Chapitre 2, manuel chapitre 3

TRAJETS ET CONTRÔLE DU MOUVEMENT

IMN501, Hiver 2021, Olivier Vaillancourt et Richard Egli
Université de Sherbrooke

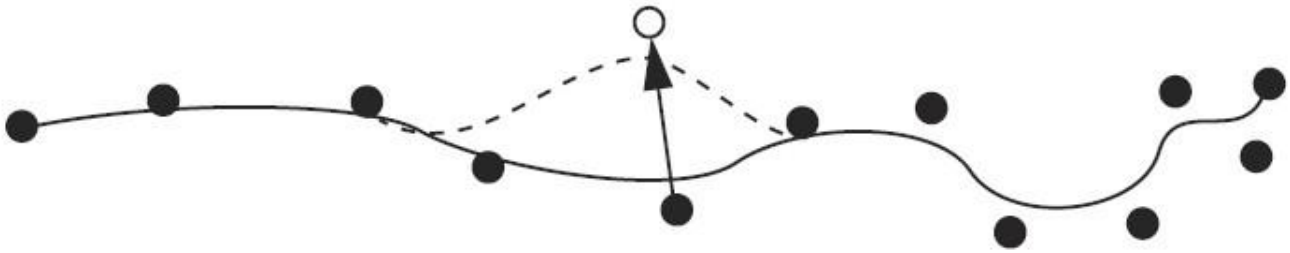
DÉFINITION D'UN TRAJET

- Suite d'un ou de plusieurs morceaux de courbe joints.
  - (Par exemple, plusieurs Bézier interconnectés.)
  - Possible de qualifier la continuité d'un trajet de la même façon que celle d'une courbe. (C_0, C_1, C_2, etc.)
- Aussi appelée "courbe composite".

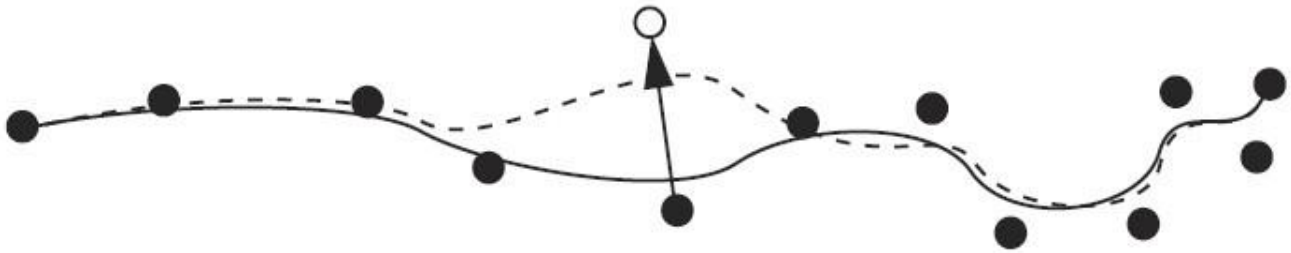
PORTÉE DE CONTRÔLE D'UNE COURBE

CONTRÔLE GLOBAL ET CONTRÔLE LOCAL

- Contrôle local vs contrôle global



Contrôle local



Contrôle global

Manuel, Figure 3.3

CONTRÔLE GLOBAL ET CONTRÔLE LOCAL

- Mode de contrôle généralement préféré: **Contrôle local**
  - Permet de modifier une partie d'un trajet sans avoir d'effet indésirable les autres segments.
  - Plusieurs courbes permettent un contrôle local :
    - Splines de Catmull-Rom
    - Courbes de Bézier cubiques
    - B-Splines cubiques

CONTRÔLE GLOBAL ET CONTRÔLE LOCAL

- En règle générale:
  - Plus le degré d'une courbe augmente, plus le contrôle devient global.

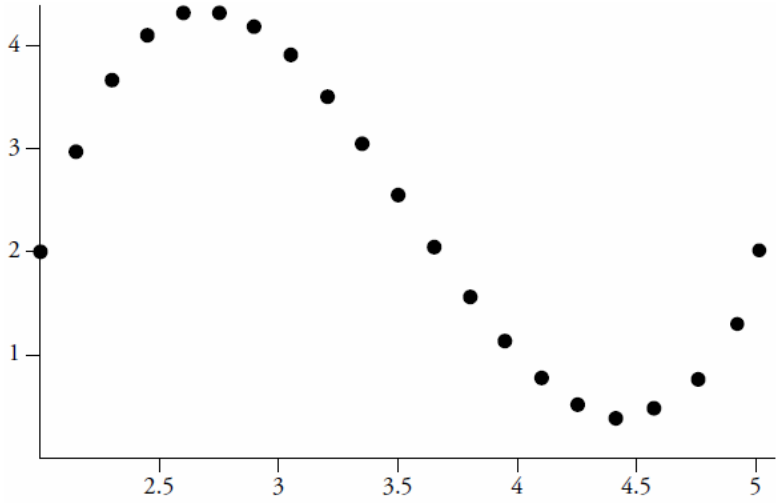
CONTRÔLE DU MOUVEMENT

2.2 - CONTRÔLE DU MOUVEMENT

- Objectif :
  - Permettre la création d'un mouvement quelconque le long d'une courbe de forme quelconque.
- Notation:
  - **P(u)** → Point dans l'espace correspondant à la progression **u** sur le trajet défini par la fonction **P** qui représente une courbe.

2.2 – CONTRÔLE DU MOUVEMENT

- Problème :
  - La variation du paramètre **u** de façon constante ne garantie pas une variation constante de la valeur résultante de **P(u)**. (Dans notre cas le déplacement spatial.)



Manuel, Figure 3.4

2.2 – CONTRÔLE DU MOUVEMENT

- Solution :

  Procéder à une **paramétrisation par longueur d'arc**.

  S'assurer que la distance parcourue sur la courbe soit proportionnelle à la progression **s** (longueur d'arc).

CALCUL DE LONGUEUR D'ARC

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Pour paramétriser, il faut en premier savoir calculer la longueur d'arc.

- Longueur d'arc → Distance parcourue sur un arc de courbe.

- Notation :
  - **s** → Longueur d'arc
  - **S(u)** → Longueur d'arc au paramètre **u.**
    (ou au point **P(u)**)
  - **U(s)** → Valeur du paramètre **u** à la longueur **s**.

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Nécessaire pour la paramétrisation:
  - s = S(u) → Trouver la longueur d'arc selon la progression. (Pour la paramétrisation en soit.)
  - U(s) → Trouver le paramètre **u** selon une progression **s** sur la courbe.
  - p = P(U(s)) → Le point dans l'espace résultant.

- 3 approches de calcul de longueur d'arc:
  1. Calcul analytique
  2. Par différentiation avant
  3. Intégration numérique

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Calcul analytique
  - Intégrale de la longueur d'arc entre un point $\mathbf{u_1}$ et $\mathbf{u_2}$, forme générale:

$$s = \int_{u_2}^{u_1} |\frac{dP}{du}| du$$

  - Où : (pour un cas 3D)

$$|\frac{dP}{du}| = \sqrt{(\frac{dx(u)}{du})^2 + (\frac{dy(u)}{du})^2 + (\frac{dz(u)}{du})^2}$$

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Calcul analytique
  - Problème: difficile, voire impossible, d'évaluer une telle intégrale pour certains types de courbe. (Notamment les B-splines).
  - Démontré dans:
    - C. Judd et P. Hartley, "Parametrization and Shape of B-Spline Curves for CAD", *Computer Aided Design*, 12(5), Septembre 1980, pp. 235-239

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Méthode de différentiation avant(forward differencing)
  - On échantillonne la courbe à plusieurs points.
  - On calcule la distance euclidienne entre les points.
  - On accumule la distance par rapport à **u** pour obtenir la valeur **s**.
  - On crée une table de correspondance **u** ←→ **s**

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Exemple de table résultante:

- **V(i)** → Paramètre **u** à l'index **i**.

- **G(i)** → Longueur d'arc **s** à l'index **i**.

Index	V(i)	G(i)
0	0.00	0.000
1	0.05	0.080
2	0.10	0.150
3	0.15	0.230
4	0.20	0.320
5	0.25	0.400
6	0.30	0.500
7	0.35	0.600
8	0.40	0.720
9	0.45	0.800
10	0.50	0.860
11	0.55	0.900
12	0.60	0.920
13	0.65	0.932
14	0.70	0.944
15	0.75	0.959
16	0.80	0.972
17	0.85	0.984
18	0.90	0.994
19	0.95	0.998
20	1.00	1.000
	u	**s**

Manuel Table 3.1

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Comment passer de **u** à **s**?
- Méthode 1 :
  1. Trouver la valeur de longueur d'arc la plus proche de celle recherchée.
  2. Calcul direct puisque u est également espacé.
- Exemple:
  - On a u = 0.73, on cherche s correspondant. d=V(i+1) -V(i)

$$i = \lfloor \frac{u}{d} + 0.5 \rfloor = \lfloor \frac{0.73}{0.05} + 0.5 \rfloor = 15$$

$$G(i) = G(15) = 0.959$$

Index	V(i)	G(i)
0	0.00	0.000
1	0.05	0.080
2	0.10	0.150
3	0.15	0.230
4	0.20	0.320
5	0.25	0.400
6	0.30	0.500
7	0.35	0.600
8	0.40	0.720
9	0.45	0.800
10	0.50	0.860
11	0.55	0.900
12	0.60	0.920
13	0.65	0.932
14	0.70	0.944
15	0.75	0.959
16	0.80	0.972
17	0.85	0.984
18	0.90	0.994
19	0.95	0.998
20	1.00	1.000
	u	**s**

Manuel Table 3.1

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Comment passer de **u** à **s**?
- Problème de la méthode 1:
  - On arrive à un seul point **s** pour plusieurs valeurs **u**, il n'y a pas d'interpolation réelle. On a donc une animation saccadée.

Index	V(i)	G(i)
0	0.00	0.000
1	0.05	0.080
2	0.10	0.150
3	0.15	0.230
4	0.20	0.320
5	0.25	0.400
6	0.30	0.500
7	0.35	0.600
8	0.40	0.720
9	0.45	0.800
10	0.50	0.860
11	0.55	0.900
12	0.60	0.920
13	0.65	0.932
14	0.70	0.944
15	0.75	0.959
16	0.80	0.972
17	0.85	0.984
18	0.90	0.994
19	0.95	0.998
20	1.00	1.000
	u	**s**

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Comment passer de **u** à **s**?

- Méthode 2 :
  1. On trouve la borne inférieure **V(i)**
  2. On trouve la borne supérieure **V(i+1)**
  3. On interpole linéairement entre les deux bornes, au prorata de la distance entre le **u** recherché et les bornes.

Index	V(i)	G(i)
0	0.00	0.000
1	0.05	0.080
2	0.10	0.150
3	0.15	0.230
4	0.20	0.320
5	0.25	0.400
6	0.30	0.500
7	0.35	0.600
8	0.40	0.720
9	0.45	0.800
10	0.50	0.860
11	0.55	0.900
12	0.60	0.920
13	0.65	0.932
14	0.70	0.944
15	0.75	0.959
16	0.80	0.972
17	0.85	0.984
18	0.90	0.994
19	0.95	0.998
20	1.00	1.000
	u	**s**

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Comment passer de **u** à **s**?

- Exemple Méthode 2 :

$$
\begin{aligned}
i &= \lfloor \tfrac{u}{d} \rfloor = \lfloor \tfrac{0.73}{0.05} \rfloor = 14 \\
w &= \tfrac{(u - V[i])}{(V[i+1] - V[i])} = \tfrac{(0.73 - 0.70)}{(0.75 - 0.70)} = 0.6 \\
s &= (1 - w)(G[i]) + (w)(G[i+1]) \\
&= (1 - 0.6)(0.944) + (0.6)(0.959) \\
&= 0.953
\end{aligned}
$$

Index	V(i)	G(i)
0	0.00	0.000
1	0.05	0.080
2	0.10	0.150
3	0.15	0.230
4	0.20	0.320
5	0.25	0.400
6	0.30	0.500
7	0.35	0.600
8	0.40	0.720
9	0.45	0.800
10	0.50	0.860
11	0.55	0.900
12	0.60	0.920
13	0.65	0.932
14	0.70	0.944
15	0.75	0.959
16	0.80	0.972
17	0.85	0.984
18	0.90	0.994
19	0.95	0.998
20	1.00	1.000

Manuel Table 3.1 **u** **s**

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Pour reparamétriser, il faut cependant passer de **s** à **u**.

- Se fait comme pour passer de **u** à **s**, à l'exception que **i** ne peut être trouvé directement.
  - Les valeurs ne sont pas espacées également

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Processus:
  1. On trouve la borne inférieure G(i)
  2. On trouve la borne supérieure G(i+1)
  3. On interpole entre les deux bornes, au prorata du **s** recherché entre celles-ci.

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Exemple:

  s = 0.75

  On cherche **u**.

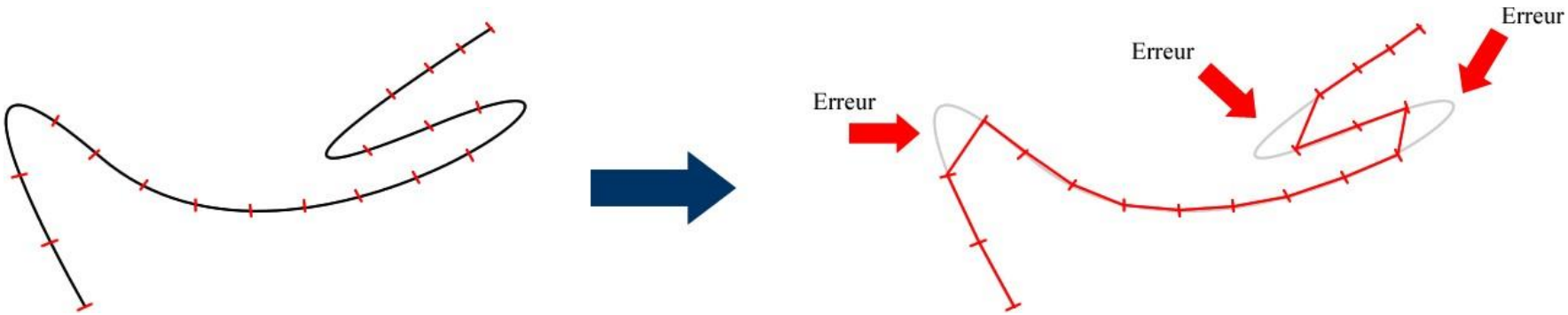
$$
\begin{aligned}
i &= RechercheIndex(0.75) = 8 \\
w &= \frac{s - G[i]}{G[i+1] - G[i]} = \frac{0.75 - 0.72}{0.80 - 0.72} = \frac{3}{8} \\
u &= (1 - w)(V[i]) + (w)(V[i+1]) \\
&= (1 - \tfrac{3}{8})(0.40) + (\tfrac{3}{8})(0.45) \\
&= 0.41875
\end{aligned}
$$

Index	V(i)	G(i)
0	0.00	0.000
1	0.05	0.080
2	0.10	0.150
3	0.15	0.230
4	0.20	0.320
5	0.25	0.400
6	0.30	0.500
7	0.35	0.600
8	0.40	0.720
9	0.45	0.800
10	0.50	0.860
11	0.55	0.900
12	0.60	0.920
13	0.65	0.932
14	0.70	0.944
15	0.75	0.959
16	0.80	0.972
17	0.85	0.984
18	0.90	0.994
19	0.95	0.998
20	1.00	1.000

Manuel Table 3.1 **u** **s**

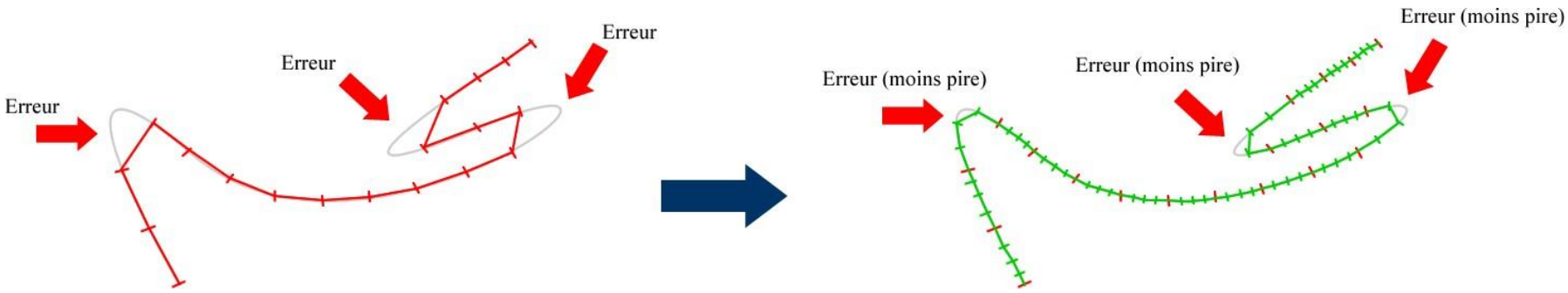
2.2.1 – CALCUL DE LONGUEUR D'ARC

- Défaut de la méthode de différentiation avant:
  - Réduit l'erreur globale mais ne tient pas compte localement de l'erreur.



2.2.1 – CALCUL DE LONGUEUR D'ARC

- Solution 1:
  - Sur-échantillonner la courbe.
    - On garde un même nombre d'entrées, mais la longueur d'arc estimée à chaque entrée est le résultat d'une évaluation à plusieurs points:

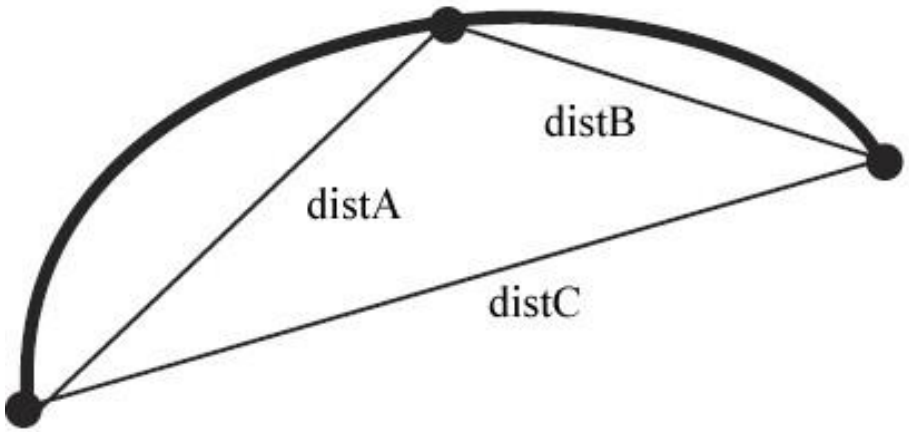


2.2.1 – CALCUL DE LONGUEUR D'ARC

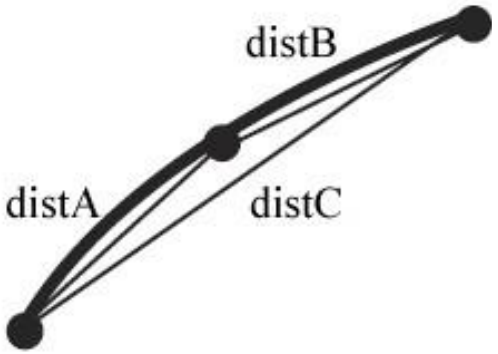
- Solution 2:
  - Échantillonnage adaptif différentiation avant:
    - Revient à détecter quelles parties de la courbe contiennent l'erreur.
    - Lors de la construction initiale de la table, on rajoute des points dans les sections de la courbe où l'erreur dépasse un seuil.
    - On mesure l'erreur en sur-échantillonnant entre les points de la courbe et en comparant la distance sur-échantillonnée avec la distance initiale.

