TELECOM PARIS
école nationale supérieure des télécommunications

Thèse

présentée pour obtenir le grade de docteur de l'Ecole Nationale Supérieure des Télécommunications

Spécialité: **Signal et Images**

Carlos Hernández Esteban

Stereo and silhouette fusion for 3D object modeling from uncalibrated images under circular motion

Modélisation d'objets 3D par fusion silhouettes-stéréo à partir de séquences d'images en rotation non calibrées

Soutenue le 4 mai 2004 devant le jury composé de :

Roger Mohr	Président
Peter Sturm	Rapporteurs
Laurent Cohen	
Michel Dhome	Examinateurs
Pascal Fua	
Francis Schmitt	Directeur de thèse

Remerciements

Je tiens a remercier mon directeur de thèse Francis Schmitt, qui m'a donné l'occasion de débuter dans le monde de la recherche, et a suivi avec enthousiasme, persévérance et même "*nocturnité*" le développement de ma thèse. À remercier aussi sa disponibilité éternelle, qui m'a permis de discuter avec lui à chaque fois que j'en ai eu besoin (et ça fait *beaucoup* de fois). Merci aussi à sa femme Pamela, toujours prête a relire et corriger les articles lorsque les *deadlines* approchent.

Je tiens aussi à exprimer mes remerciements a tous les membre du jury,

À Roger Mohr, pour avoir accepté d'être le président de mon jury de thèse,

À Peter Sturm et Laurent Cohen, pour avoir accepté d'être rapporteurs de ce travail de thèse. Ils ont également contribué avec beaucoup de suggestions et remarques à améliorer fortement la qualité du manuscrit,

À Michel Dhome et Pascal Fua d'avoir examiné ce travail avec attention et intérêt.

Je remercie Isabel Bloch, toujours disponible et prête à aider, surtout quand le thésard en question est trop fatigué pour finir un certain article dont la deadline est le jour même ...

Je voudrais aussi remercier Henry, Florence, Michel, Jean-Marie, Hans, Dominique et Sophie-Charlotte, qui ont contribué à ce que je garde un très beau souvenir de mon passage dans l'école (presque 6 ans au total, y compris ma période d'étudiant, uff!).

Je remercie mes compagnons de souffrance, les thésards de TSI, pour la bonne ambiance qu'il y a toujours eu et pour les discussions "animées" pendant le déjeuner (de n'importe quel sujet sauf de boulot, et si possible, politiquement incorrecte). Merci à Alex, Antonio, Tony, Celine, Ferdaous, Sveta, Oscar, Riadh, Pau, Jasmine, Florent, Daniel, François, Saéid et Dalila.

Enfin, je voudrais remercier mes parents Carlos et Charo, de m'avoir apporté leur soutien et affection imprescindibles pour arriver jusqu'ici, et aussi mon frère Pablo et ma soeur Laura, sans lesquels la vie serait trop simple et ennuyeuse.

Une mention spéciale à Arantxa, qui a su me supporter pendant 5 ans déjà, et à laquelle je dois, au minimum, la moitie de cette thèse.

Contents

Notation

$\mathbb{R}$	set of real numbers
$\mathbb{R}^2$	Euclidean plane
$\mathbb{R}^3$	Euclidean space
$\mathbb{P}^2$	projective plane
$\mathbb{P}^3$	projective space
M	matrix
m_{ij}	element in row i and column j of matrix M
$\mathbf{v}$	vector
v_x	x component of vector $\mathbf{v}$
$\mathbf{v} \cdot \mathbf{w}$	dot product between vectors $\mathbf{v}$ and $\mathbf{w}$
$\|\|\mathbf{v}\|\|$	Euclidean norm of vector $\mathbf{v}$
$\mathbf{v}^t, M^t$	tranposed vector $\mathbf{v}$ and matrix M
s	scalar
$\mathbf{0}_n$	vector of n zeros
I_n	identity matrix of size n
$\simeq$	equal *up to a scale*
∇	gradient operator
$\nabla^2,\ \Delta$	Laplacian operator
Δ^2	biharmonic operator

Résumé

L'évolution très rapide des logiciels et matériels en informatique graphique nous permet de prêter davantage attention à la création ou à l'acquisition de modèles 3D de haute qualité. En particulier, un effort très important est actuellement réalisé pour la représentation numérique d'objets 3D du monde réel. Les modèles 3D d'objets réels sont utilisés dans une grande variété d'applications comme par exemple le divertissement (films et jeux vidéo), la publicité (achat en ligne) ou la conservation et l'indexation d'oeuvres d'art. Dans le domaine culturel on peut citer les deux exemples d'application suivant : l'acquisition de sites entiers et la constitution de bases de données 3D à partir des collections des musées.

