Sketching 2D et 3D

Aurélien Yol

Sketching en 2D et 3D

Rapport de Stage

Aurélien YOL

24 juillet 2011

Encadrant : Pierre Poulin

Suiveur : Kadi Bouatouch

LIGUM - Laboratoire d'Informatique Graphique de l'Université de Montréal

Master Informatique spécialité Recherche en Informatique

Table des matières

Remerciements

Je tiens premièrement à remercier tout particulièrement Kadi Bouatouch sans qui ce stage n'aurait pas été possible.

Je voudrais aussi remercier Pierre Poulin, directeur du laboratoire LIGUM de l'Université de Montréal, pour son accueil, sa bonne humeur et son aide tout au long du stage.

Je remercie aussi Nicolas Rous, étudiant en doctorat au laboratoire LIGUM, qui a travaillé avec moi, tout au long du stage, sur le projet de "Sketching" et qui à toujours fait preuve d'écoute, de facilité au travail en équipe et d'une volonté d'aide dès que la situation se présentait.

Après plusieurs heures d'acharnement où il m'a, très patiemment aidé sur mes problèmes mathématiques, je voudrais remercier Gilles-Philippe Paillé.

Après cinq mois de labeur maintenant, je tiens à remercier Arnaud Rosenblatt pour m'avoir supporté à ses côtés, mais aussi Eric Blanchard, Marc-Antoine Desjardins, Hui Shao Pei, Mohammed Yengui et Luc Le Blanc pour leur bonne ambiance au quotidien.

Résumé

Depuis plusieurs années maintenant, l'intuitivité et la facilité d'utilisation sont au centre des interfaces utilisateurs. Depuis les années 90 avec la sortie de la première tablette graphique, le *sketching* est devenu un des facteurs clés améliorant ces deux notions. Notre projet se décompose en deux contributions principales : *sketch* 2D et *sketch* 3D.

Dans un premier temps, on cherche à approcher au maximum et en 2D les techniques artistiques couramment utilisées en matière de *sketching*, c'est-à-dire déterminer interactivement et incrémentalement le tracé final d'un artiste à l'aide de l'ensemble de ses tracés. Pour cela, nous utiliserons l'analyse en composantes principales, PCA, locale.

Finalement, on cherche à utiliser cette technique afin d'esquisser en 3D. Cependant, naviguer dans une scène 3D, qu'elle soit vide ou munie d'objets quelconques, n'est pas du tout trivial, et il l'est encore moins pour l'esquisse 3D. C'est pourquoi nous passerons par l'esquisse 2D$\frac{1}{2}$, où l'utilisateur esquisse sur deux types de plan, les plans de dessin permettant de tracer directement la courbe 3D, et les plans de boîte de projections permettant d'éditer la courbe déjà créée à l'aide de projections. On introduit la notion de mise en correspondance par surface minimale, permettant d'associer deux courbes lorsque l'utilisateur redéfinit une des projections de la courbe.

Introduction

Aujourd'hui, l'ordinateur est un outil indispensable, mais il ne permet pas encore de remplacer toutes les techniques de travail et d'art. Dans ce rapport, on s'intéresse à développer une technique d'esquisse (*sketching*) par ordinateur où un artiste trace plusieurs traits pour obtenir une idée générale et ensuite la raffiner. Le concept semble a priori simple mais apporte vite un grand nombre de problématiques quant à son adaptabilité. Notre but se décompose en deux contributions principales. D'un côté l'aspect 2D, où l'on cherche à approcher ce que pourrait réaliser un artiste, et comment il aimerait le réaliser, c'est-à-dire afficher en temps réel le résultat final produit par l'ensemble des tracés, sans pour autant afficher tous les traits et cacher le résultat désiré. De l'autre l'aspect 3D où l'on étend notre technique grâce aux avantages que procure l'informatique afin de pouvoir créer des esquisses 3D.

L'avantage principal de l'esquisse sur papier est que l'artiste peut graduellement approcher le résultat final tout en affinant, petit-à-petit, son tracé. Il peut appuyer plus ou moins sur le crayon pour jouer sur l'importance de ses tracés. Il peut gommer des parties insatisfaisantes et calquer les parties intéressantes. Lorsqu'on étend informatiquement cette méthode, plusieurs problématiques entrent en jeu. Comment représenter numériquement les tracés d'un artiste ? Comment déterminer le résultat final attendu par rapport à l'ensemble des tracés produits ? En représentant les tracés de l'utilisateur par un nuage de points ordonné dans le temps et l'espace, on cherche à en extraire la courbe finale résultante. Les directions locales de cette courbe finale sont principalement déduites à l'aide de l'analyse en composantes principales, PCA. Un système de poids est mis en place afin de différencier l'importance des tracés, et entre directement en compte dans le calcul du PCA.
D'un autre côté, travailler par ordinateur a aussi ses avantages sur l'esquisse sur papier. D'abord il permet de revenir en arrière, d'effacer l'inutile et d'alterner entre plusieurs représentations. Contrairement au papier, il permet d'interagir avec des scènes 3D. L'idée ici est d'étendre les techniques utilisées par les artistes pour pouvoir dessiner en 3D. Cependant, évoluer dans un espace 3D étant déjà peu intuitif pour une grande majorité des utilisateurs, l'efficacité de l'esquisse 3D est fortement compromise. Peut-on tout de même rendre le 3D utilisable et ainsi exploiter cette grande source de talents artistiques traditionnels ? La technique mise en place se base essentiellement sur la redéfinition des "ombres" de la courbe, à l'aide de boîtes de projection.

Pour bien comprendre les méthodes de *sketching* mises en place durant le stage, on considère, avant de conclure, quatre parties différentes. La première explique les difficultés rencontrées lorsqu'on esquisse par ordinateur, ainsi que les techniques déjà développées pour résoudre ces problèmes. La seconde décrit comment obtenir un tracé final à partir d'ensembles de tracés produits. La troisième montre comment étendre le *sketching* au 3D. La dernière expose les différents tests et résultats obtenus pour notre approche du *sketching* par ordinateur.

Sketching par ordinateur

Transformer son ordinateur en feuille blanche et sa souris en crayon à papier n'est pas des plus facile. En effet, "dessiner sur ordinateur" est loin d'être efficace, spécialement lorsque nous essayons de dessiner dans un environnement 3D. Dans cette section, on montre que quelque soit l'artiste, la méthode d'esquisse est assez similaire. C'est cette méthode que l'on cherche, par la suite, à reproduire par ordinateur.

1.1 Loin du crayon, proche de l'artiste...

FIGURE 1.1 – Exemple de brouillon issu de White [Whi06].

Chaque artiste a son propre style, mais tous utilisent plus ou moins la même technique. Tout débute d'une base, d'une vague ébauche, qui devient de plus en plus précise au fil des tracés. Un artiste peut repasser des dizaines et des dizaines de fois sur un tracé pour l'affiner, jusqu'à arriver au résultat désiré. Cependant, ceci ne pose pas nécessairement d'inconvénients lorsque l'on travaille sur papier car on peut toujours gommer les parties insatisfaisantes. Mais qu'en est-il si l'on se pose devant un ordinateur ?

1.1.1 Du crayon à la souris

Après le clavier, la souris est l'outil le plus populaire pour "communiquer" avec son ordinateur. En effet, d'un côté on a une souris, entrée 2D dans la plupart des cas, qui est directement en lien avec un écran (LCD, CTR, etc.), 2D lui aussi. Il n'y a donc aucun

souci à ce niveau. Néanmoins, nous verrons dans la suite de ce rapport qu'il y a beaucoup plus de difficultés lorsque la souris est utilisée pour interagir avec un espace 3D.

Pour le moment, si l'on se focalise sur une scène 2D, il n'y a aucun problème à considérer que notre souris d'ordinateur fera office de crayon et que l'écran remplacera la feuille de papier. Maintenant, étudions de plus près les techniques artistiques couramment utilisées, afin d'essayer de les reproduire numériquement.

1.1.2 Du brouillon au résultat attendu

Le but de notre recherche est d'approcher au mieux les techniques artistiques couramment utilisées en esquisse. Cependant, plusieurs hypothèses et contraintes peuvent être mises en place, notamment face au type d'application auquel cette technique est vouée à être utilisée.

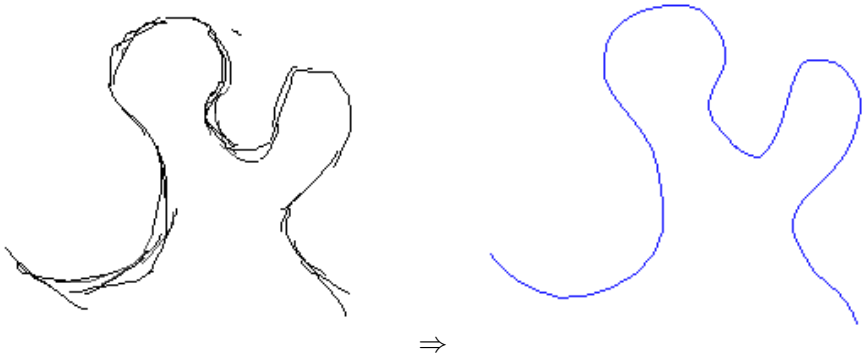
L'hypothèse principale sur laquelle se base notre travail est qu'habituellement, on considère que l'artiste peut repasser des dizaines de fois autour du même endroit pour affiner son tracé (voir figure 1.1). Les tracés peuvent être longs ou courts, légers ou gras.

Afin d'obtenir une bonne interface d'esquisse, il est aussi primordiale de limiter les interruptions dues aux processus de calculs. Ceci pourrait fortement nuire à la bonne représentation des techniques d'esquisse artistiques couramment utilisées.

L'expérience que l'on tire du 2D va pouvoir être utilisée au 3D. L'utilisateur étant beaucoup plus à l'aise au dessin de forme 2D que 3D, on veut utiliser cette information afin de faciliter l'esquisse 3D. L'utilisateur ne dessinera pas une scène 3D complète mais une succession de courbes planaires qu'il associera comme bon lui semble, et qui pourront représenter un chemin d'animation ou une déformation.

Finalement, étant donné que les courbes doivent, en partie, servir à l'animation, la courbe résultante doit, bien évidemment, être la plus proche possible des tracés, évoluer de façon intuitive, mais doit aussi être "lisse".

Objectif



(a) (b)

FIGURE 1.2 – (a) Tracés obtenus en récupérant les positions de la souris. (b) Résultat attendu.

1.2 Etat de l'art général

1.2.1 Le *sketching* en général

De jour en jour les performances informatiques sont de plus en plus élevées. C'est grâce à ces performances qu'aujourd'hui on cherche à améliorer les interfaces utilisateur, mais aussi à les rendre plus intuitives. Afin d'ajouter cette notion d'intuitivité à une interface utilisateur, le *sketching* entre souvent en jeu. Depuis la sortie de la première tablette graphique au début des années 90, les interfaces utilisateur basées sur le *sketching* sont de plus en plus présentes. Depuis plusieurs années, il existe une conférence "*Sketch-based Interfaces and Modeling*" entièrement dédiée à ce domaine, et récemment, Jorge et Samavati ont publié un livre [JS10] retraçant l'ensemble des applications impliquant du *sketching*. Ces applications regroupent un vaste domaine d'activités allant de l'animation (avec l'esquisse de courbes d'animation, de poses clés, etc.) à la modélisation (avec l'esquisse de modèles 2D ou 3D). Nous ne détaillerons ici que deux des travaux les plus reliés à notre approche.

1.2.2 "*Improving the Sketch-Based Interface*"

Le *sketching* est de plus en plus utilisé, néanmoins sa représentation actuelle se limite au dessin de courbes simples, ou de courbes successives. Peu de solutions proposées mettent en avant le calcul d'un tracé final à partir de tracés multiples. Pusch et al. [PSNW07] sont parmi les rares chercheurs à avoir étudié le problème. Leur approche est essentiellement basée sur l'analyse en composantes principales ou PCA (pour plus d'information sur le PCA, voir section 2). Leur but est d'ordonner l'ensemble des points produits par les tracés de l'utilisateur. Pour cela, ils considèrent qu'un ensemble de points est facile à ordonner s'ils ont plus ou moins la même composante principale. Le PCA est effectué à l'intérieur de boîtes englobantes alignées sur les axes X et Y. Si les tracés contenus dans chaque boîte n'ont pas une composante principale assez similaire, ils subdivisent cette boîte récursivement jusqu'à obtenir des tracés "de même direction", et donc jusqu'à pouvoir ordonner les points contenus dans chacune des boîtes. Le tracé final est obtenu en assemblant l'ensemble des résultats obtenus dans chaque boîte.

FIGURE 1.3 – Résultat de la méthode de Pusch et al. [PSNW07].

Comme on peut le voir sur la figure 1.3, les résultats obtenus sont plutôt convainquants. Néanmoins, leur solution présente quelques limites à notre vision du processus artistique. Comme ils l'expliquent, leur solution n'est pas en mesure de prendre en compte tous les types de tracés. Chaque tracé généré doit être de classe C1, c'est-à-dire que le tracé est dérivable sur un voisinage et que sa dérivée est continue. Plus concrètement, ils n'acceptent

pas de tracés aux formes extravagantes. De plus chaque tracé ne peut pas s'auto-intersecter. Finalement, leur technique n'est pas "sensible" au bruit. Or, dans la solution que nous aimerions développer, nous voudrions pouvoir donner à l'utilisateur la possibilité de créer des détails de plus en plus fins. Or est-ce que ces détails feront office de bruit dans leur algorithme ? Il s'agit donc d'une approche intéressante, avec de bons résultats, mais souffrant de quelques limitations qu'un artiste et nous aimerions éviter.

