Morphing

Ryan Bouchou, X
X, X
X

Cursus ingénieur
Première année
Génie informatique

20 janvier 2024

Tuteur entreprise :
Élisabeth X
prénom.nom@entreprise.com

Résumé

Table des matières

1 Cas des polygones simples

En premier lieu, nous allons nous intéresser au cas des polygones simples. Ce faisant, ceci sera l'occasion pour nous de faire notre premier pas dans l'*informatique graphique*, en traitant des cas élémentaires du problème de la morphose.

1.1 Généralités

Conventions Pour toute la suite, et sauf mention contraire, on se place dans le $\mathbb{R}$-evn $\mathbb{R}^2$ muni de la norme euclidienne. On notera pour tout vecteur $x \in \mathbb{R}^2$, $X \in \mathcal{M}_{2,1}(\mathbb{R})$ la matrice colonne associée à x dans la base canonique de $\mathbb{R}^2$.

Définition 1.1. Soit n un entier naturel non nul. On appelle *polygone simple* de $\mathbb{R}^2$ tout n_uplet $P = (p_1, \dots, p_n)$ de $\mathbb{R}^2$ tels que :

1. Les segments ne se croisent pas, c'est-à-dire que pour chaque paire de segments $[p_i, p_{i+1}]$ et $[p_j, p_{j+1}]$ (où p_{n+i} est p_i), les segments ne partagent pas de points autres que les sommets.
2. Chaque sommet p_i est partagé par exactement deux segments.

On notera p_{i^+} le segment $[p_i, p_{i+1}]$ et p_{i^-} le segment $[p_i, p_{i-1}]$. Enfin, on notera $\mathbb{P}$ l'ensemble des polygones simples de $\mathbb{R}^2$.

Définition 1.2. Soit $P \in \mathbb{P}$ un polygone simple de $\mathbb{R}^2$.

1. On appelle *ordre* de P le nombre n de composantes de P.
2. On appelle *arête* de P tout segment p_{i^+} ou p_{i^-} pour $i \in \{1, \dots, n\}$.
3. On appelle *sommet* de P tout vecteur p_i pour $i \in \{1, \dots, n\}$.

Subséquemment, pour n un entier naturel non nul, on note $\mathbb{P}_n$ l'ensemble des polygones simples de $\mathbb{R}^2$ d'ordre n.

Définition 1.3. Soit $P \in \mathbb{P}$ un polygone simple de $\mathbb{R}^2$. On appelle *intérieur* de P, noté $\overset{\circ}{P}$, l'ensemble des points $x \in \mathbb{R}^2$ tels que x est à gauche de chaque arête de P.

Situation À ce stade, l'enjeu de cette première partie est de déterminer un algorithme permettant la morphose d'un polygone simple P vers un autre polygone simple Q. Pour ce faire, il nous faut nous intéresser aux conditions d'une telle transformations, ainsi qu'à sa réalisation.

1.2 Un premier algorithme de morphing

1.2.1 Principe et notation

Principe L'idée de cet algorithme est de déformer progressivement le polygone P en un autre polygone Q. Pour ce faire, une première approche consiste à déformer chaque sommet p_i de P en un sommet q_i de Q par une interpolation linéaire. Ainsi, on obtient une suite de polygones $(P_k)_{0 \leq k \leq N}$ où N est le nombre d'images intermédiaires voulues et tels que $P_0 = P$, $P_N = Q$.

> Pour que l'algorithme soit correct, il est nécessaire que les polygones P et Q soient de même ordre.

Notation Soit $n, N > 0$ et $(P_k)_{0 \leq k \leq N}$ une suite de polygones de $(\mathbb{P}_n)^{\mathbb{N}}$. On notera $p_1^{(k)}, \ldots, p_n^{(k)}$ les sommets de P_k.

Ce faisant, pour le calcul des images intermédiaires on donne l'algorithme suivant :

Données : $P, Q \in \mathbb{P}_n$ deux polygones, $N > 0$ le nombre de frames
Résultat : Une suite de polygones (P_k)
$P^* \leftarrow (P_1, \ldots, P_N)$
pour $k \leftarrow 0$ **à** N **faire**
  $t \leftarrow \frac{k}{N}$
  $P_k = (p_1^{(k)}, \ldots, p_n^{(k)})$
  pour $i \leftarrow 1$ **à** n **faire**
    $p_i^{(k)} \leftarrow (1-t) * p_i + t * q_i$
  fin
fin
retourner P^*

Algorithme 0 : générationFramesNaif

1.2.2 Cas des images matricielles

Principe Dans le cadre de l'exercice, la donnée du problème est consituée de deux images matricielles P et Q de taille $L \times l$ représentant respectivement des polygones simples de couleur même couleur. Naturellement, plusieurs méthodes algorithmiques peuvent être utilisées pour encoder l'information géométrique portée par une image. Entre autre, des algorithme de *détection de contours* ou de *segmentation* peuvent être utilisés. Toutefois, et par soucis de simplicité, nous allons considérer que l'information géométrique est directement encodée par l'utilisateur lors de la saisie des *points de contrôle*. De plus, pour chaque pixel de coordonnées (i, j), on a :

$$\begin{cases} \mathbb{I}_P(i,j) = 1 & \text{si } (i,j) \in \overset{\circ}{P} \\ \mathbb{I}_P(i,j) = 0 & \text{sinon} \end{cases}$$

> Pour que l'algorithme soit correct, il est nécessaire que les images P et Q soient de même taille. De plus, si les points de contrôle saisis par l'utilisateur ne correspondent pas aux sommets des polygones, les résultats de l'algorithme peuvent être inattendus.