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Visuellement on obtient:



distA + distB - distC > seuil

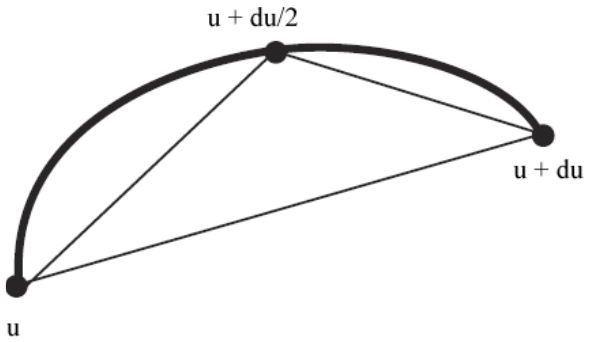


distA + distB - distC < seuil

Manuel Figure 3.7

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Solution 2 : Algorithme

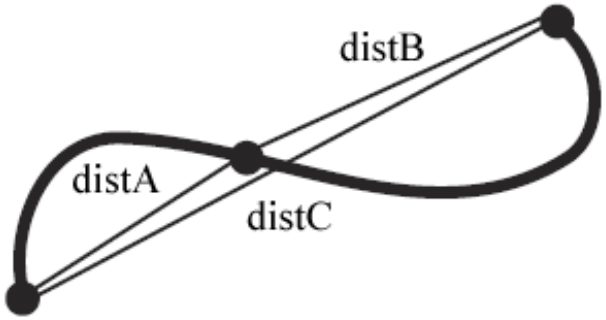



```
EchantillonnageAdaptif(u,du)
    Calculer la distance d1 = dist(u,u+du)
    Calculer la distance d2 = dist(u,u+du/2)+dist(u+du/2,u+du)
    Si d2 – d1 > seuil
        EchantillonnageAdaptif(u,du/2)
        EchantillonnageAdaptif(u+du/2,du/2)
    Sinon
        On insère le point u et le point u+du dans notre tableau de paramétrisation.
```

EchantillonnageAdaptif est appellé pour chaque duo de point de notre trajet.

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Attention! La méthode adaptive n'est pas sans faille:



distA + distB - distC < seuil

Manuel Figure 3.7

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Solution 3 :
  - Utiliser une interpolation de niveau supérieure que l'interpolation linéaire pour évaluer la longueur de la courbe.
    - Ex: Interpolation quadratique

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Intégration numérique (pas matière à examen)
  - Lorsque ce pas possible de façon analytique.
    - Méthode des trapèzes
    - Méthode de Simpson
    - Quadrature Gaussienne
    - Quadrature adaptive gaussienne

$$s = \int_{u_2}^{u_1} \left| \frac{dP}{du} \right| du$$

- L'intégration numérique fait partie de l'analyse numérique.
  - Cours MAT517 à l'Université de Sherbrooke.

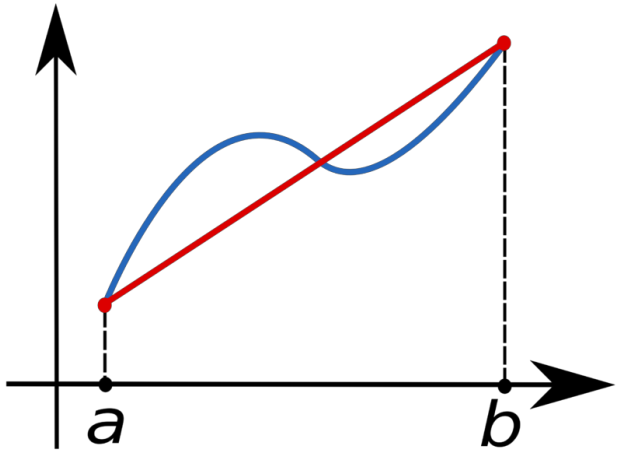
2.2.1 – CALCUL DE LONGUEUR D'ARC

- Méthode d'intégration des trapèzes.
  - L'air sous la courbe est un somme d'aires de trapèzes se rapprochant de la vraie courbe.

$$\int_a^b f(x)dx = \frac{b-a}{n}\left(\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f\left(a + k\frac{b-a}{n}\right)\right) + R_n(f)$$

n = nombre de subdivisions entre a et b

$R_n(f)$ = erreur de quadrature (approximation de la différence entre la vraie courbe et la courbe estimée numériquement.)



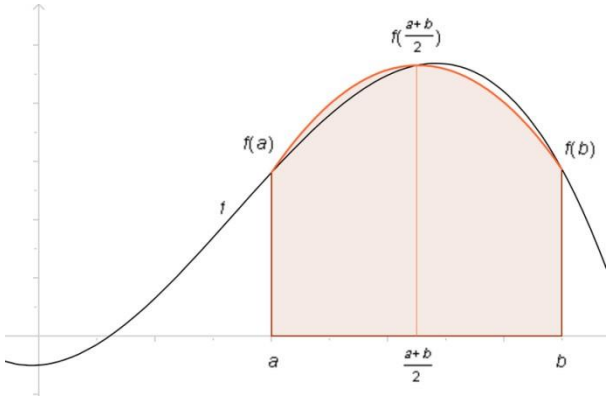
http://www.wikipedia.org, "Méthode des trapèzes", 16 janvier 2009

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Méthode d'intégration de Simpson
  - Comme la méthode des trapèzes mais utilise une interpolation quadratique pour se "calquer" sur la courbe le plus précisément possible.

$$\int_a^b f(x)dx \simeq \int_a^b P(x)dx = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

$$P(x) = f(a)\frac{(x-m)(x-b)}{a-m)(a-b)} + f(m)\frac{(x-a)(x-b)}{(m-a)(m-b)} + f(b)\frac{(x-a)(x-m)}{(b-a)(b-m)}$$



http://www.wikipedia.org, "Méthode de Simpson", 16 janvier 2009

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Quadrature gaussienne et quadrature adaptive.

- Qu'est-ce qu'une quadrature?
  - La quadrature d'une surface est la recherche d'un carré ayant la même aire que la surface en question.

  - Célèbre problème → Quadrature du cercle

    Un carré dont les côtés ont une longueur de $\sqrt{\pi}$ a le même aire qu'un cercle de rayon 1.

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Quadrature Gaussienne
  - Une fonction est évaluée à certains points fixes, sur l'intervale [-1,1]
  - La valeur de chaque évaluation est multipliée par un poids w_i variant selon la position dans la fonction.

$$\int_{-1}^{1} f(x)dx = \int_{-1}^{1} W(x)g(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

2.2.1 – CALCUL DE LONGUEUR D'ARC

- La quadrature gaussienne fonctionne sur l'intervalle [-1,1]
- Nos fonctions sont évaluées de [a,b].
- Il faut mettre à l'échelle [a,b] pour qu'il puisse être évalué de [-1,1]

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right)dx$$

$$\approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}x_i + \frac{a+b}{2}\right)$$

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Comment trouver W(x) ou w_i?
  - Tables de mathématiques
    - (*Standard mathematical tables* de CRC par exemple.)
  - Voir manuel Figure 3.8 pour un exemple d'implémentation.

2.2.1 – CALCUL DE LONGUEUR D'ARC

- Quadrature Gaussienne Adaptive

- Même principe que la différentiation avant adaptives.

  - Chaque intervalle est évalué avec la quadrature gaussienne.
  - On divise ensuite l'intervalle par 2 pour réévaluer la somme des moitiés.
  - On réévalue en réduisant récursivement l'intervalle tant que la précision n'est pas suffisante.

2.2.1 – CALCUL DE LONGUEUR D'ARC

- En résumé:
  - Calculer la longueur d'arc d'une courbe n'est pas trivial.
  - Permet la paramétrisation des courbes et donc de produire un mouvement constant en utilisant la longueur d'arc comme variable d'incrément.
  - Différentes méthodes:
    - Calcul analytique
      - Difficile voire impossible
    - Méthode par différentiation avant
      - Standard, adaptive, sur-échantillonnage
    - Intégration numériques
      - Simpson, trapèzes, quadrature Gaussienne (+adaptive)

CONTRÔLE DE LA VITESSE

2.2.2 – CONTRÔLE DE LA VITESSE

- Avec la paramétrisation par longueur d'arc, on peut obtenir une vitesse constante le long d'une courbe.
- Ceci donne assez de contrôle pour permettre l'ajout de fonctions faisant varier la vitesse, soit une fonction temps/distance
- Une telle fonction est notée: **s = S(t)** et fournie la progression **s** sur l'arc en fonction du temps **t**.
- Un point dans l'espace au cours d'un mouvement s'obtient donc tel que:

$$p = P(U(S(t)))$$

2.2.2 – CONTRÔLE DE LA VITESSE

- On a maintenant 2 types de courbes.
  - Les courbes spatiales (Catmull-Rom, Bézier, etc)
    - Indiquent *où* aller
  - Les fonctions temps-distance
    - Indiquent *quand* y aller

2.2.2 – CONTRÔLE DE LA VITESSE

- Pour simplifier l'expression de la vitesse, on utilise un temps **t** normalisé, de 0 à 1.

- 2 considérations populaires
  - La fonction temps-distance est monotone.
  - La fonction temps-distance est au moins continue C_0

- Ces considérations ne sont pas coulées dans le béton, dans certains cas il peut être désirable de les violer.

EASE-IN/EASE-OUT

2.2.3 – EASE-IN/EASE-OUT

- Très populaire dans le contrôle de la vitesse.

- Le mouvement commence à l'arrêt (vitesse initiale de 0)
- Le mouvement termine à l'arrêt (vitesse finale de 0)

- Il n'y a pas de variation instantanée de la vitesse (Continuité C_1)

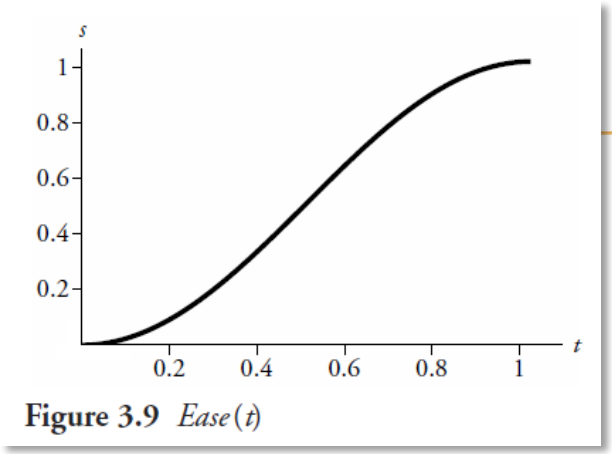


Figure 3.9 *Ease*(*t*)

Manuel Figure 3.9

Youtube, https://www.youtube.com/, "Ease-in & Ease-out", Aabid Dhamani

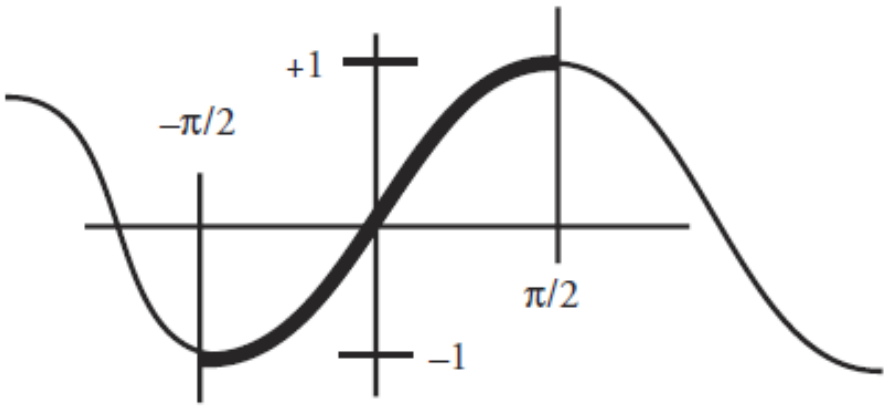
2.2.3 – EASE-IN/EASE-OUT

- Il existe différentes façon de générer une telle fonction de mouvement.

- Nous verrons:
  - Interpolation sinusoïdale
  - Fonction sinusoïdale d'accélération par morceaux
  - Polynôme cubique
  - Construction parabolique

2.2.3 – EASE-IN/EASE-OUT

- Interpolation sinusoïdale
  - La section de -π/2 à π/2 d'une courbe sinusoïdale est un segment connu possédant les propriétés d'un ease-in/ ease-out.



Manuel Figure 3.11

2.2.3 – EASE-IN/EASE-OUT

- Comment construire sin(θ) pour avoir une fonction ease-in/ease-out dont les paramètres **s** (déplacement) et **t** (temps) sont normalisés entre 0 et 1?

1. Thêta doit varier entre -π/2 et π/2 en fonction d'un paramètre t variant entre 0 et 1.

$$\{\theta \exists [-\pi/2..\pi/2] \mid \theta = (t\pi - \tfrac{\pi}{2}),\ \forall\ t \exists [0..1]\}$$

$$s = ease(t) = sin(t\pi - \tfrac{\pi}{2})$$

2.2.3 – EASE-IN/EASE-OUT

$$s = ease(t) = sin(t\pi - \tfrac{\pi}{2})$$

2. La fonction ci-haut varie en **s** de -1 à 1, (donc sur un intervalle de 2 de haut) On divise le tout par 2 pour avoir une fonction variant d'une unité. (de -0.5 à 0.5)

$$s = ease(t) = \frac{sin(t\pi - \frac{\pi}{2})}{2}$$

3. On additionne ensuite 0.5 pour faire passer la variation de [-0.5,0.5] à [0,1]

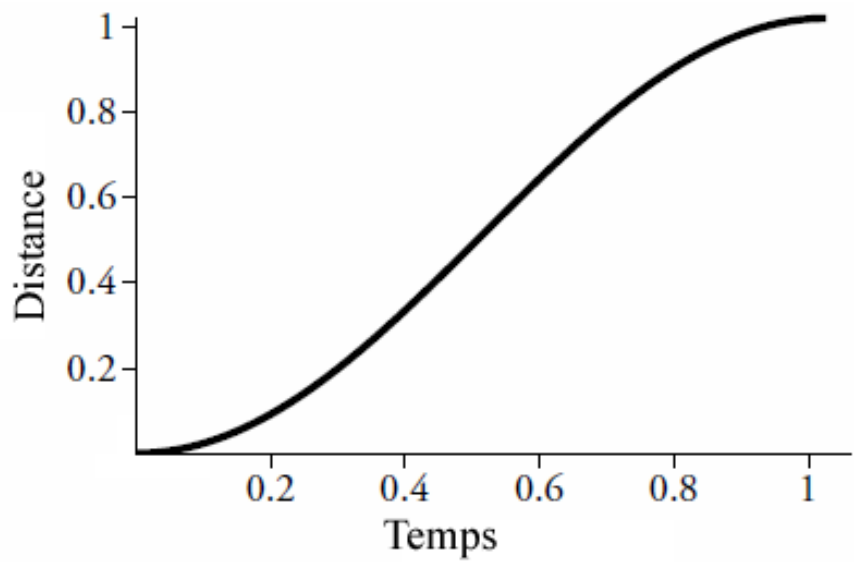
$$s = ease(t) = \frac{sin(t\pi - \frac{\pi}{2})}{2} + 0.5 = \frac{sin(t\pi - \frac{\pi}{2}) + 1}{2}$$

2.2.3 – EASE-IN/EASE-OUT

- Au final, notre fonction ease-in/ease-out sinusoïdale est exprimée selon la formule:

$$s = ease(t) = \frac{sin(t\pi - \frac{\pi}{2}) + 1}{2}$$

- Ce qui donne graphiquement:



Manuel Figure 3.12

2.2.3 – EASE-IN/EASE-OUT

- Observations par rapport à l'interpolation sinusoïdale.

- Avantages:
  - Facile à concevoir et à comprendre
  - Facile à manipuler mathématiquement

2.2.3 – EASE-IN/EASE-OUT

- Inconvénients:
  - La courbe est toujours en accélération ou en décélération.
    - Pour une courbe très longue la vitesse initiale/finale peut être très lente, très longtemps.
  - Peu de contrôle sur les différentes phases du mouvement
    - Limité à accélérer du temps 0 à 0.5 et décélérer du temps 0.5 à 1.

2.2.3 – EASE-IN/EASE-OUT

- Pour pallier à ces inconvénient, on utilise un autre type de courbe Ease-in/Ease-Out:

La Fonction sinusoïdale d’accélération par morceaux.

2.2.3 – EASE-IN/EASE-OUT

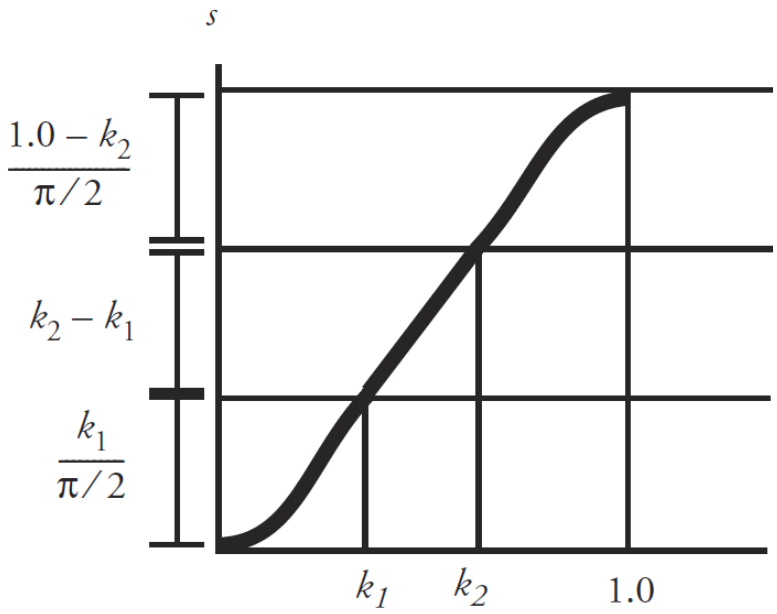
- Fonction sinusoïdale d'accélération par morceaux

  - Consiste à créer la courbe de ease-in/ease-out en 3 sections différentes, soient:
    1. Section d'accélération
    2. Section de vitesse constante
    3. Section de décélération

2.2.3 – EASE-IN/EASE-OUT

- Les sections d'accélération/décélération sont des morceaux de fonction sinusoïdale.

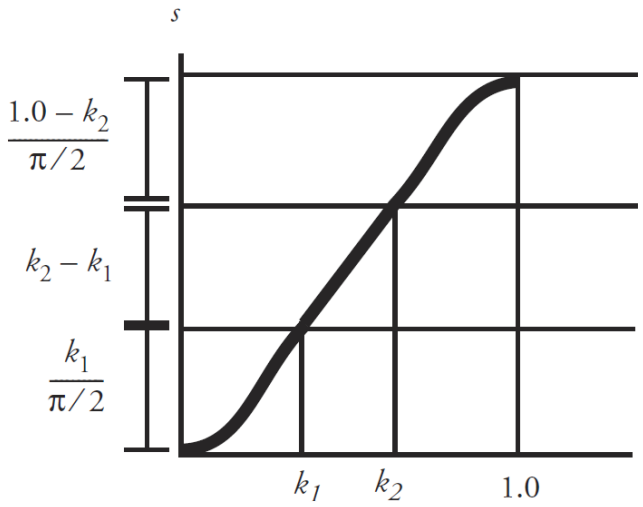
- La section centrale à vitesse constante est une simple droite.