1.2.3 "*I Love Sketch*"

Bien que les techniques d'esquisse 2D soient de plus en plus étudiées, on trouve encore peu de travaux liés à l'esquisse 3D. Bae et al. [BBS08] ont proposé *ILoveSketch*, un système d'esquisse de courbes 3D. Habituellement, le gros plus souci avec les interfaces d'esquisse 3D est qu'elles placent l'utilisateur dans un monde 3D sans pour autant lui fournir l'adaptation adéquate permettant de passer de la 2D à la 3D en terme d'esquisse. *ILoveSketch* essaie de détourner ce problème pour que l'utilisateur puisse créer des réseaux de courbes 3D complexes. Pour cela, ils partent de plusieurs objectifs principaux. Premièrement, il est nécessaire à tout bon système d'esquisse d'avoir, lors du *sketching*, un minimum d'interruptions par le processus. Ensuite, ils facilitent au maximum la navigation 2D/3D (en ajoutant par exemple des changements de caméra automatiques). Finalement, ils affichent dynamiquement les informations visuelles nécessaires pour assister l'esquisse 2D/3D. Bae et al. [BBS08] proposent un grand nombre de techniques de création de courbes 3D. Chacune de ces techniques est utilisée suivant plusieurs contraintes qui sont visuelles (chaque angle de vue ne produira pas forcément le même résultat) ou géométriques (esquisse de courbes symétriques, de courbes en lien avec les autres courbes/surfaces "proches", etc.).

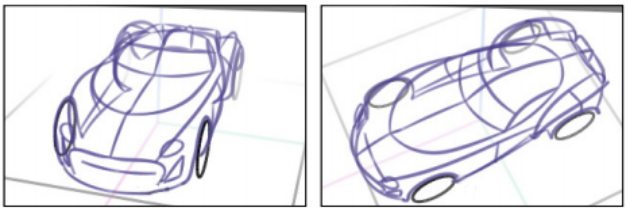


FIGURE 1.4 – Résultat de la méthode de Bae et al. [BBS08].

ILoveSketch résout en partie la problématique de l'esquisse 3D, et permet d'avoir de très bons résultats (figure 1.4). Néanmoins, il est bien précisé que le public visé pour leur système est assez restreint. En effet, utilisant un grand nombre de contraintes géométriques comme les symétries, et de contraintes visuelles comme le dessin en perspective, ce système est fortement déconseillé aux utilisateurs n'ayant pas de connaissances avancées en matière de *sketching*. Notre but est de pouvoir développer une technique d'esquisse 3D, un peu plus ouverte au grand public.

Bae et al. [BBS08] ont aussi ajouté la fonctionnalité de déterminer le tracé principal à partir de tracés multiples à leur système. Néanmoins, celui-ci n'est pas perfectionné et ne peut pas différencier des tracés de raffinements des tracés généraux comme nous aimerions faire.

Calcul de courbe principale

Numériquement, les tracés produits par l'utilisateur correspondent à une suite temporelle de points, ou un nuage de points, pour l'instant limité au 2D. Cette suite est obtenue en récupérant successivement dans le temps l'ensemble des positions de la souris à l'écran lors de son déplacement.

Cette section montre comment déterminer le tracé principal ou courbe, à partir d'un ensemble de points. Pour cela, on dispose de deux informations essentielles qui sont la position des points des tracés dans l'espace et à quel moment chaque point a été créé. Lorsqu'un artiste dessine sur papier, il cherche à approcher petit à petit le résultat désiré. La courbe principale représente, en temps réel, le résultat final obtenu avec les tracés déjà produits. Notre solution se base essentiellement sur le parcours du nuage de points généré par l'ensemble des tracés. C'est le résultat du parcours qui détermine la courbe principale.

2.1 Etat de l'art

2.1.1 Quelques notions sur le PCA

Partant d'un nuage de points, notre but est de pouvoir le parcourir. Pour ce faire, il est nécessaire d'extraire, à partir des position des points dans l'espace, d'autres informations qui faciliteront ce parcours. Le PCA (ou ACP en français, analyse en composantes principales) permet d'extraire les directions principales d'un nuage de points. C'est cette notion de "direction de nuage" qui est importante au calcul de la courbe principale.

Le PCA utilise les transformations orthogonales (voir appendice 1) afin de convertir un ensemble d'observations de variables en un ensemble de valeurs de variables non corrélées, appelées composantes principales. La transformation est définie de telle sorte que la première composante principale ait la plus forte variance. Les composantes principales suivantes, quant à elles, sont aussi ordonnées successivement selon la plus forte variance, sous la contrainte d'être orthogonale à la composante précédente.

Explication en image

Les points rouges (figure 2.1) symbolisent, ici, les variables sur lesquelles on effectue l'analyse.

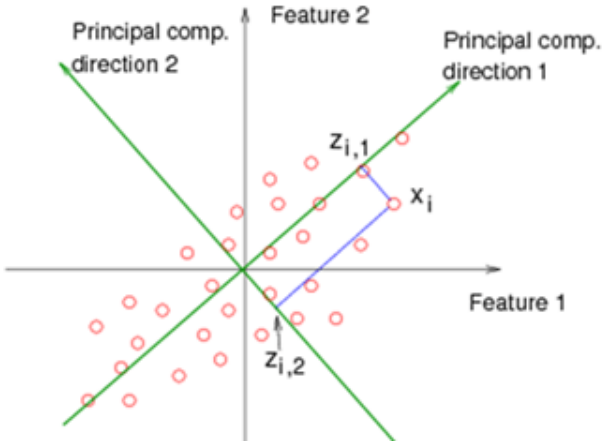


FIGURE 2.1 – Explication du PCA.

Calcul mathématique

Les explications mathématiques ci-dessous sont principalement issues du tutoriel proposé par Garcia [Gar09].

Supposons $\mathbf{X}$, une matrice de n observations x_{ij} où le nombre de colonnes correspond au nombre de dimensions des données. Supposons maintenant que l'on soustrait à chaque colonne de l'ensemble de départ la moyenne $\mu_\mathbf{j}$ (moyenne de la dimension correspondante). On obtient alors une nouvelle matrice $\mathbf{Y}$ ayant sa moyenne centrée en zéro. Le calcul du PCA, est généralement obtenu avec une décomposition en valeurs singulières (SVD), qui est définie de la sorte :

$$\begin{aligned} \mathbf{X}/\mu_\mathbf{j} &\rightarrow \mathbf{Y} \\ \frac{1}{(n-1)}\mathbf{Y}\mathbf{Y}^\mathrm{T} &\rightarrow \mathbf{A} \\ \mathbf{A} &\rightarrow \mathbf{U}\mathbf{S}\mathbf{V}^\mathrm{T} \end{aligned}$$

Calculer $\mathbf{Y}\mathbf{Y}^\mathrm{T}$ revient à produire une matrice contenant les sommes des déviations. Multiplier $\mathbf{Y}\mathbf{Y}^\mathrm{T}$ par $1/(n-1)$ permet d'obtenir la matrice $\mathbf{A}$ où les éléments situés sur la diagonale correspondent à la variance σ_{ij}^2 et les autres éléments à la co-variance $\sigma_i\sigma_j$. $\mathbf{A}$ est souvent appelée la matrice de covariance de $\mathbf{X}$.

$\mathbf{A}$ est décomposée en trois matrices grâce à la SVD : $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^\mathrm{T}$, où $\mathbf{S}$ est une matrice diagonale contenant les valeurs singulières. $\mathbf{U}$ et $\mathbf{V}$ sont des matrices orthogonales, leurs colonnes sont les vecteurs propres "droits et gauches" de $\mathbf{A}$. La première colonne de $\mathbf{V}$ correspond à la composante principale ayant la plus forte variance, la deuxième colonne à la deuxième composante principale, et ainsi de suite.

Algorithme itératif

Tous ces calculs sont souvent fastidieux et peuvent considérablement ralentir leur application. Pour éviter ces calculs et déterminer rapidement les composantes principales d'un nuage de points, il existe un algorithme itératif [Wik11]. Cet algorithme est déconseillé lorsque l'on fait face à des données de grandes dimensions car il biaise le résultat. Néanmoins,

ce sera la solution retenue pour notre projet, puisque, premièrement les données dont l'on dispose sont 2D ou 3D (données de faibles dimensions), et on ne cherche pas à avoir un calcul exact des composantes principales, puisqu'une approximation est largement suffisante.

En reprenant les données précédentes, l'algorithme permettant d'obtenir la première composante principale est défini de la sorte :

Algorithm 1 PCA Itératif

$\mathbf{p} = 0$
$c = 0$
while c < nombre d'itérations max **do**
  $\mathbf{t} = 0$ /*un vecteur de taille égale au nombre de dimensions*/
  for chaque ligne $\mathbf{y} \in \mathbf{Y}^{\mathrm{T}}$ **do**
    $\mathbf{t} = \mathbf{t} + (\mathbf{y} \cdot \mathbf{p})\mathbf{y}$ /*$\mathbf{y} \cdot \mathbf{p}$ est le produit scalaire entre $\mathbf{y}$ et $\mathbf{p}$*/
  end for
  $\mathbf{p} = \frac{\mathbf{t}}{|\mathbf{t}|}$
  $c = c + 1$
end while

Les composantes principales suivantes sont obtenues en soustrayant la composante $\mathbf{p}$ à $\mathbf{Y}^{\mathrm{T}}$ et ensuite en répétant l'algorithme.

Que ce soit pour servir de droite sur laquelle on projette des points, ou de direction de parcours, le PCA est à la base de toutes les solutions qui ont été explorées, et est aussi à la base de notre solution finale.

2.1.2 Algorithme de Hastie et Stuetzle

L'algorithme de Hastie et Stuetzle [HS89] a été le premier étudié, et est aussi l'un des premiers algorithmes à avoir mis en place cette notion de courbe principale.

Ils se servent du PCA comme direction de projection. Après avoir calculé les moyennes des ensembles de points se projetant dans un même voisinage, c'est l'ensemble de ces points "moyens" qui représentent la courbe principale. Néanmoins, limiter le calcul de la courbe principale à une simple projection sur la direction principale de l'ensemble du nuage de points peut vite poser problème.

Algorithme

Soit $\mathbf{X}$ un vecteur de n points 2D. La première composante principale de $\mathbf{X}$ est une droite dont la distance euclidienne entre celle-ci et $\mathbf{X}$ est minimale.

Considérons $f(t) = (f_1(t), ..., f_n(t))$ représentant une courbe lisse de $\mathbf{X}$. Pour tout $\mathbf{x} \in \mathbb{R}^n$, on a : $t_f(\mathbf{x})$ la plus grande valeur du paramètre t pour laquelle la distance entre $\mathbf{x}$ et $f(t)$ est minimisée (figure 2.2). Plus formellement, les indices de projection $t_f(\mathbf{x})$ sont définis par :

$$t_f(\mathbf{x}) = \sup\left[t : ||\mathbf{x} - f(t)|| = \inf_\tau ||\mathbf{x} - f(\tau)||\right]$$

Voici pas à pas l'évolution de l'algorithme de Hastie et Stuetzle :

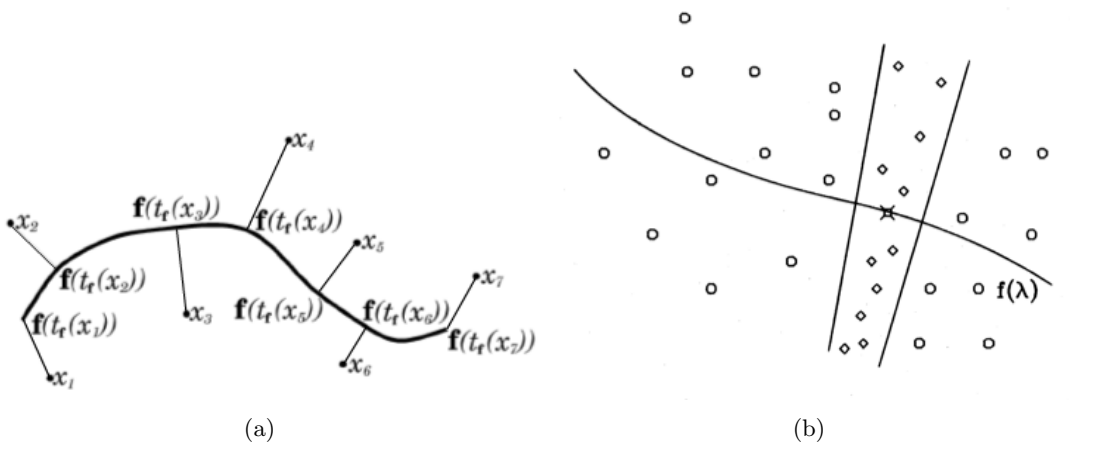
Algorithm 2 Algorithme HS

Étape 0. (Initialisation) $f^{(0)}(t)$ est la première composante principale de $\mathbf{X}$; $j = 1$.

Étape 1. On définit $f^{(j)}(t) = E(\mathbf{X}|t_{f^{j-1}}(\mathbf{X})) = t$.
Ceci correspond à la moyenne de tous les points se projetant dans un voisinage fixé.

Étape 2. $t_{f^j}(\mathbf{x}) = \max[t : ||\mathbf{x} - f^j(t)|| = \min_\tau ||\mathbf{x} - f^j(\tau)||]$
On met à jour les couples projection/point.

Étape 3. $\Delta(f^j) = E||\mathbf{X} - f^j(t_{f^j}(\mathbf{X})||^2$
Si $|\Delta(f^j) - \Delta(f^{j-1})| < seuil$, alors on arrête d'itérer. Sinon $j = j+1$ et retour à l'étape 1.
Condition d'arrêt de stabilité.



(a) (b)

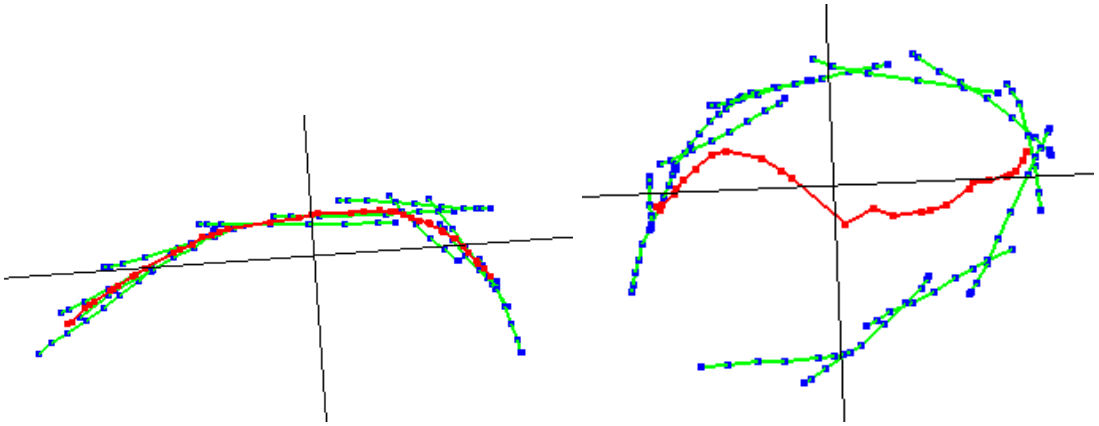
FIGURE 2.2 – (a) Illustration des projections. (b) Un point de la courbe résultat est obtenu par moyenne des points se projetant dans un voisinage fixé.