Propriété 1.1. Soit $u, v \in \mathbb{R}^2$ deux vecteurs. Alors u est à gauche de v si et seulement si $det(u, v) > 0$.

Démonstration : Admis. o.ε.δ.

Propriété 1.2. Soit $p = [u, v], (u, v) \in \mathbb{R}^2$ un segment et $x \in \mathbb{R}^2$ un vecteur. Alors x est à gauche de p si et seulement si $det(p, [u, x]) > 0$.

Démonstration : Soit $u, v \in \mathbb{R}^2$ deux vecteurs. On note θ l'angle orienté entre u et v. Alors,

$$\begin{aligned} u \text{ est à gauche de } v &\iff \det(u,v) > 0 && (1)\\ &\iff \|u\| \cdot \|v\| \sin(\theta) > 0 && (2)\\ &\iff \sin(\theta) > 0 && (3)\\ &\iff \theta \in]0, \pi[&& (4)\\ &\iff u \text{ est dans le demi-espace à gauche de } v && (5) \end{aligned}$$

o.ε.δ.

Ce faisant, pour le calcul des images intermédiaires, et la determination de l'intérieur du polygone, de on donne la suivante :

Données : Une liste de points *tab* représentant le polygone, un point P
Résultat : Un booléen indiquant si le point P est à l'intérieur du polygone
$nbp \leftarrow$ taille de *tab* **pour** $i \leftarrow 0$ **à** $nbp - 1$ **faire**
  $A \leftarrow tab[i]$ $B \leftarrow tab[(i+1) \mod nbp]$
  $D \leftarrow (B.x - A.x, B.y - A.y)$ $T \leftarrow (P.x - A.x, P.y - A.y)$
  $d \leftarrow det(D, T)$
  si $d < 0$ **alors**
    retourner Faux
  fin
fin
retourner Vrai ;

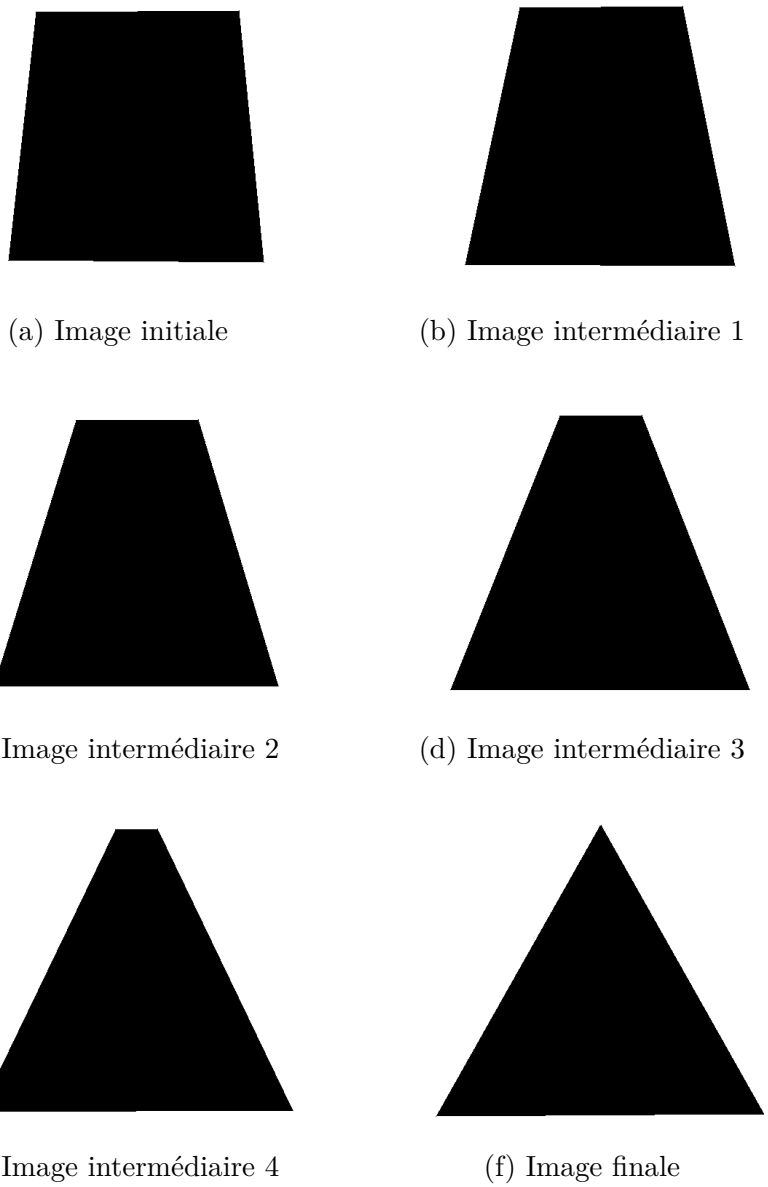
Algorithme 1 : isInside

On en déduit l'algorithme naif pour le mophing de formes unies simples :

