
INF555 – PROJET

Sketch-Based Shape Retrieval

L’article étudié présente une méthode permettant de retrouver, à partir d’un croquis, un modèle en trois dimensions dans une base de données de 1800 objets.

1 Méthode

Idée générale. La méthode présentée dans l’article repose sur la notion de *bag-of-features*, c’est-à-dire d’ensemble de vecteurs qui caractérisent une image. La recherche d’un modèle dans une base de données se ramène donc à la recherche d’un vecteur. Pour mener à bien cette démarche, il faut dans un premier temps générer une bibliothèque de vecteurs associés aux modèles 3D dans laquelle on va effectuer la recherche du vecteur généré à partir du croquis de requête.

Étapes principales de l’algorithme.

1. La première étape consiste donc en la génération de croquis, à partir d’un modèle, tel qu’un humain le ferait.
2. On découpe les croquis générés en morceaux, et pour chaque morceau, on calcule une *feature*. Cela représente donc un ensemble de plusieurs milliers de *features*, qu’on réduit à un millier de représentants par une méthode de classification non supervisée. Ces 1000 représentants vont constituer notre alphabet.
3. Enfin on représente chaque croquis par un histogramme correspond au nombre de fois que chaque mot de l’alphabet apparaît dans celui-ci. Cet histogramme est une sorte de carte d’identité de l’image.
4. Pour trouver les croquis générés (et donc les modèles) les plus proches d’un dessin, on compare simplement les histogrammes. Des histogrammes proches indiquent des dessins proches.



FIGURE 1 – Une orientation a été choisie, et seules certaines lignes sont dessinées.

2 Implémentation

L'algorithme, que nous avons choisi d'implémenter en **C++**, se découpe en deux grandes étapes :

- un pré-traitement des 1800 modèles est d'abord effectué, permettant d'obtenir une nouvelle base de données, composée de l'alphabet de 1000 mots et des histogrammes des croquis générés (*offline processing*) ;
- la recherche à partir d'un croquis dans cette nouvelle base de données (*online querying*).

2.1 Pré-traitement des modèles

Lorsqu'une personne réalise un croquis (en deux dimensions) d'un objet, elle effectue plusieurs choix (figure 1) :

- elle choisit une direction depuis laquelle regarder, ou selon laquelle projeter l'objet ;
- elle choisit de quelle façon transformer l'objet texturé qu'elle observe en simple lignes.

Lors de la première étape, pour chaque modèle 3D on génère un ensemble de 50 directions au hasard (mais uniformément réparties) selon lesquelles projeter. Comme suggéré par l'article on tire 50 points aléatoirement sur une sphère (provenant d'un fichier `.off` composé de 16 000 sommets) puis on applique l'algorithme *k-means* permettant de séparer la sphère en 50 régions de même taille uniformément réparties. On prend ensuite le centre de chaque région comme direction de projection.

Pour la deuxième étape, le choix que nous avons fait est de calculer une carte de profondeur de la projection (une image de l'objet où la couleur d'un point dépend de sa distance à l'observateur : un point proche sera noir et un point au loin sera blanc). On applique ensuite à l'image obtenue un algorithme de détection des contours (Canny). Nous ouvrons le modèle au format `.off` avec **OpenGL**, qui nous permet d'obtenir la carte de profondeur. Nous utilisons la bibliothèque **OpenCV** pour ensuite appliquer le filtre de Canny.

On obtient ainsi un ensemble d'images pour chaque modèle qui ressemblent à des croquis (figure 2 page suivante).

Nous effectuons ensuite les étapes suivantes.

