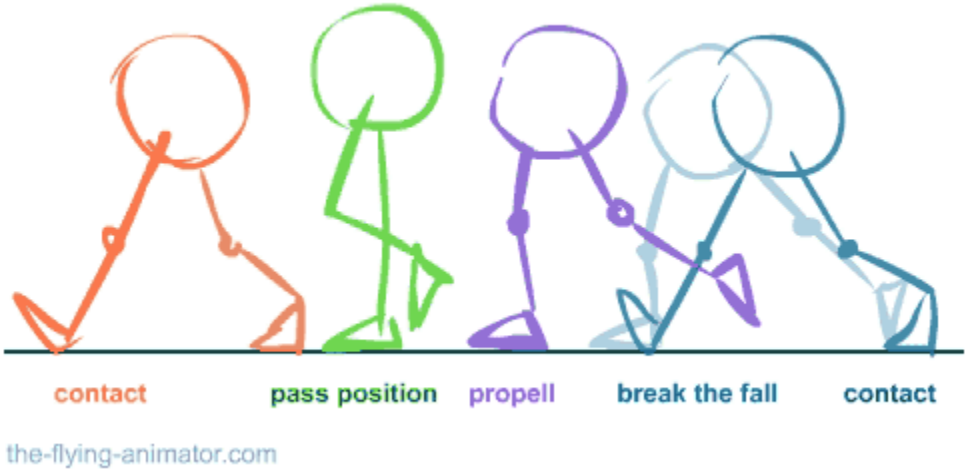
Deux représentations possibles pour les bones:

- Un bone est représenté par la liaison entre deux joints (nœuds de la hiérarchie)
- Un bone est associé à un nœud de transformation de la hiérarchie (le bone et le joint sont indifférenciés)

Animation par keyframes

Les animations du squelette sont stockées sous la forme de keyframes, série de poses pour chacun des nœuds du squelette:

- Vecteur 3D pour le stockage de la translation
- Quaternion pour le stockage de la rotation



Le format d'animation résultant est donc très compact (7 floats stockent l'équivalent d'une matrice 4x3 rotation/translation).

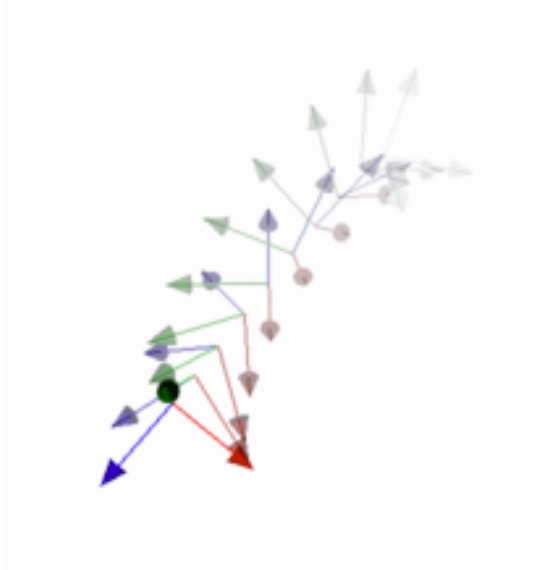
Les keyframes sont pré-calculées à partir:

- des données produites par les animateurs
- des données issues de la motion capture

Interpolation

L'animation entre les différentes keyframes se fait par interpolation
- Généralement linéaire (LERP) pour la translation
- Sphérique pour l'orientation (SLERP sur les quaternions)

Animation du squelette par interpolation entre les poses



Interpolation sphérique d'un quaternion

Skinning

- Une fois le squelette d'animation construit, on va lui associer un ou plusieurs modèles géométriques qui vont lui servir d'enveloppe, enveloppe qui sera déformée lors de l'animation.
- Historiquement, trois techniques existent:
  - Rigid bones
  - Hard skinning
  - Soft skinning

Rigid bones

Les enveloppes du type « rigid bones » sont les modèles les plus simples. Le principe est d'associer, à chacun des bones du squelette, un modèle géométrique différent. Chaque nœud de transformation du squelette est donc associé à un maillage indépendant et un seul.

Aucune déformation (autre que la transformation du bone dans l'espace) n'est donc appliquée sur les vertices des maillages, ce qui en fait une méthode très rapide.

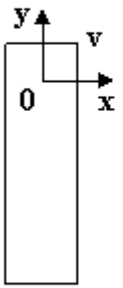
Cette transformation étant réalisée par le hardware (pipeline fixe), c'était la méthode d'animation privilégiée avant l'avènement des pipelines programmables.

Modèle « rigid bones ».

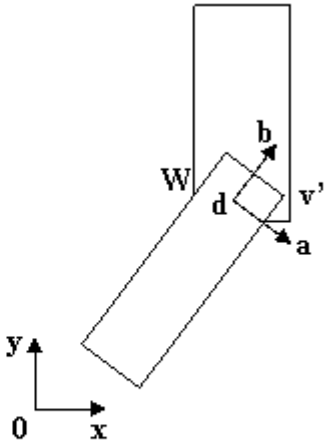
Artefacts

Le problème de cette technologie est que, lors de l'animation, des artefacts visuels apparaissent dans le modèle:

- Interpénétration des différents bones
- Apparition d'espaces disjoints (cracks) entre les bones

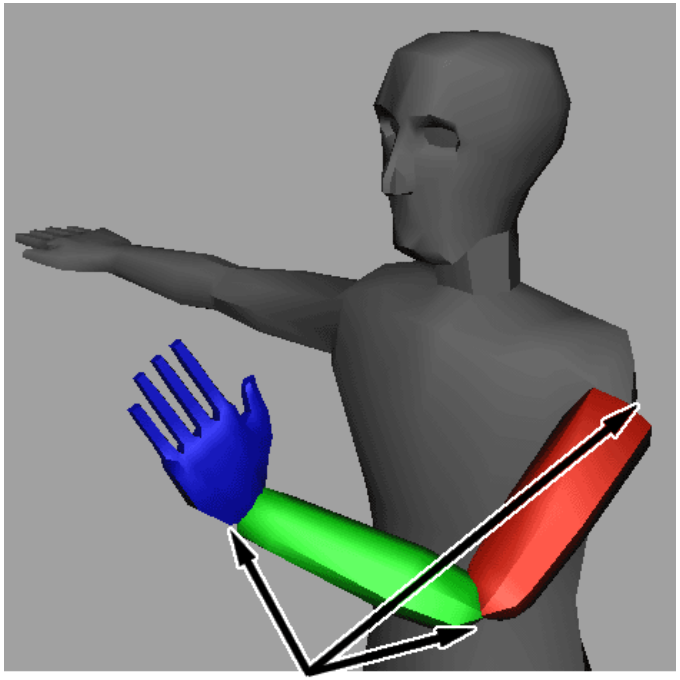


Vertex **v** defined in joint's local space



Vertex **v** transformed to world space using matrix **W**

Animation de type « rigid bones ». Des artefacts apparaissent au niveau des jointures.