Données : Deux images matricielles P et Q de taille $L \times l$, un entier N
Résultat : Une suite d'images matricielles P^*
$nbp \leftarrow$ taille de *tab*
$P^* \leftarrow$ générationFramesNaif(P, Q)
pour $i \leftarrow 0$ **à** $nbp - 1$ **faire**
  pour $x \leftarrow 0$ **à** $L - 1$ **faire**
    pour $y \leftarrow 0$ **à** $l - 1$ **faire**
      if$(isInside(P_i^*, (x, y)))$ $P_i^*[x][y] \leftarrow color$
    fin
  fin
fin
retourner P^*

Algorithme 2 : morphingNaif

Résultats Considérons une exécution de l'algorithmes sur deux instances du problème.



(a) Image initiale (b) Image intermédiaire 1

(c) Image intermédiaire 2 (d) Image intermédiaire 3

(e) Image intermédiaire 4 (f) Image finale

FIGURE 2 – Résultats de l'algorithme de morphing

> Dans le cas de polygones simples, relativements semblables dans leur géométrie, l'algorithme de morphing naif donne des résultats satisfaisants.

En pratique Toutefois, il apparaît qu'une telle approche ne préserve pas les propriétés géométriques du polygone. En effet, les images intermédiaires peuvent contenir des segments qui se croisent, ou des sommets qui ne sont pas partagés par exactement deux segments. Dans un tel cas, la transformations ne paraît pas naturelle.

FIGURE 3 – Interpolation linéaire, observez les auto-intersections. [2]

Ainsi, il est nécessaire de trouver une approche plus rigoureuse pour le morphing de polygones simples.

1.3 Morphing par interpolation longeur-angle

1.3.1 Principe et notation

Principe L'idée de cette méthode, issue des travaux de Sederberg [5], est de déformer progressivement le polygone P en un autre polygone Q, en interpolant linéairement la mesure des angles entre chaque arêtes consécutives, et leurs normes respectives. Ce faisant, l'avantage est de préserver les propriétés géométriques souhaitées pour un rendu visuel naturel. On parle d'*invariance par mouvement rigide.* de morphing améliorée est.

Notation Soit $n, N > 0$ et $(P_k)_{0 \leq k \leq N}$ une suite de polygones de $(\mathbb{P}_n)^{\mathbb{N}}$. On pose, pour $k \in \{0, \dots, N\}$:

$\rightarrow$ $l_i^{(k)}$ la longueur de l'arête $p_{i^+}^{(k)}$,

$\rightarrow$ $\theta_i^{(k)}$ l'angle entre les arêtes $p_{i^-}^{(k)}$ et $p_{i^+}^{(k)}$.

Subséquemment, pour une image intermédiaire $k \in \{0, \dots, N\}$, avec $t = \frac{k}{N}$ on a :

$$l_i^{(k)} = (1-t)l_i + tq_i$$
$$\theta_i^{(k)} = (1-t)\theta_i + t\theta_i$$

Ce faisant, pour le calcul des images intermédiaires on donne l'algorithme suivant :

Données : $P, Q \in \mathbb{P}_n$ deux polygones, $N > 0$ le nombre de frames
Résultat : Une suite de polygones (P_k)
$P^* \leftarrow (P_1, \dots, P_N)$
$P_0 \leftarrow P$, $P_N \leftarrow Q$
pour $k \leftarrow 0$ **à** N **faire**
  $t \leftarrow \frac{k}{N}$
  $P_k = ((l_1^{(k)}, \theta_1^{(k)}), \dots, (l_n^{(k)}, \theta_n^{(k)}))$
  pour $i \leftarrow 1$ **à** n **faire**
    $l_i^{(k)} \leftarrow (1-t)l_i + tq_i$
    $\theta_i^{(k)} \leftarrow (1-t)\theta_i + t\theta_i$
  fin
fin
retourner P^*

Algorithme 3 : générationFramesLA

1.3.2 Cas des images matricielles

> En l'état, et de par des contraintes de temps, nous n'avons pas implémenté ce segond algorithme. On donne cependant une comparaison des résultats obtenus par *Sederberg et al* [5] avec les deux méthodes.

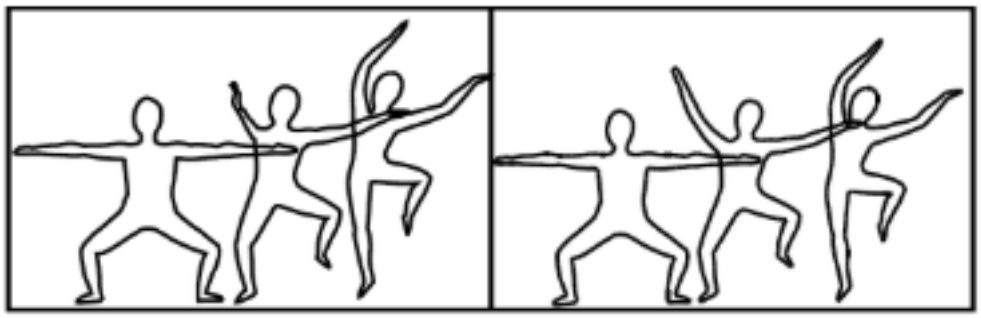


FIGURE 4 – Morphing par interpolation longueur-angle à droite, naive à gauche. [5]

2 Morphing de formes courbes

Les polylignes à courbure nulle ne permettent pas une représentation précise de formes courbes. Pour pallier ce problème, nous allons nous appuyer sur les *courbes B-splines*, couramment utilisées en CAO pour représenter des formes courbes [4].