Manuel Figure 3.12

2.2.3 – EASE-IN/EASE-OUT

- De cette construction on déduit un avantage majeur:
  - Contrôle indépendant du temps d'accélération, de décélération et de vitesse constante.



Manuel Figure 3.12

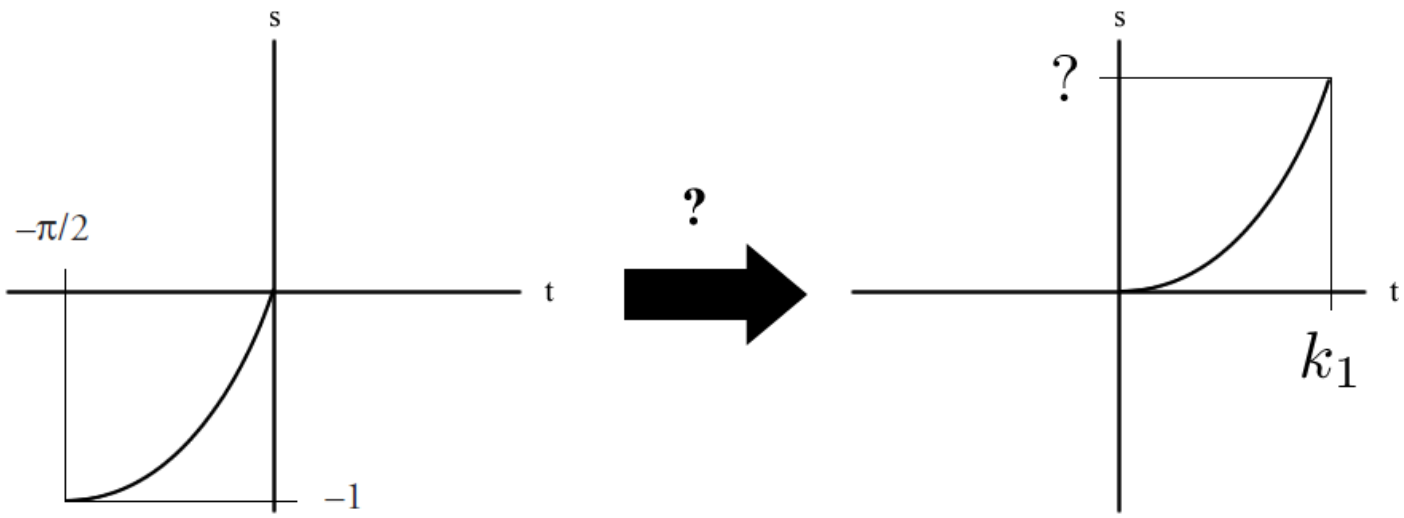
2.2.3 – EASE-IN/EASE-OUT

- Construction de la fonction d'interpolation
  - Les sections de la fonction sont séparés par des valeurs $\mathbf{k_1}$ et $\mathbf{k_2}$.
  - $\mathbf{k_1}$ et $\mathbf{k_2}$ sont régis par les propriétés suivantes :

$$k_1, k_2 \exists [0..1]$$
$$k_2 \geq k_1$$

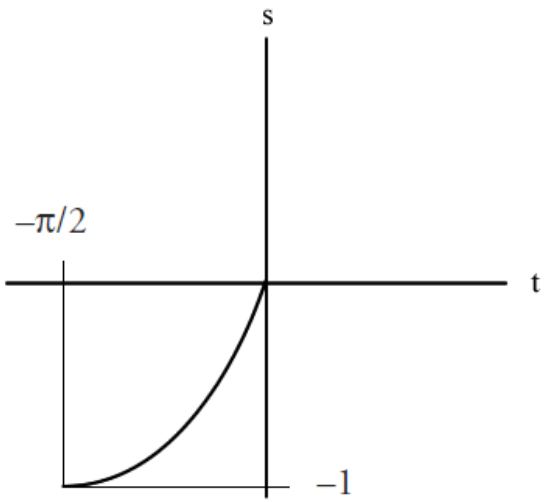
2.2.3 – EASE-IN/EASE-OUT

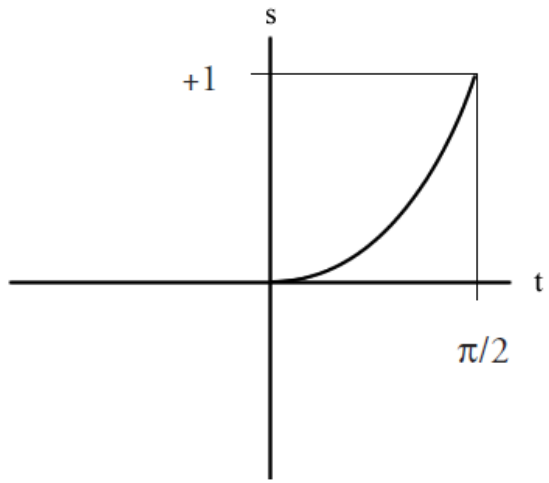
- Construction de la fonction d'interpolation
  - Morceau pour l'accélération:
    - Comment passer du sinus standard à un morceau évoluant dans les positifs et de 0 à k_1 sur l'axe t?



2.2.3 – EASE-IN/EASE-OUT

- Étape 1 : Décallage du morceau de fonction pour l'emmener dans le quadrant positif



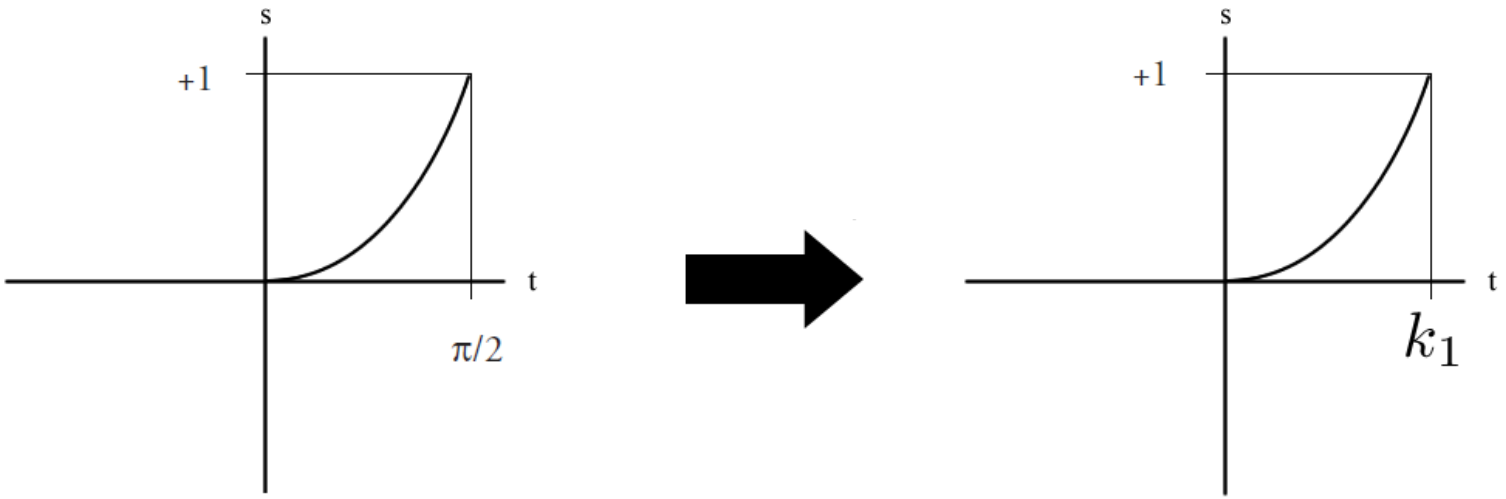


$$sin(t)\ \forall\ t = [\frac{-\pi}{2}, 0]$$

$$sin(t - \frac{\pi}{2}) + 1\ \forall\ t = [0, \frac{\pi}{2}]$$

2.2.3 – EASE-IN/EASE-OUT

- Étape 2: On redimensionne la courbe sur l'axe **t** pour qu'elle accélère du temps 0 au temps k_1.

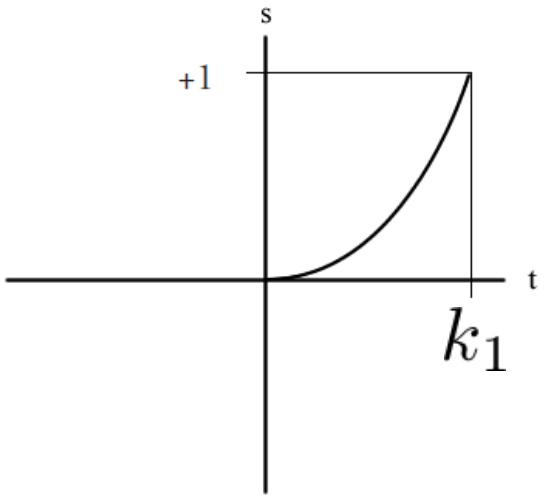


$$sin(t - \frac{\pi}{2}) + 1 \; \forall \; t = [0, \frac{\pi}{2}]$$

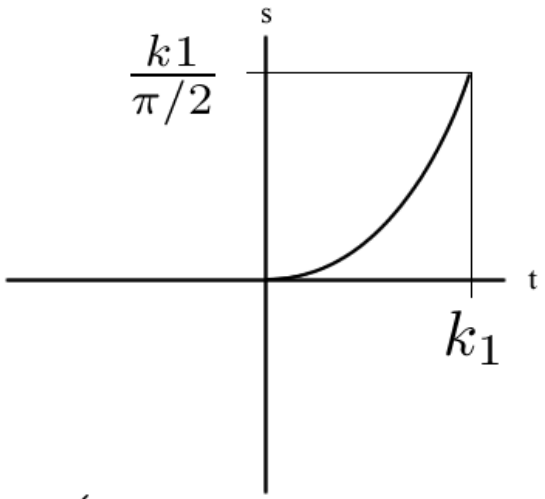
$$sin(\frac{t}{k_1}\frac{\pi}{2} - \frac{\pi}{2}) + 1 \quad \forall \; t = [0, k_1]$$

2.2.3 – EASE-IN/EASE-OUT

- Étape 3: Pour conserver la proportionalité entre l'axe **t** et l'axe **s**, on redimensionne l'axe **s** du même coefficient:



$$s \rightarrow s(\frac{k1}{\pi/2})$$



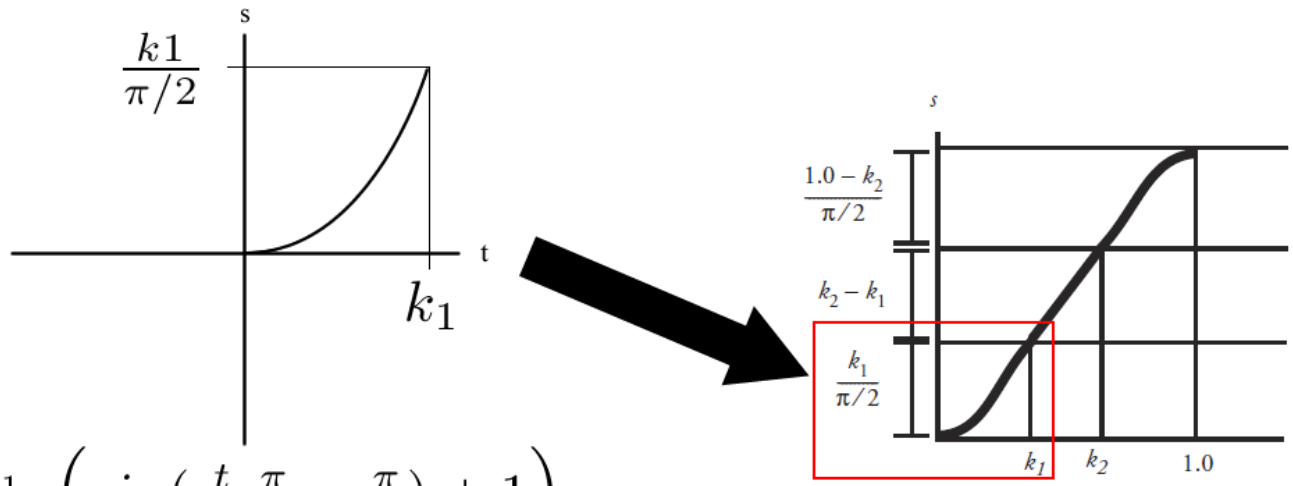
$$sin(\frac{t}{k_1}\frac{\pi}{2} - \frac{\pi}{2}) + 1 \quad \forall\ t = [0, k_1]$$

$$\frac{k_1}{\pi/2}\left(sin(\frac{t}{k_1}\frac{\pi}{2} - \frac{\pi}{2}) + 1\right)$$

$$\forall\ t = [0, k_1]$$

2.2.3 – EASE-IN/EASE-OUT

- La section d'accélération est maintenant complétée:

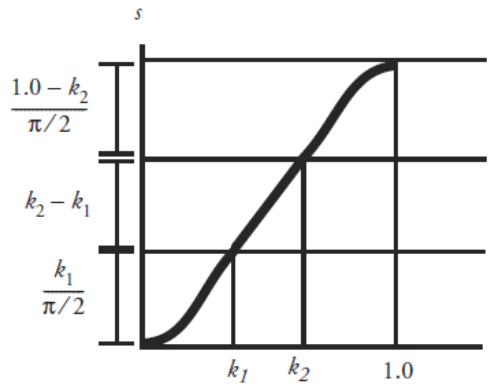


2.2.3 – EASE-IN/EASE-OUT

- La section de vitesse constante est en fait une droite.
  - Sa pente est égale à la dérivée de sin(t) lorsque t = 0, soit 1.

$$sin(0)' = cos(0) = 1$$

  - Le segment de droite est décallé de k_1 en **t** et de $k_1/(\pi/2)$ en **s**. (voir image →)



2.2.3 – EASE-IN/EASE-OUT

- On a donc une droite suivant la formule:

$$(\frac{k_1}{\pi/2} + t - k_1) \; \forall \; t = [k1, k2]$$

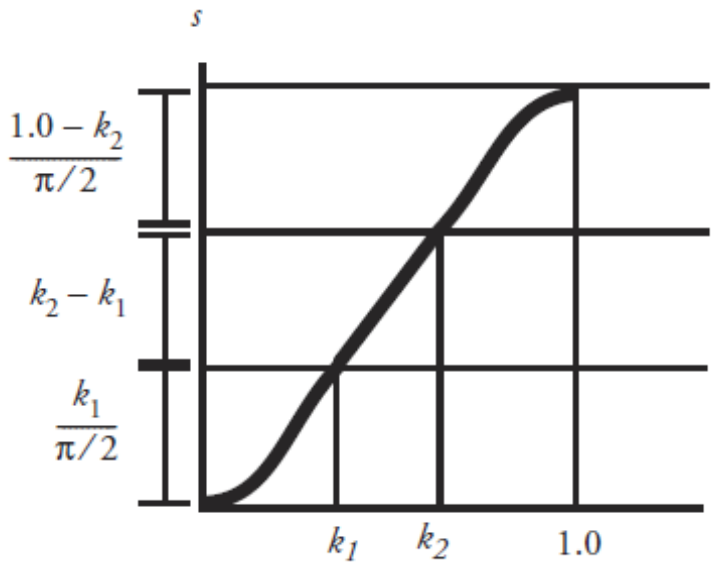
2.2.3 – EASE-IN/EASE-OUT

- Construction de la section de décélération
  - Revient au même que la section d'accélération, à quelques différences près:
    - On décale de $\mathbf{k_2}$ en **t.**
    - On décale de $\mathbf{k_1 / (\pi/2)} + \mathbf{k_2 - k_1}$ en **s**.
    - Le facteur de mise à l'échelle au niveau des axe est de $\mathbf{(1-k_2)/(\pi/2)}$.
- On se retrouve avec la formule:

$$\frac{k_1}{\pi/2} + k_2 - k_1 + \left((1-k_2)\frac{2}{\pi}\right) sin\left(\left(\frac{t-k_2}{1-k_2}\right)\frac{\pi}{2}\right) \quad \forall\, t = [k_2, 1]$$

2.2.3 – EASE-IN/EASE-OUT

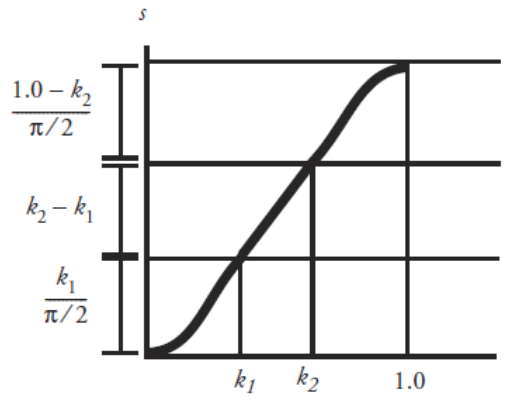
- Il reste un dernier détail à régler.
- Si on regarde le graphique, on remarque que l'interpolation évolue de 0 à 1 en **t** mais que ce n'est pas le cas en **s**.



2.2.3 – EASE-IN/EASE-OUT

- Pour régler ce problème, on normalise tous les résultats obtenus par la somme des éléments qui composent la progression sur **s**. Soit:

$$f = (k_1(2/\pi)) + (k_2 - k_1) + ((1 - k_2)(2/\pi))$$



2.2.3 – EASE-IN/EASE-OUT

- En résumé, une interpolation sinusoïdale par morceaux s'exprime telle que:

$$ease(t) = \left\{ \begin{array}{ll} \left(k_1 \frac{2}{\pi}(sin(\frac{t}{k_1}\frac{\pi}{2} - \frac{\pi}{2}) + 1)\right)/f & t = [0..k_1] \\ \left(\frac{k_1}{\pi/2} + t - k_1\right)/f & t = [k_1..k_2] \\ \left(\frac{k_1}{\pi/2} + k_2 - k_1 + ((1-k_2)\frac{2}{\pi})sin(\frac{t-k_2}{1-k_2}\frac{\pi}{2})\right)/f & t = [k_2..1] \end{array} \right\}$$

$$f = k_1 \frac{2}{\pi} + k_2 - k_1 + (1 - k_2)\frac{2}{\pi}$$

2.2.3 – EASE-IN/EASE-OUT

- La méthode d'interpolation sinusoïdale par morceaux règle la plupart des problèmes de l'interpolation sinusoïdale simple:
  - Phase de mouvement constant sans accélération/décélération.
  - Contrôle sur la durée de l'accélération et de la décélération.