La plupart des modèles 3D d'objets réels que nous voyons dans la vie quotidienne sont issus de graphistes qui utilisent des logiciels spécialisés comme 3D Studio ou Maya. La quantité de travail requise pour obtenir des modèles de haute qualité rend le processus de création long et cher. Même si les communautés scientifiques de la vision par ordinateur, de l'infographie et des scanners 3D traitent ce problème depuis une trentaine d'années, l'acquisition automatique d'objets 3D reste une tache difficile. En fait, la raison pour laquelle les gens continuent à créer les modèles 3D "à la main", même après 30 ans d'expérience dans ce domaine, est qu'il n'y a pas de technique 3D idéale qui permette de scanner tout objet correctement. La difficulté de trouver un système approprié de reconstruction 3D fonctionnant pour une large classe d'objets justifie la littérature abondante qui existe à ce sujet.

Mis à part les techniques de Conception Assistée par Ordinateur (CAO), il existe deux approches principales au problème de la représentation d'objets 3D réels : les techniques fondées exclusivement sur le rendu par les images (*Image-Based Rendering ou IBR*), et les techniques de reconstruction 3D.

Les techniques fondées sur le rendu par les images permettent de générer des vues de synthèse à partir d'un ensemble d'images originales. Elles n'estiment pas la vraie structure 3D à partir des images, mais elles *interpolent* directement l'ensemble des vues originales pour générer une nouvelle vue. Dans ce type d'approche, l'objectif est de générer des vues cohérentes de la scène réelle, pas d'avoir des mesures précises. Pour beaucoup d'applications comme les effets spéciaux, ces techniques peuvent se révéler très efficaces.

A l'opposé des techniques IBR, les algorithmes de reconstruction 3D essaient d'estimer complètement la structure 3D sous-jacente. Parmi les techniques de reconstruction 3D, nous pouvons distinguer deux groupes principaux : les méthodes actives et les méthodes passives. Les méthodes actives sont utilisées principalement

par la communauté des scanners 3D et utilisent une source de lumière contrôlée tel qu'un laser ou une lumière structurée pour récupérer l'information 3D (*3D scanning*). Les méthodes passives sont développées par la communauté de la vision par ordinateur et n'utilisent que l'information contenue dans les images de la scène. Dans ce type d'algorithmes, les seules données en entrée sont les images de l'objet, éventuellement calibrées. Les principaux avantages sont un coût réduit du système d'acquisition et la possibilité de capter la couleur directement. Le principal désavantage est la plus faible résolution des modèles reconstruits comparée à celle des modèles issus des méthodes actives.

Parmi les techniques de modélisation 3D à partir des images, les approches basées sur les silhouettes sont couramment utilisées. Elles fournissent généralement un modèle initial pour d'autres techniques plus avancées de reconstruction 3D et sont également suffisamment rapides pour des applications temps réel. D'autres types d'information peuvent être utilisés pour la modélisation 3D, tels que la stéréo ou l'albédo. Nous proposons une nouvelle technique de modélisation 3D qui est capable de fusionner les informations liées aux silhouettes et à la stéréo de manière à reconstruire des objets 3D de haute qualité. Cet algorithme a été testé avec succès dans le cadre du projet européen SCULPTEUR IST-2001-35372, avec plus de 100 objets de musées reconstruits par notre méthode. L'algorithme de reconstruction a été amélioré à fin de simplifier le processus d'acquisition. En particulier, nous nous sommes attaqués à l'une des étapes les plus critiques pour n'importe quel algorithme de vision par ordinateur : le calibrage. Nous proposons une nouvelle technique de calibrage qui utilise comme seule source d'information les silhouettes de l'objet. Cette technique met en oeuvre une nouvelle façon d'utiliser les silhouettes et nous permet d'estimer les paramètres de la caméra sous l'hypothèse d'un mouvement circulaire. Le mouvement circulaire, grâce à sa simplicité de mise en oeuvre, est couramment utilisé pour la modélisation 3D à partir des images. Supposant les silhouettes correctement extraites, nous définissons un **critère de cohérence d'un ensemble de silhouettes** et le proposons comme mesure globale de qualité pour le calibrage d'un sous-ensemble des paramètres de la caméra. Le calibrage est accompli en maximisant le critère de cohérence entre silhouettes. En l'ajoutant à la méthode proposée de reconstruction 3D, le calibrage à partir des silhouettes nous permet d'obtenir des reconstructions de haute qualité sans l'utilisation d'une quelconque mire géométrique.