Limite principale de l'algorithme

Le calcul de la courbe principale développé par Hastie et Stuezle a été un bon début de recherche. Cependant, il présente un désavantage majeur : il ne permet pas de gérer les courbes fermées, voire semi-fermées (cf. figure 2.3b). Plusieurs améliorations, comme celles présentées par Kégl et al. [KLZ00], ont été testées afin de pouvoir gérer les courbes fermées. Néanmoins, les résultats obtenus ne nous satisfaisaient pas suffisamment. De plus, ces techniques ne permettaient pas à l'utilisateur de raffiner son tracé.

Résultat

A la figure 2.3, en vert correspondent les tracés générés par l'utilisateur. Les points bleus sont les points obtenus par tracés correspondant aux positions de la souris à l'écran. Le nombre de points obtenus en déplaçant la souris est fortement lié aux performances graphiques de l'ordinateur. Finalement, la courbe rouge symbolise la courbe produite avec l'algorithme de Hastie et Stuetzle.



(a) (b)

FIGURE 2.3 – (a) Résultat sur une courbe ouverte. (b) Résultat sur une courbe semi-fermée.

2.1.3 Algorithme LPC

Bien d'autres techniques, comme le noyau PCA de Mika et al. [MSS+99], permettent de parer ce problème de courbes fermées. Or, d'après les hypothèses décrites précédemment, on veut donner la possibilité à l'artiste de ré-éditer ses courbes, mais aussi, une fois encore, de pouvoir lui ajouter des détails de plus en plus fins s'il le souhaite. Afin d'arriver à ce résultat, notre solution est essentiellement basée sur l'algorithme LPC de Einbeck et al. [ETE05].

Présentation de l'algorithme

Considérons $\mathbf{X}$ un nuage de n points 2D : $\mathbf{X}_i \in \mathbb{R}^2, i = 1, ..., n$ où $\mathbf{X}_i = (\mathbf{X}_{i1}, ..., \mathbf{X}_{id})$. L'algorithme LPC parcourt le nuage de points grâce à une succession de PCA locaux (figure 2.4). L'objectif est de trouver une courbe "lisse" passant par le milieu du nuage de points. Cette courbe est calculée en déterminant les "centres de masse locaux" (c'est-à-dire les points moyens des voisinages locaux), grâce à l'algorithme suivant :

Algorithm 3 Algorithme LPC

Étape 0. Choisir d'un point de départ $x_{(0)}$. Fixer $x = x_{(0)}$.

Étape 1. Calculer le centre de masse μ^x autour de x.

Étape 2. Effectuer un PCA localement en x.

Étape 3. Trouver la nouvelle valeur de x en suivant la première composante principale γ^x et en débutant de μ^x.

Étape 4. Répéter les étapes 1 à 3 jusqu'à ce que μ^x soit approximativement constant.

La série des μ^x déterminera la courbe principale.

Explication en image

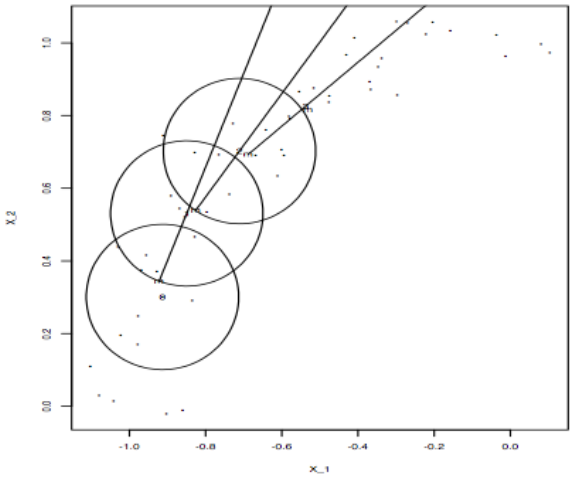


FIGURE 2.4 – Explication de l'algorithme LPC.

Remarques

Le choix du premier point est important pour cet algorithme. Dans notre cas, comme le dernier tracé est toujours considéré comme le plus important, c'est le point central de ce dernier tracé qui sera choisi comme point de départ.
Le centre de masse est la moyenne des points contenus dans le voisinage local.

Cet algorithme permet de "parcourir" localement la courbe. Ne pas oublier que l'on a débuté du centre du nouveau tracé dans une direction fixée, et donc que la direction opposée n'a pas encore été traitée. Il est donc important, une fois arrivé en "fin de nuage", de repartir en sens inverse. Depuis $x_{(0)}$, il suffit de faire $\gamma^x_{(0)} = -\gamma^x_{(0)}$ et de recommencer l'algorithme.

Un autre point à prendre en compte est le maintien de la direction. Le PCA calcule la direction principale du nuage mais n'est pas nécessairement stable au niveau du "sens" de ce vecteur direction. C'est pourquoi on assure le maintient de la direction par un simple produit scalaire entre la direction courante et la précédente :

$$\cos(\alpha^x_{(i)}) = \gamma^x_{(i)} \cdot \gamma^x_{(i-1)}.$$

Si $\cos(\alpha^x_{(i)}) < 0$, alors on fixe $\gamma^x_{(i)} = -\gamma^x_{(i)}$ et on poursuit l'algorithme normalement.

Einbeck et al. [ETE05] proposent une méthode de "pénalisation d'angle" permettant, entre autres, de pouvoir maintenir la direction et gérer les "croisements". Cependant, celle-ci ne sera pas décrite car, après tests, il s'est avéré que leur méthode n'est pas assez stable pour notre utilisation.

Limites

En comparaison à l'algorithme de Hastie et Stuetzle [HS89], l'algorithme LPC permet de régler le problème des croisements. Même si leur technique ne semble pas assez stable pour notre système, plusieurs pistes peuvent être explorées et améliorées sur cet aspect.

Cependant, dans la version proposée par Einbeck et al. [ETE05], il n'est toujours pas possible de différencier les points symbolisant des raffinements des points des tracés plus globaux. Notre solution résout ce problème en utilisant principalement la notion de voisinage adaptatif.

2.2 Notre solution

Notre solution utilise la notion de temporalité des tracés pour différencier leur importance. De plus, on émet pour hypothèse que, comme un tracé est un ensemble de points récupérés grâce aux positions de la souris, plus il y a de points autour d'un même endroit, plus l'utilisateur cherche à affiner son tracé à cet endroit.

2.2.1 Mise en place du système de poids

L'idée de cette partie est de différencier l'importance de chacun des tracés en prenant en compte que les tracés les plus récents, effectués à un endroit précis, ont plus d'importance que les anciens. Chaque point d'un tracé est initialisé avec un poids de 1.0. Le poids d'un point peut diminuer au fur et à mesure, et en fonction des nouveaux tracés. Par conséquent on peut fixer une règle symbolisant la décroissance des poids de chacun des points d'un tracé. Dans notre cas la décroissance d'importance est linéaire. Soit p_i le i^{eme} point du tracé p :

Algorithm 4 Mise à jour des poids

```
if le poids est à diminuer then
    Poids(p_i) = αPoids(p_i)
end if
if Poids(p_i) < seuil then
    Suppression du point
    if nombre de points du tracé p = 0 then
        Suppression du tracé
    end if
end if
```

Le valeurs de α et du *seuil* déterminent l'évolution de l'importance des tracés. Dans notre cas nous avons fixé $\alpha = 0.5$ et le $seuil = 0.3$, les poids étant initialement fixés à 1.0, ce qui correspond à garder les trois derniers points tracés à un même endroit.

La figure 2.5 montre l'évolution des poids d'un *sketch*, les numéros correspondent à l'ordre dans lequel les tracés ont été créés.

Les points en rouge représentent des points de poids 1.0, en vert de poids 0.5 et en bleu de poids 0.25. Sur la figure 2.5 (a), le tracé 2 modifie les poids du tracé 1. Sur la figure 2.5 (b), le tracé 3 modifie à la fois les poids des tracés 1 et 2.

(a) (b)

FIGURE 2.5 – Evolution des poids entre deux (a) et trois (b) tracés.

Les boîtes englobantes orientées

La première étape, avant même de mettre à jour les poids, est de détecter quels sont les poids à mettre à jour, avec les boîtes englobantes orientées (OBB).

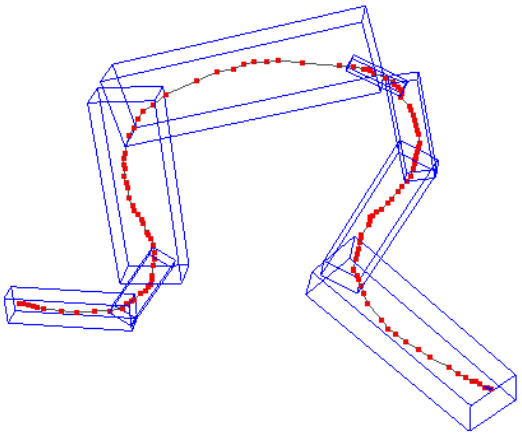


FIGURE 2.6 – Exemple de boîtes englobantes orientées le long d'un seul tracé en 3D.

Si l'on regarde la figure 2.6, chacune des suites de points est contenue dans une boîte. Cette boîte s'adapte à la configuration de sa suite de points associés et donc adapte son orientation.

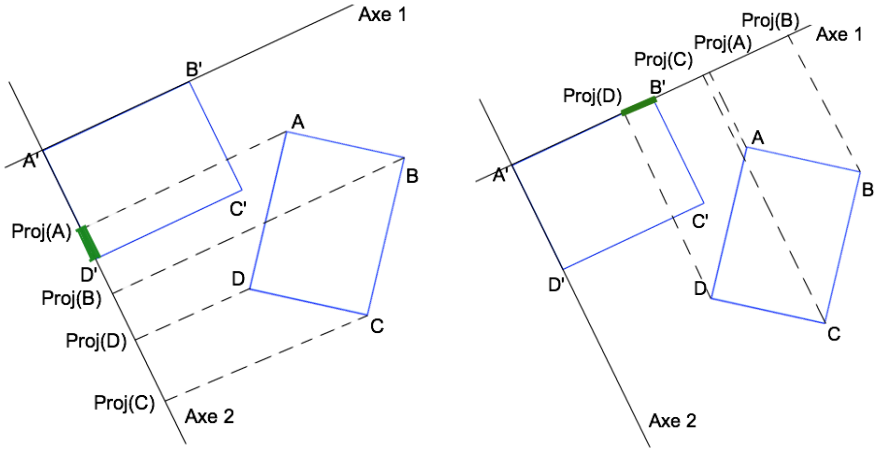
Après avoir été généré, chaque tracé est re-découpé selon sa courbure. La découpe d'un tracé est simple. La courbure en un point i du tracé correspond à la tangente en ce point (pour plus d'information voir la partie "Un nuage de point, un champ vectoriel" de la section 2.2.2). Si le produit scalaire entre la direction $\mathbf{d_i}$ au point i et la direction $\mathbf{d_0}$ du premier point (0) de découpe dépasse un certain seuil, on découpe le tracé, i.e. on crée une nouvelle courbe.

$$\frac{\mathbf{d_i}}{||\mathbf{d_i}||} \cdot \frac{\mathbf{d_0}}{||\mathbf{d_0}||} > seuil$$

A sa création, la boîte englobante suit de près le tracé. Or il est nécessaire d'y ajouter un *offset* afin de détecter les intersections entre boîtes. Cet *offset* est relatif à la longueur du tracé. Plus le tracé est long, plus l'*offset* de sa boîte englobante sera important. L'*offset* ajouté est le même sur chaque dimension.

Intersections

Une fois que les boîtes des tracés sont générées, il ne reste plus qu'à tester les intersections entre les boîtes englobantes du nouveau tracé et celles des anciens tracés afin de déterminer si on doit réduire ou non le poids de points des anciens tracés. Pour cela on teste les chevauchements entre boîtes.



(a) (b)

FIGURE 2.7 – (a) Test axe 2. (b) Test axe 1.

Pour tester l'intersection, la première étape est de projeter les "coins" d'une des boîtes sur les axes principaux de l'autre (en 3D on étend le schéma avec le troisième axe principal).

Après projection, il existe deux types de chevauchements. Le premier est illustré à la figure 2.7. Le second est obtenu lorsque les projections d'une boîte englobent les projections de l'autre.
Une fois que les étapes précédentes ont été effectuées pour une boîte, et que les chevauchements ont été validés, on recommence en projetant sur les deux axes principaux de l'autre boîte. On remarque que dans l'exemple présent, l'intersection n'est pas validée (voir figure 2.8).

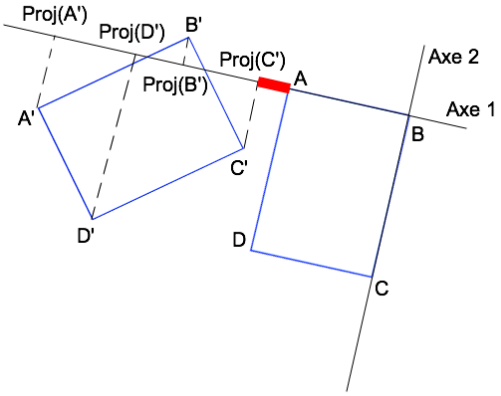
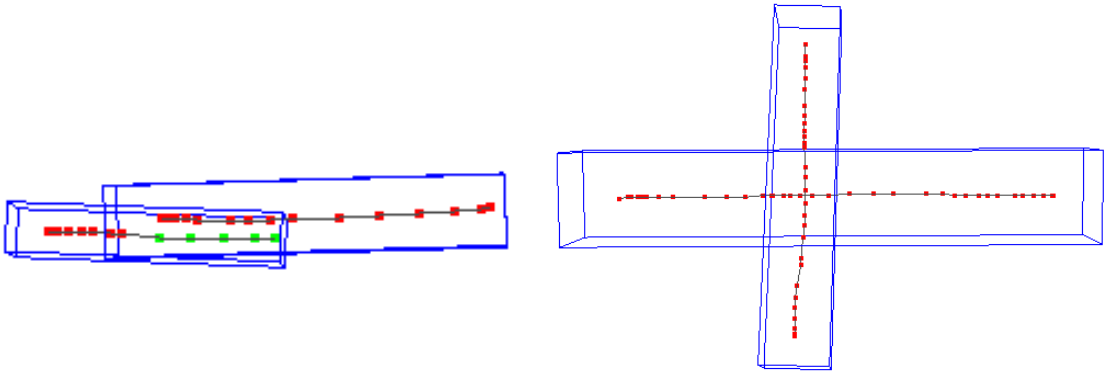


FIGURE 2.8 – Chevauchement négatif.