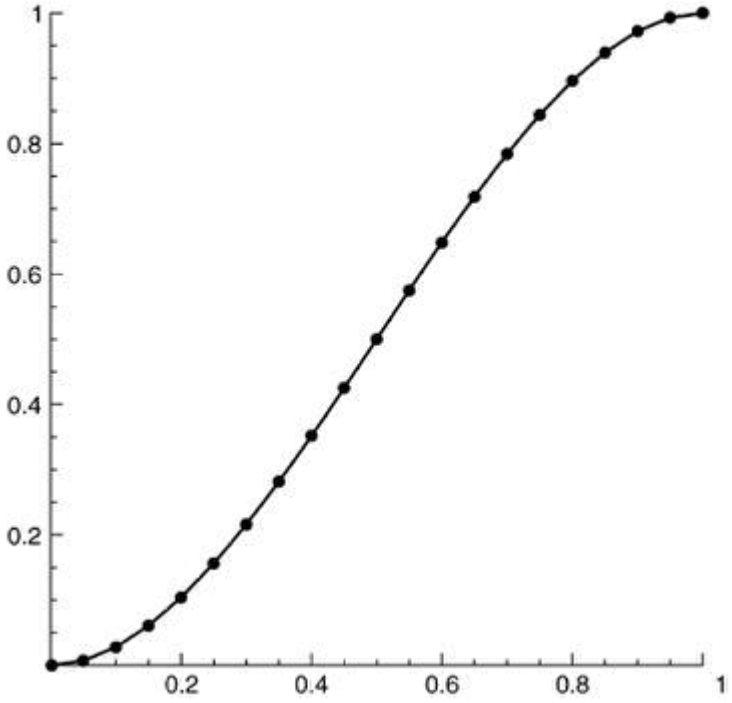
2.2.3 – EASE-IN/EASE-OUT

- Il reste malheureusement un petit défaut :
  - La fonction “sinus” est une fonction transcendante. (log, sin, cos, exp, etc. → fonction transcendante)
    - Elle ne peut pas être exprimée en une série finie de termes algébriques.
    - Requiert donc des fonctions “spéciales” (comme le sinus) qui sont plus difficile à manipuler et beaucoup de calcul.

2.2.3 – EASE-IN/EASE-OUT

- Méthode par polynôme cubique:
  - Possibilité de générer un polynôme cubique qui génère une courbe ayant les propriétés d'un "Ease-in/Ease-out".
  - Exemple: $s = -2t^3 + 3t^2$
  - Par contre, pas de section vitesse constante

2.2.3 – EASE-IN/EASE-OUT



Manuel Figure 3.14

$$s = -2t^3 + 3t^2$$

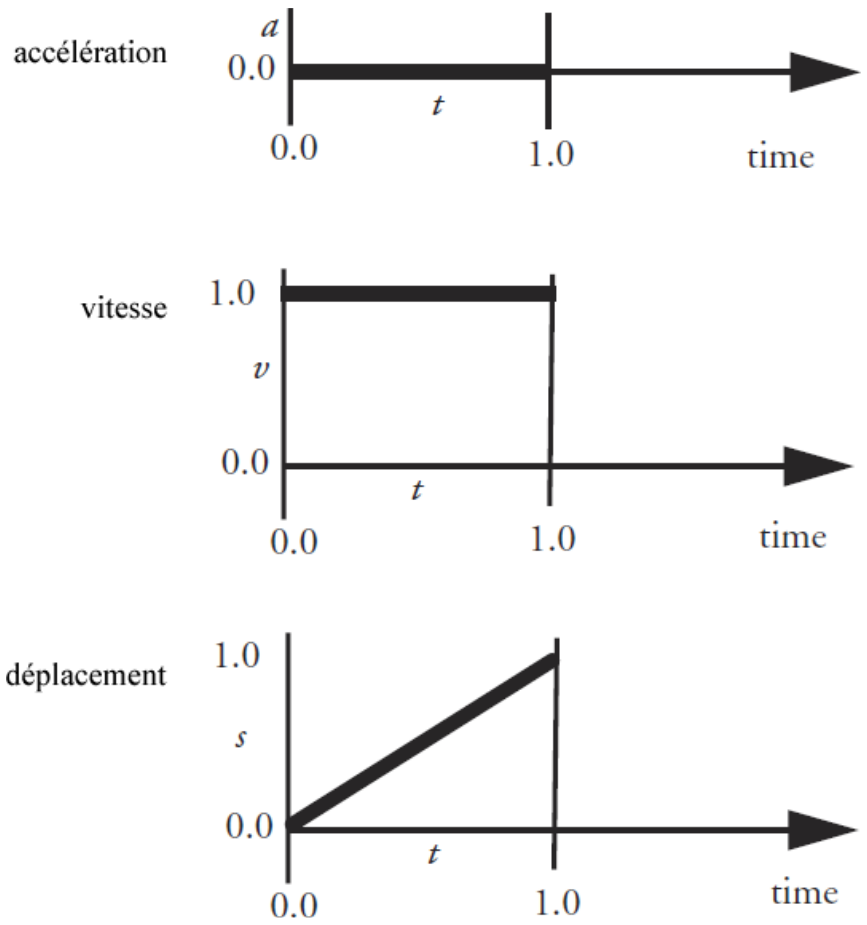
2.2.3 – EASE-IN/EASE-OUT

- 4e et dernière méthode : Construction parabolique

- On considère 3 façons d'exprimer un déplacement:
  1. Accélération en fonction du temps.
  2. Vitesse en fonction du temps.
  3. Distance en fonction du temps.

2.2.3 – EASE-IN/EASE-OUT

- Pour un déplacement à vitesse constante on aurait donc:



Manuel Figure 3.15

2.2.3 – EASE-IN/EASE-OUT

- Petit rappel de mécanique classique :

1. L'intégrale indéfinie d'une fonction d'accélération donne la fonction de vitesse correspondante.

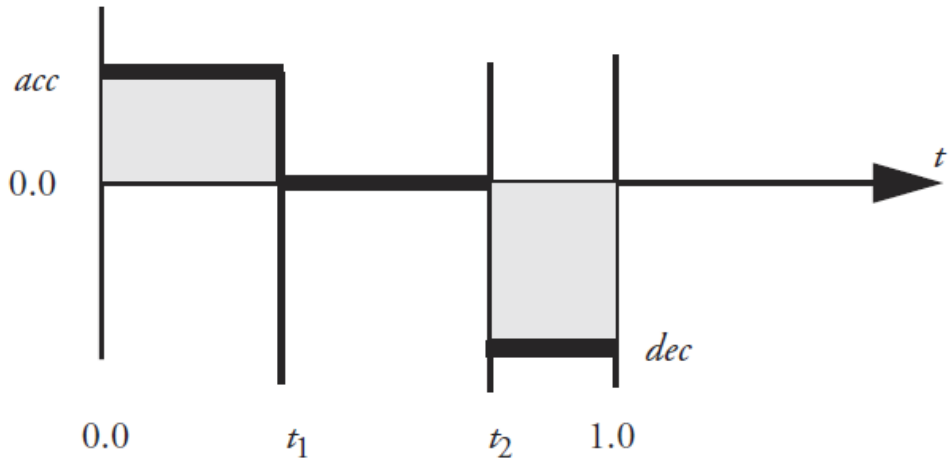
$$v(t) = \int a(t)dt$$

2. L'intégrale indéfinie d'une fonction de vitesse donne la fonction de déplacement correspondante

$$s(t) = \int v(t)dt$$

2.2.3 – EASE-IN/EASE-OUT

- On peut exprimer un déplacement avec un graphique d'accélération :



$$a = acc \quad 0 < t < t_1$$
$$a = 0.0 \quad t_1 < t < t_2$$
$$a = dec \quad t_2 < t < 1.0$$

Manuel Figure 3.16

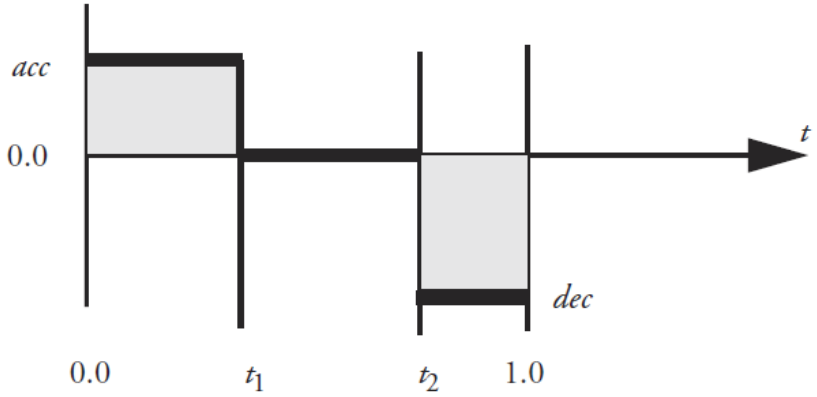
2.2.3 – EASE-IN/EASE-OUT

- Si l’aire sous la courbe à l’accélération est égale à l’aire sous la courbe à la décélération:
  - La vitesse initiale est égale à la vitesse finale
  - Si la vitesse initiale/finale est 0, on a un ease-in/ease-out.

2.2.3 – EASE-IN/EASE-OUT

- En d'autres mots, il est possible d'exprimer un ease-in, ease-out avec 4 paramètres, soit:

1. acc
2. t_1
3. t_2
4. dec



Manuel Figure 3.16

- Problème: Difficile de respecter les contraintes d'aire en assignant manuellement les valeurs aux variables.

2.2.3 – EASE-IN/EASE-OUT

- Solution:
  - On fixe 3 des 4 paramètres manuellement.
  - On laisse l'ordinateur résoudre le système (avec respect des contraintes) pour déterminer le paramètre restant.

2.2.3 – EASE-IN/EASE-OUT

- **Exemple:** Taux d'accélération (**acc**), la durée de l'accélération ($\mathbf{t_1}$) et le début de la décélération ($\mathbf{t_2}$) sont fixes. On cherche une décélération (**dec**>0) qui respecte les contraintes de vitesse initiale/finale.
  - Pour respecter la contrainte, on doit avoir:

$$(t_1)(acc) = (1 - t_2)(dec)$$

  - On connait $\mathbf{t_1}$, $\mathbf{t_2}$ et **acc**, on résoud pour trouver **dec**.

$$\frac{(t_1)(acc)}{1-t_2} = (dec)$$

← On exprime en terme d'aire mais il ne faut pas oublier que dec est appliqué négativement sur la vitesse.

2.2.3 – EASE-IN/EASE-OUT

- À partir du graphe d'accélération (fonction temps-accélération), on peut extraire le graphe de vitesse (fonction temps-vitesse) en faisant l'intégrale de l'accélération.

$$
\begin{aligned}
v(t) &= \int (acc)d(t-0) & \\
&= (acc)(t-0)+c_1 & \\
&= (acc)(t)+c_1 & 0<t<t_1
\end{aligned}
$$

$$
\begin{aligned}
v(t) &= \int (0)d(t-t_1) & \\
&= (0)(t-t_1)+c_2 & \\
&= c_2 & t_1<t<t_2
\end{aligned}
$$

$$
\begin{aligned}
v(t) &= \int -(dec)d(t-t_2) & \\
&= -(dec)(t-t_2)+c_3 & t_2<t<1
\end{aligned}
$$

2.2.3 – EASE-IN/EASE-OUT

- **c_1** correspond à la vitesse initiale. On a donc:

$$c_1 = 0$$

- **c_2** et **c_3** correspondent à la vitesse constante:

$$c_2 = c_3 = (acc)(t_1) = v_0$$

Cette vitesse est la vitesse constante du ease-in/ease-out. On la note **v_0**.

2.2.3 – EASE-IN/EASE-OUT

- En ajoutant nos constantes aux équations précédemment obtenues, on a :

$$\begin{aligned} v(t) &= (acc)(t) + c_1 \\ &= (acc)(t) \qquad 0 < t < t_1 \end{aligned}$$

$$\begin{aligned} v(t) &= c_2 \\ &= v_0 \qquad t_1 < t < t_2 \end{aligned}$$

$$\begin{aligned} v(t) &= -(dec)(t - t_2) + c_3 \\ &= -(dec)(t - t_2) + v_0 \qquad t_2 < t < 1 \end{aligned}$$

2.2.3- EASE-IN/EASE-OUT

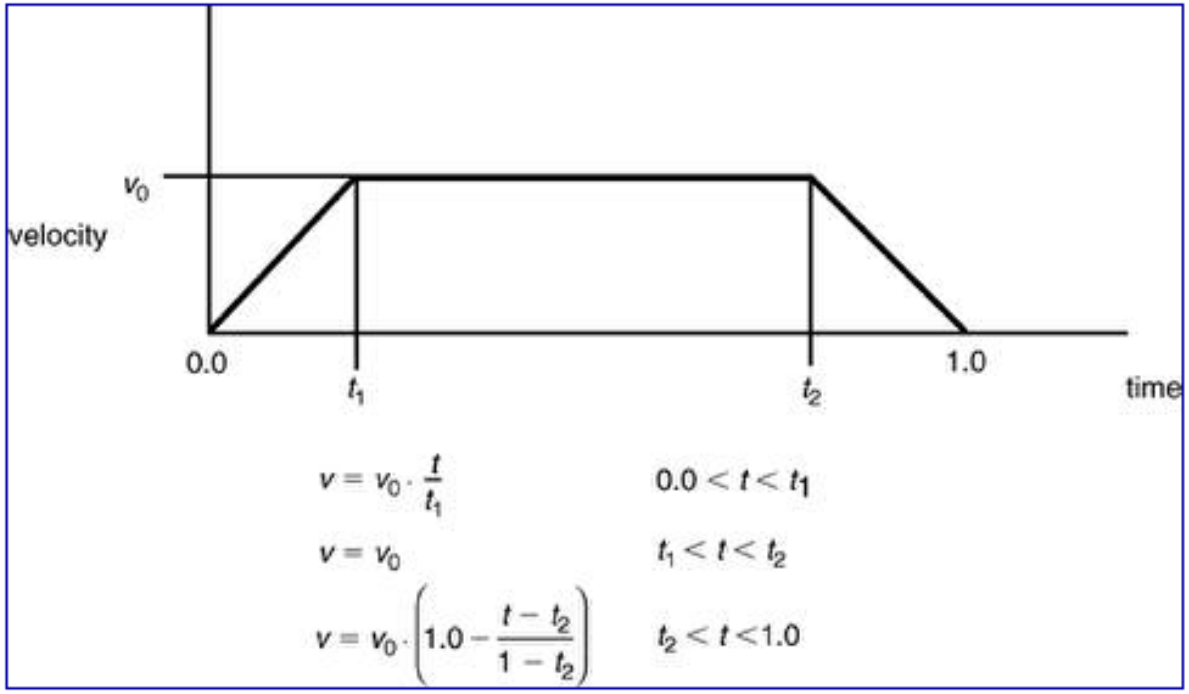
- On peut encore plus simplifier les expressions en réutilisant deux égalités :

$$c_2 = (acc)(t_1) = v_0 \quad \text{et} \quad \frac{(t_1)(acc)}{1-t_2} = (dec)$$

- Pour ensuite obtenir la forme suivante possédant 3 paramètres:

$$\begin{aligned} v &= v_0\left(\frac{t}{t_1}\right) && 0.0 < t < t_1 \\ v &= v_0 && t_1 < t < t_2 \\ v &= v_0\left(1.0 - \frac{t-t_2}{1-t_2}\right) && t_2 < t < 1 \end{aligned}$$

2.2.3- EASE-IN/EASE-OUT



Manuel Figure 3.17

2.2.3 – EASE-IN/EASE-OUT

- La courbe de vitesse peut donc être formée avec 3 paramètres, soit v_0, t_1 et t_2.

- L'aire sous la courbe (intégrale) représente le déplacement total. Elle est donnée tel que:

$$1 = \frac{1}{2}v_0 t_1 + v_0(t_2 - t_1) + \frac{1}{2}v_0(1 - t_2)$$

(On se rappelle que le déplacement est normalisé de 0 à 1, d'où la somme = 1)

2.2.3 – EASE-IN/EASE-OUT

- Comme c'était le cas pour les courbes d'accélération, on peut spécifier 2 paramètres et déduire le 3e.

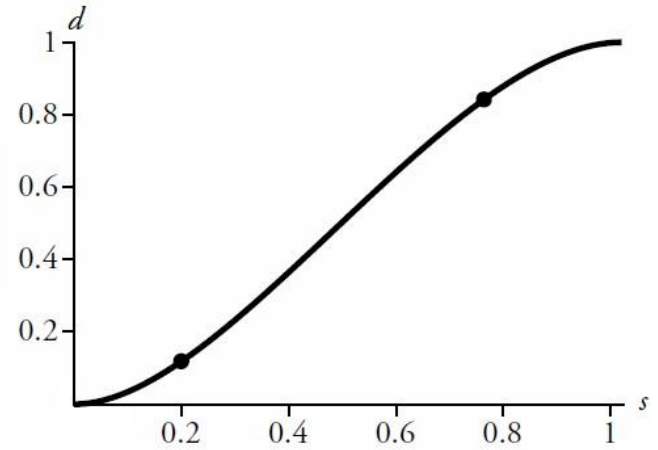
- Exemple: $\mathbf{t_1}$ et $\mathbf{t_2}$ sont spécifiés. On cherche $\mathbf{v_0}$:

$$
\begin{aligned}
1 &= \tfrac{1}{2}v_0 t_1 + v_0(t_2 - t_1) + \tfrac{1}{2}v_0(1 - t_2) \\
&= v_0\left(\tfrac{1}{2}t_1 + t_2 - t_1 + \tfrac{1}{2}(1 - t_2)\right) \\
&= v_0\left(-\tfrac{1}{2}t_1 + \tfrac{1}{2}t_2 + \tfrac{1}{2}\right) \\
&= v_0 \tfrac{t_2 - t_1 + 1}{2}
\end{aligned}
$$

$$
v_0 = \frac{2}{t_2 - t_1 + 1}
$$

2.2.3 – EASE-IN/EASE-OUT

- Si on intègre une dernière fois, on obtient la courbe distance/temps : (Selon le même principe que l'intégration précédente.)



Manuel Figure 3.18

$$d = v_0 \cdot \frac{t^2}{2 \cdot t_1} \qquad 0.0 < t < t_1$$

$$d = v_0 \cdot \frac{t_1}{2} + v_0 \cdot (t - t_1) \qquad t_1 < t < t_2$$

$$d = v_0 \cdot \frac{t_1}{2} + v_0 \cdot (t_2 - t_1) + \left(v_0 - \frac{\left(v_0 \cdot \frac{t - t_2}{1 - t_2} \right)}{2} \right) \cdot (t - t_2) \qquad t_2 < t < 1.0$$

2.2.3 – EASE-IN/EASE-OUT

- Avantages d'une construction parabolique:
  - Calculs relativement simples une fois le système développé.
  - Évite les fonctions transcendantes
  - Permet de représenter la même interpolation de plusieurs façons (accélération/vitesse/distance)

2.2.3 – EASE-IN/EASE-OUT

- Désavantages d'une construction parabolique:
  - Plus complexe à comprendre.
  - Peut être relativement long à implémenter si on veut le faire de façon complète. (toutes les représentations, tous les inconnus possibles, etc.)

SOMMAIRE

- Ease-in/Ease-out
  - Vitesse initiale et vitesse finale → 0
  - Vitesse médiane → constant

- Il existe différentes façons de créer un Ease-in/Ease-Out.
  - Interpolation sinusoïdale
  - Interpolation sinusoïdale par morceaux
  - Interpolation par un polynôme cubique
  - Construction parabolique
    - Accélération vs temps
    - Vitesse vs temps
    - Déplacement vs temps
    - Intégrales, contraintes d'aire sous la courbe, etc.