2.1 Propédeutique au morphing de courbes splines

2.1.1 Splines

Définition 1.4. Considérons des réels $u_0 < u_1 < \ldots < u_m$, et $p \in \mathbb{N}$. On définit les *fonction B-Splines* $B_{i,p}$ par récurrence sur p et i dans $\mathbb{N}$ comme suit :

$$\begin{cases} \text{Pour } 0 \leq i \leq m-1 \\ B_{i,0}(u) = 1 \text{ si } u \in [u_i, u_{i+1}[, \quad B_{i,0}(u) = 0 \text{ sinon} \end{cases} \tag{6}$$

$$\begin{cases} \text{Pour } 0 \leq i \leq m-1 \\ B_{i,p}(u) = \frac{u-u_i}{t_{i+p}-t_i} B_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} B_{i+1,p-1}(u) \end{cases} \tag{7}$$

Notation. Soit $j = 1, \ldots, m+1-i.$, on note : $\begin{cases} \omega_{i,j}(u) = \frac{u-u_i}{u_{i+j}-u_i} & \text{si } u_i < u_{i+1}, \\ \omega_{i,j} = 0 & \text{sinon.} \end{cases}$

Convention. Pour la suite, toute fonction dont le dénominateur est nul sera considérée comme nulle.

Définition 1.5. Ainsi, on a pour $i \in \{0, \ldots, m-p-1\}$ et $p \in \mathbb{N}$:

$$\begin{aligned} B_{i,0}(u) =& 1 \quad \text{pour} \quad t \in [u_i, t_{i+1}[= 0, \\ B_{i,p}(u) =& \sum_{j=1}^{m+1-i} \omega_{i,j}(u) B_{i,p-1}(u) \quad \text{pour } p > 0. \end{aligned}$$

2.1.2 Courbes B-Splines

Définition 1.6. Soit $m \in \mathbb{N}$. On appelle $(u_i)_{0 \leq i \leq m}$, *vecteur de noeuds*, et p, *degré de la B-spline*. On considère aussi des *points de contrôle* $\mathbf{P}_1, \ldots, \mathbf{P}_m$ de $\mathbb{R}^n$. De fait, $(\mathbf{P}_i)_{0 \leq i \leq m}$ forme un *polygone de contrôle*. La *courbe B-Spline* d'ordre p associée à ces données est définie par :

$$u \longmapsto C(u) = \sum_{i=0}^{m} B_{i,p}(u) \mathbf{P}_i. \tag{8}$$

Propriété 1.3. Supposons que $C(u)$ soit une courbe B-spline de degré p définie comme suit :

$$C(u) = \sum_{i=0}^{n} B_{i,p}(u) \mathbf{P}_i$$

Soit le point de contrôle $\mathbf{P}_i$ déplacé vers une nouvelle position $\mathbf{P}_i + \mathbf{v}$. Alors, la nouvelle courbe B-spline $D(u)$ de degré p est la suivante [4] :

$$D(u) = C(u) + N_{i,p}(u)\mathbf{v} \tag{9}$$

Démonstration :

$$\begin{aligned} D(u) &= \sum_{i=0}^{n} B_{i,p}(u)(\mathbf{P}_i + \mathbf{v}) \\ &= \sum_{i=0}^{n} B_{i,p}(u)\mathbf{P}_i + \sum_{i=0}^{n} B_{i,p}(u)\mathbf{v} \\ &= C(u) + \sum_{i=0}^{n} B_{i,p}(u)\mathbf{v} \\ &= C(u) + N_{i,p}(u)\mathbf{v} \end{aligned}$$ o.ε.δ.

En pratique Desormais, il nous est possible, sur la base de points de contrôle, d'un vecteur de noeud et du degré de la B-spline, de générer une courbe B-spline comme ci-dessous :