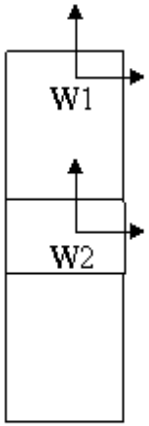


Unnatural cracks at the joints

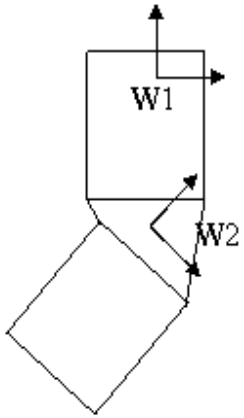
Skinning

On introduit le concept de « skin »: un seul maillage pour l'ensemble de l'enveloppe. A chacun des vertex de la skin, on associe en méta-donnée l'index du bone dans le squelette qui sera utilisé pour la transformation lors de l'animation (hard skinning).

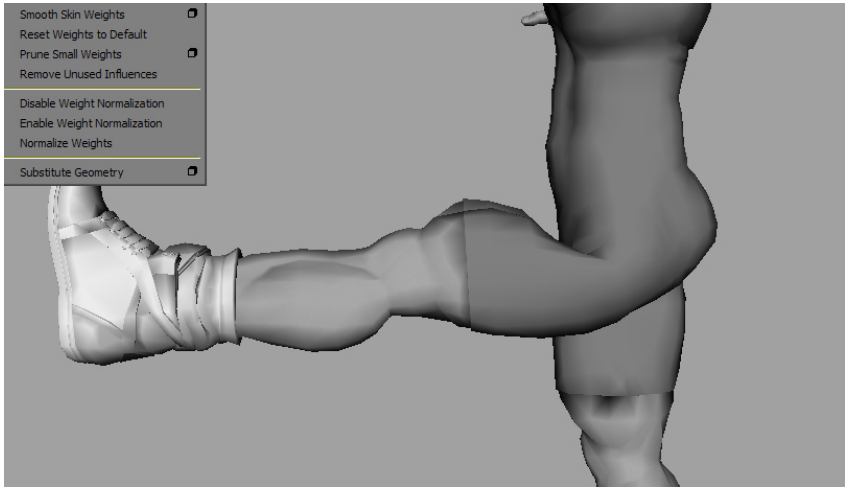
Certains défauts notables sont toujours présents. La présence de ces défauts est due au fait que les vertices ne sont attachés qu'à un seul bone.



An unbent knee with skin attached to joints 1 and 2



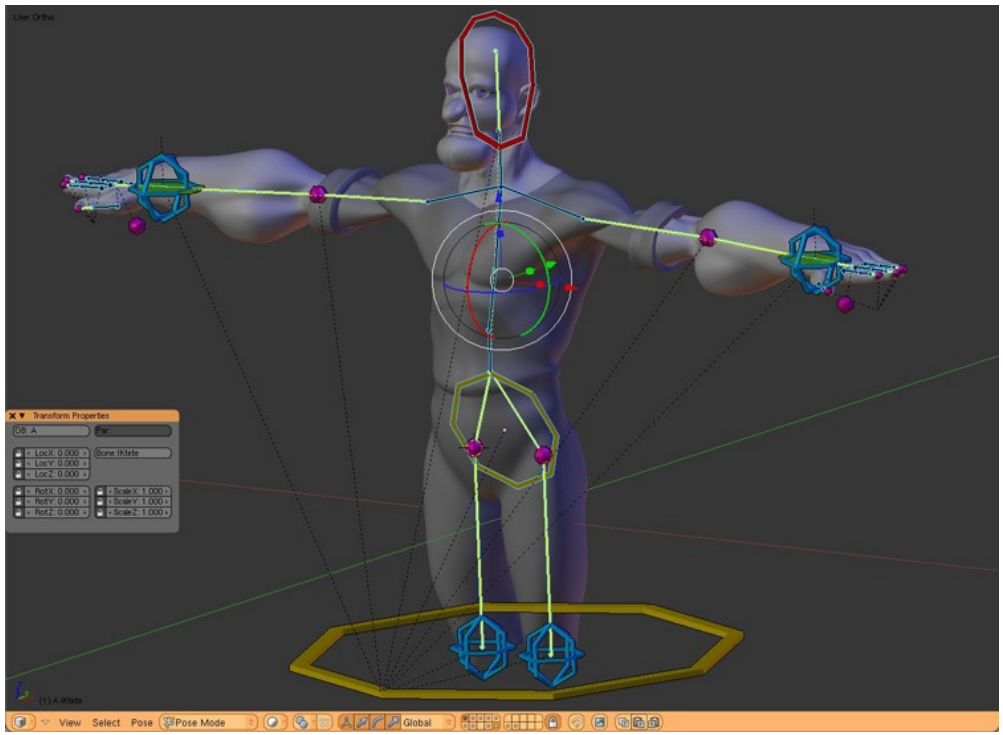
Every vertex is attached to exactly one joint, so as the knee bends, we get some distortion



Déformation à la jointure de la cuisse avec la hanche.