FONCTIONS DISTANCE-TEMPS GÉNÉRALES

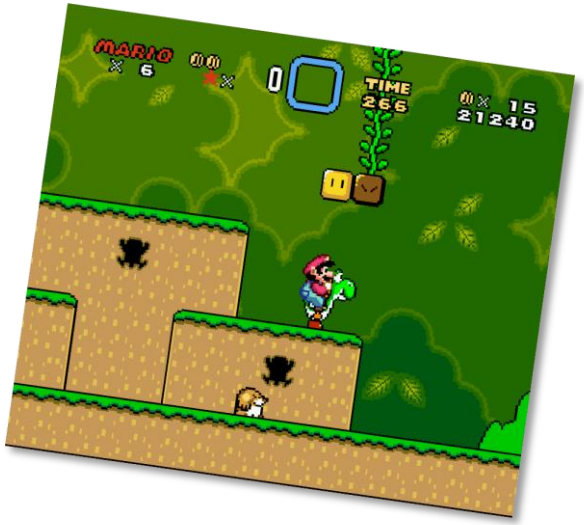
2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

- Tel que vu à la section précédente, on peut exprimer, entre autre, un déplacement selon:
  - L'accélération en fonction du temps
  - La vitesse en fonction du temps

- En ne limitant pas l'air sous la courbe de ces fonctions, le déplacement n'est lui-même pas limité.
  - Le déplacement n'a pas de valeur fixe en fonction du temps.

2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

- Dans les faits, en animation, le déplacement est presque toujours limité en fonction du temps.



- Ex :
  - Personnage court 2 unités/seconde
  - Main attrapant un objet lancé → position de la main = position de l'objet à un temps **t**.

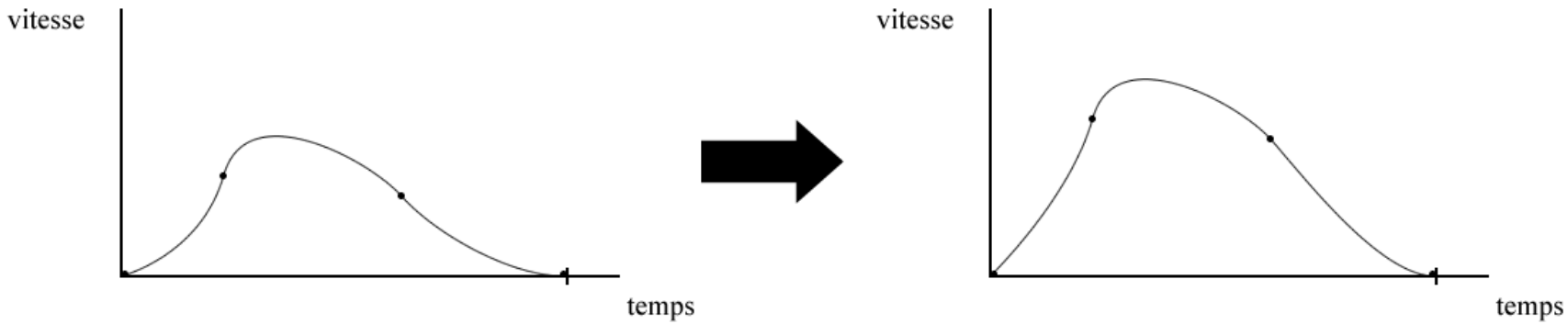
2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

- En contraignant la distance en fonction du temps, on contraint la vitesse moyenne.

- Ceci ajoute une contrainte à l'expression du déplacement pour:
  - Vitesse vs temps
  - Accélération vs temps.

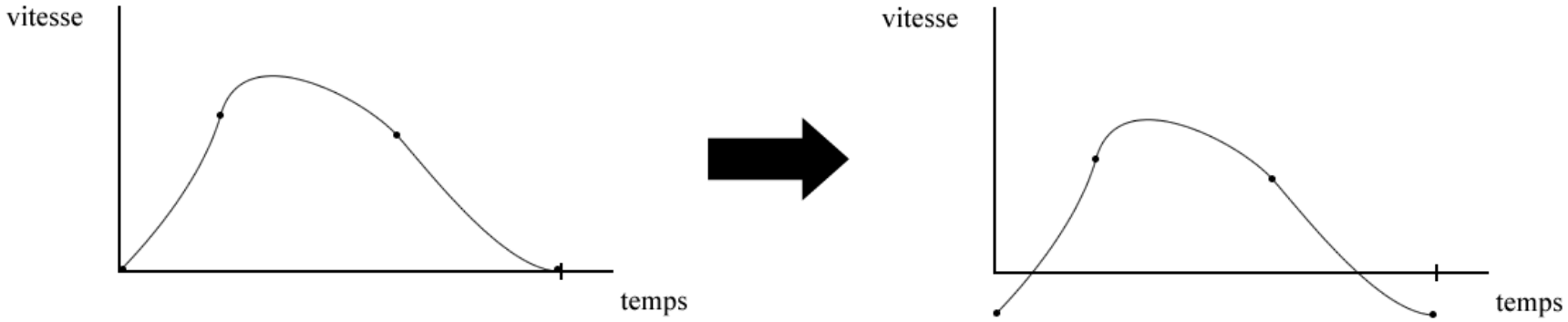
2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

- Exemple: Vitesse vs temps
  - Augmentation de la vitesse:
    - Problème : La contrainte de déplacement fixe n'est plus respectée



2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

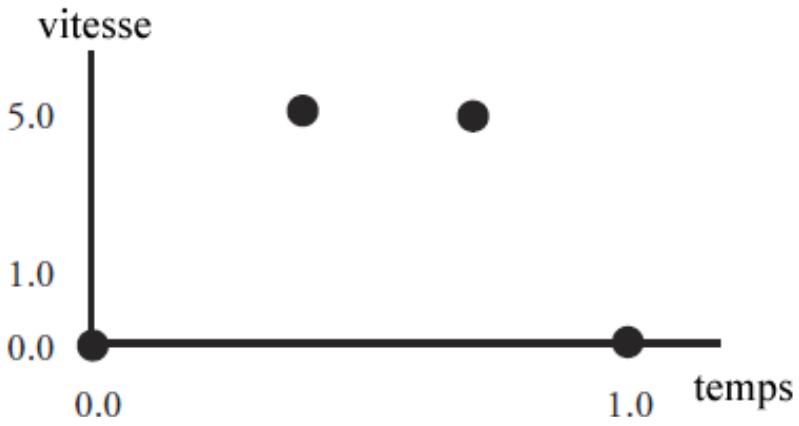
- Solution:
  - Diminuer globalement la vitesse pour rétablir la contrainte de déplacement.

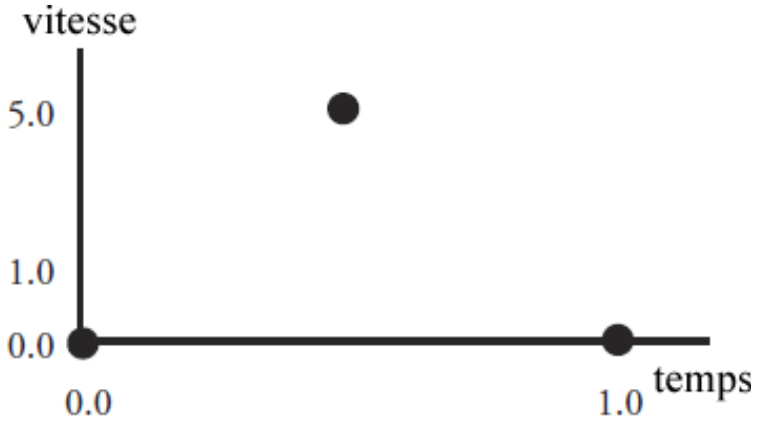


- Problèmes :
  - Chaque fois qu'on modifie la vitesse à un point, toutes les autres vitesses sont altérées.

2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

- Approche différente:
  - Demander à l'utilisateur de fixer lui-même la vitesse à différents point et générer les courbes reliant ces points pour respecter la contrainte.

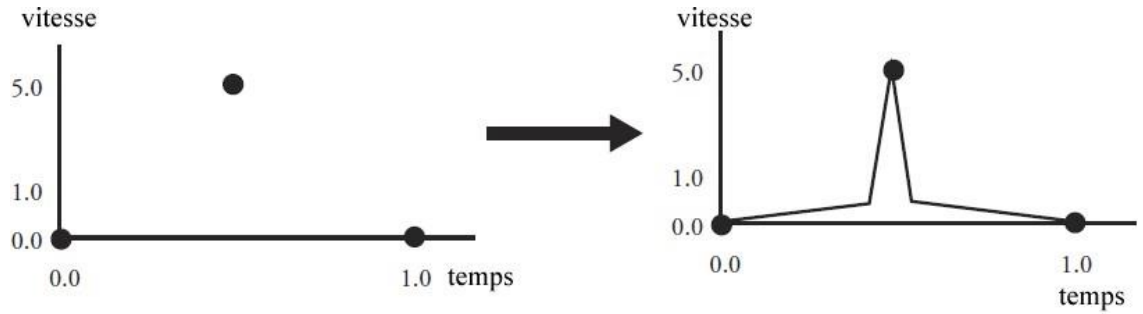


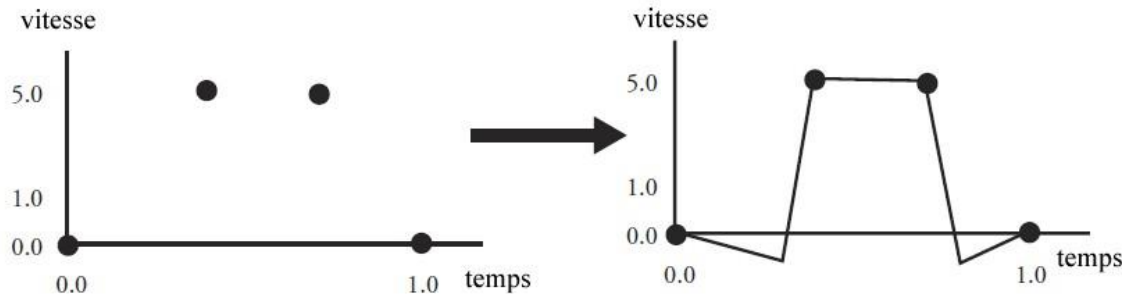


Manuel Figure 3.19

2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

- Problème (encore):
  - Aucune garantie que la courbe générée correspondra au mouvement souhaité.





Manuel Figure 3.19

2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

- On réalise rapidement que les systèmes vitesse/temps, accélération/temps sont plutôt limités dans certaines situations.

- On préfère généralement l'utilisation de courbes distance/temps.

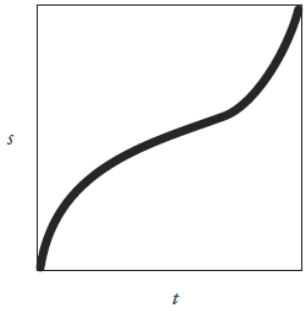
2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

- Pour un espace de temps et de distance normalisé de 0 à 1, le même déplacement possède des contraintes faciles à suivre:
  - Commencer à (0,0)
  - Terminer à (1,1)
  - (Le reste de la courbe peut prendre n'importe quelle forme.)

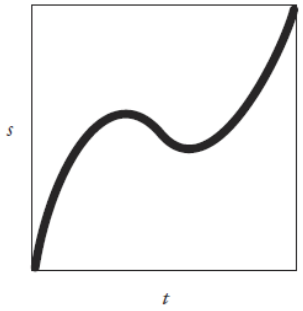
- Les utilisateurs préfèrent donc souvent l'utilisation de courbe distance/temps.

2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

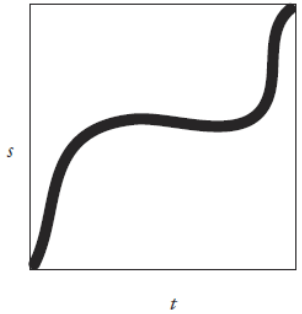
- Exemples de courbe distance/temps qui suivent les dites contraintes:



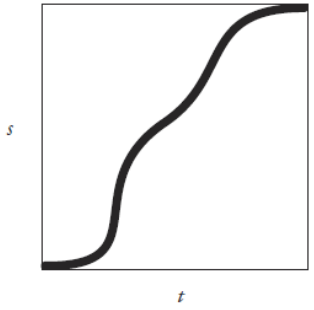
Début/Fin abruptes



Progression "recule" sur la courbe



Arrêt pendant le déplacement

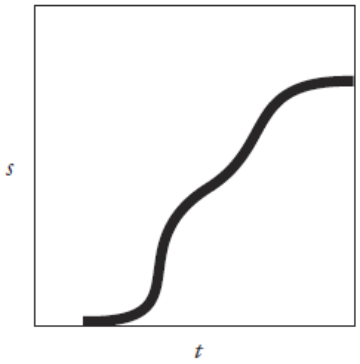


Début/Fin lente

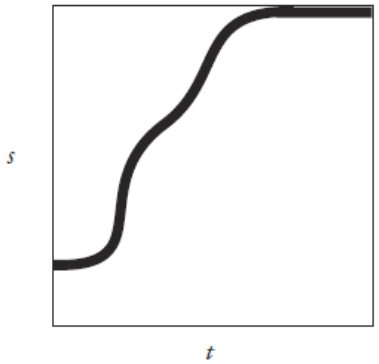
Manuel Figure 3.20

2.2.4 – FNC. DISTANCE-TEMPS GÉNÉRALES

- Autre avantage des fonctions distances/temps:
  - Permettent de générer des déplacements partiels en espace et en temps.



Pause avant de commencer.
Arrête le déplacement avant la fin.



Le déplacement commence le long de la courbe
Atteint la fin avant t = 1.0

  - Note: Les contraintes de début à (0,0) et de fin à (1,1) ne sont cependant plus respectées.

PAIRES DE POSITION/TEMPS

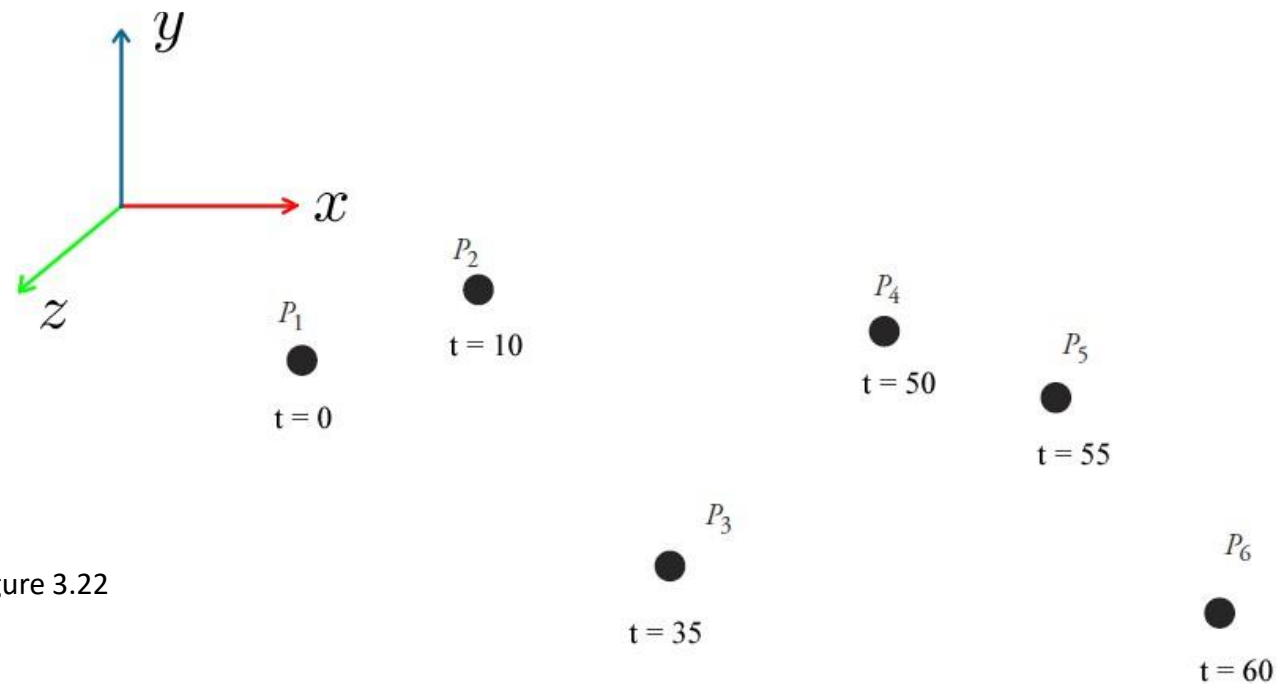
2.2.5 – PAIRES DE POSITION/TEMPS

- Nous avons vu comment atteindre des déplacement particuliers le long d'une courbe. (Donc en faisant uniquement varier le paramètre **s** sur la courbe.)

- On peut aborder le problème différemment:
  - Comment faire pour interpoler entre une série de positions spatiales **P** (en 3D) précises à des moments **t** précis?

2.2.5 – PAIRES DE POSITION/TEMPS

- L'utilisateur passe donc une série de points situés dans l'espace ayant chacun une contrainte de temps :



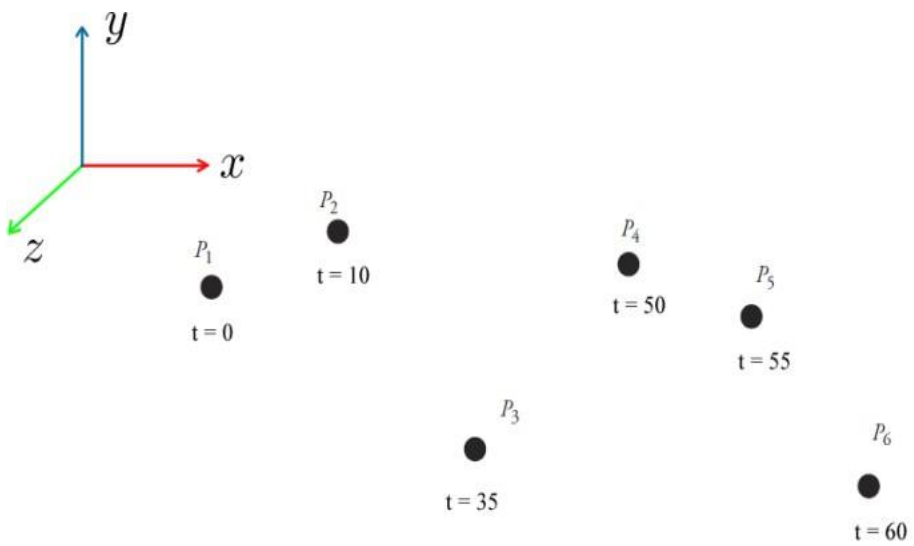
Manuel Figure 3.22

2.2.5 – PAIRES DE POSITION/TEMPS

- Si l'animateur a des contraintes de position spécifiques qui doivent être satisfaites à des moments spécifiques, la courbe d'espace paramétrée dans le temps peut être déterminée directement.
  - Les paires temps-position peuvent être spécifiées par un animateur, comme dans la figure 3.22
  - les points de contrôle qui produisent une courbe d'interpolation peuvent être calculés à partir de cette information

2.2.5 – PAIRES DE POSITION/TEMPS

- Exemple:
  - Relier les points avec un B-Spline de degré 5.



Manuel Figure 3.22

2.2.5 – PAIRES DE POSITION/TEMPS

- Un B-Spline peut être exprimé sous sa forme de base.
  - Pour un B-Spline de degré **k**, ayant **n+1** points de contrôle, on a :

$$P(t) = \sum_{i=1}^{n+1} B_i N_{i,k}(t)$$

  - Où $\mathbf{B_i}$ est le **i**ème point de contrôle (inconnu).
  - Et $\mathbf{N_{i,k}(t)}$ est la fonction de mélange d'un B-Spline pour le **i**ème point de contrôle.