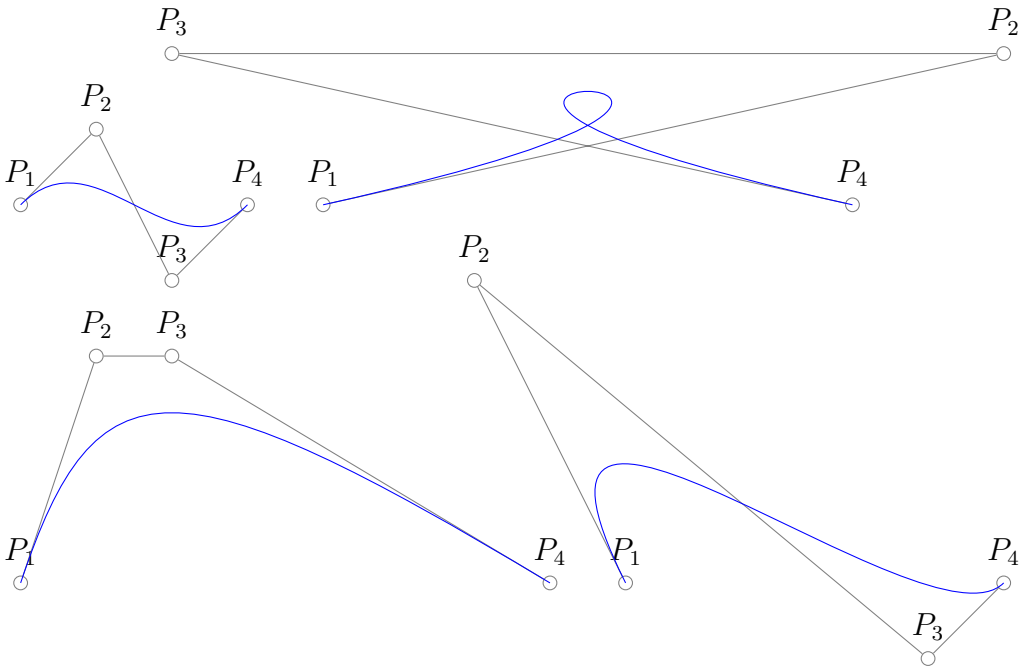


Figure 5 – Courbes B-splines

Lemme 2. — *Soit $\mathcal{C} = (U, P, d)$ une courbe B-Spline. Alors, $\mathcal{C}$ est une courbe fermée si et seulement si :*

1. $\mathbf{P}_0 = \mathbf{P}_m$, *où* $m = ord(P)$,
2. U *est un vecteur de noeuds uniforme.*

Corollaire 3. — *Soient $m, N > 0$ et $v = (\mathbf{v}_{i,k})_{(i,k)\in[\![0,m]\!]\times[\![0,N]\!]} \in \mathcal{M}_{m+1,N+1}(\mathbb{R}^2)$ Soit $\mathcal{C}_0 = (U, P^{(0)}, d)$ et $\mathcal{C}_k = (U, P^{(k)}, d)$, $k \in \mathbb{N}$, deux courbes B-Splines telles que :*

1. $\forall k \in [\![0, N]\!], \forall i \in [\![0, m]\!]$, $\mathbf{P}_i^{(k)} = \mathbf{P}_i^{(0)} + \mathbf{v}_{i,k}$,
2. U *est un vecteur de noeuds uniforme.*

2.2 Morphing de courbes B-splines

Principe Notre approche se distingue en deux phases. Premièrement, laisser à l'utilisateur le soin de réaliser l'ajout de points de contrôles, et subséquemment, l'ajustement de la courbe qui modélise la forme souhaitée. Ensuite, nous procédons au calcul des fonctions de base $B_{i,p}(u)$ ainsi qu'aux points de la courbes $C(u)$ pour l'image source. Ce faisant, en réalisant une interpolation linéaire des points de controles entre les polygones de contrôles de l'image source et de l'image destination, nous sommes en mesure de déterminer pour chaque image intermédiaire $0 \leq k \leq N$ la position $C^{(k)}(u)$ de la courbe en utilisant le corollaire 3.

2.2.1 Calcul d'une courbe B-spline

Calcul de la courbe Dans le sous-domaine mathématique de l'analyse numérique, l'algorithme de de Boor [6] est un algorithme polynomial et numériquement stable pour évaluer les courbes splines sous forme de B-splines. Il s'agit d'une généralisation de l'algorithme de de Casteljau pour les courbes de Bézier. Cet algorithme a été conçu par le mathématicien germano-américain Carl R. de Boor. Des variantes simplifiées et potentiellement plus rapides de l'algorithme de de Boor ont été créées, mais elles souffrent d'une stabilité comparativement plus faible.

Fonction `calculerPoint`(u, *controlPoints*, *nodeVector*) **:**
- $k \leftarrow trouverIntervalle(u, nodeVector)$;
- $d \leftarrow controlPoints.subList(k - deg, k + 1)$;
- `// Algorithme de De Boor`
- **pour** $r \leftarrow 1$ **à** deg **faire**
  - **pour** $j \leftarrow k$ **à** $k - r$ **pas** -1 **faire**
    - $i \leftarrow j - k + deg$;
    - $a \leftarrow \frac{u - nodeVector.get(j)}{nodeVector.get(j+deg-r+1) - nodeVector.get(j)}$;
    - $interpolated \leftarrow d.get(i-1).nextPoint(d.get(i), a)$;
    - $d.set(i, interpolated)$;
- **retourner** $d.get(deg)$;

Algorithme 4 : Calcul courbes B-splines

Remarque. On donne le pseudo-code de la fonction `trouverIntervalle` dans l'algorithme 9 en annexe.

Algorithme On donne ci-après l'algorithme principal pour le morphing de courbes B-splines, basé sur l'approche évoquée plus haut. Malheureusement, faute d'appréciation, nous n'avons pas pu tester l'algorithme. Nous en donnons une implémentation en Java complète dans les sources du projet. Par ailleurs, la javadoc est disponible pour une meilleure compréhension de l'implémentation.