Ce manuscrit est organisé en trois chapitres. Le premier est consacré au problème du calibrage de la caméra et le second à l'algorithme de reconstruction 3D. Le troisième chapitre décrit en détail un ensemble de résultats expérimentaux que nous avons sélectionnés pour illustrer différents points d'intérêt.

Dans le **chapitre 1** nous présentons une nouvelle approche pour l'estimation du mouvement et de la distance focale d'une caméra sous l'hypothèse de mouvement circulaire. Cette approche s'appuie sur la définition du concept de cohérence d'un ensemble de silhouettes. La cohérence est définie comme la similitude entre l'ensemble de silhouettes originales et les silhouettes de leur enveloppe visuelle. Cette approche a été testée avec succès dans différents problèmes d'optimisation tels que l'estimation du mouvement et de la distance focale d'une séquence en rotation ou le recalage de deux

séquences indépendantes. La bonne précision des résultats est due à l'utilisation du contour complet de la silhouette dans le calcul de la cohérence, alors que les méthodes basées sur les tangentes épipolaires n'emploient qu'un faible nombre de points. Nous avons validé la méthode en utilisant des séquences réelles, ce qui nous a permis de reconstruire des objets 3D à partir de séquences d'images en rotation, sans l'utilisation d'une quelconque information additionnelle telle que la prise de vue d'une mire de calibrage.

Nos principales contributions sont:

- la définition et l'implantation du nouveau concept de cohérence d'un ensemble de silhouettes,
- la comparaison entre le nouveau critère proposé de cohérence entre silhouettes, et le critère des tangences épipolaires proposé par [Wong and Cipolla, 2001],
- la comparaison de la précision du critère des tangences épipolaires en fonction du nombre de paires de silhouettes utilisées dans le calcul du critère.

Nous prouvons qu'une implantation rapide du critère des **tangences épipolaires** à partir d'une représentation polygonale des silhouettes peut fournir de très bons résultats. De plus, bien que dans l'article original le critère soit seulement employé pour estimer le mouvement, nos résultats montrent qu'il est également possible d'estimer la distance focale, comme nous le démontrons avec les nombreux résultats expérimentaux obtenus sur des données synthétiques et réelles. Finalement, et à la différence de la formulation originale de [Wong and Cipolla, 2001], nous prouvons que cela vaut la peine d'employer toutes les paires disponibles de silhouettes pour calculer le critère plutôt que de n'employer que les paires les plus proches.

Le critère proposé de cohérence d'un ensemble de silhouettes devrait encore pouvoir bénéficier de plusieurs améliorations. Nous étudierons la robustesse de cette technique de calibrage pour des mouvements plus complexes que le mouvement circulaire. D'après la précision obtenue pour l'estimation des différentes variables avec le processus d'optimisation, nous devrions aussi pouvoir améliorer la convergence de l'algorithme en employant une technique d'optimisation utilisant des dérivées, avec une évaluation robuste des dérivées adaptée à ce problème. Un autre point très important à considérer serait l'aspect discret de l'actuelle implantation du critère de cohérence dont il faudrait se débarrasser. L'idée principale serait de calculer de manière exacte les silhouettes de l'enveloppe visuelle comme des polygones fermés. Ceci nous permettrait de définir un critère de cohérence par la simple comparaison des deux polygones définissant la silhouette originale et la silhouette de l'enveloppe visuelle. L'algorithme pour calculer la silhouette de l'enveloppe visuelle serait :

- calcul des surfaces 3D générées par l'intersection de chaque silhouette avec les $n-1$ autres silhouettes,
- projection des arêtes des surfaces 3D dans la vue désirée,
- calcul du polygone minimal contenant toutes les arêtes projetées.