2.2.5 – PAIRES DE POSITION/TEMPS

- Pour un point $\mathbf{P}_j$ donné, si on développe la sommation on obtient:

$$P_j = N_{1,k}(t_j)B_1 + N_{2,k}(t_j)B_2 + ... + N_{n+1,k}(t_j)B_{n+1}$$

- Hors, nous avons dans l'exemple 6 points pour un B-Spline de degré 5, soit:

$$\begin{aligned}
P_1 &= N_{1,6}(t_1)B_1 + N_{2,6}(t_1)B_2 + ... + N_{6,6}(t_1)B_6 \\
P_2 &= N_{1,6}(t_2)B_1 + N_{2,6}(t_2)B_2 + ... + N_{6,6}(t_2)B_6 \\
&... \\
P_6 &= N_{1,6}(t_6)B_1 + N_{2,6}(t_6)B_2 + ... + N_{6,6}(t_6)B_6
\end{aligned}$$

2.2.5 – PAIRES DE POSITION/TEMPS

$$
\begin{aligned}
P_1 &= N_{1,6}(t_1)B_1 + N_{2,6}(t_1)B_2 + ... + N_{6,6}(t_1)B_6 \\
P_2 &= N_{1,6}(t_2)B_1 + N_{2,6}(t_2)B_2 + ... + N_{6,6}(t_2)B_6 \\
&... \\
P_6 &= N_{1,6}(t_6)B_1 + N_{2,6}(t_6)B_2 + ... + N_{6,6}(t_6)B_6
\end{aligned}
$$

- En considérant les $\mathbf{N_{i,k}(t_i)}$ séparément des $\mathbf{B_i}$, on peut former un système matriciel, soit:

$$P = NB$$

- Où
  - P → matrice-colonne contenant les points dans l'espace.
  - N → matrice contenant tous les $\mathbf{N_{i,k}(t_i)}$ évalués.
  - B → matrice-colonne contenant les points de contrôle.

2.2.5 – PAIRES DE POSITION/TEMPS

- De ce système, on connait **P** et **N**. (Grâce aux paires position/temps).
  - Il ne reste qu'à trouver **B**, soit les points de contrôle qui génèrent une courbe passant par toutes les paires position/temps.

$$B = N^{-1}P$$

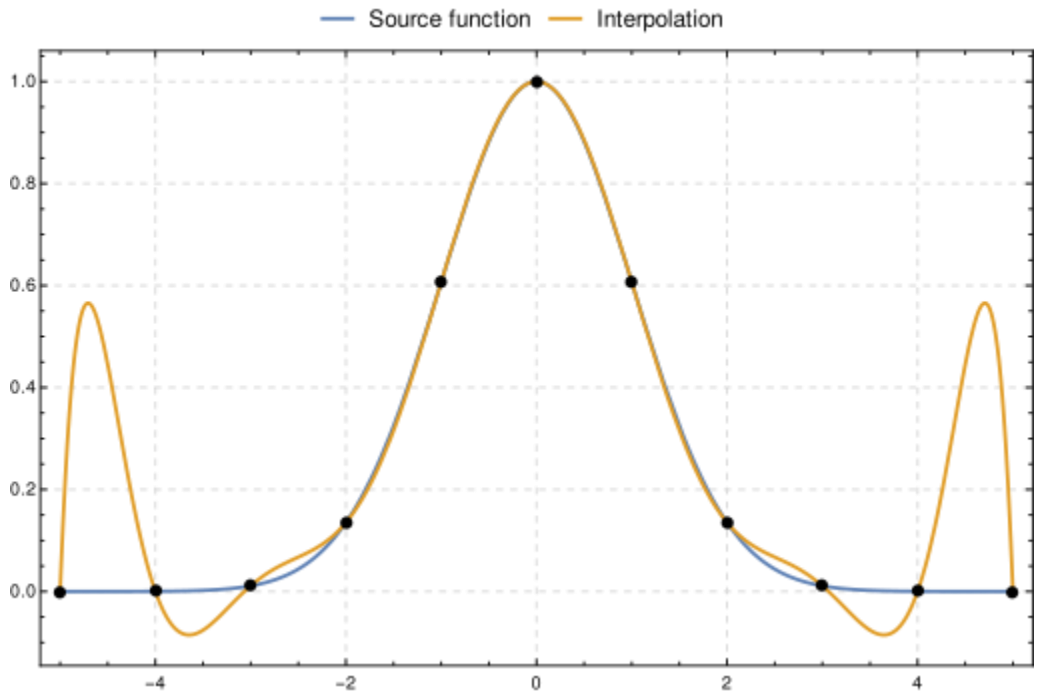
  - Dans notre cas, nous avons 6 points pour une courbe de degré 5 (donc 6 pts de contrôle). N est donc carré et facilement inversible pour résoudre le système.

2.2.5 – PAIRES DE POSITION/TEMPS

- Hors, avoir un nombre trop élevé de points de contrôle peut générer des fluctuations indésirées dans la courbe.
  - On réduit le nombre de point de contrôles à trouver pour générer la courbe.
    - Matrice rectangulaire → pseudo inverse

$$B = [N^T N]^{-1} N^T P$$

2.2.5 – PAIRES DE POSITION/TEMPS



Fluctuations indésirées dans la courbe, en particulier lorsque le polynôme est de degré élevé (degré 10 ici).

2.2.5 – PAIRES DE POSITION/TEMPS

- Note:
  - Si on réduit le nombre de point de contrôle, il est possible de ne pas pouvoir résoudre le système.
  - On dit dans ce cas que le système est **sur-contraint**.
  - On peut aussi utiliser la méthode des moindres carrés (peut résulter en une approximation mais c'est plus stable numériquement).

INTERPOLATION D'ORIENTATIONS

2.3 - INTERPOLATION D'ORIENTATIONS

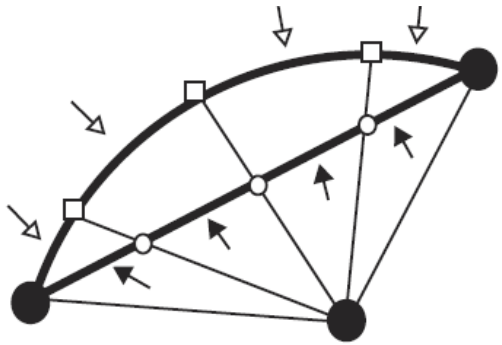
- Nous avons vu plusieurs façons de représenter une orientation:
  - Matrices de transformation
  - Rotation à angles fixes
  - Rotation à angles d'Euler
  - Angle et Axe
  - Quaternions

2.3 - INTERPOLATION D'ORIENTATIONS

- L'utilisation des quaternions était favorisée pour les raisons suivantes:
  - Pas de Gimbal Lock
  - Léger en temps de calcul
  - Facilement convertible en matrice de transformation.
  - Facile d'interpoler avec les Slerp.

2.3.1 – INTERPOLATION DE QUATERNIONS

- Pourquoi ne pas simplement interpoler les nombres linéairement?
  - Interpolons entre 2 valeurs situées sur l'hypersphère des quaternions.
  - Problème:
    - Inégalité de la vitesse d'interpolation.



○ Interpolation linéaire
□ Point résultant sur l'hypersphère
→ Intervales égaux sur l'interpolation linéaire
⇢ Intervales inégaux sur l'hypersphère

Manuel Figure 3.23

2.3.1 – INTERPOLATION DE QUATERNIONS

- Solution: Les slerps (**S**pherical **l**inear int**erp**olation)

  - Rappel de la formule:

$$cos(\theta) = q_1 \cdot q_2 = s_1 s_2 + v_1 \cdot v_2$$

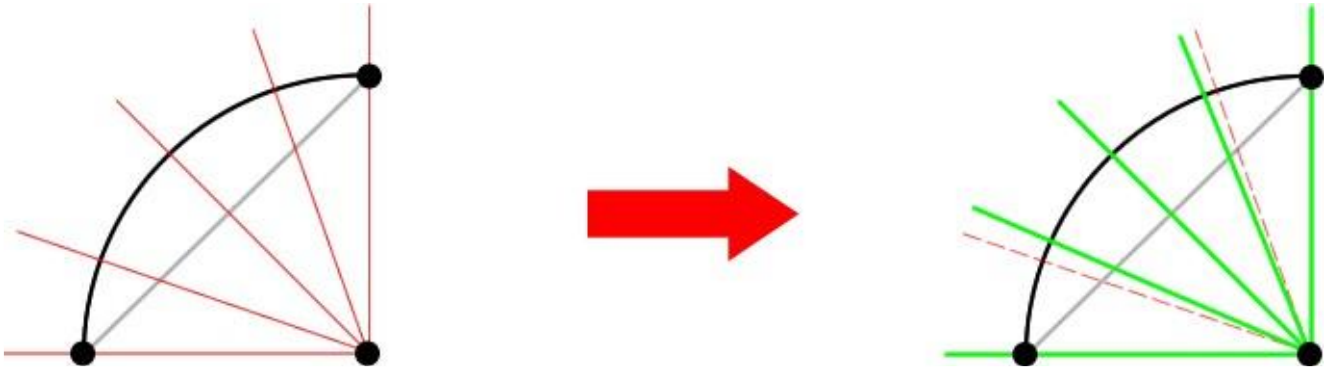
$$slerp(q_1, q_2, u) = \frac{sin((1-u)\theta)}{sin(\theta)} q_1 + \frac{sin(u\theta)}{sin(\theta)} q_2$$

(**q**$_1$ et **q**$_2$ sont normalisés)

2.3.1 – INTERPOLATION DE QUATERNIONS

$$slerp(q_1, q_2, u) = \frac{sin((1-u)\theta)}{sin(\theta)} q_1 + \frac{sin(u\theta)}{sin(\theta)} q_2$$

- Fonctionnement:
  - Le sinus permet de faire évoluer l'interpolation pour qu'elle soit constante sur un arc circulaire. (Ce qui correspond l'interpolation sur l'hypersphère d'un quaternion.)



2.3.1 – INTERPOLATION DE QUATERNIONS

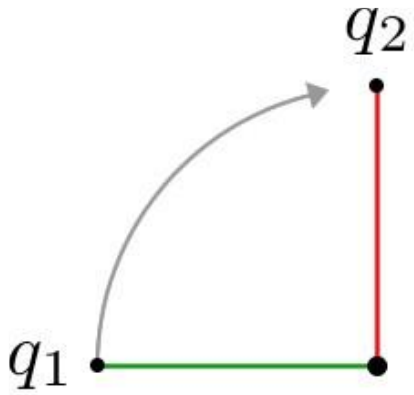
- Direction d'interpolation linéaire sphérique
  - On se souvient que pour une orientation donnée:

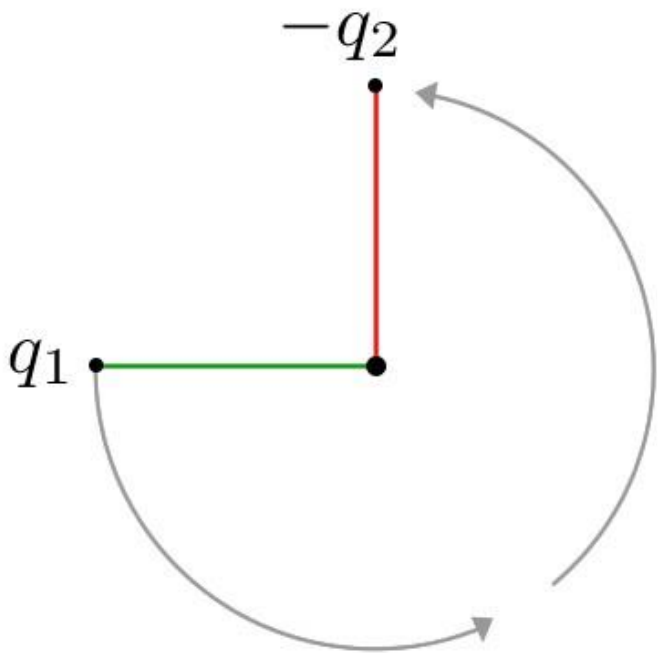
$$q = -q$$

  - On peut donc dire qu'une interpolation de $\mathbf{q_1}$ à $\mathbf{q_2}$ est égale à une interpolation entre $\mathbf{q_1}$ et $\mathbf{-q_2}$.

2.3.1 – INTERPOLATION DE QUATERNIONS

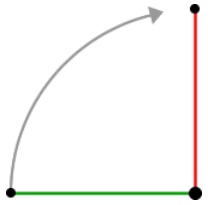
- Direction d'interpolation linéaire sphérique.
  - Ceci est vrai mais la direction dans laquelle se fait l'interpolation change.



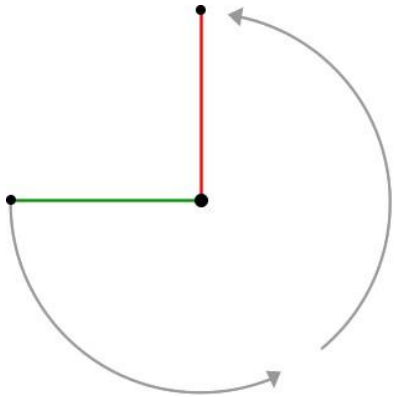


2.3.1 – INTERPOLATION DE QUATERNIONS

- Pour déterminer si l'interpolation utilisera le grand angle ou le petit angle:
  - On utilise le produit scalaire entre 2 quaternions.
  - Si le résultat est > à 0.
    - L'interpolation suit le petit angle.



  - Si le résultat est < à 0
    - L'interpolation suit le grand angle.

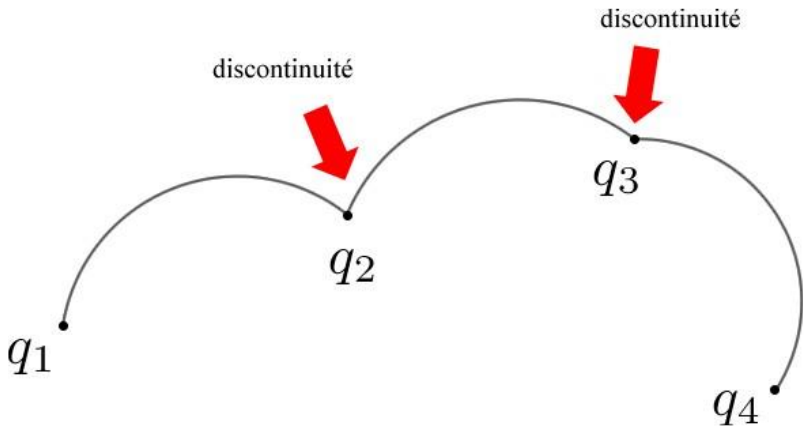


2.3.1 – INTERPOLATION DE QUATERNIONS

- Une fois la direction vérifiée, on peut simplement inverser le signe du second quaternion pour changer la direction si elle ne convient pas.
- L’interpolation avec le petit angle est généralement choisie.

2.3.1 – INTERPOLATION DE QUATERNIONS

- Problème avec les slerp:
  - Slerp → Interpolation à l'aide d'arcs circulaires sur une hypersphère à 4 dimensions.
  - Problème: De multiples interpolations sphériques linéaires ne garantissent pas une continuité **C1**.

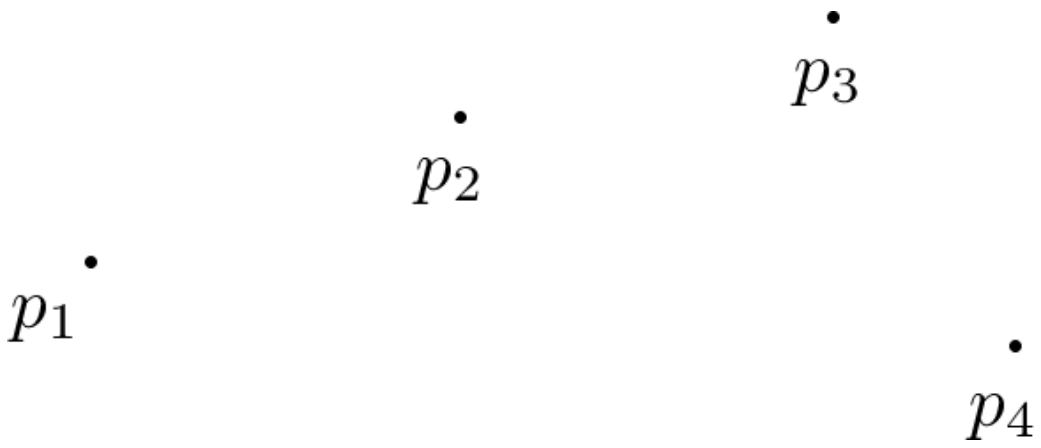


2.3.1 – INTERPOLATION DE QUATERNIONS

- Solution proposée:
  - Remplacer l'interpolation sphérique linéaire par une courbe de Bézier cubique.
    - On défini une courbe de Bézier entre chaque paire d'interpolation.
    - On génère d'autres points de contrôle automatiquement pour chacune des paires.

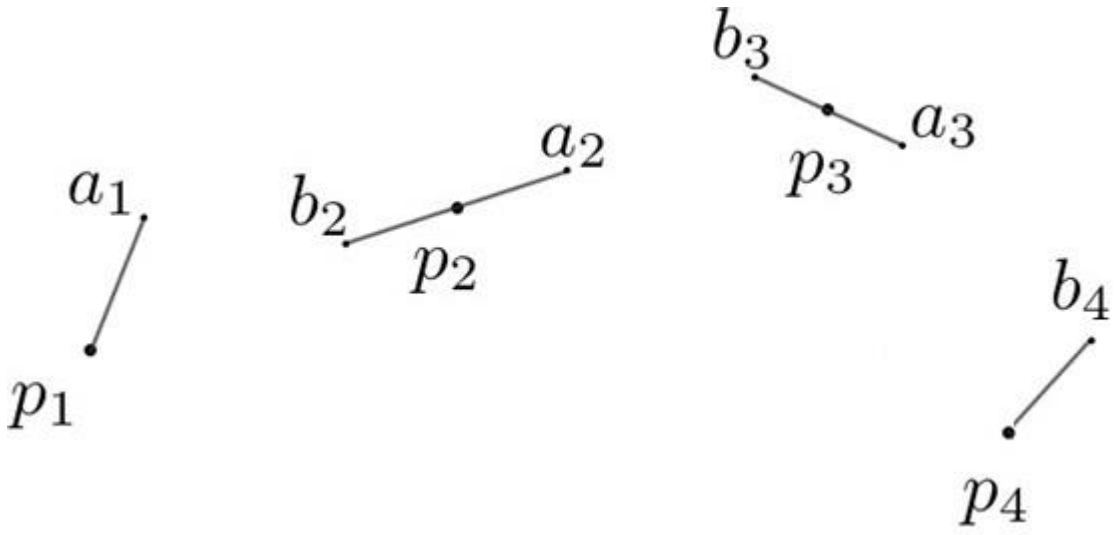
2.3.1 – INTERPOLATION DE QUATERNIONS

- On reprend l'exemple précédent:
  - (Les quaternions sont remplacés par des points 2D le temps de l'explication, on verra après comment procéder avec les quaternions.)



