Détection de symétries

Lina Mezghani & Sarah Perrin

January 18, 2018

Introduction

Le projet que nous avons choisi est basé sur l'article «Symmetry in 3D Geometry : Extraction and Applications» (Mitra, Pauly, Wand, Ceylan, 2012). Cet article introduit les concepts basiques nécessaires pour étudier les symétries présentes dans des objets 3D et présente plusieurs méthodes pour les trouver. Nous avons choisi d'implémenter l'une d'entre elles, décrite plus précisément dans l'article «Partial and Approximate Symmetry Detection for 3D Geometry» (Mitra, Guibas, Pauly, 2006). Cet article expose comment trouver des symétries - définies comme combinaisons des opérations élémentaires translation, rotation, réflection et changement d'échelle - en trouvant la transformation associée à deux points de surfaces de la forme, en la représentant dans un autre espace, en identifiant les clusters formés et en extrayant un représentant de chaque cluster. Nous avons donc cherché à reproduire les algorithmes décrits, puis à les tester sur des nuages de points.

1 Description de la méthode implémentée

Pour détecter des symétries, l'article décrit une méthode qui consiste en plusieurs étapes distinctes. Ces étapes sont : sampling, analysis, pairing, clustering et patching sont résumées dans l'image (Figure 1) tirée de l'article ci-dessous.

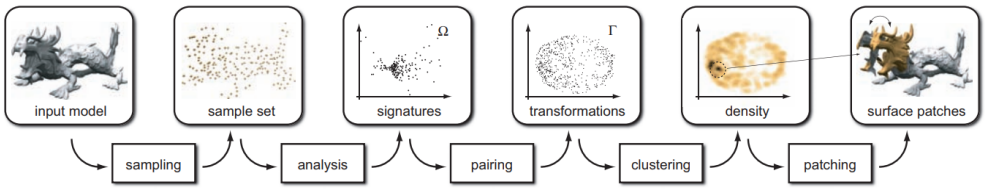

Figure 1: Schéma des différentes étapes

1.1 Sampling

Cette première étape consiste à sélectionner les points de surfaces de la forme 3D qui serviront ensuite pour calculer les symétries.

1.2 Analysis

L'article présente d'abord une méthode simple sur un exemple en 2D. Pour toutes les paires des points de surface, on calcule la droite qui correspond au plan de réflexion, c'est-à-dire la droite médiatrice entre les deux points. Chaque droite peut être décrite par une distance d et un angle phi, par exemple en prenant les coordonnées polaires du point de la droite le plus proche de l'origine et en recentrant le centre de masse à l'origine du repère si nécessaire.

Cependant, cette technique ne permet de détecter que des symétries réflexives, et pas par exemple de translation ou de rotation. Une deuxième technique est donc introduite, cette fois-ci basée sur les tenseurs de courbure (curvature tensor). L'idée est de calculer ces tenseurs, qui donnent alors les deux directions et normes principales de courbure pour chaque point. Puis, pour

une paire, on peut calculer quelle rotation est associée à chaque point en cherchant à aligner ces tenseurs. Le changement d'échelle peut aussi être obtenu en regardant la variation de normes des tenseurs. Enfin, la translation peut-être calculée à partir de la rotation, du changement d'échelle et des points d'origine. On obtient donc des points dans un espace en dimension 7 (car trois dimensions pour la rotation et la translation et une pour le changement d'échelle).

1.3 Pairing

Cette étape permet d'éliminer certains points du nouvel espace avant d'appliquer le clustering, ce qui permet de réduire le temps de calcul et de trouver des clusters plus pertinents. Elle se base sur des considérations relatives aux tenseurs de courbure et nous ne l'avons donc pas implémentée dans notre projet.

1.4 Clustering

Le clustering est l'étape clé permettant de trouver les symétries. L'idée est que si une symétrie est présente, alors beaucoup de couples de points voteront pour cette symétrie, c'est-à-dire qu'ils auront des coordonnées proches dans l'espace de transformation. Si l'on repère ces groupes de points, il est alors possible de remonter aux symétries présentes dans l'objet.

Pour cela, on utilise l'algorithme `MeanShift`, qui a l'avantage de ne pas avoir de nombre de cluster prérequis contrairement aux k-moyennes. On traite l'ensemble des points comme provenant d'une fonction de densité de probabilité : les régions de l'espace où la densité est forte correspondent aux maximaux locaux de la distribution sous-jacente. `MeanShift` consiste à effectuer des estimations locales du gradient de la densité aux points de données, puis à bouger ces points le long du gradient estimé de manière itérative, jusqu'à ce qu'il y ait convergence : les points stationnaires de ce procédé correspondent aux maximaux locaux de la distribution, et donc dans notre cas aux principales symétries.

1.5 Patching

Cette dernière étape consiste d'une part à affiner les symétries obtenues par `MeanShift`, mais également à rendre les résultats visuellement intéressants, par exemple en traçant les plans obtenus ou en colorant les points participant à une symétrie d'une certaine couleur.