Le chapitre est organisé de la façon suivante. Nous présentons d'abord le modèle de caméra et donnons une brève explication des méthodes classiques de calibrage et des méthodes d'auto calibrage en section 1.1. Ensuite nous décrivons dans la section 1.2 les techniques existantes qui emploient les silhouettes pour l'estimation de mouvement ou le calibrage d'une caméra. Dans la section 1.3 nous détaillons le concept de cohérence d'un ensemble de silhouettes et le proposons comme mesure globale de qualité pour le calibrage d'un sous-ensemble des paramètres de la caméra. Dans la section 1.4 nous prouvons que le critère de cohérence apparaît comme une extension naturelle de l'approche avec les tangences épipolaires. Puisque nous travaillons avec des séquences en mouvement circulaire, bien connu pour être un mouvement critique pour l'auto calibrage, nous indiquons en section 1.5 quel est le sous-ensemble des paramètres qui peuvent être estimés en utilisant la cohérence de silhouettes et précisons pourquoi avec une explication physique. Dans la section 1.6 nous proposons une implantation rapide du critère de cohérence et discutons les différentes techniques d'optimisation dans la section 1.7. Enfin nous présentons des résultats expérimentaux avec des données synthétiques et réelles dans les sections 1.8 et 1.9.

Dans le **Chapitre 2** nous présentons une nouvelle approche pour la modélisation d'objets 3D basée sur la fusion des informations de texture et des silhouettes. Nous proposons un système complet où différentes techniques connues sont employées et améliorées pour fournir des résultats de haute qualité. La méthode utilise un modèle déformable classique qui nous permet de fusionner les informations de texture et de silhouettes pour estimer la géométrie et la couleur de l'objet. Nous proposons une nouvelle formulation de la contrainte des silhouettes et l'utilisation d'une approche par diffusion multi-résolution du vecteur gradient (GVF) pour la composante stéréo.

Nos contributions principales sont les suivantes:

- l'approche par décision majoritaire 1-D, qui permet de prendre une décision robuste à partir de n courbes de corrélation stéréo,
- l'implantation d'une technique multi-corrélation par décision majoritaire 3D, qui nous permet d'extraire l'information géométrique d'un ensemble d'images de façon robuste, améliorant les résultats dans des conditions réelles d'illumination, et tout particulièrement dans le cas de reflets spéculaires,
- l'adaptation de la technique de GVF, généralement employé dans le domaine de l'imagerie médicale, à notre problème spécifique de modélisation 3D. Cette adaptation a nécessité une implantation sur une grille multi-résolution ce qui, à notre connaissance, n'a jamais été développé auparavant pour la modélisation d'objets 3D,
- la définition d'une force définie par les silhouettes qui permet d'intégrer facilement les silhouettes dans l'évolution du modèle déformable,
- l'adaptation à un maillage triangulaire de l'opérateur bi-harmonique classique généralement utilisé avec les maillages simpliciaux,

- l'implantation d'une technique de placage de texture qui permet de spécifier le niveau de détail de la texture de chaque triangle, tout en gardant des transitions douces par interpolation bilinéaire entre des triangles adjacents,

- la définition d'une technique pratique d'échantillonnage de la texture des triangles permettant de créer une carte de texture sans artefacts, en particulier le long de deux triangles adjacents n'ayant pas la même résolution.

- et pour finir, l'aspect global du système qui a été développé avec soin dans chacune de ses parties et qui fournit des résultats très satisfaisants comme le prouve la quantité et la qualité des multiples objets reconstruits dans des conditions expérimentales très variées.

Les deux limitations les plus importantes de cette approche sont aussi ses deux sources de robustesse : le volume de corrélations de décision majoritaire et l'utilisation d'un modèle déformable avec une topologie constante. Le cumul des corrélations permet d'avoir de bonnes reconstructions en présence de reflets, mais le fait qu'il soit effectué dans un volume discret limite la résolution maximum du modèle 3D. Cette limitation pourrait être surmontée en introduisant le modèle final dans une autre évolution où l'énergie liée à la texture prendrait en compte la surface courante du modèle 3D (corrélation fondée sur le plan tangent ou l'approximation par une quadrique). Étant donné que le modèle est déjà très près de la solution, nous n'aurions besoin que de très peu d'itérations. La seconde limitation est celle de la topologie constante. Elle permet de garantir la topologie du modèle final mais c'est aussi une limitation pour les objets dont la topologie ne peut pas être captée par l'enveloppe visuelle. Une solution possible consisterait à détecter les auto-collisions du maillage et, pour récupérer la bonne topologie, d'effectuer soit une étape fondée sur les ensembles de niveaux, soit une modification topologique du modèle 3D. En outre, comme déjà discuté en section 3.3, nous voudrions étudier de manière plus approfondie la gestion de la visibilité de la surface de corrélation, trouver une force de silhouettes plus générale, et avoir une meilleure gestion des objets possédant des arêtes vives. D'autres améliorations sont également envisageables :