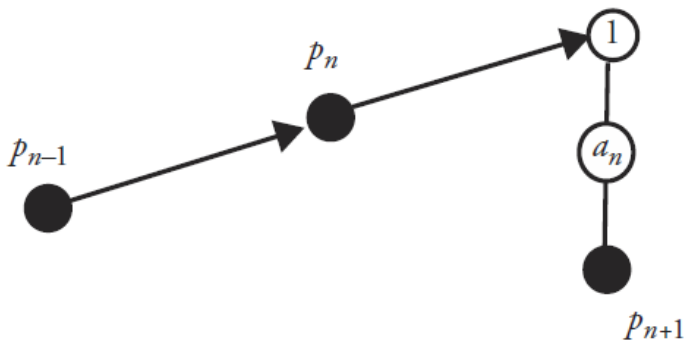
2.3.1 – INTERPOLATION DE QUATERNIONS

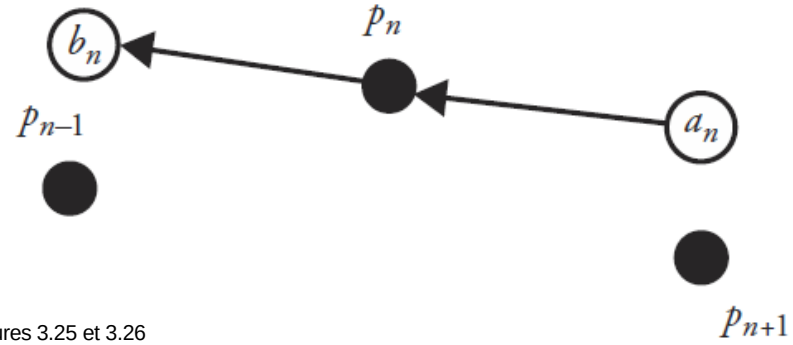
- Pour générer une courbe de Bézier cubique, on doit d'abord générer les points de contrôles.
  - Les points de contrôle avant ($\mathbf{b}_n$) et après ($\mathbf{a}_n$) chaque point doivent être diamétralement opposés pour préserver la continuité **C1**.



2.3.1 – INTERPOLATION DE QUATERNIONS

- On utilise une heuristique pour générer les points de contrôle.
  - Par exemple :





Manuel Figures 3.25 et 3.26

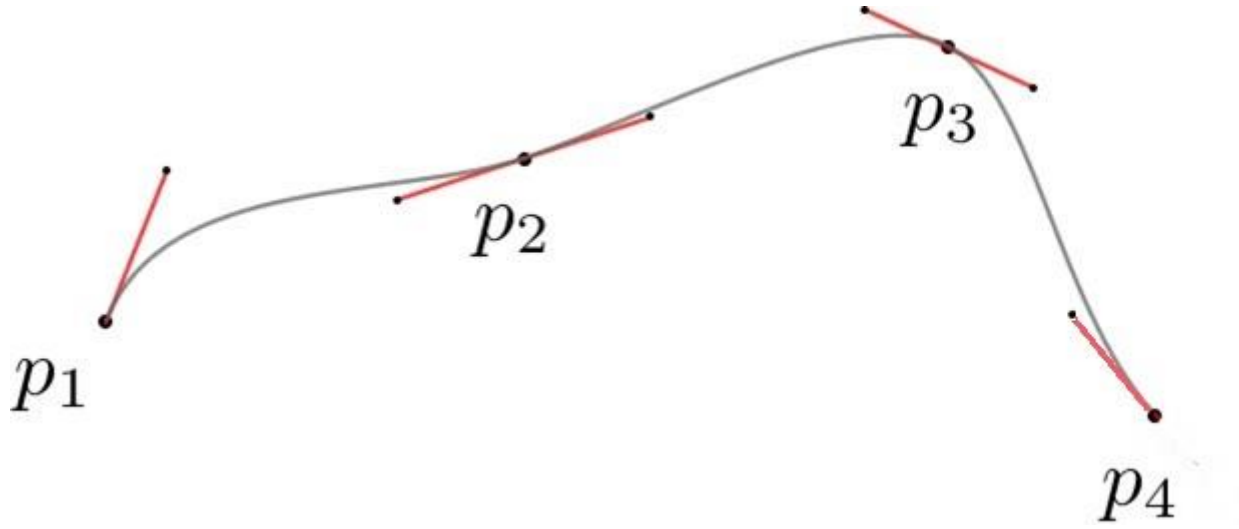
$$pt\ 1\ encerclé = P_n + (P_n - P_{n-1})$$

$$b_n = P_n + (P_n - a_n)$$

$$a_n = (pt1\ encerclé + P_{n+1})/2$$

2.3.1 – INTERPOLATION DE QUATERNIONS

- Un fois les points de contrôle générés, on obtient une courbe d'interpolation continue en **C1**.



2.3.1 – INTERPOLATION DE QUATERNIONS

- On sait comment générer les points de contrôle.
- Comment générer des quaternions de contrôle?
  - Mêmes principe mais:
    - On obtient le point **1** (voir image, 2 slides avant) en effectuant :

$$point\ 1\ encerclé = P_n + (P_n - P_{n-1})\ \text{en 2D}$$

Dans l'espace de quaternions :

$$quat\ 1\ encerclé = slerp(Q_{n-1}, Q_n, 2)$$

    - On obtient $\mathbf{a_n}$ en effectuant :

$$a_n = (pt1\ encerclé + P_{n+1})/2\ \text{en 2D}$$

Dans l'espace de quaternions :

$$quat\ a_n = slerp(quat\ 1\ encerclé, Q_{n+1}, 0.5)$$

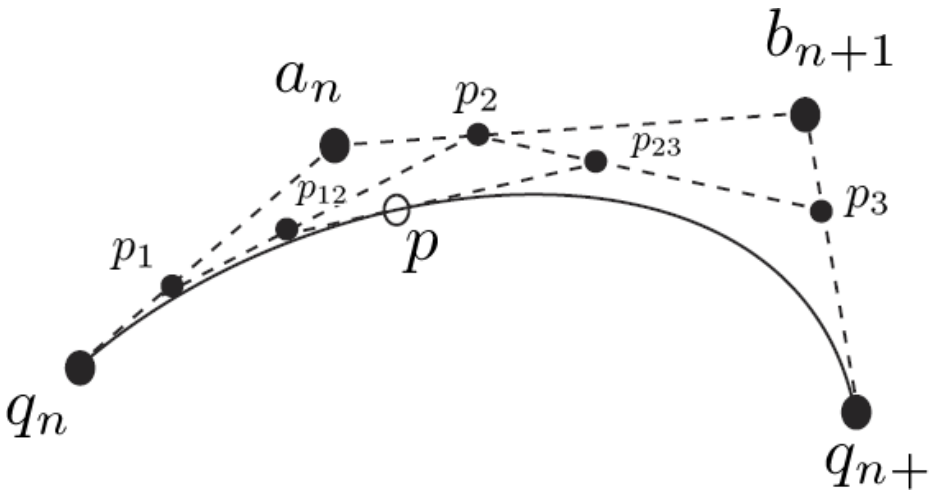
    - $\mathbf{b_n}$ en quaternion est laissé en exercice.

2.3.1 – INTERPOLATION DE QUATERNIONS

- Les quaternions de contrôle générés, il ne reste plus qu'à faire la courbe.
  - On utilise la méthode De Castlejau:
    - On sait que De Castlejau permet de générer des courbes entre les points à partir d'une série d'interpolations linéaires.
    - On peut utiliser la même méthode et générer la courbe entre les quaternions à partir d'une série d'interpolations linéaires sphériques (slerp).

2.3.1 – INTERPOLATION DE QUATERNIONS

- Exemple, si on est à **u = 1/3**.



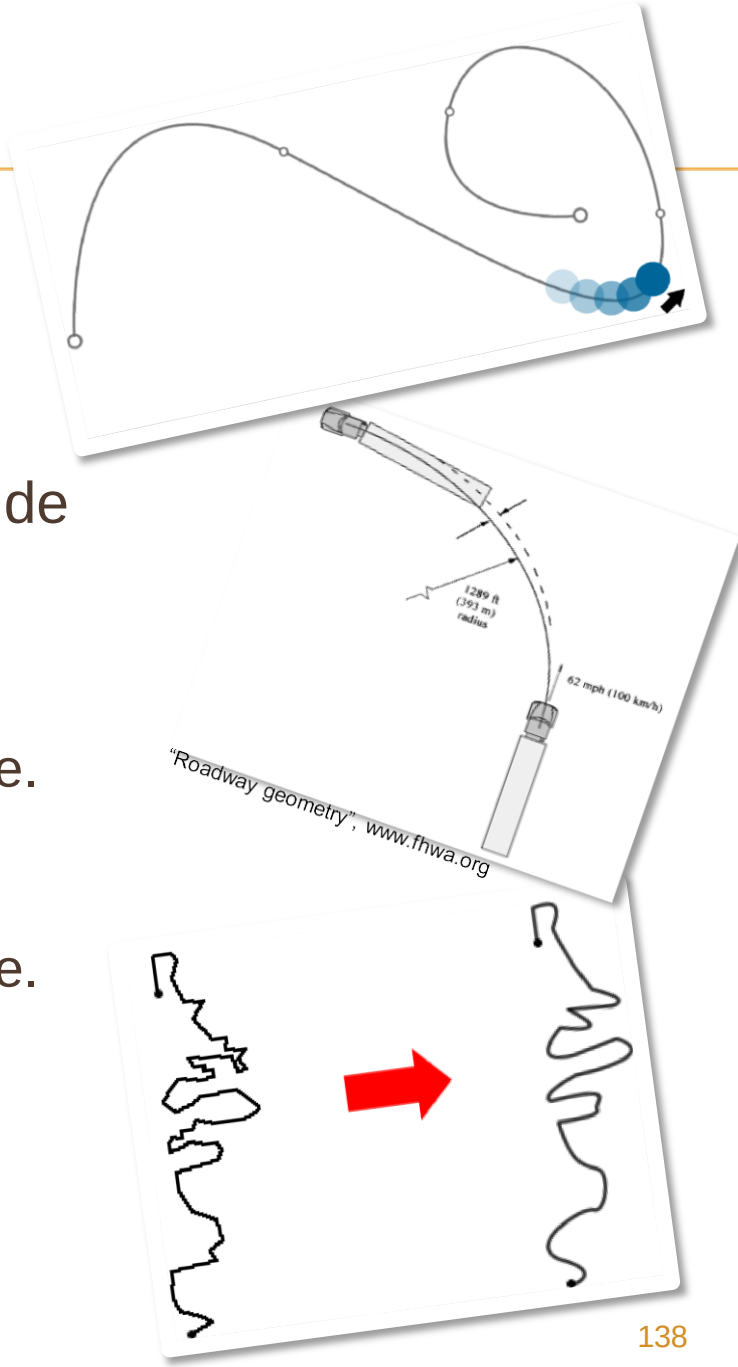
$$
\begin{aligned}
p_1 &= slerp(q_n, a_n, 1/3) \\
p_2 &= slerp(a_n, b_{n+1}, 1/3) \\
p_3 &= slerp(b_{n+1}, q_{n+1}, 1/3) \\
p_{12} &= slerp(p_1, p_2, 1/3) \\
p_{23} &= slerp(p_2, p_3, 1/3) \\
p &= slerp(p_{12}, p_{23}, 1/3)
\end{aligned}
$$

Manuel Figure 3.29

TRAVAILLER AVEC DES TRAJETS

2.4.1 – SUIVI DE TRAJET

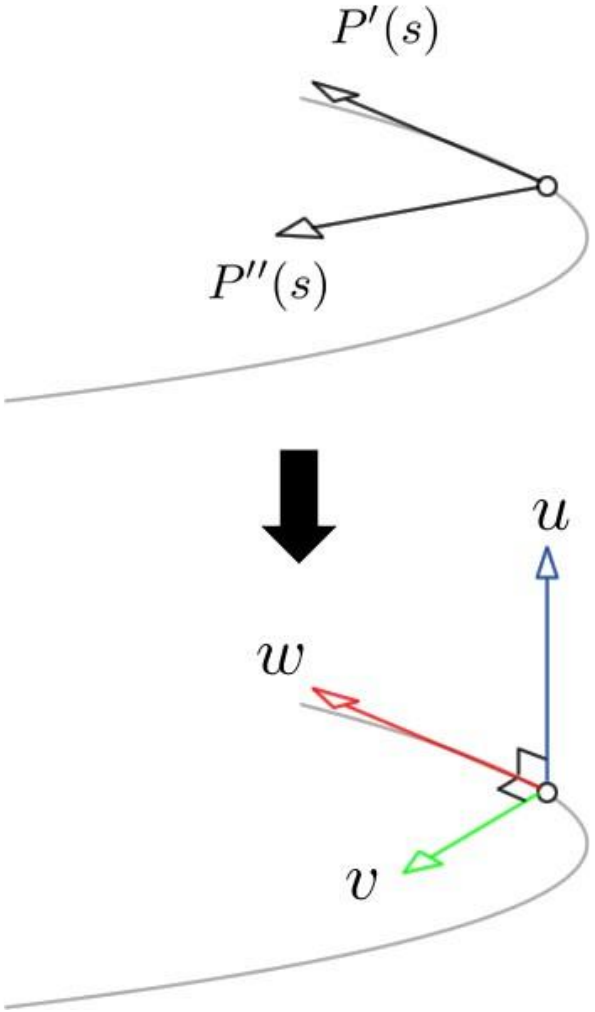
- Énormément utilisée pour déplacer des objets complets.
- Moins populaire pour l'animation locale de modèle. (marcher, lancer, etc.)
- Dans sa forme la plus triviale:
  - Translation de l'objet le long de la courbe.
- Peut devenir plus complexe:
  - Orientation de l'objet le long de la courbe.
  - "Nettoyage" du trajet lors de tracking (capteurs)



2.4.2 – ORIENTATION LE LONG D'UN TRAJET

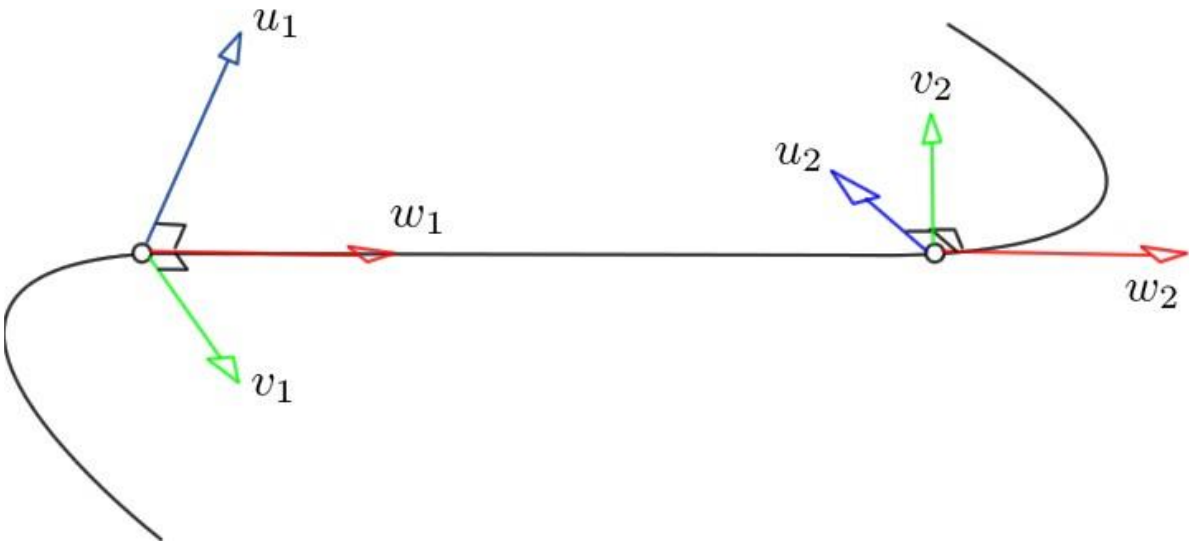
- Repère de frenet
  - Système de coordonnées qui dépend entièrement du trajet qu'il suit.
  - Se construit tel que:

$$
\begin{aligned}
w &= P'(s) \\
u &= P'(s) \times P''(s) \\
v &= u \times w
\end{aligned}
$$



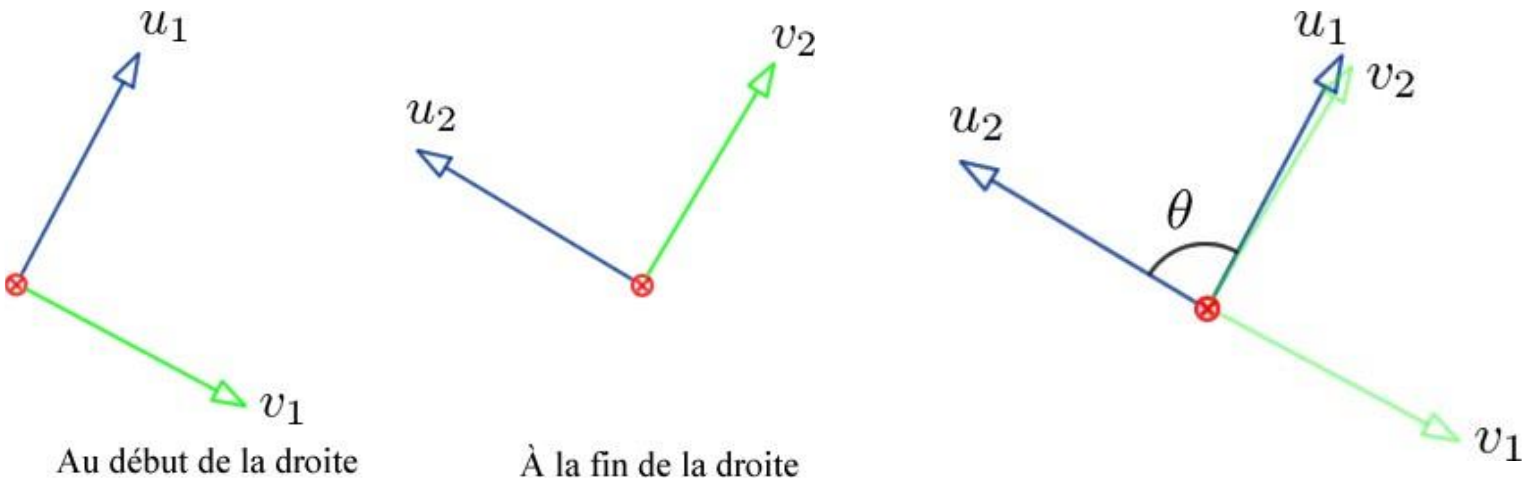
2.4.2 – ORIENTATION LE LONG D'UN TRAJET

- Problème avec le cadre de Frenet :
  - S'il n'y a pas de courbure (segment de droite), la deuxième dérivée est zéro.
  - **u** et **v** tournent librement autour de **w**.



2.4.2 – ORIENTATION LE LONG D'UN TRAJET

- Solution:
  - Prendre le repère de Frenet au début et à la fin du segment de droite et interpoler entre les 2 repères selon l'angle autour de **w**.