Pour affiner les résultats, les auteurs proposent pour chaque cluster de partir du mode T_k (maxima local du cluster), de sélectionner un point au hasard dans l'espace de transformation du cluster - qui correspond à une paire de points (p_i, p_j) de la surface - de regarder si ses voisins vérifient la même symétrie T_k, et si c'est le cas de les ajouter au patch. De plus, la transformation $T_k$est corrigée au cours du processus grâce à l'algorithme Iterative Closest Point. En effet, à chaque étape, on peut trouver quelle transformation permet de superposer le mieux possible les deux surfaces symétriques grâce à ICP et donc corriger T_k.

2 Description de l'implémentation

Nous avons globalement réutilisé des structures de classes similaires à celles vues en TD, comme `PointSet`, `Point_3`, `Translation_3`, et nous avons représenté les nuages de points grâce à Processing.

2.1 Sampling

Nous n'avons pas eu à implémenter cette étape car lors de nos tests nous avons pris en entrée des nuages de points où seuls les points de surface étaient présents. Lorsque les nuages de points étaient trop volumineux (nombre de points supérieur à 1000), nous pouvions définir la proportion de points que nous souhaitions garder.

2.2 Analysis

Cette partie nous a demandé un travail important pour comprendre les sous-entendus de l'article et les détails qui n'étaient pas explicites. Nous avons choisi d'implémenter la première méthode (réflexions) plus simple et mieux décrite, d'abord en 2D puis en 3D.

Pour cela, en 2D, pour chaque paire de point, nous avons implémenté une fonction qui permet de calculer le plan de réflexion entre ces points et qui renvoie le point du plan le plus proche de l'origine en coordonnées polaires (FIGURE 2). Nous avons choisi d'utiliser les coordonnées polaires et non cartésiennes car celles-ci permettent de donner une information sur l'inclinaison du plan qui est essentielle. Nos nouveaux points sont donc dans un nouvel espace de transformation en dimension 2 (r, θ). Nous avons également fait attention à replacer le centre de masse du `Point Cloud` à l'origine par une translation.

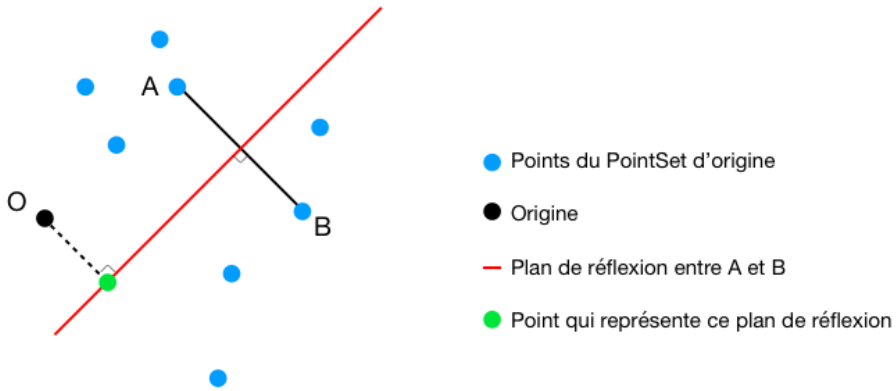

Figure 2: Construction du point caractéristique du plan de réflexion en 2D

En dimension 3, nous avons fait de même en coordonnées sphériques. Nous avons de plus fait attention à ce que les points dans le nouvel espace de transformation (r, theta, phi) soient bien proches si jamais les symétries sont proches. Par exemple, pour theta qui varie entre 0 et 2π, il fallait que deux points qui auraient par exemple un angle theta proche de 0 et l'autre proche de 2π soient proches également. Pour cela, nous avons défini une nouvelle distance `myDistance` qui tient compte de cette caractéristique.

Nous avons néanmoins cherché à comprendre en profondeur l'autre méthode décrite, et réfléchi à ce que serait une possible implémentation. Nous avons par exemple lu dans l'article «Normal Based Estimation of the Curvature Tensor for Triangular Meshes» (Holger Theisel, Christian Rossl, Rhaleb Zayer, Hans-Peter Seidel) qu'il était possible de calculer κ_1 et κ_2, les valeurs propres de T matrice du tenseur, à partir du vecteur normal et d'un triangle mesh, notions qui nous étaient davantage familières. Cependant, nous n'avons pas essayé d'implémenter la méthode décrite et nous avons préféré nous concentrer sur la méthode précédente en 2D et 3D, sur le clustering et le rendu final.

2.3 Clustering

Pour implémenter l'algorithme de `MeanShift`, nous sommes parties d'un squelette que nous avons trouvé par hasard sur internet, qui est le code écrit pour un TD du cours INF562. Ce code avait l'avantage pour nous d'utiliser à peu près les mêmes classes et bibliothèques que le cours d'INF555, ce qui nous a facilité l'implémentation. Cependant, nous avons effectué plusieurs modifications pour que ce code fonctionne dans notre cas. Par exemple, nous avons ajouté une fonction pour convertir les `PointCloud` en `PointSet` et redéfinit la notion de distance pour qu'elle corresponde à `myDistance`.