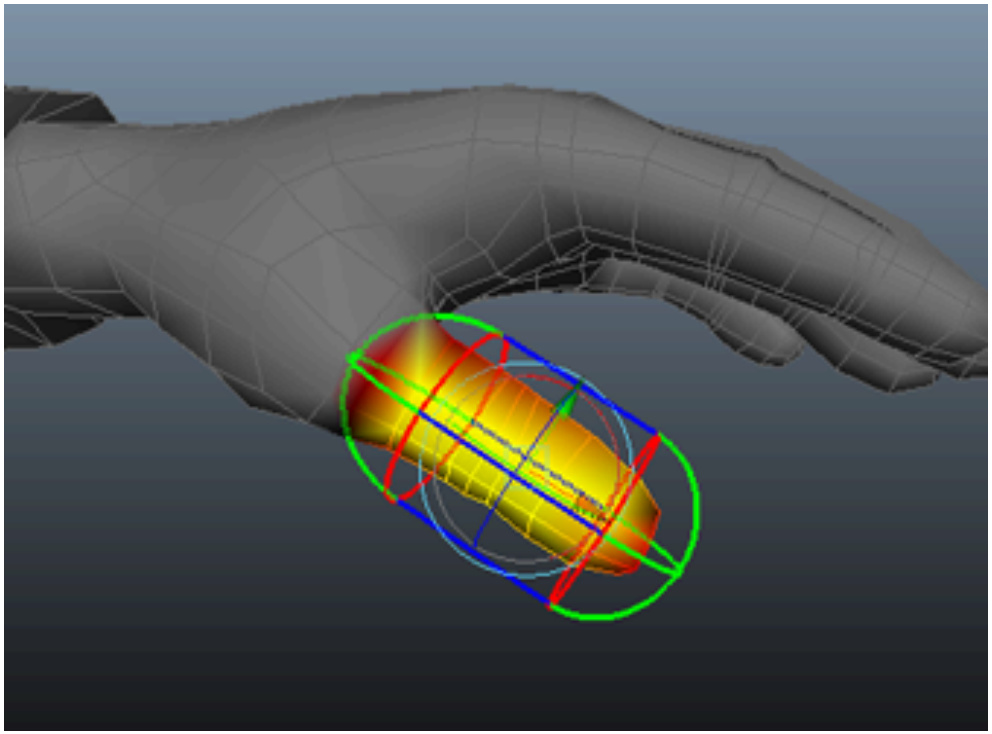
Skinning

L'animateur commence par placer sa skin dans un état initial connu (pour un humanoïde, généralement debout avec les bras à l'horizontale : « T-pose »). Cette étape va permettre d'initialiser les matrices de transformation du squelette dans le même espace que celui de la skin (skin binding).



Skin blending

Puis, à l'aide d'outils spécifiques, l'infographiste va peindre directement sur le modèle les influences des différents bones du squelette.



Affectation des poids de skinning.

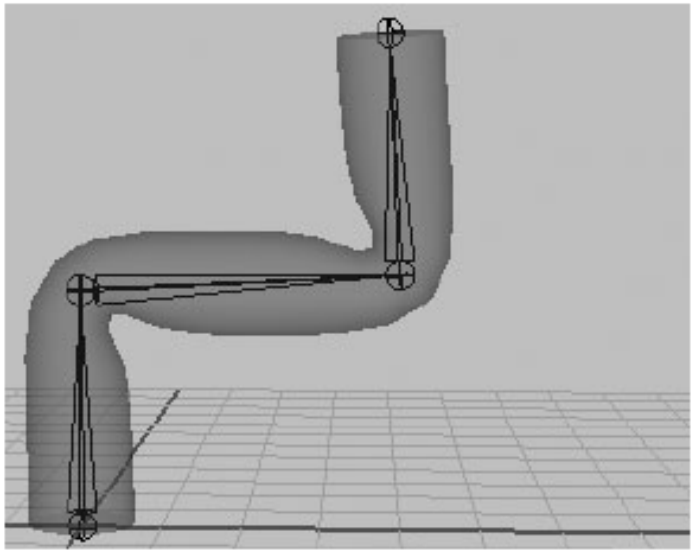
Smooth skinning

La skin est exprimée dans le repère global (centré)

- Les transformations du squelette d'animation sont exprimées dans le repère global (centré)

On fournit au moteur de rendu toutes les matrices de transformations

- On stocke pour chaque vertex de la skin les indices des matrices dans la palette, et les poids de skinning (également appelés Beta-weights)
- Les influences des bones sont combinées pour chacun des vertex de la skin dans le vertex shader



Déformation d'un maillage par smooth skinning

Linear blending (smooth skinning)

Chaque matrice de skinning est de la forme

$$Ms = MB * MI\text{-}1$$

avec:

- Ms : Matrice de skinning de la palette
- MB : Matrice du bone (animé) dans le repère global centré
- MI-1 : Inverse de la matrice de pose du bone

L'application des poids de skinning se fait par moyenne pondérée des positions du vertex obtenues après transformation par les matrices de skinning associées.

$$P = \sum i\ \beta i * Ms\ i * P0$$

avec:

- P : Position du vertex après skinning
- βi : Influence du ième bone du vertex
- Ms i : Index de la ième matrice de skinning
- P0 : Position originale du vertex dans la skin

Afin de réduire les transferts de données sur le bus, les poids de skinning sont normalisés (leur somme vaut 1), ce qui rend le dernier poids implicite (il n'est donc pas transmis).

Vertex tweening vs. Soft skinning

Vertex tweening	Soft skinning
Un maillage par pose	Keyframing du squelette, un seul maillage pour toutes les animations
Déformations dues à l'interpolation linéaire	Pas de déformations
Dupliquer l'animation pour chaque modèle différent	Un seul squelette d'animation convient pour plusieurs skins différentes
Combinaison d'animations coûteuse (ensemble des vertices de la géométrie)	Combinaison d'animations simplifiée (combinaison des squelettes)
Sous-animations complexes à mettre en œuvre	Sous-animations très simples à mettre en œuvre