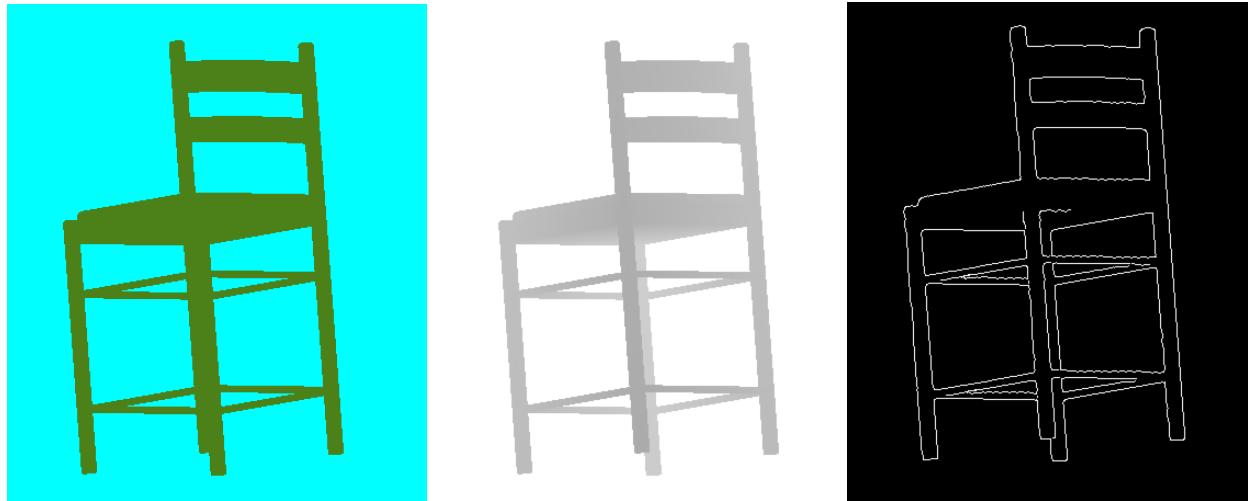


FIGURE 2 – Le modèle 3D, la carte de profondeur, puis le résultat après la détection des contours via Canny.

Filtre de Gabor. Afin d'améliorer les contours, on applique un filtre de Gabor à chaque image, suivant 4 directions. Cela permet de renforcer les contours dans la direction choisie et d'atténuer les autres. On calcule pour ça la transformée de Fourier discrète d'une image (avec OpenCV), on la multiplie pixel par pixel au filtre (dans le domaine spectral), puis on revient dans le domaine spatial. L'article restait assez flou sur les détails techniques de cette partie, qui s'est révélée être chronophage et difficile. Nous avons en effet rencontré quelques difficultés, en particulier avec la définition du filtre : la formule définissant le filtre dans le domaine spectral était valable pour des fonctions continues mais devait être adaptée pour un usage dans le cas discret. De plus, la multiplication de matrices complexes implémentées dans OpenCV n'a pas fonctionné. Ce problème a été particulièrement difficile à détecter mais a pu être résolu. Enfin, les résultats de filtrage obtenus présentent des artefacts ou effets de bord que nous n'avons pas réussi à éliminer.

Calcul des *features*. On découpe les 4 images obtenues en 1024 blocs d'aire 7,5 % de l'aire de l'image. Chaque bloc est ensuite divisé en 64 cellules. Pour chaque cellule on somme les valeurs des pixels qu'elle contient. Ces $64 \times 4 = 256$ valeurs forment un mot (ou *feature*). Nous stockons ces vecteurs sous forme de tableaux de flottants. Nous avons pu constater la difficulté de gérer correctement la mémoire en C++, notamment quand on emploie des *pointeurs*. Finalement, au lieu de stocker les vecteurs avec des `float*`, nous avons utilisé le conteneur fourni par la librairie standard `std::array<float, FEAT_SIZE>`, ce qui nous a permis de nous affranchir de l'usage des pointeurs.

Constitution d'un vocabulaire visuel. On élimine aléatoirement une partie de ces mots afin de n'en garder qu'un million. Cela représente néanmoins un volume d'environ 256 Mo. Grâce à l'algorithme *k-means*, on réduit ce million de mots à un millier, qui vont composer

notre alphabet.

Indexation des modèles. Enfin on calcule l'histogramme de chaque image en regardant pour chacun de ses 1024 mots de quel mot de l'alphabet il est le plus proche, au sens de la métrique euclidienne.

Cet ensemble d'étapes pouvant s'avérer très long (une vingtaine d'heures pour les 1800 modèles), nous stockons le résultat obtenu (l'alphabet et les histogrammes) dans un fichier. De façon générale, l'article passait sous silence toute question relative à l'efficacité, mais c'est un problème auquel nous avons dû être très attentifs. En effet, avec 1800 modèles, 50 projections par modèles et 1024 mots de longueur 256 par projection, on peut facilement atteindre 20 Go en mémoire.

2.2 Recherche d'un modèle

On commence par charger en mémoire l'alphabet et les histogrammes des modèles à partir du fichier dans lequel nous avions sauvégardé ces données.

Lorsqu'on donne un croquis au programme, on calcule son histogramme comme précédemment (on applique Canny, on filtre par Gabor suivant différentes orientations et on découpe en blocs pour calculer les *features*). Ensuite on cherche simplement de quels histogrammes il est le plus proche.

3 Résultats

3.1 Pré-traitement

Filtre de Gabor. On a commencé par vérifier que le filtre de Gabor donnait des résultats attendus, à savoir, intensifier les contours dans une certaine direction. Sur la figure 3 page suivante, on peut constater que pour $\theta = \pi/4$, ce résultat est obtenu, malgré quelques effets de bords.

Chaine de traitement complète. Pour constituer notre vocabulaire visuel et notre base de données d'histogrammes, on a choisi les paramètres suivants :

- nombre de modèles 3D : 100 (chaises, voitures, avions, ...);
- nombre de vues par modèle : 50;
- taille du vocabulaire : 50 mots¹.

Le temps d'exécution a été d'environ 50 minutes. Il est à noter que la mémoire utilisée n'a pas excédée 300 Mo. On obtient alors un fichier contenant :

- la longueur des mots, le nombres de mots (taille du vocabulaire) et le nombre de total de vues sur la première ligne ;

1. on a ainsi le même rapport nombre total de vues sur taille du vocabulaire que les auteurs, à savoir 100.