Mise à jour des poids

Pour résumer, si la boîte englobante d'un nouveau tracé intersecte la boîte englobante d'un ancien tracé, on réduit le poids de certains points de l'ancien tracé que l'on considère proches du nouveau tracé. Pour cela on projette les points de l'ancien tracé sur la composante principale du premier. Les points situés "à l'intérieur" du nouveau tracé voient leur poids réduit (voir figure 2.9 (a)).

Néanmoins, il y a une petite exception. Afin de ne pas affecter le poids de points lorsque l'intersection de boîte correspond à une boucle ou un retour sur le tracé lui-même, on décide de regarder le produit scalaire entre les deux axes principaux des boîtes. S'il est supérieur à un certain seuil, l'intersection est rejetée, et les poids ne sont pas réduits (voir figure 2.9 (b)).



(a) (b)

FIGURE 2.9 – (a) Intersection validée. (b) Intersection rejetée.

Utilisation des poids

Pour prendre en compte les poids attribués, on modifie l'algorithme du PCA itératif afin de donner plus d'importance aux points ayant les poids les plus importants dans le calcul des composantes principales. La direction va alors tendre plus vers les points de poids élevés. Pour mieux comprendre l'algorithme 5 (nouvel algorithme du PCA itératif), voici quelques informations essentielles :

La fonction *normalisationPoids* renormalise entre 0.0 et 1.0 les poids de l'ensemble sur lequel on calcule le PCA.

La fonction f augmente l'importance des points de poids élevé. Dans notre cas c'est une fonction exponentielle fixée entre $b = 0$ et $a = 5$. Ainsi les points de poids 1.0 prennent entièrement le dessus pour la détermination de la direction.

On remarque qu'une étape de stabilisation a été ajoutée. Si l'ancien tracé **ancienP** et le nouveau tracé **p** sont "quasiment" alignés, on arrête l'algorithme.

Comme l'algorithme LPC parcourt un nuage de points grâce à des PCA locaux, favoriser la direction sur les points les plus importants entre directement en lien avec la manière dont sera parcouru le nuage.

Algorithm 5 PCA itératif avec prise en compte des poids

```
normalisationPoids(Y)
p = 0
ancienP = 0
c = 0
while c < nombre d'itérations max do
  t = 0 /*un vecteur de taille égale au nombre de dimensions*/
  for chaque ligne y ∈ Y^T do
    poids = [f(a * poids(y)) − f(b)]/f(a) /*a et b ∈ ℝ*/
    t = t + poids * (y · p)y
  end for
  ancienP = p
  p = t/|t|
  if ancienP · p > 0.99 then
    break
  end if
  c = c + 1
end while
```

2.2.2 Modifications apportées à l'algorithme LPC

On rappelle que la faiblesse principale pour notre application de l'algorithme LPC est qu'il ne permet pas de différencier les points symbolisant des raffinements locaux de ceux faisant partie de tracés plus globaux. De plus après tests, il s'est avéré que la gestion de croisements n'était pas assez stable pour bien répondre aux désirs de l'artiste. Voyons maintenant l'ensemble des spécificités modifiées ou ajoutées à l'algorithme pour résoudre ces problèmes.

Un nuage de points, un champ vectoriel

Chaque tracé, constitué d'une suite ordonnée de points, va maintenant être considéré comme une suite de particules ayant un vecteur direction associé (voir figure 2.10). La direction associée est la tangente au point, c'est-à-dire la moyenne entre le vecteur i et le vecteur $i-1$.

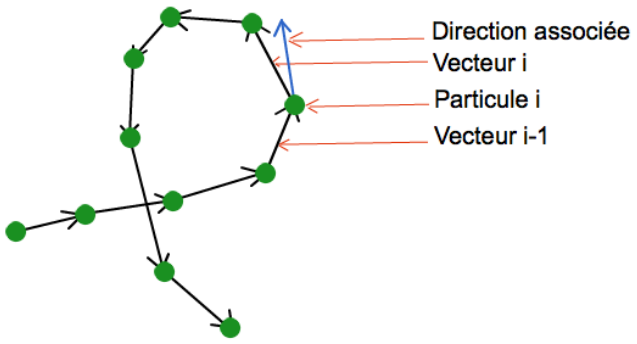


FIGURE 2.10 – Champ vectoriel sur une suite de points formant un tracé.

En reprenant maintenant l'algorithme de calcul de courbe principale, le calcul du centre

de masse sur un voisinage autour de x va comporter une étape supplémentaire. Pour chacun des points du voisinage, si le produit scalaire entre la direction de parcours (1^{ere} composante principale) et la direction du point en question est inférieur à un certain seuil, le point ne sera plus pris en compte dans le voisinage et donc dans le calcul du centre de masse. Ceci élimine des points à éviter. La gestion des croisements est donc beaucoup plus stable.

Un parcours adaptatif de type "lampe torche"

Comme on a pu le voir sur la figure 2.4 représentant l'algorithme LPC [ETE05], le voisinage local est déterminé par un cercle de rayon fixe. Pour étendre cette technique à un voisinage adaptatif, un nombre de points maximal est fixé pour un voisinage. De cette façon, plus il y aura de points dans un petit périmètre, plus le parcours sera affiné. L'utilisateur pourra alors ajouter des détails à son *sketch*. De plus, on remarquera que l'on n'utilise pas des cercles mais des torches orientées qui favorisent la direction et limite le nombre de points du voisinage. L'algorithme suivant détaille ce fonctionnement.

Algorithm 6 Parcours "Lampe Torche"

```
while !stop do
  stop = faux
  voisinage = 0 /*On vide le vecteur voisinage*/

  for p ∈ pointsAConsiderer do
    /*On regarde si le point courant est contenu dans la torche centrée sur centre et
    orientée sur direction*/
    if direction · p.direction ≥ cos(angleRecherche) then
      if voisinage.taille() = nbPointsMaxVoisinage then
        /*On garde, en considérant le nouveau point, les nbPointsMaxVoisinage points
        les plus proches du centre de masse*/
        voisinage = plusProcheCentre(centre, voisinage)
      else
        voisinage.ajouter(p) /*On ajoute le points au voisinage*/
      end if
    end if
  end for

  if voisinage.taille() < nbPointsMinVoisinage then
    if tailleCouranteTorche < tailleMaxTorche then
      tailleCouranteTorche *= 2.0 /*Elargissement de la recherche*/
    else
      stop = vrai
    end if
  else
    tailleCouranteTorche = 2.0*plusLongueDist(center,voisinage)
    center = moyenne(voisinage) /*On calcule le nouveau centre de masse*/
    direction = pca(voisinage).premierePC() /*Nouvelle direction de torche*/
    courbeFinale.ajouter(centre)
  end if
end while
```

Etant donné qu'à l'initialisation, nous ne disposons pas encore de direction de parcours, on ne cherchera pas les points dans une torche mais dans un cercle, comme dans l'algorithme de base. On considère qu'après l'initialisation, on est en possession d'un centre de masse et d'une direction, ce qui est suffisant pour définir une torche.

$nbPointsMinVoisinage$ et $nbPointsMaxVoisinage$ sont des variables fixes. Le seuil minimal sera égal à 2. Nombre de points minimal nécessaire pour obtenir une direction de parcours. Le seuil maximal permet de jouer sur le lissage de la courbe. Il est, pour nous, fixé à 15.
direction est la première composante principale du voisinage.
$angleRecherche$ est une variable fixe définissant l'angle d'ouverture de la torche.

Au début de l'étape de recherche, $tailleCouranteTorche$ est fixée à deux fois la taille de la torche précédente (grâce à la fonction $plusLongueDist$ qui détecte la plus longue distance entre les points du voisinage et du centre de masse). Ceci évite de chercher dans de grands espaces lorsque l'on est en train de parcourir des petits raffinements.

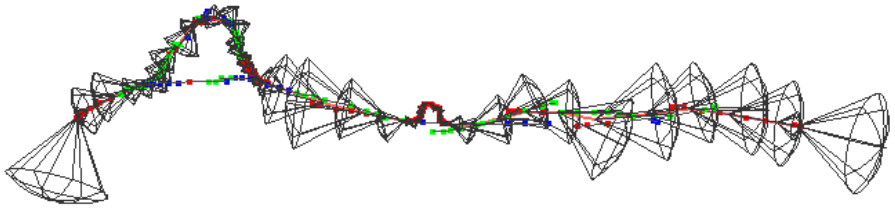
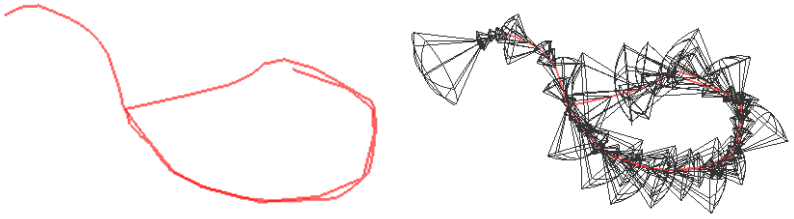


FIGURE 2.11 – Illustration du parcours avec les torches correspondantes.

En figure 2.11, on remarque le changement de direction des torches qui est du au fait que le point de départ du parcours est toujours le point central du dernier tracé. Sur cet exemple, le dernier tracé a été fait aux alentours du milieu de tous les tracés. De plus, on remarque que le parcours tend vers les points de poids élevé (en rouge sur figure 2.11).

Néanmoins, un point de l'algorithme reste à éclaircir. On remarque que l'algorithme boucle sur "tous les points à considérer". On pourrait croire que ceci correspond à tous les points du nuage de points, c'est-à-dire à l'ensemble des points correspondant à tous les tracés. Cependant, comme on recherche un nombre de points fixe (afin d'adapter le voisinage), s'il n'y a pas assez de points, la taille de la torche va grossir afin de voir s'il n'y a pas, plus loin, de points pouvant être pris en considération. Malgré l'étape de suppression présentée en sous-section suivante, il est fort possible que quelques points n'ont pas été supprimés, car non parcourus. On peut voir sur la figure 2.13 que ceci peut très vite poser problème :

FIGURE 2.12 – Tracés en entrée.



(a) (b)

FIGURE 2.13 – (a) Courbe générée. (b) Torches de parcours.

Pour parer ce problème, on décide de ne pas rechercher les points du cône dans tous le nuage de points mais dans un ensemble de points ciblés.

Avec ce type de parcours, le centre du cône est situé à l'intérieur d'une ou de plusieurs boîtes englobantes de tracés de base. Le "tous les points à considérer" correspond alors à l'ensemble des points des tracés t dont la boîte englobante contient le centre du cône, et des tracés dont la boîte englobante intersecte celle des tracés t.

FIGURE 2.14 – Résultat du parcours avec ensemble de points ciblés.

De plus, cette optimisation permet d'améliorer le parcours sur des courbes composées de croisements. Avant d'ajouter les points dans l'ensemble des points à prendre en considération, on peut aussi regarder si la boîte est "orientée" dans la direction du cône, et donc supprimer les tracés intersectant, mais ne devant pas être traités.

Suppression de points

Afin d'éviter à l'algorithme de boucler indéfiniment et de reconsidérer des points déjà traités, on y ajoute une étape qui supprime, dans le voisinage local, les points derrière le nouveau centre de masse (voir figure 2.15).

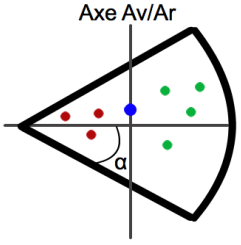


FIGURE 2.15 – Suppression de points.

Le point en bleu est le nouveau centre de masse, les points en vert sont les points conservés, et ceux en rouge sont supprimés de l'ensemble des points à traiter. α correspond à l'angle d'ouverture de la torche.

2.2.3 Résultats et limites

Les résultats obtenus par notre méthode sont assez convainquants même s'ils présentent quelques limites. En effet, il n'est pas possible générer une courbe qui repasserait plusieurs fois au même endroit, comme par exemple, une droite qui ferait des aller-retour sur elle-même. Cependant, pour parer ce problème, nous avons ajouté la possibilité de commencer de nouvelles courbes, sans pour autant avoir développé de méthodes fiables pour remettre par la suite toutes ces courbes ensemble. De plus, comme notre solution fonctionne essentiellement à l'aide de voisinages adaptatifs, l'algorithme ne peut fonctionner s'il n'y a pas assez de points dans une courbe.

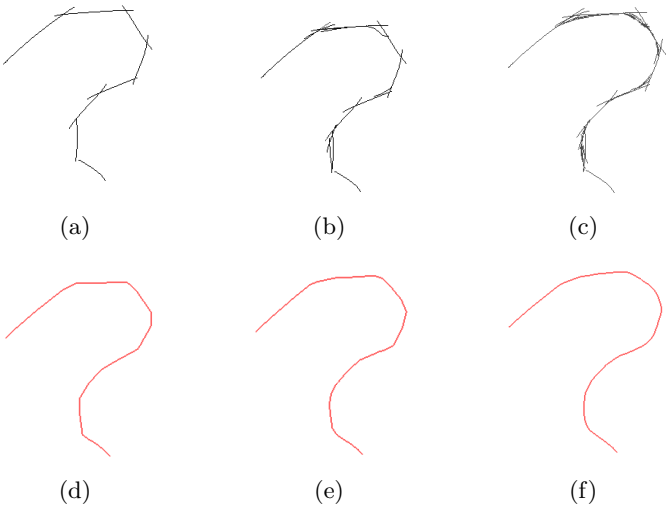


FIGURE 2.16 – Évolution chronologique d'un tracé. (d) (e) (f) résultats de (a) (b) (c).