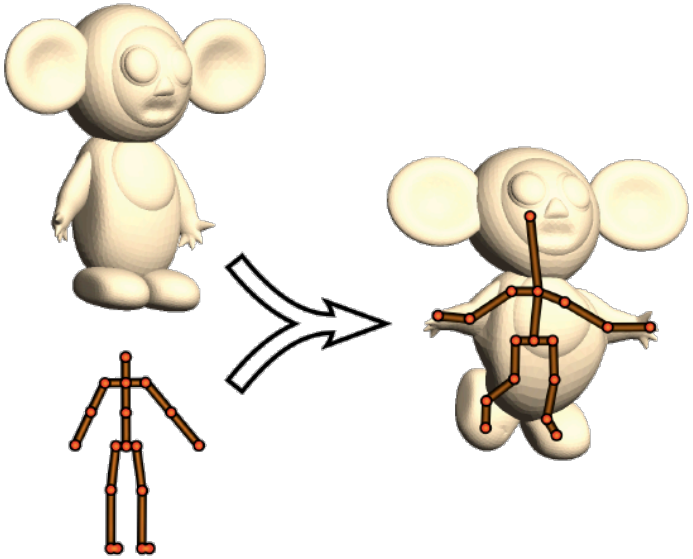
Dual quaternion skinning

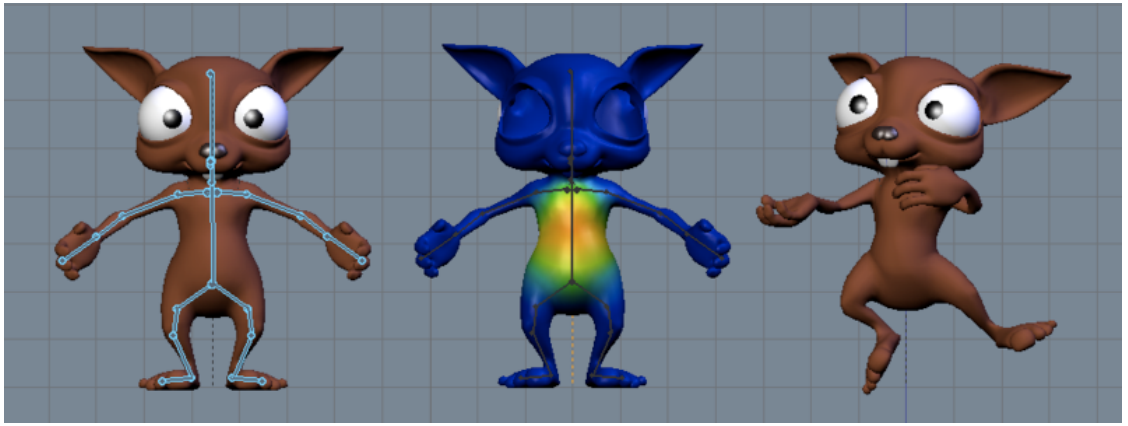
- Linear blend
  - $P(v_i)= (sum\_f\ w_{if}\ T_f\ )P_0(v_i)$ où les poids w_{if} sont normalisés ($sum\_f\ w_{if} = 1$)
- Non-linear blend
  - $P(v_i)= B_i\ (w; T_1 ,\dots ,T_f ,\dots ,T_N\ )P_0(v_i)$
- The blending should be coordinate invariant; always returns a valid rigid transformation; and interpolate two rigid transformations along the shortest path
- Dual quaternion blending

$$DLB(\mathbf{w}; \hat{\mathbf{q_1}}, \dots, \hat{\mathbf{q_N}}) = \frac{w_1\hat{\mathbf{q_1}} + \dots + w_N\hat{\mathbf{q_N}}}{||w_1\hat{\mathbf{q_1}} + \dots + w_N\hat{\mathbf{q_N}}||}$$

Automatic rigging and skinning

- Automatic Rigging and Animation of 3D Characters
  Ilya Baran and Jovan Popovic, SIGGRAPH 2007.





Animation réaliste

- Intégrer l'animation d'un acteur de manière réaliste dans notre scène nécessite de pouvoir répondre aux questions suivantes:
  - Le maillage est certes animé, mais comment le déplacer de manière réaliste dans l'espace ?
  - Comment faire pour établir des transitions correctes entre les différentes animations ?
  - Comment limiter le nombre total d'animations produites par les infographistes ?
  - Comment synchroniser des évènements avec l'animation d'un acteur ?

Cinématique inverse

Comme il devient impossible de générer toutes les animations pouvant gérer correctement tous les cas de figure, on utilise la cinématique inverse (IK – Inverse kinematics) qui est une méthode itérative:

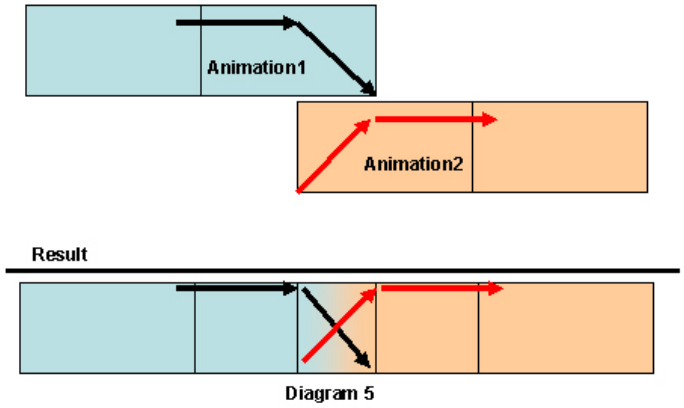
- On tire parti de la structure des squelettes d'animation, à savoir une chaîne de joints (articulations) et de bones.
- On définit, pour chaque articulation, des degrés de liberté (DOF – Degrees Of Freedom), qui fixent les limites en rotation (et éventuellement en translation)
- On applique l'animation traditionnelle, puis on va chercher à placer le bout de la chaîne (ex: le pied au bout de la jambe) sur une position cible

Pour chacune des articulations de la chaîne à partir de la source, on va réaliser une petite rotation (en respectant les DOF) qui va tendre à faire approcher le bout de la chaîne articulaire de la position destination

On itère les petites rotations jusqu'à ce que la position cible soit suffisamment proche ou que le système se soit stabilisé (pas de solution « exacte »).

La mise en place d'une procédure de cinétique inverse est difficile car il peut y avoir plusieurs solutions possibles (ou aucune). Néanmoins les résultats sont souvent très convaincants lorsque l'utilisation reste très localisée.