Entrée : Les points de contrôle de l'image source *sourcePoints*, les points de contrôle de l'image destination *destinationPoints*, le nombre d'images intermédiaires à générer *numFrames*
Sortie : Une liste d'images intermédiaires représentant le morphing de courbes B-splines

```
// Vérifier que les listes de points de contrôle ont la même taille
si sourcePoints.size() ≠ destinationPoints.size() alors
    retourner null;

// Calculer les vecteurs de noeuds pour les courbes source et destination
sourceNodeVector ← calculerNodeVector(sourcePoints.size(), deg);
destinationNodeVector ← calculerNodeVector(destinationPoints.size(), deg);

// Générer les images intermédiaires
intermediateImages ← une liste vide;
pour i ← 0 à numFrames faire
    t ← i / numFrames;
    // Calculer les points de contrôle intermédiaires par interpolation linéaire
    intermediatePoints ← une liste vide;
    pour j ← 0 à sourcePoints.size() faire
        sourcePoint ← sourcePoints.get(j);
        destinationPoint ← destinationPoints.get(j);
        x ← (1 − t) · sourcePoint.getX() + t · destinationPoint.getX();
        y ← (1 − t) · sourcePoint.getY() + t · destinationPoint.getY();
        intermediatePoints.add(newPoint(x, y));

    // Calculer les points de la courbe B-spline intermédiaire
    intermediateCurve ← une liste vide;
    pour u ← 0 à 1 pas 0.01 faire
        point ← calculerPoint(u, intermediatePoints, sourceNodeVector);
        intermediateCurve.add(point);

    // Ajouter l'image intermédiaire à la liste
    intermediateImage ← genererImage(intermediateCurve);
    intermediateImages.add(intermediateImage);

retourner intermediateImages;
```

Algorithme 5 : Morphing de courbes B-splines

3 Morphing d'images quelconques

Après avoir abordé la morphose dans des cas simples, notamment celui de formes unies simples et courbes, nous allons désormais nous intéresser à la morphose d'images quelconques. De ce fait, l'objectif à atteindre est de pouvoir passer d'une image à une autre de manière fluide, en conservant les caractéristiques de chacune des images, tout en **évitant l'effet juxtaposition**.

(a) Image de départ (b) Image d'arrivée

FIGURE 6 – Paires d'images à morpher [1]

Pour ce faire, nous allons nous appuyer sur les travaux de Beier-Nelly [1], qui proposent une méthode de morphing basée sur la paramétrisation de l'image par des vecteurs de contrôle. Ainsi, chaque pixel de l'image est repéré par une association de positions relatives à ces vecteurs de contrôle. En conséquence de quoi, pour deux images différentes ayant le même nombre de vecteurs de contrôle, il est possible d'associer chaque pixel de l'une, à un pixel de l'autre (ceci ne signifie pas qu'il existe une bijection entre les pixels des deux images).

Principe Le principe de la morphose entre deux images repose sur deux étapes majeures. La première consiste à déformer les images de départ et d'arrivée, et la seconde à interpoler les images résultantes. Ce faisant, il nous est possible d'éviter l'effet de superposition des images. Naturellement, plus le nombre de vecteurs de contrôle est élevé, plus la morphose sera précise. De même, un grand nombre d'images intermédiaires permettra d'obtenir une animation plus naturelle.

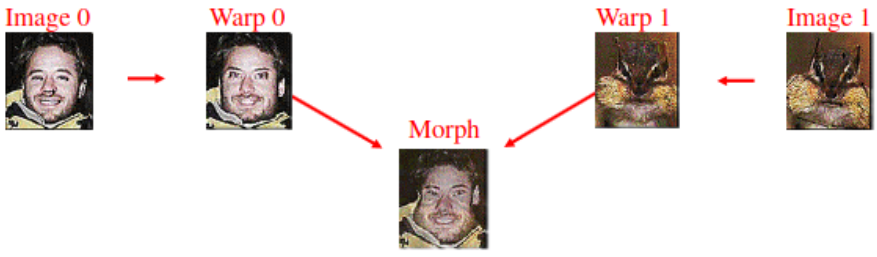


FIGURE 7 – Calcul d'un image intermédiaire [3]

> Sur la figure 7, on peut observer le calcul d'une image intermédiaire *Morph*. Explicitement, on a : $Morph = (1 - \alpha) \times Wrap_0 + \alpha \times Wrap_1$, où α est un paramètre variant de 0 à 1.

3.1 Déformation des images

3.1.1 Préliminaires

Définition 1.7. Soient P une image. On note w sa largeur et h sa hauteur, ainsi que $\mathcal{D}(P) = [0, w] \times [0, h]$. On appelle **vecteur de contrôle** de P tout couple de points du plan $c = (c_1, c_2)$ tel que $c \in \mathcal{D}(P)$.

Définition 1.8. Soient P et Q deux images, ainsi que p et q deux vecteurs de contrôle de respectivement P et Q. On définit la relation $\sim$ telle que $p \sim Q$ si et seulement si p et q sont appariés. Id est, p et q sont une seule et même entité, mais dans deux contextes (*images*) différents.

Conditions. Considérons deux images P et Q à morpher en $N > 0$ étapes. Alors, chacune possède $n + 1$ vecteurs de contrôle $p_0, \ldots, p_n$ et $q_0, \ldots, q_n$ tels que pour tout $i \in \{0, \ldots, n\}$, $p_i \sim q_i$.