	Nb Tracés	Nb Tracés utilisés	Nb Points	Nb Points Utilisés
Étape 1	8	8	164	164
Étape 2	18	18	292	277
Étape 3	78	63	1287	934

Le tableau ci-dessus montre les statistiques du test de la figure 2.16. L'étape 1 correspondant aux figures 2.16 (a) et (d), l'étape 2 aux figures 2.16 (b) et (e), et l'étape 3 aux figures 2.16 (c) et (f). L'étape 2 montre que certains points de poids trop faible ont été supprimés. Et l'étape 3 montre que certains tracés inutiles, car n'ayant plus de points, ne sont plus pris en compte.

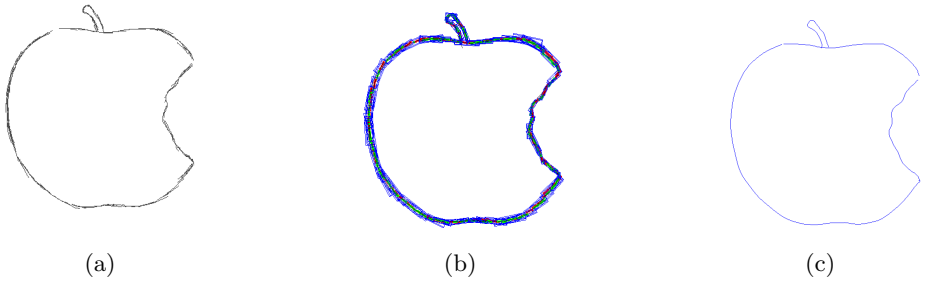


FIGURE 2.17 – (a) Tracés produits par l'utilisateur. (b) Poids et boîtes englobantes correspondants. (c) Esquisse générée par notre algorithme.

Propagation au 3D

Grâce à l'informatique, on peut désormais pousser beaucoup loin les techniques de "sketching" et esquisser en 3D fait partie d'une des améliorations possible. L'extension de l'algorithme de calcul de courbe principale au 3D ne pose pas de problème en lui-même. Le plus gros soucis reste sur la problématique posée par "Comment dessiner en 3D" ? On le rappelle, la souris n'est pas le meilleur outil pour interagir avec une scène 3D, et il est difficilement pensable de pouvoir dessiner en toute liberté dans cette scène 3D contenant aussi des espaces vides. Cependant, on essaie de détourner ce problème en instaurant deux approches : l'édition de courbe et la création de courbes.

3.1 Etat de l'art

3.1.1 "*On Expert Performance in 3D Curve-Drawing Tasks*"

Schmidt et al. [SKKS09] ont testé la précision d'un artiste lorsqu'on lui demande de dessiner sur un plan ou sur un objet 3D, suivant plusieurs points de vue. Dans un des tests, illustré à la figure 3.1, ils demandent à l'artiste de dessiner, dans un premier temps, un cercle sur un plan 3D, puis un trait suivant la courbe sur la forme d'un cylindre.

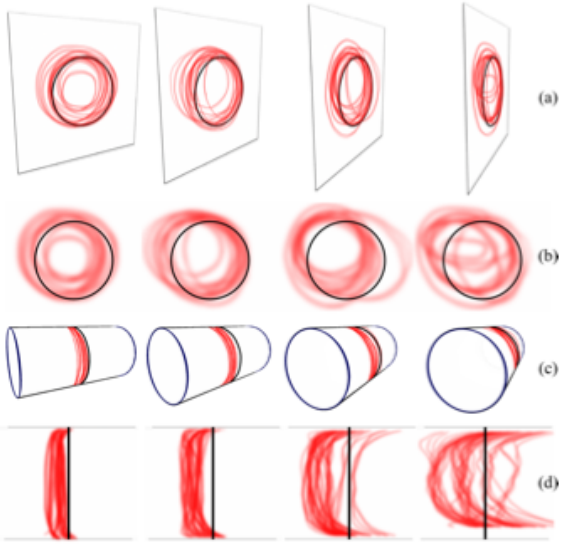


FIGURE 3.1 – Résultats des tests Schmidt et al. [SKKS09].

En rouge, les tracés produits par l'utilisateur, le résultat correct est en noir. On remarque,

en (a), que plus le plan de dessin est oblique au point de vue, moins précis est l'utilisateur dans son *sketch*. Ce biais est encore plus présent lorsqu'on lui demande de dessiner sur une surface 3D, même aussi simple et régulière qu'un cylindre, en (c) et (d). Cette série de tests illustre bien le fait que l'utilisateur n'est pas à l'aise lorsqu'il s'agit d'interagir avec une scène 3D, et encore moins lorsqu'il doit dessiner dans celle-ci.

3.1.2 "*An Interface for Sketching 3D Curves*"

Quelques techniques ont été développées pour faciliter le dessin en 3D, mais aucune d'entre elles ne se démarque vraiment des autres. La méthode retenue pour notre solution est celle proposée par Cohen et al. [CMZ+99]. Comme expliqué précédemment, l'utilisateur est beaucoup plus à l'aise en dessinant sur un plan. Cohen et al. permettent à l'utilisateur de créer des courbes 3D en dessinant sur différents plans. Ils utilisent l'ombre, c'est-à-dire la projection de la courbe sur un plan, pour extraire la 3^{eme} dimension de la courbe. L'utilisateur dessine une première courbe, qui est projetée suivant un vecteur fixé au vecteur Y du repère orthogonal. Ceci projette en quelque sorte la courbe sur le "sol". Finalement, une fois que l'utilisateur a redessiné l'ombre celle-ci est déprojetée dans l'espace 3D. Pour déprojeter, leur technique utilise des "points critiques", soient des points de forte tangente sur la courbe. Ils précisent que leur technique n'accepte pas les "nouvelles ombres" ne pouvant pas correspondre à la courbe. De plus, la nouvelle ombre doit comporter le même nombre de points critiques que la courbe réelle pour une meilleure correspondance.

Leur technique semble être une bonne approche. Elle ne représente pas directement du *sketch* 3D, mais plutôt 2D$\frac{1}{2}$, et permet d'obtenir de bons résultats. De plus, l'utilisateur semble beaucoup plus à l'aise avec ce genre de technique. Cependant plusieurs choses peuvent être améliorées, qu'ils ont d'ailleurs en partie suggérées dans leur article. Au niveau de l'initialisation, la première courbe dessinée est censée comporter plusieurs informations 3D (profondeur, hauteur, etc.). Ici, la dimension manquante est déterminée soit grâce à l'environnement proche, soit entre le plan du "sol" ou un des plans lui étant parallèles. Le choix est défini en fonction de l'angle de la caméra. Ensuite, au niveau de la déprojection, un élément essentiel à prendre en compte est au niveau de la précision. Que se passe-t-il si la précision est totalement différente entre la courbe et son ombre ? Finalement, ils mettent aussi en avant la possibilité d'étendre leur technique à des courbes et plans quelconques, d'où notre technique.

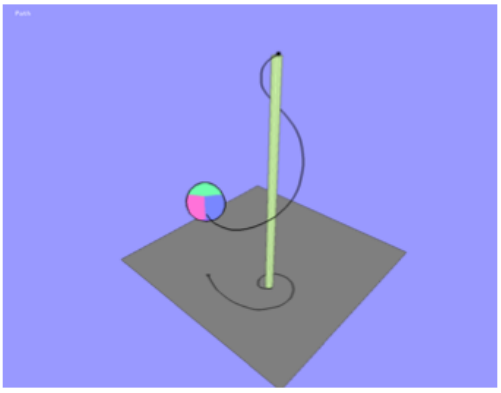


FIGURE 3.2 – Exemple de courbe obtenue par Cohen et al. [CMZ+99].

3.2 Propagation au 3D - Notre solution

Au vu des résultats des performances artistiques obtenus par Schmidt et al. [SKKS09], nous avons décidé pour notre projet d'utiliser une approche de dessin 2D$\frac{1}{2}$. Notre solution est adaptée à l'utilisateur, qui est beaucoup plus à l'aise en dessinant sur un plan 2D car ceci se rapproche mieux du dessin sur papier. Chaque trait dessiné à l'écran est projeté sur un plan de la scène 3D. En différenciant la notion d'édition et de création, on instaure les plans de dessin essentiellement voués à la création de courbes. Opposés aux boîtes englobantes, elles-aussi constituées de plans de dessin, mais utilisées pour l'édition de courbes.

3.2.1 Les plans de dessin

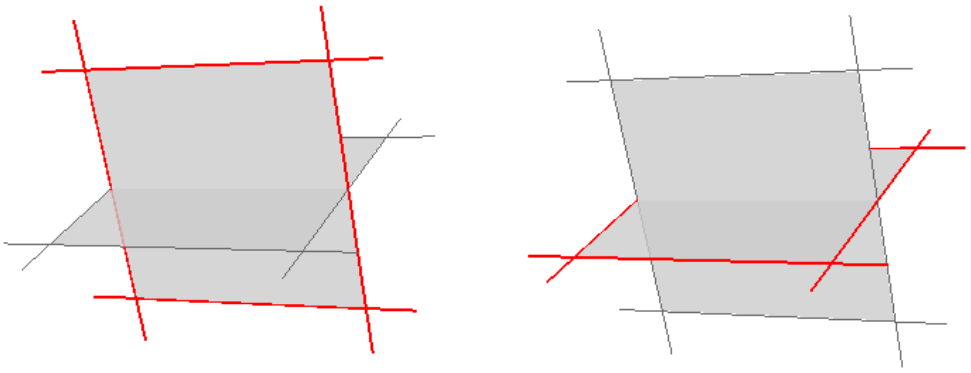


FIGURE 3.3 – Représentation des plans de dessin.

L'utilisateur dispose de deux plans de dessin perpendiculaires l'un à l'autre, et il peut passer d'un plan à un autre en appuyant simplement sur la touche "espace". Tout ce que l'utilisateur dessine à l'écran est projeté sur ce plan et représente directement la courbe dans l'espace 3D.

Remarque : Pour plus d'information sur comment projeter un point dessiné en 2D à l'écran sur un plan quelconque et par rapport à une caméra courante, voir l'appendice 2.

En laissant à l'utilisateur un contrôle complet du plan, il est capable de le positionner dans le monde 3D. Ainsi, il peut facilement placer son angle de vue, en déplacement soit le plan, soit la caméra, perpendiculaire au plan de dessin, ce qui réduit les erreurs de tracé. On peut même imaginer qu'avec un tel contrôle, si l'utilisateur est muni d'une souris 3D, il est capable de dessiner directement la courbe 3D souhaitée, sans avoir à passer par le dessin de projections comme nous l'expliquerons plus tard. Cependant, afin que les utilisateurs munis d'une souris 2D soient capables d'avoir ce "contrôle complet" sur le plan, nous avons mis en place un système de transformations, illustré à la figure 3.4.

Les flèches rouges, bleues et vertes représentent respectivement les translations en X, Y et Z du repère orthogonal. Les flèches oranges symbolisent les rotations aux axes associés. De plus, un changement d'échelle des plans de dessin a été associé à sur la roulette de la souris.

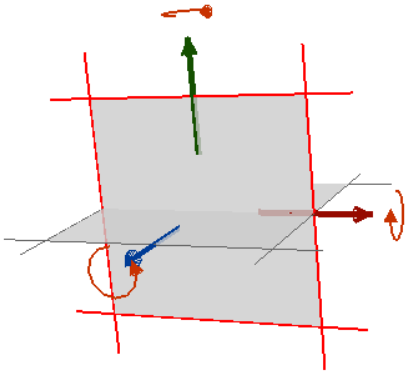


FIGURE 3.4 – Interface de transformation des plans de dessin.

Remarque : Pour plus d'information sur les transformations de points 3D, voir l'appendice 3.

3.2.2 La boîte de projection

L'idée des boîtes de projection s'inspire de la solution de Cohen et al. [CMZ+99]. On cherche ici à donner la possibilité à l'utilisateur de redessiner les projections de la courbe 3D sur des plans définis. Redessiner les projections correspond à l'idée de redessiner "l'ombre de la courbe".

Cependant, l'objectif était de donner un contrôle "semi-complet" à l'utilisateur sur cette boîte de projection. On entend par "semi-complet" que l'utilisateur a uniquement la possibilité d'effectuer des rotations sur cette boîte. Les translations ont été désactivées. La figure 3.5 illustre l'influence et les avantages que peut amener ce contrôle semi-complet sur la redéfinition des projections.

L'utilisateur peut générer la boîte de projection de deux manières différentes. La première, en appuyant sur la touche "entrée", génère la boîte selon les derniers tracés produits par l'utilisateur. La seconde lui donne la possibilité de sélectionner la partie de courbe sur laquelle il veut afficher la boîte de projection (figure 3.5). Comme pour les plans de dessin, l'utilisateur peut dessiner sur chacun des côtés de la boîte de projection en pressant la touche "espace".

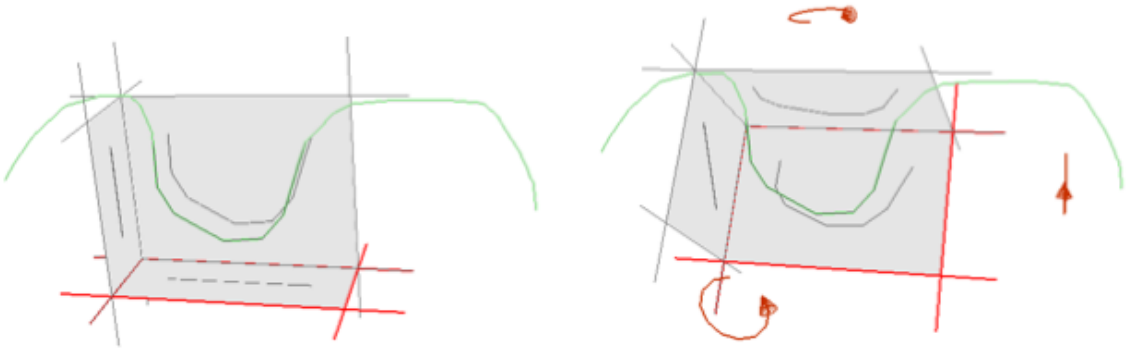


FIGURE 3.5 – Boîte de projection.