Blending d'animations



Blending d'animations

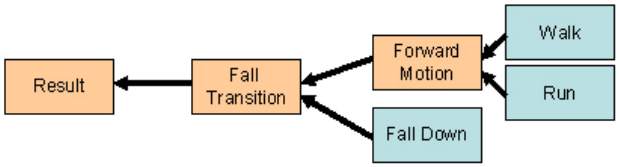
Comment gérer correctement les transitions entre deux animations (ex: marche/course/saut/chute), sachant que la transition peut intervenir à n'importe quel point dans l'animation ?

Comme vu précédemment avec l'animation procédurale, où l'on combine une animation traditionnelle avec un calcul de cinématique inverse, il est possible de mixer plusieurs animations entre elles.

Pour enchaîner correctement les animations, on va donc utiliser le même principe: le squelette d'animation va, le temps de la transition, être une moyenne pondérée des deux animations source et cible.

Blending d'animations

On peut pousser le concept un peu plus loin en mettant en place un gestionnaire d'animation capable de mixer plusieurs couches d'animation pour générer une animation complexe. La gestion de ces couches d'animation se fait généralement à partir d'une structure en arbre. Ces animations peuvent être locales (ex: le joueur recharge son arme pendant sa course). Ceci permet de réduire le nombre total d'animation que les infographistes vont devoir produire (les combinaisons de sous-animations au runtime permettant la génération automatique de nouvelles animations).




```
BlendLayer *forward_motion = new BlendLayer();
forward_motion->AddAnimation(WalkAnim, walk_weight);
forward_motion->AddAnimation(RunAnim, run_weight);

BlendLayer *fall_transition = new BlendLayer();
fall_transition->AddAnimation(FallAnim, fall_weight);
Fall_transition->AddAnimation(forward_motion, motion_weight);

result_layer = new BlendLayer();
result_layer->AddAnimation(fall_transition, 1.0f);

result_layer->Blend();
```

Diagram 2

Gauche : un arbre décrit la combinaison des différentes couches d'animation. Droite: Le personnage de Drake (Uncharted) combine jusqu'à 25 animations.

Effets sonores

- Pour finaliser l'intégration réaliste de notre modèle dans la scène, il reste à être capable de synchroniser l'interaction entre les visuels de l'animation et la scène: bruits de pas, cris, recharge d'une arme, etc…

- Une méthode possible consiste à mettre en place un gestionnaire d'évènements dans la boucle du jeu. Ce gestionnaire va puiser les évènements (de types connus) dans une file, et les traiter. Ces évènements peuvent par exemple être de type sonore, ou bien déclencher des actions spécifiques, et sont issus de plusieurs sous-systèmes du jeu, dont le moteur d'animation.

- Au niveau des animations, on va associer, pour les keyframes qui le nécessitent, des marqueurs qui seront des déclencheurs d'évènement. Le moteur d'animation va donc, au moment où il joue une animation, détecter le franchissement d'un évènement, et en notifier le gestionnaire d'évènements. Les actions deviennent donc parfaitement synchrones avec le déroulement de notre animation.

Animation faciale

- L'animation faciale réaliste couvre plusieurs domaines de recherche.
- Dans sa version « temps réel », telle qu'elle peut aujourd'hui être implémentée dans les jeux, on notera les spécificités importantes suivantes:
  - Expressions réalistes
  - Synchronisation labiale (lipsync)

Blendshapes

La méthode la plus commune pour mettre en œuvre une animation faciale temps réel « réaliste » est d'utiliser des « morph targets » ou « blend shapes ».

Cette méthode consiste à définir plusieurs maillages représentant les expressions remarquables, et à moyenner la pondération obtenue par vertex tweening de chacune de ces expressions relativement au modèle de base.

Cette méthode se basant sur une déformation du maillage réalisée manuellement en amont, elle permet de rendre les effets à la fois des expressions, mais aussi des déformations des muscles.

Exemples de blend shapes pour l'animation faciale

Blendshapes

Techniquement, la création des blend shapes est réalisée par les artistes à partir de la pose statique.

La déformation étant par essence très localisée, seuls les vertices subissant une déformation nécessitent d'être stockés, ce qui réduit l'empreinte mémoire.

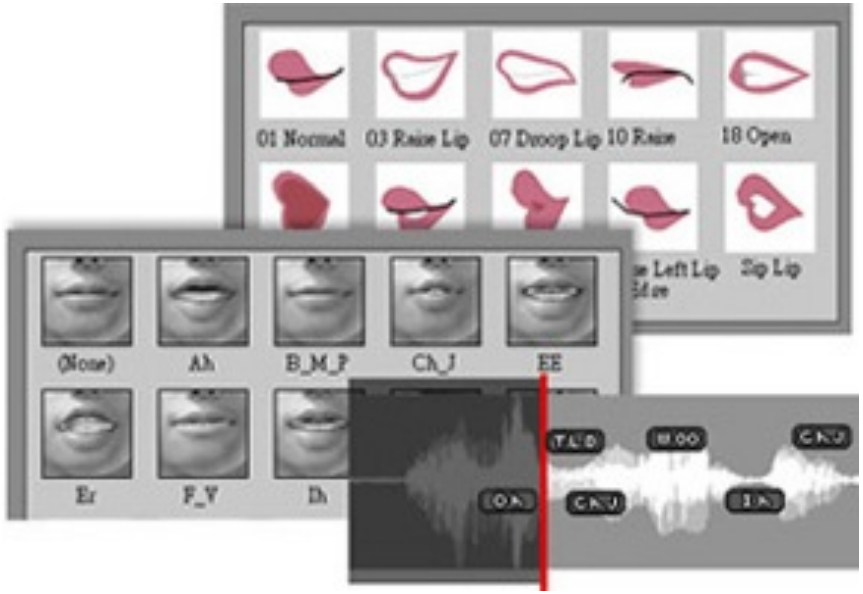
Côté animation, les morph targets sont appliquées sur le mesh avant la phase de skinning, de manière à pouvoir subir l'influence de l'animation globale.