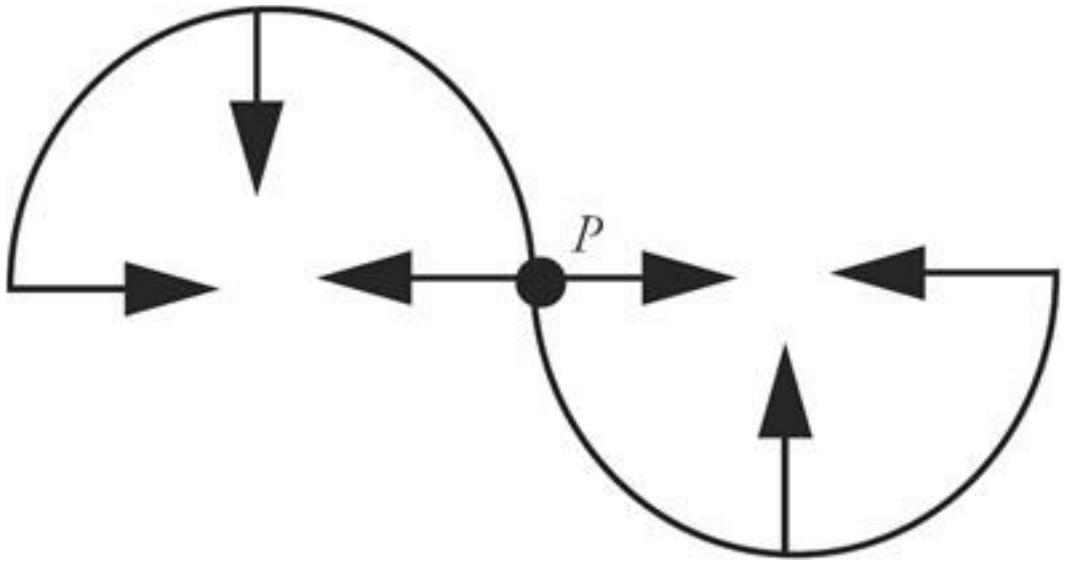


Au début de la droite

À la fin de la droite

2.4.2 – ORIENTATION LE LONG D'UN TRAJET

- Autre problème :
  - Si la courbure n'est pas continue, le repère peut changer d'orientation instantanément:
    - (Comme dans le cas de 2 demi-cercles.)



2.4.2 – ORIENTATION LE LONG D'UN TRAJET

- Dernier problème:
  - Problèmes lorsqu'utilisé dans une courbe variant beaucoup dans les 3 dimensions:
    - Inversion subite du vecteur "haut" du repère.
    - Changements d'orientations brusques autour de l'axe **w**.

2.4.2 – ORIENTATION LE LONG D'UN TRAJET

- Malgré les points faibles du cadre de Frenet, certains de ses concepts peuvent être réutilisés.
  - Utiliser la courbure pour déterminer l'inclinaison d'un objet dans les courbes.
  - Utiliser la tangente pour déterminer la direction de l'objet dans la courbe.

2.4.2 – ORIENTATION LE LONG D'UN TRAJET

- Suivi de trajet avec une caméra
  - La caméra peut être décrite selon certains paramètres:
    - Centre d'intérêt (**COI**)
    - Position (**POS**)
    - Vecteur Haut (**Up-Vector**)
    - Vecteur de vision (souvent donné t.q **v = COI – POS**)

2.4.2 – ORIENTATION LE LONG D'UN TRAJET

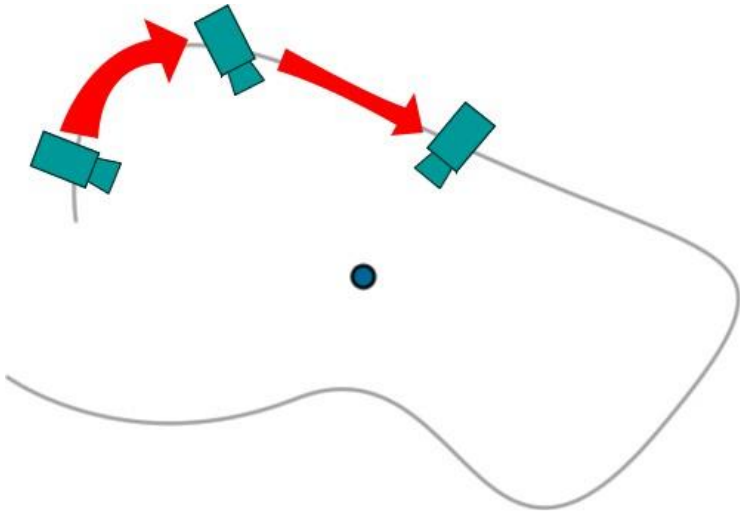
- La position se détermine comme on le fait pour n'importe quel objet.

- Le vecteur haut est souvent fixé à (0,1,0).
  - Peut être altéré pour donner une inclinaison à la caméra (en utilisant la courbure par exemple)

- Différentes façons de déterminer le vecteur de vision (direction de regard).

2.4.2 – ORIENTATION LE LONG D'UN TRAJET

- Génération automatique du vecteur de vision.
  - Le centre d'intérêt peut être un point fixe que la caméra suit en se déplaçant.

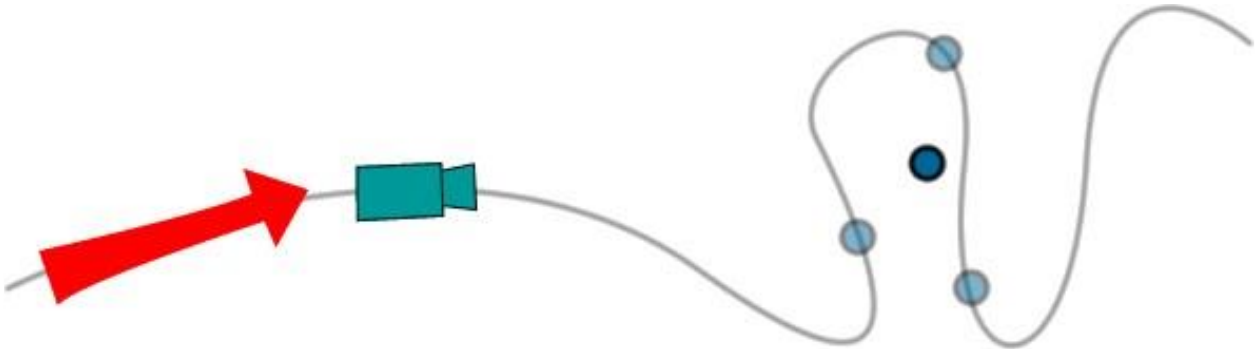


2.4.2 – ORIENTATION LE LONG D'UN TRAJET

- Génération automatique du vecteur de vision.
  - Le centre d'intérêt peut être un point plus loin sur le trajet de la caméra.

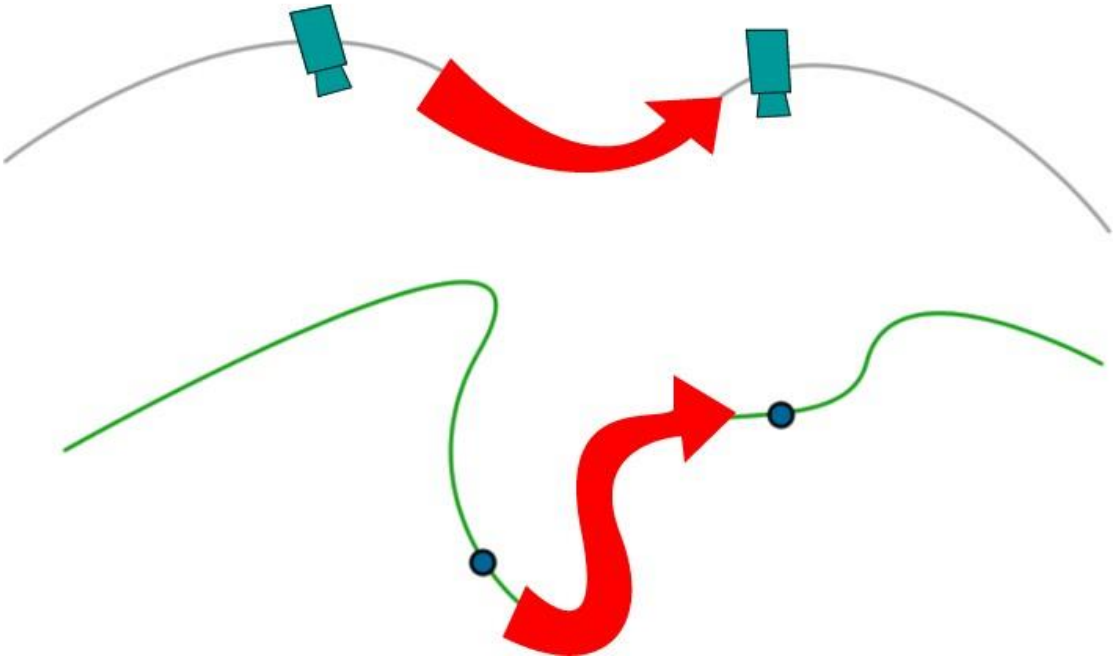
2.4.2 – ORIENTATION LE LONG D'UN TRAJET

- Génération automatique du vecteur de vision.
  - Si jamais la courbe varie brusquement, on peut vouloir faire une moyenne de plusieurs points pour adoucir le déplacement du centre d'intérêt.



2.4.2 – ORIENTATION LE LONG D'UN TRAJET

- Génération automatique du vecteur de vision.
  - Pour un meilleur contrôle, on peut finalement fixer le point d'intérêt sur un trajet indépendant de celui de la caméra.



ADOUCISSEMENT DE TRAJET

2.4.3 – ADOUCISSEMENT DE TRAJET

- Un trajet aura parfois besoin d'être adoucis lorsqu'il n'est pas suffisamment régulier ou bruité. C'est particulièrement le cas lorsque les données proviennent de capteurs.

- Pour adoucir un trajet, diverses méthodes sont possibles:

  - Interpolation linéaire des valeurs adjacentes.

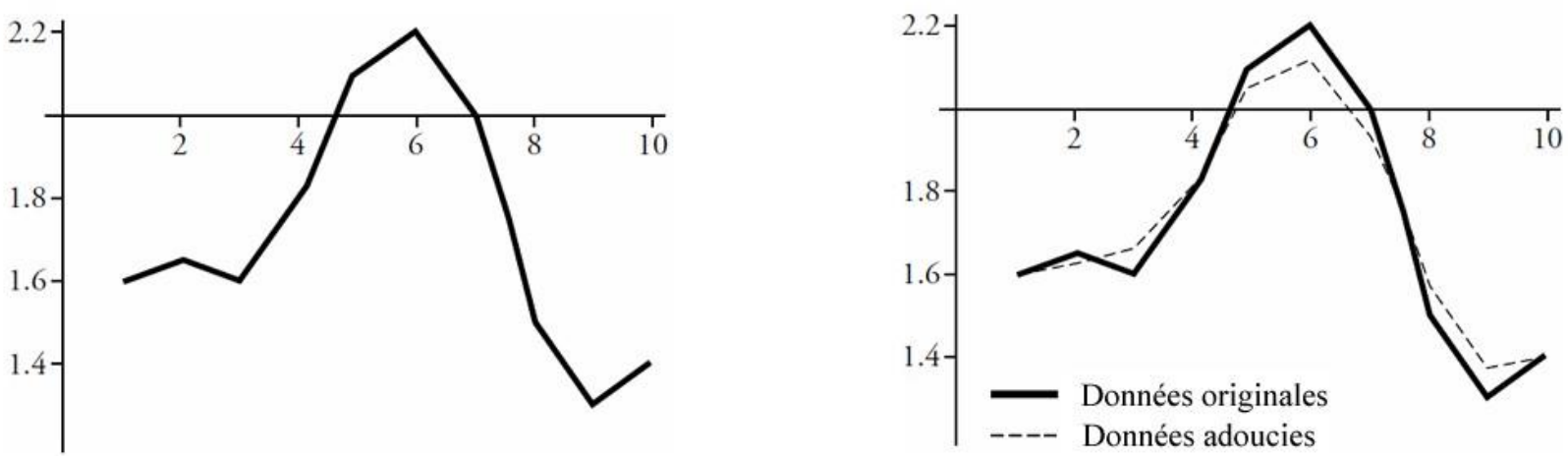
  - Estimation par noyau de convolution.

  - Approximation par B-Spline.

2.4.3 – ADOUCISSEMENT DE TRAJET

- Interpolation linéaire des valeurs adjacentes.

$$P'_i = \frac{1}{4}P_{i-1} + \frac{1}{2}P_i + \frac{1}{4}P_{i+1}$$



- On peut appliquer le processus de façon successive pour augmenter l'adoucissement.

2.4.3 – ADOUCISSEMENT DE TRAJET

- Estimation par noyau de convolution

  (n'est pas matière à examen)

  - Quand les données peuvent être exprimées comme un fonction de la forme y = f(x), il est possible de les adoucir avec une **estimation par noyau de convolution**.

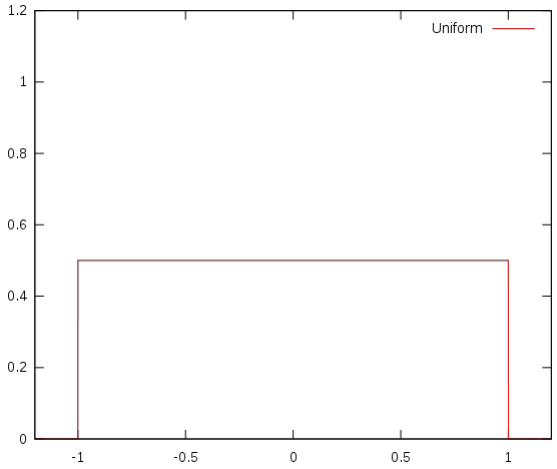
2.4.3 – ADOUCISSEMENT DE TRAJET

- Le noyau de convolution souhaité est défini tel que:
  - Il est centré à 0.
  - Son support est préférablement fini (le domaine où la fonction n'est pas égale à 0 est fini)
  - L'aire sous la courbe est égale à 1.

2.4.3 – ADOUCISSEMENT DE TRAJET

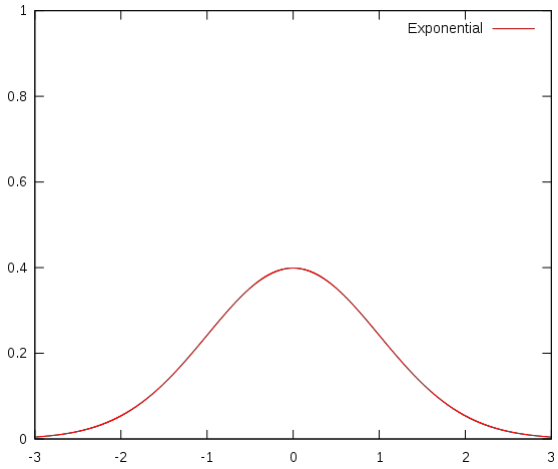
- Exemples de noyaux de convolution :

Uniforme



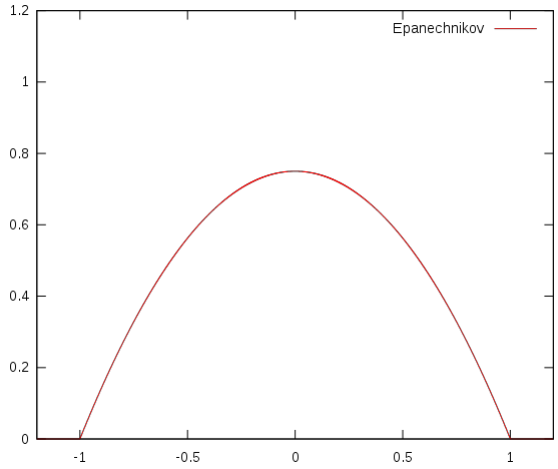
$$K(u) = \frac{1}{2}\ 1_{(|u|\leq 1)}$$

Gaussien (ou “exponentiel”)



$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$$

Epanechnikov



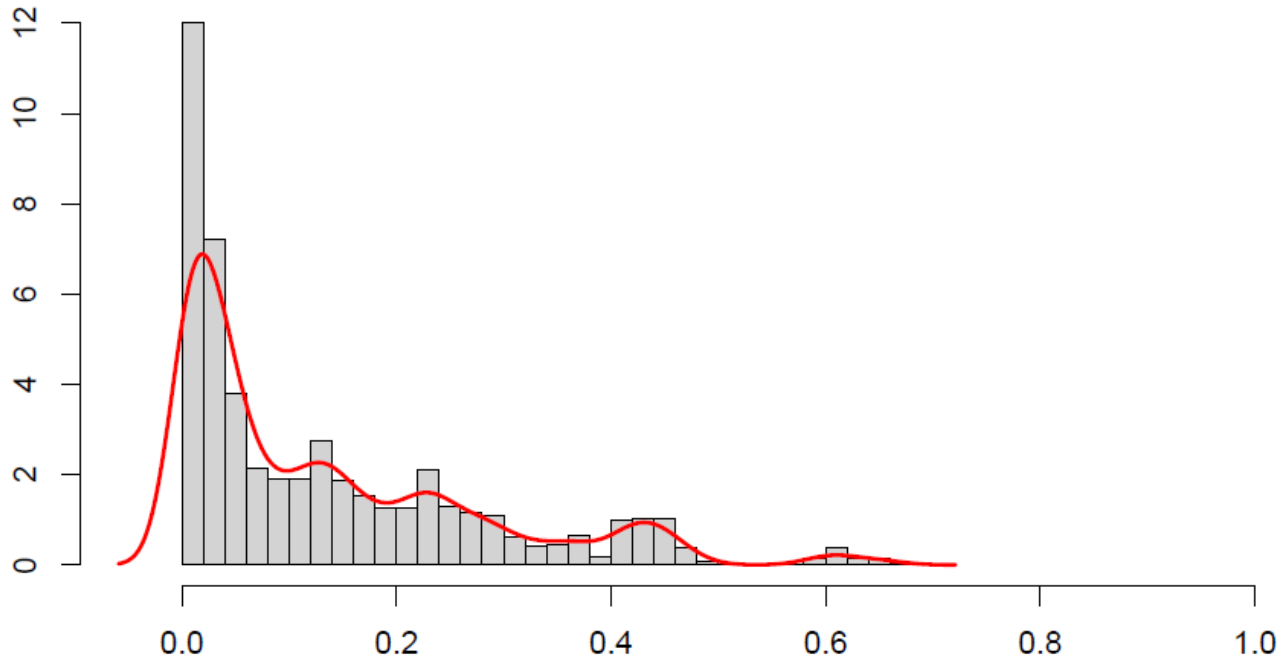
$$K(u) = \frac{3}{4}(1-u^2)\ 1_{(|u|\leq 1)}$$

2.4.3 – ADOUCISSEMENT DE TRAJET

- Procédé:
  1. On prend notre courbe
  2. On convolue chacun des points avec notre noyau.
     1. Si la courbe est une série de points discrets
        → Convolution discrète (avec la sommation)
     2. Si la courbe est continue
        → Convolution standard (avec l'intégrale)
  3. Le résultat final donne la courbe adoucie.

2.4.3 – ADOUCISSEMENT DE TRAJET

- Exemple:



2.4.3 – ADOUCISSEMENT DE TRAJET

- Approximation par B-Spline uniforme
  - Si une approximation est suffisante on utilise directement les points pour calculer une courbe B-Splines.
  - Tous les points n’ont pas à être utilisés.

SOMMAIRE, CHAP 2

- Calcul de longueur d'arc
  - Paramétrisation
- Contrôle du mouvement
  - Ease-In/Ease-Out
  - Fonction distance/temps générales
  - Paires position/temps
- Interpolation d'orientations
  - Interpolation de quaternions
- Orientation le long d'un trajet
  - Cadre de Frenet
  - Contrôle de caméra
- Adoucissement de trajets

RÉFÉRENCES

- Manuel, Parent, Rick : ”*Computer Animation, Algorithms and techniques*“, 3nd edition, Morgan Kaufmann, 2012
- H. Eberly, David: “*3D Game Engine Architecture”, Morgan Kaufmann, 2005*