Afin de faciliter l'utilisation de ces boîtes de projection à un artiste n'étant pas forcément à l'aise avec le contrôle de scènes 3D et de caméras, nous avons ajouté plusieurs spécificités comme la sélection intelligente, l'orientation automatique ou encore la caméra adaptative.

Sélection intelligente

Lorsque l'utilisateur sélectionne la partie de courbe sur laquelle il veut afficher la boîte de projection, s'il sélectionne plusieurs morceaux de courbes, on ne gardera que le plus long morceau (morceau vert de la figure 3.5). Les autres morceaux (rouges sur la figure) ne sont pas traités.

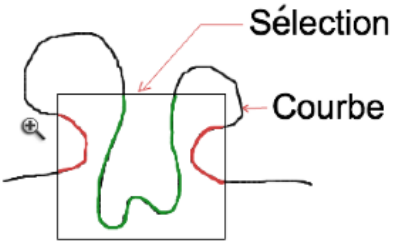


FIGURE 3.6 – Sélection de courbe.

Orientation automatique

A sa création, la boîte de projection est automatiquement orientée sur le PCA des points qu'elle contient. On rappelle que le PCA permet d'obtenir, dans un espace 3D, les trois directions principales d'un nuage de points. Par conséquent, en orientant la boîte de projection sur le PCA dès sa création, on affiche les ombres les plus représentatives de la courbe.

Caméra adaptative

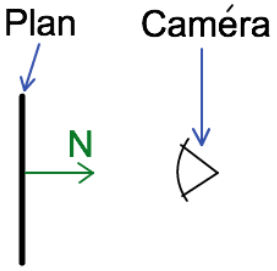


FIGURE 3.7 – Alignement de la caméra.

Le contrôle de la caméra est souvent la partie la moins intuitive lorsque l'on demande à un utilisateur de naviguer dans une scène 3D. Pour réduire ce problème nous avons ajouté des déplacements de caméra automatiques. Comme expliqué précédemment, l'utilisateur a la possibilité de choisir le plan de la boîte de projection sur lequel il veut dessiner en appuyant sur la touche "espace". Désormais, lorsque l'utilisateur change de plan, la caméra

est déplacée afin d'être alignée selon la normale au plan (figure 3.7). Cette position est, comme on l'a vu avec les tests de Schmidt et al. [SKKS09], celle minimisant les erreurs de dessin dues à la perspective.

Néanmoins, après déplacement automatique de la caméra, l'utilisateur a toujours la possibilité de la repositionner comme bon lui semble.

Un autre élément important à traiter est la correspondance entre l'ancienne projection de la courbe et la nouvelle, c'est-à-dire comment déprojeter la nouvelle courbe. Comme on a pu le voir, cette partie était assez limitée et contraignante dans la solution fournie par Cohen et al. [CMZ+99]. Nous avons cherché des méthodes plus avancées. Afin de bien comprendre notre solution, il est important d'introduire la notion de surface minimale.

3.2.3 Introduction aux surfaces minimales

Soit **S** une surface définie par un ensemble de points (voir figure 3.9). Une surface minimale est une surface minimisant son aire. Ce minimum est réalisé sous contrainte que l'ensemble des points et le bord de la surface sont déterminés d'avance. Intuitivement, une surface minimale est une surface dont l'aire ne peut qu'augmenter lorsqu'on lui applique une déformation. L'exemple typique de surface minimale est la bulle de savon, où s'appuyant sur un contour, le film de savon tend à minimiser son énergie, donc sa surface (voir illustration 3.8).

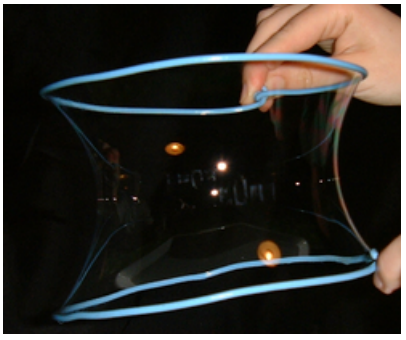


FIGURE 3.8 – Surface minimale réelle d'une bulle de savon.

Une surface minimale d'une surface quelconque minimise son aire, et donc minimise sa courbure. La courbure peut être symbolisée par l'opérateur Laplacien, soit l'opérateur différentiel défini par l'application de l'opérateur gradient (grandeur vectorielle qui indique de quelle façon une grandeur physique ou une fonction varie), suivie de l'application de l'opérateur divergence (elle mesure comment le flot d'un champ de vecteurs déforme un volume). L'expression générale de l'opérateur Laplacien est la suivante :

$$\Delta = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

Localement, et en version discrète, l'opérateur est représenté de cette façon :

$$\Delta f_{i,j} = \frac{f_{i+1,j}+f_{i-1,j}-2f_{i,j}}{dy^2} + \frac{f_{i,j+1}+f_{i,j-1}-2f_{i,j}}{dx^2}$$

où $f_{i,j}$ représente un point du maillage de la surface minimale (3.9). Les variables dx et dy représentent l'inverse du nombre de subdivisions du maillage en hauteur et en largeur (X et Y sur la figure 3.9).

Dans l'initialisation du maillage, partant de deux courbes 2D dessinées sur un même plan, on ajoute un vecteur déplacement à une des deux courbes, afin de les différencier en 3D (primordial pour minimiser la courbure d'une surface). Le maillage est alors initialisé comme ceci : un point à un pourcentage p d'une courbe est associé au point de même pourcentage sur l'autre courbe. Tous les points situés entre p et son correspondant sont interpolés linéairement. La surface **S** est appelée surface réglée.

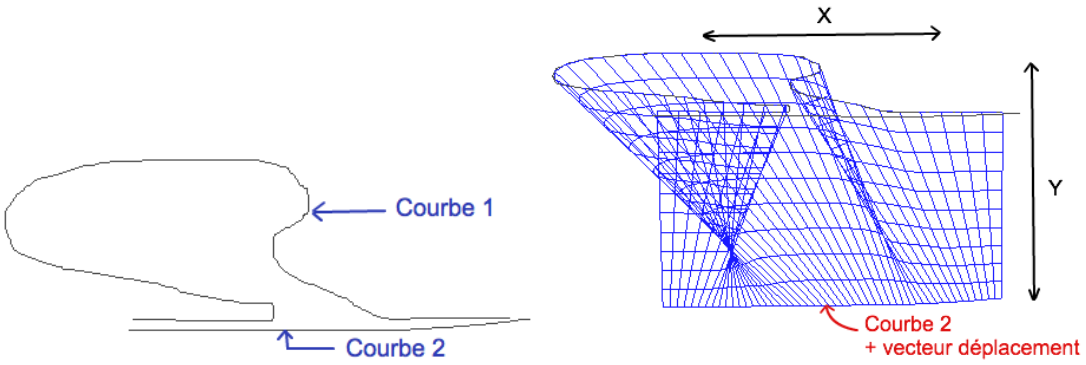


FIGURE 3.9 – Courbes de base et maillage initial.

Ensuite en minimisation de courbure, on essaie localement de rendre le Laplacien égal à zéro. Pour ce faire, on utilise un algorithme de descente de gradient. Le principe général d'une minimisation par Laplacien est de soustraire, itérativement et localement, les valeurs du Laplacien :

$$f_{i,j} = f_{i,j} - \Delta f_{i,j} dt$$

Ici, dt représente un pas de temps de la simulation qui peut fausser la minimisation s'il est fixé à une valeur trop élevée. Cependant, il est possible de trouver sa valeur optimale à chaque itération. L'algorithme ci-dessous est dérivé de ceux présentés par Shewchuk [She94]. Soit **X** l'ensemble des points du maillage :

Algorithm 7 Descente de Laplacien

$i = 0$
while $i < i_{max}$ et $Erreur > \epsilon$ **do**
  $\mathbf{R} = \Delta_{\mathbf{X}}$
  $\mathbf{Q} = \Delta_{\mathbf{R}}$
  $\delta = \mathbf{R^T R}$
  $dt = \frac{\delta}{\mathbf{R^T Q}}$
  $i = i + 1$
end while

Remarque 1 : *Erreur* est la somme des normes de tous les Laplaciens locaux divisée par le nombre de Laplaciens.

Remarque 2 : Dans le cas d'une minimisation pour surface minimale générale, les Laplaciens aux bords "haut" et "bas" (i.e. au niveau des courbes) sont fixés à zéro.

La figure 3.10 montre le résultat qu'il est possible d'obtenir après plusieurs itérations.

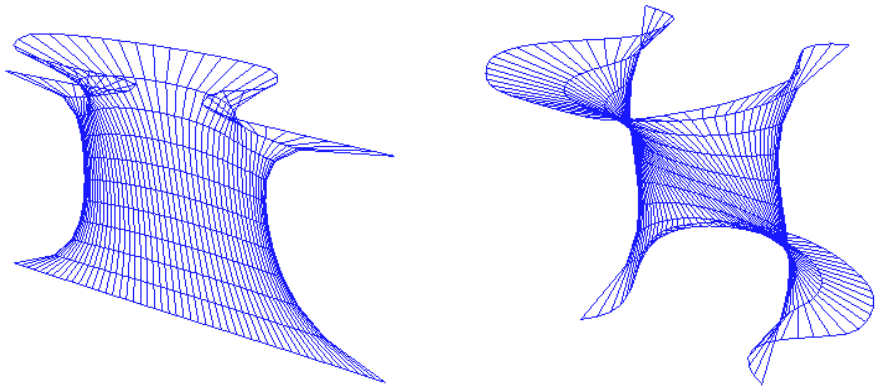


FIGURE 3.10 – Deux surfaces minimales par descente de Laplacien.

3.2.4 Déprojection avancée

Une association par pourcentage n'est souvent pas assez précise. Sur la figure 3.11, on illustre le résultat que l'on aimerait atteindre lorsque l'on met en correspondance deux courbes différentes. Le tout est utilisé pour obtenir le résultat présenté en figure 3.13.

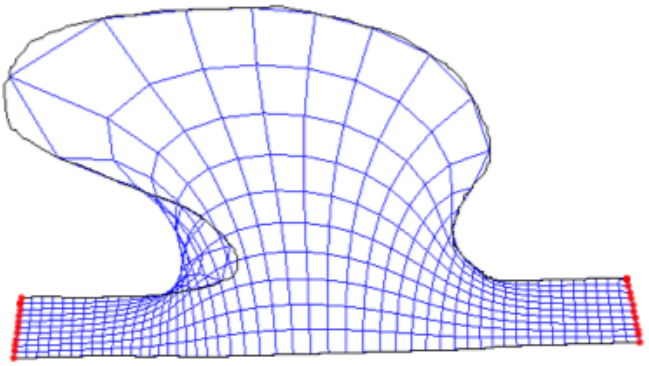


FIGURE 3.11 – Résultat attendu.

Pour ce faire, le calcul de la surface minimale est à la base de notre solution. Néanmoins, plusieurs modifications ont été apportées, comme pour les contraintes aux bords. Afin d'obtenir une mise en correspondance, comme celle illustrée ci-dessus, il est primordial que les points aux bords "haut" et "bas" (c'est-à-dire les points sur les courbes) aient la possibilité de se déplacer sur leur courbe associée.

Mobilité des points aux courbes

La première modification a été au niveau des bords "gauche" et "droit" (points en rouge sur la figure 3.11). Désormais, il sont fixes, contrairement aux points aux bords "haut" et "bas", c'est-à-dire au niveau des courbes qui eux (exceptés premiers et derniers points) sont mobiles.

Le calcul du Laplacien aussi a été modifié, spécialement au niveau des points “haut” et “bas”. Maintenant, le calcul de la dérivée partielle en x est :

$$B = \frac{t_i(l_{i,j+1} - l_{i,j-1})}{dx^2}$$

où t_i est la tangente au point x_i. $l_{i,j+1}$ représente la distance sur la courbe entre les points x_i et x_{i+1}, et $l_{i,j-1}$ entre les points x_i et x_{i-1}. Le calcul du Laplacien sur les bords “haut” et “bas” devient :

$$\Delta f_{i,j} = \frac{f_{i+1,j} + f_{i-1,j} - 2f_{i,j}}{dy^2} + B$$

Cependant, afin d'assurer le déplacement du point **sur** la courbe on va modifier l'étape “$f_{i,j} = f_{i,j} - \Delta f_{i,j} dt$” au niveau des bords. Pour bien faire, on cherche à trouver le pourcentage de déplacement du point sur la courbe, et correspondant au Laplacien calculé. Pour cela, on utilise le produit scalaire,

$$\alpha = \Delta f_{i,j} \cdot \mathbf{d}$$

ù $\mathbf{d}$ représente toujours la tangente à la courbe au point x_i. Le nouveau point x_i est donc le point se situant au pourcentage de $x_i + \alpha$ sur la courbe.

Maximiser l'orthogonalité

La dernière modification maximise l'orthogonalité du maillage. Pour ce faire on considère que la dérivée partielle en y doit être plus importante que la dérivée partielle en x :

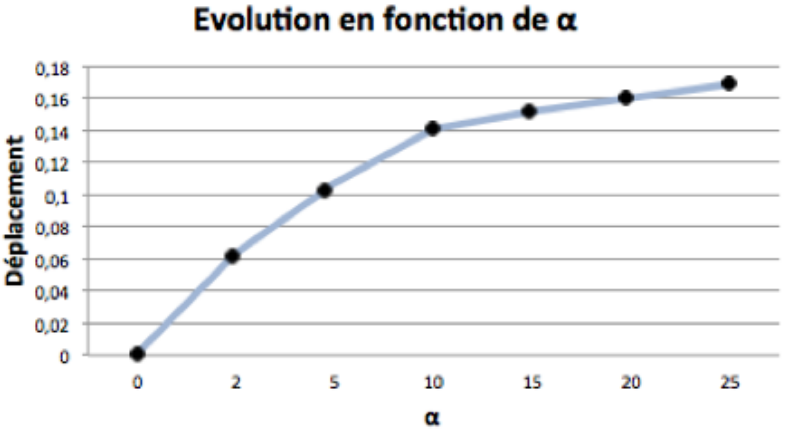
$$\frac{\partial^2}{\partial y^2} \gg \frac{\partial^2}{\partial x^2}$$

donc :

$$\Delta = \frac{\partial^2}{\partial x^2} + \alpha \frac{\partial^2}{\partial y^2}$$

avec $\alpha > 1.0$.