3.1.2 Cas simple : un seul vecteur de contrôle

Notations. Soit v un vecteur de controle et x un point du plan
- → On note v_1 et v_2 les composantes de v,
- → On note $\widehat{v} = v_2 - v_1 \in \mathbb{R}^2$,
- → On note $v^\perp = \frac{\widehat{v}}{\|\widehat{v}\|} \times \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$,
- → On note $d(X, v)$ la distance entre x et la droite passant par la première composante de v_1 et portée par v.

Propriété 1.4. Soit $c = (P, Q)$ un vecteur de contrôle, X un point du plan, alors :

$$u = \frac{c \cdot (P, X)}{\|c\|^2} \tag{10}$$

$$v = d(X, c).v^\perp \tag{11}$$

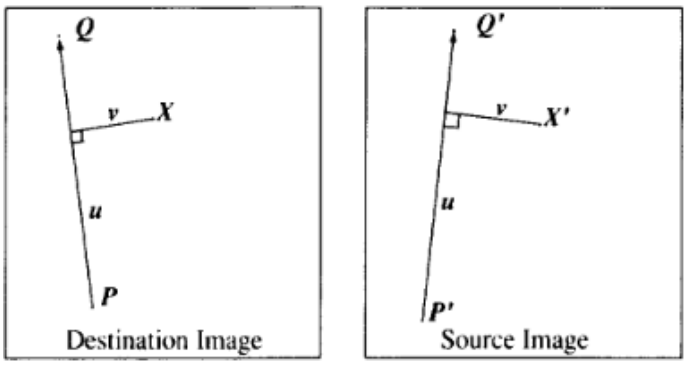


FIGURE 8 – Apparaiement à un seul vecteur [1]

Algorithme. L'algorithme de déformation d'une image avec un seul vecteur de contrôle est donné par l'algorithme 6 suivant.

Données : Deux images D et S à morpher, $(P, Q) \sim (P', Q')$ des vecteurs de contrôle de D et S resp.
Résultat : Une image S déformée
pour *chaque* $X \in \mathcal{D}(D)$ **faire**
  déterminer (u, v);
  déduire $X' \in \mathcal{D}(S)$ pour ce couple (u, v);
Couleur(X) dans D $\leftarrow$ Couleur(X') dans S;

Algorithme 6 : Déformation d'une image avec un seul vecteur de contrôle [1]

3.1.3 Cas général : plusieurs vecteurs de contrôle

Principe L'idée est de déformer l'image d'arrivée en fonction des vecteurs de contrôle de l'image de départ. Pour ce faire, - et à fortiori, bénéficier de l'ensemble des vecteurs de controle - on détermine le barycentre des positions X' calculées dans l'image source pour chaque vecteur de contrôle. Dans la logique des chose, le poids attribué à chaque ligne doit être le plus fort lorsque le pixel est exactement sur la ligne, et diminuer au fur et à mesure que le pixel s'en éloigne.

Poids. Le poids attribué à chaque ligne est donné par l'équation suivante :

$$\text{weight } = \left(\frac{\text{length }^{p}}{(a + \text{ dist })} \right)^{b} \tag{12}$$

où `length` est la longueur d'une ligne. `dist` est la distance du pixel à la ligne, et `a, b,` et `p` sont des constantes qui peuvent être utilisées pour modifier l'effet relatif des lignes.

> En pratique, nous avons utilisé $a = 0.1$, $b = 0.5$ et $p = 1$ qui ont demblé donné des résultats très satisfaisants. Toutefois, il est essentiel de garder à l'esprit que ces valeurs peuvent être ajustées en fonction des images à morpher et de la sensibilité voulues.

Vecteurs de contrôle. Pour chaque déformation intermédiaire d'ordre k, les vecteurs de controles sont déterminés par interpolation linéaire des vecteurs de contrôle de départ et d'arrivée.

$$\begin{aligned} P_{i,k} &= (1-u)P_{i,0} + uP_{i,N} \\ Q_{i,k} &= (1-u)Q_{i,0} + uQ_{i,N} \end{aligned} \tag{13}$$

En notant N le nombre d'étapes de la morphose, et $u = \frac{k}{N}$.

Algorithme. L'algorithme de déformation d'une image avec plusieurs vecteurs de contrôle est donné par l'algorithme 7 suivant.

Données : Deux images D et S à morpher, $(P_i, Q_i) \sim (P'_i, Q'_i)$ des vecteurs de contrôle de D et S resp.
Résultat : Une image S déformée
pour *chaque* $X \in \mathcal{D}(D)$ **faire**
  $DSUM \leftarrow (0, 0)$;
  $sommeDesPoids \leftarrow 0$;
  pour *chaque ligne* (P_i, Q_i) **faire**
    déterminer (u, v);
    déduire $X'_i \in \mathcal{D}(S)$ pour ce couple (u, v);
    déterminer $D_i = X'_i - X_i$;
    déterminer la distance la plus courte de X à (P_i, Q_i);
    déterminer le poids associé à cette distance;
    $DSUM \leftarrow DSUM + D_i \times poids$;
    $sommeDesPoids \leftarrow sommeDesPoids + poids$;
  $X' \leftarrow X + \frac{DSUM}{sommeDesPoids}$;
  Couleur(X) dans D ← Couleur(X') dans S;

Algorithme 7 : Déformation d'une image avec deux vecteurs de contrôle [1]

Exemple. La figure 9 illustre la déformation de l'image source en image destination, mais surtout la détermination de la position X' calculée comme une combinaison linéaire des vecteurs (X, X'_1) et (X, X'_2) [1].