Synchronisation labiale (lipsync)

La mise en œuvre d'une synchronisation labiale se déroule en 2 phases:

L'infographiste modélise les blend shapes correspondant aux différentes formes de bouche (environ 9 par approximation)

(1) A,I
(2) E
(3) F, V
(4) C, D, G, J, K, N, S, T, Y, Z
(5) L, T
(6) O
(7) U
(8) W, Q
(9) M, B, P (maillage de base)

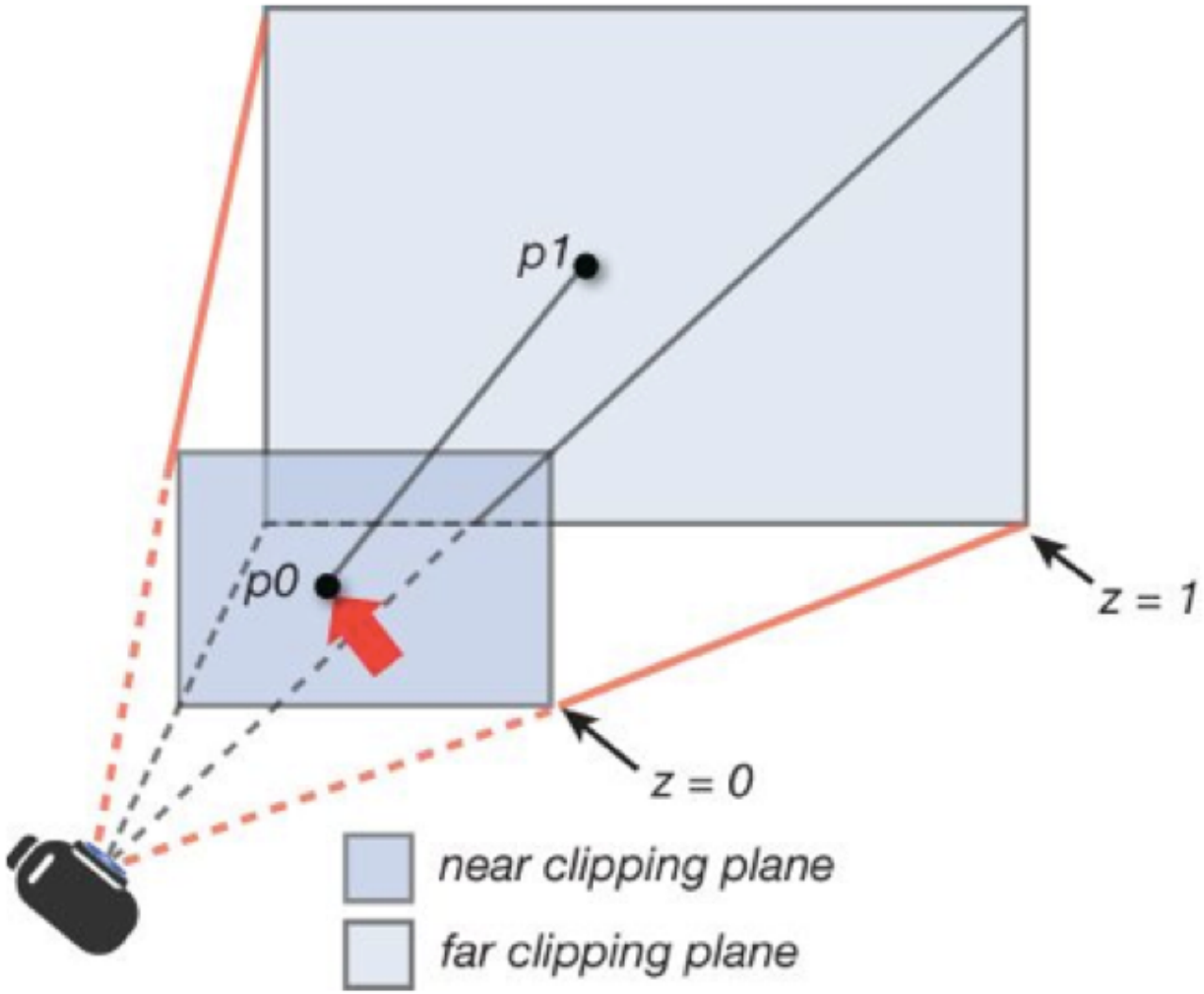


On réalise une analyse fréquentielle du texte à synchroniser, et on en extrait les phonèmes remarquables (à associer à nos blend shapes). On peut alors générer une série de keyframes qui encodent l'animation à appliquer sur les blend shapes.