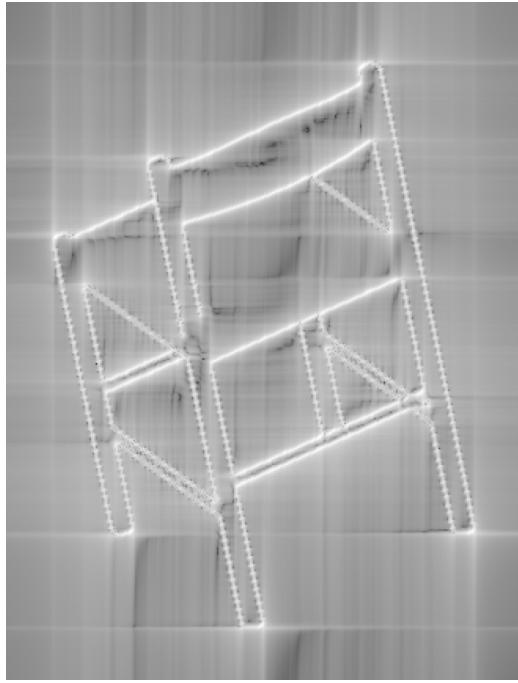


FIGURE 3 – Filtrage d'une chaise par Gabor.

- une ligne par mot : la fréquence de ce mot dans le vocabulaire, suivie des 256 coordonnées du mot ;
- une ligne par vue : le nom du modèle associé, suivi des valeurs de l'histogramme.

Ce fichier pèse quelques Mo.

3.2 Requêtes

L'étape de *querying* est beaucoup plus rapide. Après l'importation des données du fichier d'histogrammes, qui s'effectue quasiment instantanément, on calcule simplement un histogramme. Ensuite, on cherche les histogrammes les plus proches dans la base de données. On peut choisir le nombre de résultats souhaités. La figure 4 page suivante montre les résultats obtenus pour un croquis donné en entrée. On constate que notre implémentation ne donne pas des résultats satisfaisants. Cela peut s'expliquer par le faible nombre de modèles utilisés lors du pré-traitement ou par un bug dans le code qui fausserait les histogrammes calculés.

4 Pistes d'amélioration

Tout au long du projet, nous avons fait des choix, car l'article proposait souvent plusieurs solutions.

Nous choisissons les directions selon lesquelles projeter le modèle de manière aléatoire mais uniforme. Une autre possibilité est d'essayer de calculer la probabilité d'une projection en

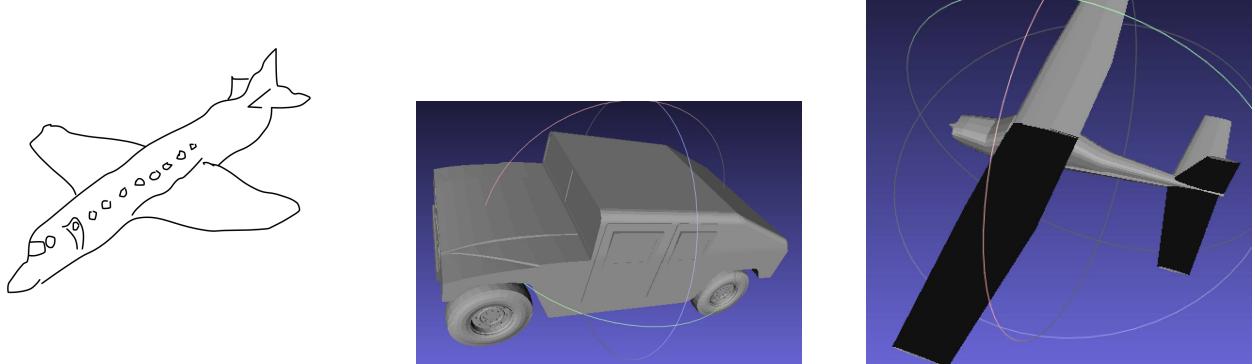


FIGURE 4 – Le croquis passé en entrée et les deux modèles trouvés par l'algorithme.

utilisant plusieurs facteurs tels que la longueur de la projection, l'aire de la projection, et les variations de profondeur.

Pour l'obtention des croquis à partir des modèles 3D, nous avons décidé d'appliquer Canny à la carte de profondeur. Cela donne de bons résultats, mais l'algorithme *Suggestive Contours* qui s'applique directement au modèle 3D pourrait donner de meilleurs résultats.

Nous pourrions également augmenter la résolution des projections, qui est actuellement limitée à 800×800 . Généralement les contours sont assez fins (4 à 5 pixels) et peu lisses.

Il est également possible d'agir sur le nombre de projections réalisées pour chaque modèle. Nous nous sommes limités à 50 pour des raisons d'efficacité, mais les auteurs suggèrent 100.