Néanmoins, il est primordial d'étudier l'évolution de la surface avec plusieurs valeurs de α fixées pour voir s'il est possible de déterminer empiriquement une valeur de convergence.



Afin d'obtenir ce graphique, nous avons pris pour initialisation un premier point à 50% sur la première courbe et son correspondant aussi à 50% sur la seconde. La courbe résultat montre de quel pourcentage le point a évolué sur la seconde courbe afin de respecter au mieux l'orthogonalité. On voit qu'au bout d'un certain moment, et ça pour la majorité des courbes, augmenter la valeur de α n'a plus énormément d'importance sur le résultat. Après plusieurs séries de tests, on a pu en déduire qu'au-dessus de $\alpha = 20$, l'amélioration des résultats pouvait être négligeable. De plus, dans le cas de α trop grand, ceci peut fausser la minimisation et faire augmenter, voire exploser l'erreur. On rappelle que l'erreur correspond à la somme des normes des Laplaciens, on cherche à la minimiser, donc que ces Laplaciens soient de norme nulle.

La figure 3.12 montre les résultats visuels de l'évolution de la surface pour différentes valeurs de α :

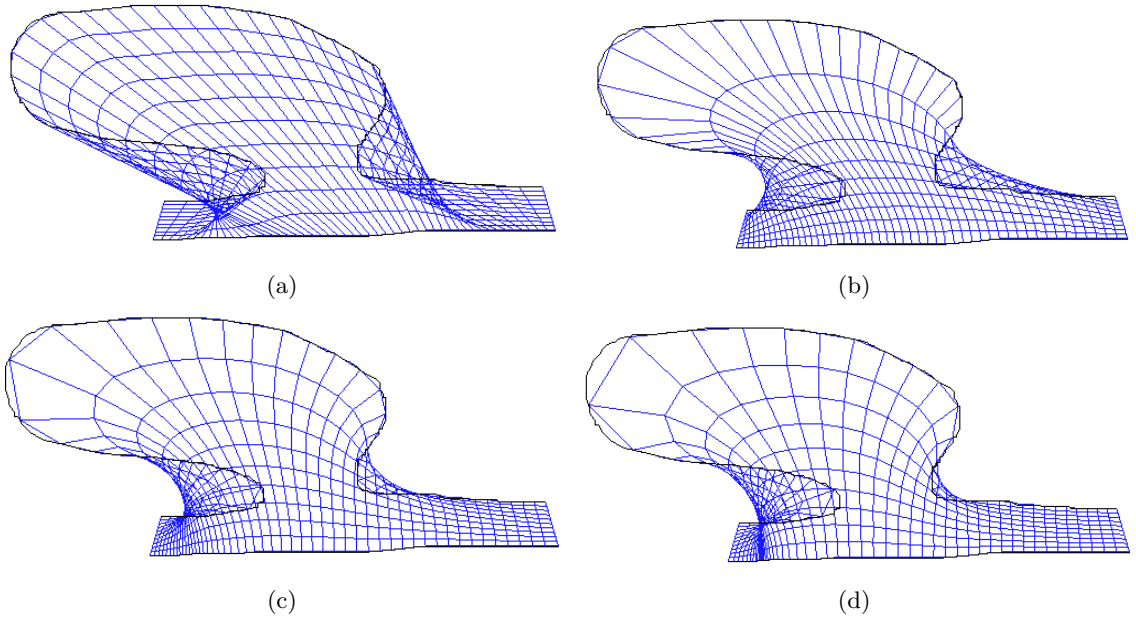


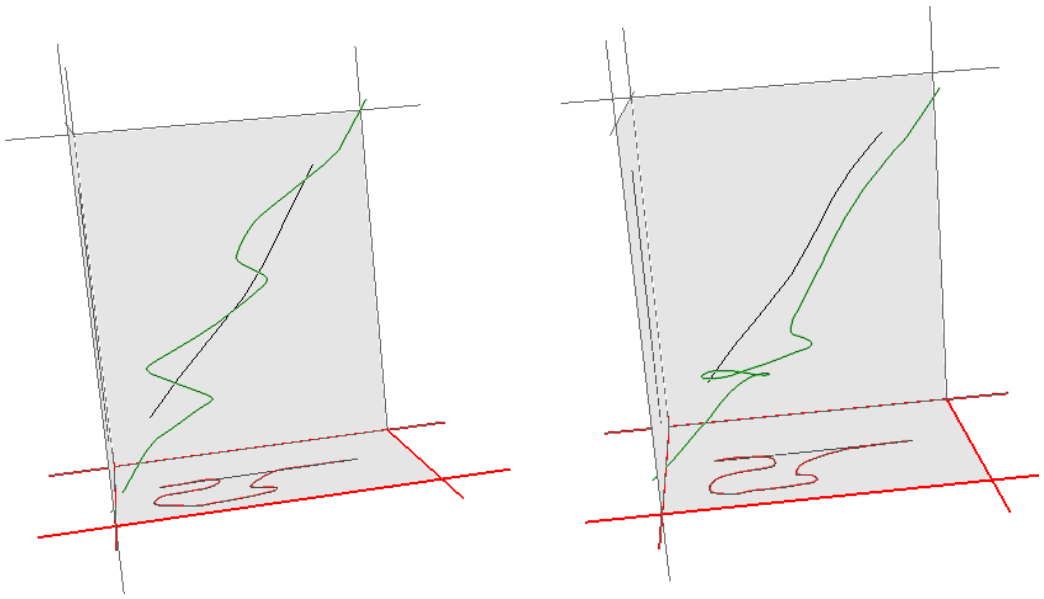
FIGURE 3.12 – (a) Surface non minimisée, initialisée avec les pourcentages. (b) Surface minimisée avec $\alpha = 2$. (c) Surface minimisée avec $\alpha = 10$. (d) Surface minimisée avec $\alpha = 20$.

On remarque qu'avec $\alpha = 2$, les résultats sont déjà plus satisfaisants qu'à l'initialisation. Cependant, l'orthogonalité n'est pas encore tout à fait respectée.

A la figure 3.12 (d), on peut voir le résultat final pour $\alpha = 20$. L'orthogonalité est respectée et la correspondance entre les deux courbes est beaucoup plus représentative. A la Figure 3.13, on peut voir le résultat appliqué. Dans les deux cas, on démarre d'un trait diagonal, où l'on redessine sa projection au sol, le remplaçant par quelque chose d'un peu plus complexe. On essaie de garder au maximum l'aspect diagonal de la courbe. A gauche la correspondance non-optimisée, à droite avec une optimisation à $\alpha = 20$.

Cependant, l'utilisateur ne désirera peut-être pas toujours utiliser la version optimisée. D'une part car la déprojection prend un peu plus de temps et d'autre part parce qu'elle n'arrive pas toujours au résultat voulu. Dans le cas d'un dessin de spirale par exemple, on utilisera une déprojection simple. Grossièrement, on dessinera un premier trait vertical, puis un enchaînement de cercles au sol, correspondant à son "ombre". On rappelle que

l'algorithme de calcul de courbes principales ne permet pas de gérer les courbes repassant plusieurs fois sur elles-mêmes, mais que l'utilisateur a la possibilité de démarrer de nouveaux tracés, et de les relier.



(a) (b)

FIGURE 3.13 – (a) Non-optimisé. (b) Optimisé avec $\alpha = 20$.

3.2.5 Résultat

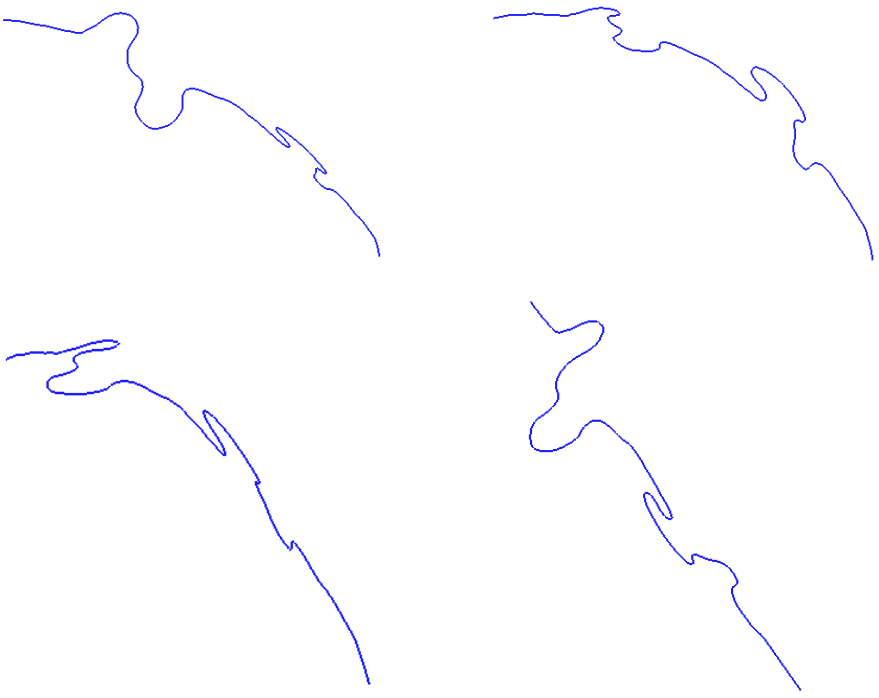


FIGURE 3.14 – Courbe 3D générée avec notre solution et vue sous plusieurs angles.

Applications

Afin de mieux adapter et améliorer notre système nous l'avons testé avec différentes personnes ayant des capacités artistiques de tous niveaux. Les tâches qui leur ont été demandées se sont divisées en deux parties. Premièrement l'aspect 2D où l'utilisateur doit esquisser des silhouettes/contours avec support à l'appui, ce qui nous permet d'avoir des retours sur la technique générale de courbes principales par tracés multiples. Ensuite pour l'aspect 3D, on demande à l'utilisateur de produire un chemin passant par des anneaux 3D générés aléatoirement.

4.1 Esquisse de contours 2D

4.1.1 Tâche à effectuer

Ici, le but est d'avoir des retours sur la technique générale de courbe principale. On entend par là l'avis de l'utilisateur quant à la stabilité du système, sa qualité de représentation du tracé, etc.

L'esquisse de contours 2D est divisée en deux étapes. Premièrement, l'utilisateur doit, à l'aide d'un support, esquisser une silhouette ou un contour. Ensuite, on lui demande de ré-effectuer l'opération mais cette fois-ci sans support. Afin de bien pouvoir comparer les performances de chaque utilisateur ainsi que la ressemblance entre la version sans support de celle avec support, il est nécessaire que la forme demandée ne soit pas compliquée. Pour les tests suivants, nous utilisons pour support un cercle que l'utilisateur doit calquer, puis tenter d'approcher.

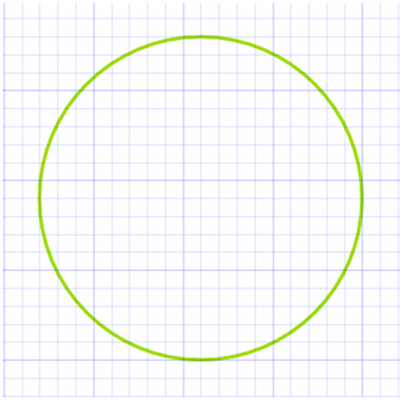
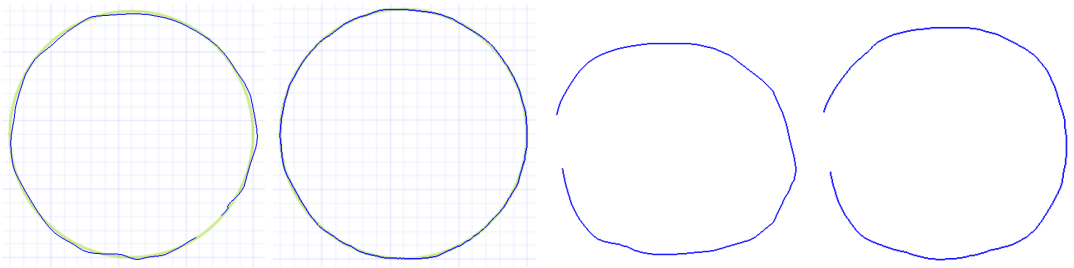


FIGURE 4.1 – Support utilisé pour la série de tests d'esquisse de contours 2D.

4.1.2 Performances



(a) (b) (c) (d)

FIGURE 4.2 – (a) Avec support 30sec. (b) Avec support 3min. (c) Sans support 30sec. (d) Sans support 3min.

	Temps	Nb Tracés	Nb Tracés utilisés	Nb Points	Nb Points utilisés
Avec support	30sec	22	20	632	580
Avec support	3min	68	35	3107	1621
Sans support	30sec	20	19	525	494
Sans support	3min	57	33	2602	1273

4.1.3 Retour de l'utilisateur

Dans la version actuelle du système, la courbe principale est recalculée, à chaque nouveau tracé, sur l'ensemble des tracés produits par l'utilisateur. Le processus de calcul de courbe principale n'étant plus toujours instantané, l'utilisateur est arrêté dans son esquisse et on perd cette fluidité de tracé recherchée. De plus, le calcul de courbe principale étant sur un nuage de points, si on ré-effectue ce parcours sur l'ensemble des tracés, le parcours ne sera probablement pas complètement identique au parcours précédent, et ceci amène quelques instabilités dans le système.

Néanmoins, l'utilisateur est satisfait de la courbe principale obtenue et pense que celle-ci illustre très bien ce qu'il voulait esquisser.