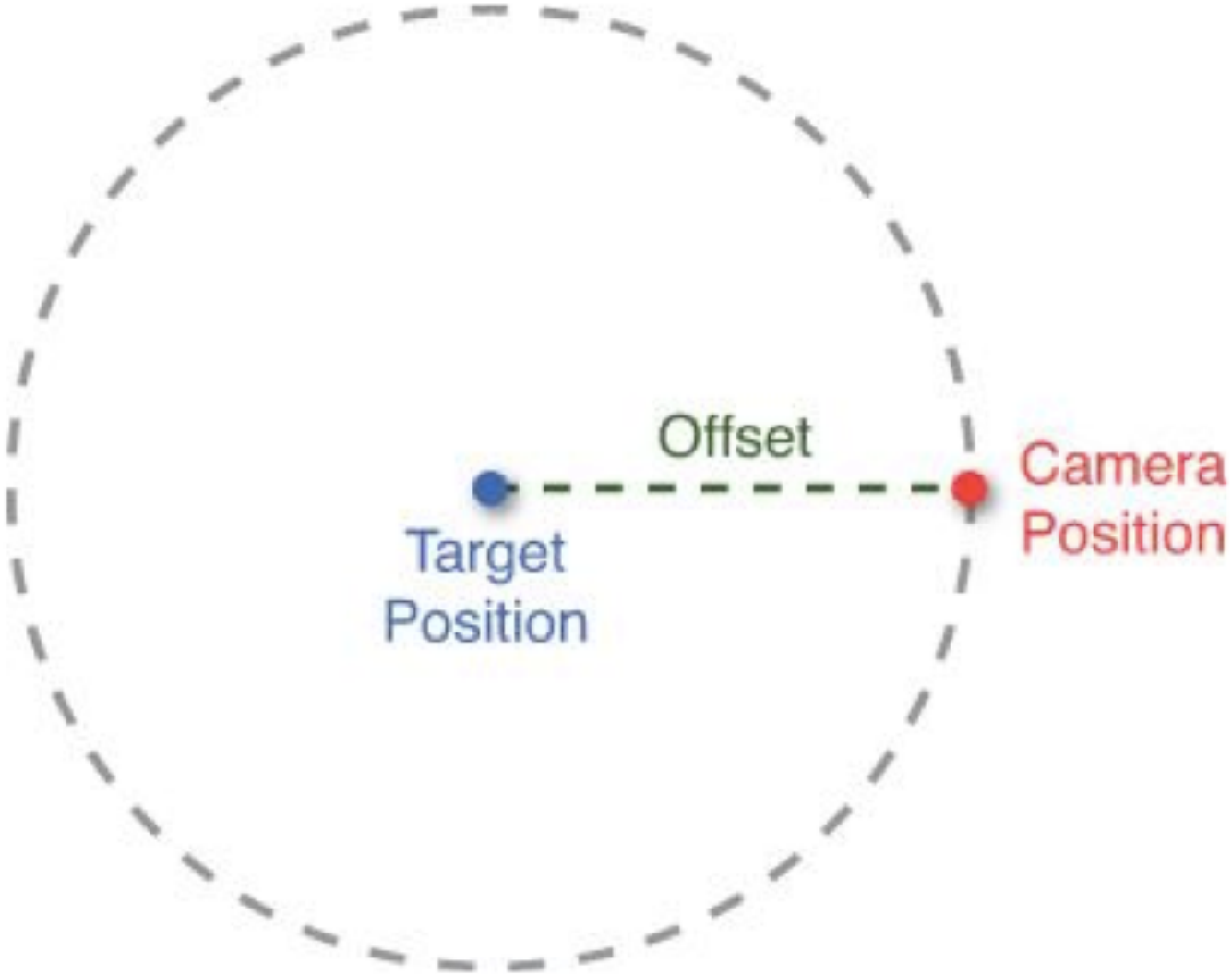
Animation de la caméra

- La gestion de la caméra est une partie importante dans une application 3D interactive:
  - Elle doit suivre l'action et offrir le meilleur champ de vision possible
  - Elle doit correctement interagir avec l'environnement (collisions, masquages, effets spéciaux)
- On distingue deux types de caméra:
  - Caméra immersive (first person)
  - Caméra de suivi (tracking ou third person)