2.4 Patching

Afin de rendre les résultats plus visuels, nous avons implémenter des fonctions afin de tracer les plans de symétries ainsi que des fonctions pour colorer les points participant à des symétries de différentes couleurs.

3 Résultats

3.1 Complexité de l'algorithme

Pour un nombre de points n du Point Cloud, il y a $\frac{n(n-1)}{2}$ paires différentes. L'étape la plus longue est celle du `MeanShift`. Dans la fonction `computeClusters()`, `detectCluster` est appelé $O(n^2)$ fois, et a une complexité $O(n)$ (recherche slowRange). La complexité totale est donc $O(n^3)$.

En pratique, on obtient les temps suivants lorsqu'on fait tourner l'algorithme

3.2 Discussion du paramètre bandWidth

`bandWidth` est le seul paramètre de `MeanShift` et permet de régler les différents rayons caractéristiques :

- $rayondeconvergence^2 = bandWidth \times 10^-3$
- $rayondelafenetre^2 = bandWidth$
- $rayond'influence^2 = bandWidth/4$
- $distancedefusion^2 = bandWidth$

Expérimentalement, nous avons obtenu des résultats très différents suivant les valeurs de bandWidth. Pour des valeurs élevées, nous ne détectons qu'un seul cluster. Souvent, ce cluster révèle la symétrie principale du `Point Cloud`.

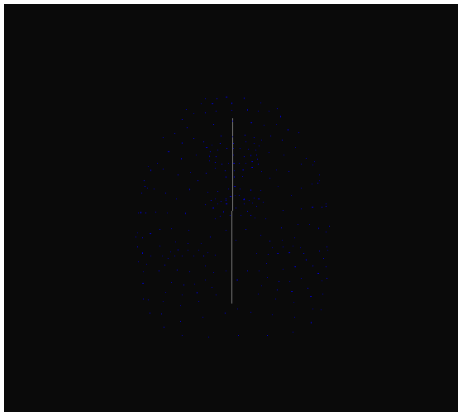

Figure 3: Nefertiti en 2D bandWidth = 2.0, ratio = 1

En 3D, le plan de réflexion est représenté par plusieurs points rouges qui appartiennent au plan.

Lorsque plusieurs clusters sont détectés, les symétries sont plus ou moins approchées suivant les exemples. Pour certaines valeurs de `bandWidth`, nous détectons des symétries qui semblent cohérentes. Cependant, il arrive que pour un nombre de cluster identique, avec deux valeurs de bandWidth différentes nous trouvions des résultats différents.

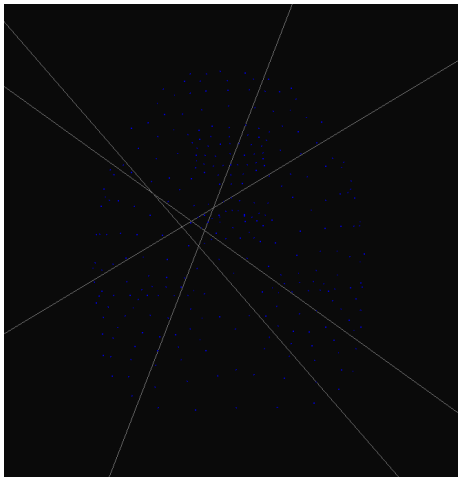

Figure 4: Nefertiti en 2D bandWidth = 1.0, ratio = 1

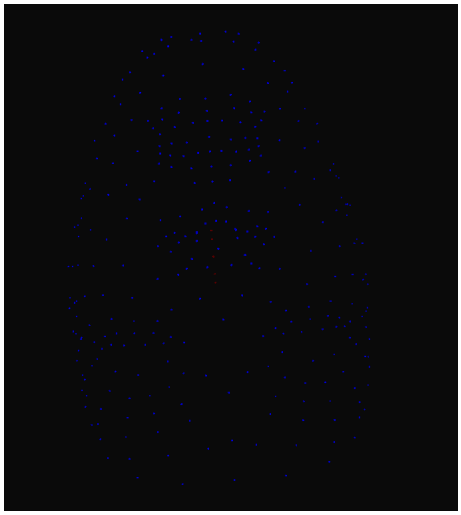

Figure 5: Nefertiti en 3D bandWidth = 3.0, ratio = 0.5

4 Conclusion et extensions

Ainsi, nous avons réussi à implémenter, au moins partiellement, les algorithmes décrits dans l'article. Ceux-ci semblent fonctionner en 2D comme en 3D, bien que souvent les symétries trouvées restent approchées. Il serait bien sûr possible d'améliorer notre solution, notamment en implémentant une correction des symétries grâce à Iterative Closest Point. De plus, il reste à généraliser le résultat en implémentant la méthode qui utilise les tenseurs de courbure afin d'être en mesure de détecter également les symétries qui sont des combinaisons de rotations, translations et changement d'échelle.