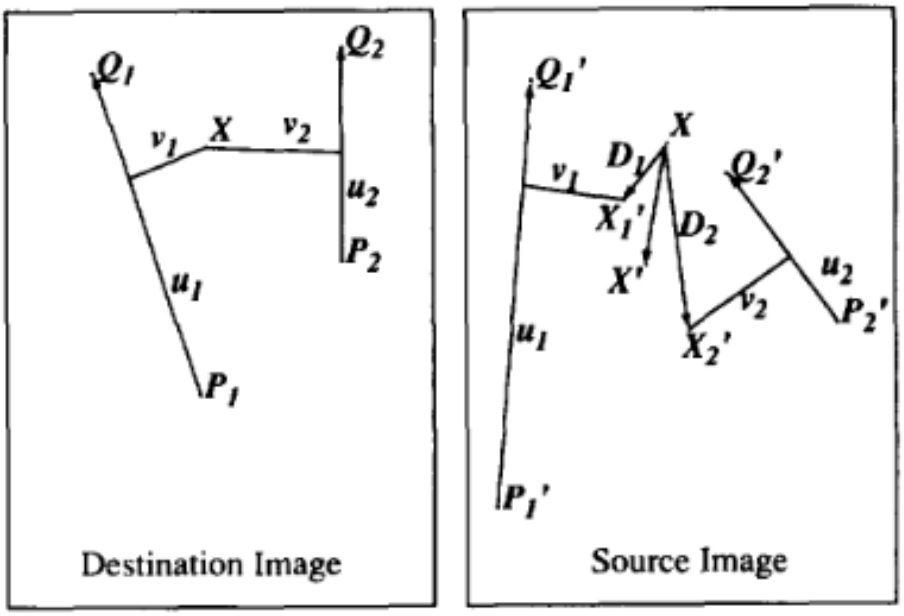


FIGURE 9 – Multiple lignes [1]

3.2 Interpolation des images

Principe L'interpolation des images consiste à déterminer une image intermédiaire entre l'image de départ et l'image d'arrivée. Pour ce faire, nous avons précalculé des images temporaires, étant pour chacune une deformation judicieuse de l'image de départ et d'arrivée. Desormais, nous pouvons déterminer une image intermédiaire en interpolant les images temporaires, *i.e.* , en combinant les couleurs des pixels de ces images.

Algorithme. L'algorithme d'interpolation des images est donné par l'algorithme 8 suivant.

Données : Deux images D et S à morpher, N le nombre d'étapes de la morphose
Résultat : Une image intermédiaire
pour *chaque étape k de la morphose* **faire**
    wrapSrc $\leftarrow$ wrapImage(D,S,k);
    wrapDst $\leftarrow$ wrapImage(S,D,k);
    frame $\leftarrow$ Image vide de taille convenable;
    pour *chaque* $X \in \mathcal{D}(frame)$ **faire**
        $\text{Couleur}_{frame}(X) \leftarrow$
        $\leftarrow (1-\alpha) \times \text{Couleur}_{wrapSrc}(X) + \alpha \times \text{Couleur}_{wrapDst}(X)$ S;

Algorithme 8 : Interpolation des images [1]

Liste des Algorithmes

Table des figures

Annexes

A Morphing de courbes splines

```
// Trouve l'intervalle dans le vecteur de noeuds qui contient
   le paramètre u
Fonction trouverIntervalle(u, nodeVector) :
    n ← nodeVector.size() − 1;
    si u ≥ nodeVector.get(n) alors
        retourner n − 1;
    pour i ← 0 à n − 1 faire
        si u ≥ nodeVector.get(i) and u < nodeVector.get(i + 1) alors
            retourner i;
    retourner −1 // Interval not found
```

Algorithme 9 : Trouver intervalle

Références

[1] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. *Computer Graphics*, 26(2) :35–42, 1992.

[2] Mélanie Cornillac. *Morphing multirésolution de courbes*. Modélisation et simulation, Université de Grenoble, 2010. NNT : ff, tel-00581474f.

[3] Department of Computer Science, University of Toronto and Flores-Mangas, Fernando. Csc320w : Introduction to visual computing. Course Material, 2021. Course syllabus, lecture slides, and other materials may be available on the course website or through the department.

[4] Pierre Pansu. Courbes b-splines, 2004. Accessed : 2024-05-30.

[5] T.W. Sederberg, P. Gao, G. Wang, and H. Mu. 2-d shape blending : An intrinsic solution to the vertex path problem. In *Computer Graphics (SIGGRAPH 93 Proceedings)*, volume 27, pages 15–18, 1993.

[6] C.-K. Shene. Cs3621 introduction to computing with geometry notes, Year Not Specified. Professor, Department of Computer Science, Michigan Technological University.