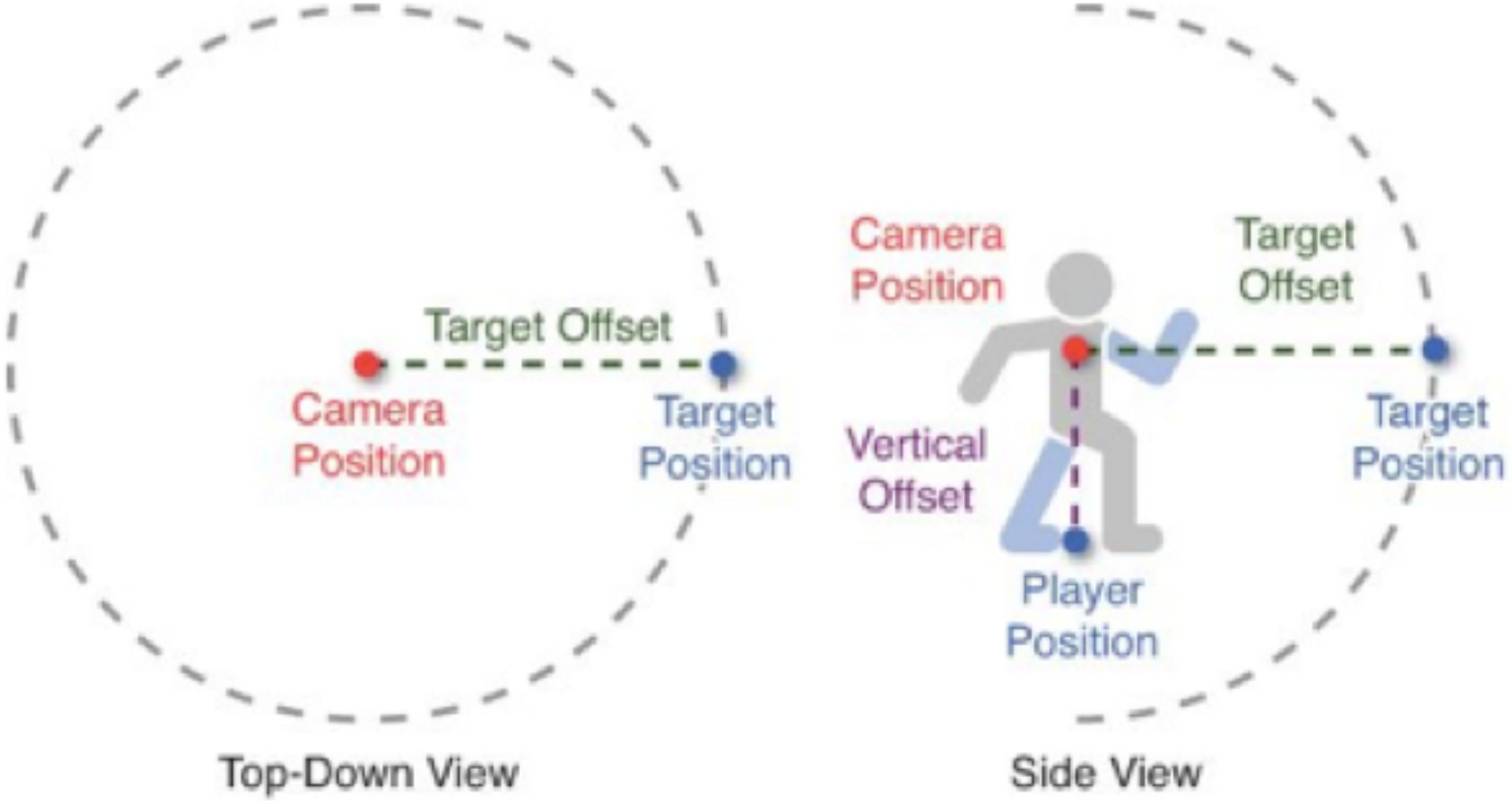
Picking et lancer de rayons



Orbit camera



First-person camera



Third-person camera

```
// tPos, tUp, tForward = Position, up, and forward vector of target
// hDist = horizontal follow distance
// vDist = vertical follow distance
function BasicFollowCamera(Vector3 tPos, Vector3 tUp, Vector3 tForward,
                           float hDist, float vDist)
   // Eye is offset from the target position
   Vector3 eye = tPos - tForward * hDist + tUp * vDist

   // Camera forward is FROM eye TO target
   Vector3 cameraForward = tPos - eye
   cameraForward.Normalize()

   // Cross products to calculate camera left, then camera up
   Vector3 cameraLeft = CrossProduct(tUp, cameraForward)
   cameraLeft.Normalize()
   Vector3 cameraUp = CrossProduct(cameraForward, cameraLeft)
   cameraUp.Normalize()

   // CreateLookAt parameters are eye, target, and up
   return CreateLookAt(eye, tPos, cameraUp)
end
```

Animation de la caméra

- Quel que soit son type (immersive ou tracking), l'objet caméra est souvent attaché à la hiérarchie (squelette) d'animation de l'acteur suivi (en position et/ou orientation). Il va donc directement subir toutes les transformations induites par l'animation du personnage.

- Le gros problème est que les données d'animation peuvent être bruitées (surtout si elles proviennent d'une source type motion capture), ou bien les déplacements de l'objet suivi peuvent subir des variations brutales de vitesse. Ces mouvements sont directement retransmis à la caméra, induisant un comportement relativement rigide et non naturel.

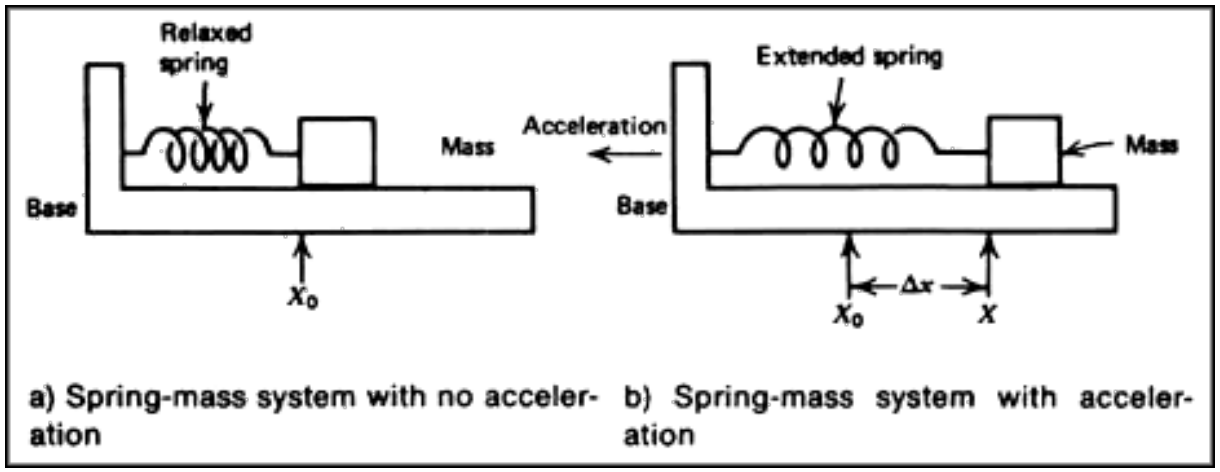
- Afin de réduire ces artefacts, il faut implémenter un modèle de caméra qui permette d'amortir les imperfections. On choisit en général un modèle masse-ressort pour appliquer les transformations à la caméra.

Animation masse-ressort

La force de compression/extension appliquée par un ressort est

$$F = -k*(X-X0)$$

k : Constante de raideur du ressort
X-X0 : Variation de longueur du ressort le long de son axe



a) Spring-mass system with no acceleration

b) Spring-mass system with acceleration

La position de la caméra est obtenue en résolvant l'équation de Newton

$$\Sigma F = m*a$$
$$dV = (\Sigma F/m)*dT$$
$$dP = V * dT$$

Animation physique d'une caméra

On remplace la liaison rigide entre la squelette d'animation et la caméra par une liaison élastique de type ressort

- On définit un coefficient de raideur approprié pour la liaison élastique (ni trop fort : la caméra suivrait trop brutalement les variations de vitesse comme pour une liaison rigide, ni trop faible : la caméra oscillerait autour de sa position cible)

Lors d'un déplacement de la cible de la caméra, on applique une force sur la liaison élastique (soit approximée, soit en dérivant la vitesse)

- Après l'intégration physique, on applique un coefficient d'atténuation artificiel (en principe sur la vitesse) à la masse, afin de converger rapidement vers la position de destination et limiter les oscillations
- On peut optionnellement définir des longueurs minimales et maximales autorisées pour la liaison élastique

Le même principe de déplacement élastique peut s'appliquer aussi bien aux translations de la caméra qu'à ses rotations (inertie).

Gestion des collisions

Gestion des collisions avec les objets infranchissables de la scène (murs, parois, etc). La gestion des forces physiques appliquées à la caméra peut être intégrée directement dans la gestion des liaisons élastiques.

Gestion des objets pouvant masquer le champ visuel d'une caméra de type tracking.

Une solution : tester en permanence si un objet s'interpose entre la caméra et sa cible, et si c'est le cas se rapprocher de la cible pour se placer devant l'objet

Autre solution : rendre transparents les objets interposés

- Optimisation du champ visuel

Pour suivre l'action au plus près, la caméra peut avoir à « anticiper » les déplacements de sa cible

Animation et cinématographie

Controle du cadrage et de la profondeur de champ

Variation dynamique de la profondeur de champ pour focaliser l'attention sur la cible

Variation dynamique de la focale pour renforcer les effets de vitesse (zooms compensés)

Ajout de bruit sur la position/orientation de la caméra pour simuler une caméra à l'épaule ou un effet d'aspiration

Ajout d'effets graphiques sur la caméra (pluie, buée, etc)

Pour aller plus loin