4.2 Esquisse de chemins 3D

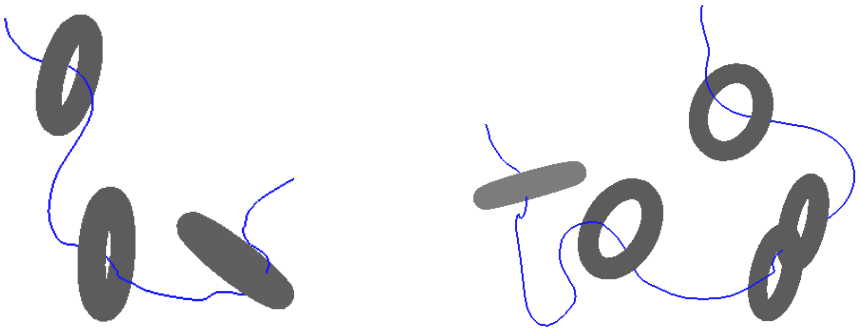
4.2.1 Tâche à effectuer

L'esquisse de chemins 3D tester, en priorité, la bonne adaptabilité de notre techniques de courbe principale au 3D, mais elle doit aussi fournir des retours sur notre solution "d'esquisse 3D", surtout au niveau des changements automatiques de camera, de l'utilisation de la 2D$\frac{1}{2}$ au lieu d'une 3D à proprement dit, etc.

Pour ce faire, on génère une série d'anneaux 3D dans une scène 3D restreinte sur sa taille (afin de ne pas générer les anneaux sur un espace 3D trop large). Le but de l'utilisateur est d'esquisser un chemin 3D passant par l'ensemble de ces anneaux. On ne contraint pas l'utilisateur, pour ces tests, sur l'ordre dans lequel il doit parcourir les anneaux.

FIGURE 4.3 – Anneaux définissant le chemin à tracer.

4.2.2 Performances



(a) (b)

FIGURE 4.4 – (a) 3 anneaux. (b) 5 anneaux.

Temps	Nb Tracés utilisés	Nb Points utilisés	Nombre d'anneau
1.44min	29	1209	3
4.07min	68	4612	5

4.2.3 Retour de l'utilisateur

L'utilisateur reste peu à l'aise avec les contrôles de caméra. C'est pourquoi l'adaptation automatique de caméra est fortement appréciée. De plus, l'idée de toujours esquisser sur un plan semble être, pour lui, ce qu'il y a de plus logique et de plus intuitif.

L'utilisateur suggère qu'il serait bon d'ajouter un outil d'effacement, il voudrait aussi pouvoir ré-éditer des parties de courbes plus grandes et plus complexes.

Le passage du 2D au 3D n'a aucune influence sur l'algorithme de calcul de courbe principale.

Conclusion

De nos jours, l'intuitivité et la facilité deviennent deux mots d'ordre dans la conception d'interfaces utilisateur. Néanmoins comme on a pu le présenter, les techniques de "sketching" utilisées se rapprochaient rarement des méthodes d'esquisse artistique. C'est pourquoi nous avons proposé une méthode de "sketching" s'approchant des techniques de dessins au crayon, en espérant qu'elle corresponde mieux aux techniques classiques, et ainsi qu'elle soit plus naturelle d'utilisation. Nous avons développé une représentation numérique des traits d'esquisse et l'avons adapté à la fois aux contraintes de traits multiples, et à la notion temporelle qui s'apparente avec une notion de traits en gras ou à peine effleurés. Nous avons aussi donner une version efficace pour que le traitement se fasse en temps réel. Finalement, nous avons étendu cette notion d'esquisse au 3D en développant une méthode de projection sur des plans et à l'adaptation des positions de la caméra par rapport à ces plans.

Même si notre méthode répond à plusieurs de nos contraintes initiales, il s'agit d'un problème très complexe qui demande une étude en profondeur des diverses options qui s'offrent à nous. Dans ce sens, nos deux études expérimentales sur des sujets nous ont fait ressortir les points positifs de notre approche, et aussi des problèmes encore à régler et des pistes de solutions. Comme on a pu le signaler, notre solution n'était pas toujours assez stable (il arrivait de perdre la courbe principale car le parcours n'est pas exactement le même à chaque fois que l'on ajoute un nouveau trait). C'est pourquoi nous essayons actuellement de corriger ce problème. Par la suite, on pourrait même essayer d'imaginer ajouter la gestion de niveaux de détail. L'utilisateur pourrait retracer des grands traits sur des petits raffinements sans pour autant les supprimer car il dessine sur un niveau de détail supérieur.

Encore plusieurs changements restent à être effectuer avant d'arriver à une version stable de notre solution. Cependant, elle démontre qu'il reste encore à faire du côté du "sketching" et qu'il est toujours aussi difficile d'imaginer une solution parfaite du "sketching" 3D. En cas de bon fonctionnement, notre système serait voué à être utilisé par l'industrie cinématographique. Bien que celle-ci pourrait bien évidemment être adaptée dans bien d'autres situations.

Appendices

4.3 Projections

4.3.1 Projection orthogonale

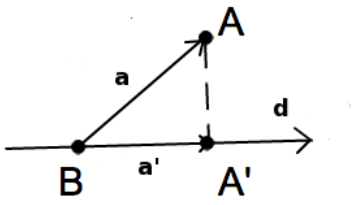


FIGURE 4.5 – Schéma de projection orthogonale.

La projection orthogonale d'un point A sur une droite de direction $\mathbf{d}$ et passant par B correspond à trouver un vecteur reliant A à la droite, tout en étant perpendiculaire à celle-ci. Pour ceci, quatre étapes sont nécessaires.

$$\begin{aligned} \mathbf{ab} &= A - B \\ \alpha &= \mathbf{d} \cdot \mathbf{ab} \\ \mathbf{a}' &= \mathbf{d} * \alpha \\ A' &= B + \mathbf{a}' \end{aligned}$$

Tout d'abord, on crée le vecteur $\mathbf{a}$. Ensuite, on calcule α, le produit scalaire entre $\mathbf{a}$ et $\mathbf{d}$ ($\mathbf{d}$ toujours normalisé). Puis, on multiplie $\mathbf{d}$ par ce coefficient, afin de projeter le vecteur $\mathbf{ab}$ sur $\mathbf{d}$. Et finalement, on déplace B le long de ce vecteur.

4.3.2 Déprojection planaire

Cet appendice trouve la projection d'un point dessiné, en 2D à l'écran, sur un plan quelconque d'une scène 3D. On considère que le point 2D est dessiné dans une fenêtre OpenGL et qu'il est projeté dans la scène 3D affichée dans cette fenêtre et regardé par une "camera", comme celle représentée sur le schéma 4.2.

En considérant le schéma 4.2, le point que l'on cherche est A'. Le point A correspond à la projection du point 2D sur un plan parallèle à la camera (i.e. perpendiculaire au vecteur vue) et à distance *depth* fixée. P correspond à la position du plan dans l'espace, C à la position de la camera et le vecteur $\mathbf{n}$, correspond à la normal au plan sur lequel on projette

le point dessiné à l'écran.

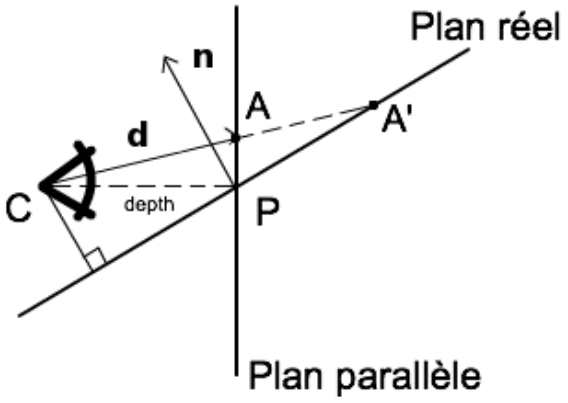


FIGURE 4.6 – Schéma de déprojection planaire.

La première étape pour déterminer A' et de calculer le vecteur $\mathbf{d}$. Pour cela, on cherche à trouver le point A.

En considérant un point 2D p dont les coordonnées sont normalisées . Lorsque p se trouve dans le coin en haut à gauche de la fenêtre, ses coordonnées sont $(-1.0, 1.0)$, au centre de la fenêtre $(0.0, 0.0)$ et dans le coin en bas à droite $(1.0, -1.0)$. Puis une caméra caractérisée comme sur le schéma 4.3.

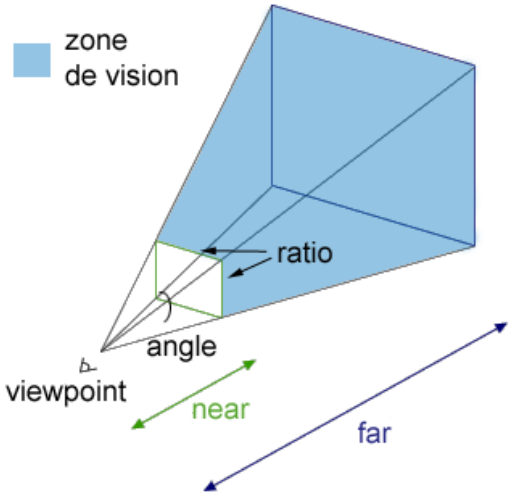


FIGURE 4.7 – Schéma de caméra.

Soient px et py les coordonnées x et y du point 2D normalisé. Le point A se trouve de cette façon :

$$px = \frac{px}{1.0/tan(angle)}$$
$$py = \frac{py}{1.0/(tan(angle)*ratio)}$$
$$A = (axisX * px - axisY * py - axisZ) * depth + C$$

où $axisX$, $axisY$ et $axisZ$ sont les vecteurs d'orientation de la caméra, correspondant aux

trois colonnes de sa matrice de rotation. Pour plus de détails sur les matrices de rotation, voir l'appendice sur les matrices de transformation.
Avec A, il est facile d'avoir le vecteur $\mathbf{d}$, et A' se calcule de telle manière :

$$A' = C + \mathbf{d}\frac{\mathbf{n}\cdot(P-C)}{\mathbf{d}\cdot\mathbf{n}}$$

4.4 Matrice de Transformation

Les matrices de transformation sont utilisées pour déplacer un point dans l'espace. Elles ont été redéfinies dans notre solution afin d'avoir un meilleur contrôle sur les objets. La forme générale des matrices de transformation est la suivante :

$$\mathbf{T} = \mathbf{R}.\mathbf{S}.\mathbf{t}$$

Où $\mathbf{R}$ est une matrice de rotation, $\mathbf{S}$ une matrice de mise à l'échelle et $\mathbf{t}$ un vecteur translation. Pour toutes les explications suivantes, $\mathbf{p}$ symbolisera un point 3D.

L'application d'une translation (définie par un vecteur cas si $\mathbf{p}$ est un point 3D) à $\mathbf{p}$ a pour résultat :

$$\mathbf{p} = \mathbf{p} + \mathbf{t}$$

L'application d'une matrice de mise à l'échelle (définie par une matrice 3×3 cas si X est un point 3D) à X a pour résultat :

$$\mathbf{p} = \mathbf{S}\mathbf{p} = \begin{pmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & sz \end{pmatrix} \mathbf{p}$$

Où sx, sy et sz représentent, respectivement, les coefficients de mise à l'échelle sur l'axe X, Y et Z.

En ce qui concerne les matrices de rotation, il en existe différents types. Tout d'abords les matrices de rotation d'angle θ et d'axe X, Y et Z (de gauche à droite ci-dessous).

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & \cos\theta & -\sin\theta \\ 0.0 & \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\theta & 0.0 & -\sin\theta \\ 0.0 & 1.0 & 0.0 \\ \sin\theta & 0.0 & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0.0 \\ \sin\theta & \cos\theta & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

Puis les matrices plus générique prenant un axe $v = (vx, vy, vz)$ et un angle θ. On considère $\mathbf{Rc}$ et $\mathbf{Rs}$ tel que :

$Rc = v * (1.0 - \cos\theta)$
$Rs = v * \sin\theta$
$\mathbf{Rc}$ et $\mathbf{Rs}$ de la forme $\mathbf{Rc} = (rcx, rcy, rcz)$ et $\mathbf{Rs} = (rsx, rsy, rsz)$, on obtient la matrice de rotation R avec :

$$\begin{pmatrix} \cos\theta + vx * rcx & ax * rcy - rsz & ax * rcz + rsy \\ ax * rcy + rsz & \cos\theta + vy * rcy & vy * rcz - rsx \\ vx * rcz - rsy & vy * rcz + rsx & \cos\theta + vz * rcz \end{pmatrix}$$

Et le nouveau $\mathbf{p}$ est obtenu avec :

$$\mathbf{p} = \mathbf{R}\mathbf{p}$$

Bibliographie

[BBS08] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. Ilovesketch : as-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '08, pages 151–160, New York, NY, USA, 2008. ACM.

[CMZ+99] Jonathan M. Cohen, Lee Markosian, Robert C. Zeleznik, John F. Hughes, and Ronen Barzel. An interface for sketching 3d curves. In *SI3D '99 : Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 17–21, New York, NY, USA, 1999. ACM Press.

[ETE05] Jochen Einbeck, Gerhard Tutz, and Ludger Evers. Local principal curves. *Statistics and Computing*, pages 301–313, 2005.

[Gar09] E Garcia. PCA & SPCA tutorial. *Computing*, pages 1–6, 2009.

[HS89] T. Hastie and W. Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84 :502–516, 1989.

[JS10] Joaquim Jorge and Faramarz Samavati. *Sketch-based Interfaces and Modeling.* Springer, 2010.

[KLZ00] Balázs Kégl, Tamás Linder, and Kenneth Zeger. Learning and design of principal curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22 :281–297, 2000.

[MSS+99] Sebastian Mika, Bernhard Schölkopf, Alex Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel PCA and de-noising in feature spaces. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 11*, pages 536–542. MIT Press, 1999.

[PSNW07] Richard Pusch, Faramarz F. Samavati, Ahmad H. Nasri, and Brian Wyvill. Improving the sketch-based interface. *The Visual Computer*, 23 :955–962, Sep 2007.

[She94] Jonathan R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, Pittsburgh, PA, USA, 1994.

[SKKS09] Ryan Schmidt, Azam Khan, Gord Kurtenbach, and Karan Singh. On expert performance in 3d curve-drawing tasks. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, pages 133–140, New York, NY, USA, 2009. ACM.

[Whi06] Tony White. *ANIMATION - From Pencils To Pixels.* Focal Press, 2006.

[Wik11] Wikipédia. Principal Component Analysis, Consulté en 2011.