- une amélioration de la stratégie de convergence afin d'une part d'arrêter l'évolution du modèle dans les régions qui ont déjà convergées, et d'autre part d'accélérer son évolution dans les régions vides de l'espace de corrélation,

- l'utilisation de la courbure pour permettre une évolution multi-résolution du maillage,

- des développements supplémentaires au niveau de la génération de la texture. En particulier, puisque nous disposons de la géométrie de l'objet, nous pourrions appliquer des techniques plus avancées de traitement de signal pour mieux filtrer les ombres portées et, dans des conditions d'éclairage calibré, pour estimer partiellement la fonction de distribution bidirectionnelle de réflectance (BDRF) en chaque point de la surface de l'objet.

Chaque section du chapitre est consacrée à une brique de base de la théorie des modèles déformables appliquée à notre problème. Le chapitre est donc organisé de la façon suivante. Dans la section 2.1 nous décrivons l'état de l'art et comment la technique proposée se compare aux méthodes existantes. Dans la section 2.2 nous donnons une vue d'ensemble rapide de la théorie des modèles déformables et justifions le choix d'un modèle déformable classique par rapport à la méthode d'ensembles de niveaux qui est plus récente et actuellement très étudiée. Dans la section 2.3 nous décrivons comment initialiser le modèle déformable tandis que dans les sections 2.4 et 2.5 nous développons les deux forces externes qui vont diriger le modèle déformable : la force définie par la texture et la force définie par les silhouettes. Dans la section 2.6 nous détaillons les forces de régularisation utilisées et précisons leur formulation dans le cas d'un maillage triangulaire. La boucle d'itération du modèle déformable est récapitulée en section 2.7. La dernière étape consistant à calculer une carte de texture pour un modèle 3D à partir d'un ensemble d'images, est décrite dans la section 2.8.

Enfin nous présentons dans le **Chapitre 3** quelques-uns uns des résultats les plus intéressants obtenus par la combinaison des algorithmes décrits dans les deux chapitres précédents. Le chapitre est divisé en 5 sections. Les 3 premières sections présentent des résultats de reconstruction n'utilisant qu'une seule séquence en rotation. Ils sont classifiés selon la quantité d'information que nous fournissons à l'algorithme allant de 3 séquences d'images (une séquence pour la texture, une séquence pour les silhouettes et une séquence pour le calibrage) à une seule séquence d'images dont sont automatiquement extraites les silhouettes de l'objet permettant de calibrer le système. Nous discutons également les limites de la technique proposée, qui sont liées aux types d'information que nous employons (les silhouettes et la stéréo) et à la manière de les fusionner (un modèle déformable). Nous présentons finalement des résultats additionnels qui exploitent *deux* séquences en rotation du même objet, et un aperçu des meilleures reconstructions obtenues avec la technique proposée.

Introduction

As computer graphics and technology become more powerful, attention is being focused on the creation or acquisition of high quality 3D models. As a result, a great effort is being made to exploit the biggest source of 3D models: the real world. 3D models of real objects are used in a great variety of applications ranging from entertainment (games and movies), to advertising (online shopping) or cultural heritage. Two examples of cultural heritage applications are the acquisition of entire outdoor sites, and the construction of 3D databases from museums art collections.

Many of the 3D real models that we see every day have been created by a graphic designer using specialized tools such as 3D Studio or Maya. However, the required work to obtain a good quality model makes the entire process very long and expensive. Although the Computer Vision, Computer Graphics and 3D scanner communities have been studying the 3D modeling problem for at least 30 years, automatically acquiring 3D models is not an easy task. The reason why, even with 30 years of experience in this domain, people continue to create themselves *by hand* the 3D models is that, in fact, it does not exist any Computer Vision or 3D scanner technique that works "out of the box" for any object. The difficulty of finding a suitable 3D reconstruction system that works for a large class of objects justifies the abundant literature that exists on this subject.

Discarding Computer Aided Design (CAD) tools, there are two major approaches to the problem of 3D real object representation: image-based rendering techniques (IBR) and 3D modeling techniques. IBR algorithms do not estimate the real 3D structure behind the images, they only *interpolate* a given set of images to generate a new synthetic view. This kind of techniques may be enough for many applications and are intensively used by the special effects community. In opposition to IBR, 3D modeling algorithms try to recover the full 3D structure. Among the 3D modeling approaches, two main groups are to be distinguished: active methods and passive ones. Active methods are used by the 3D scanner community and require a controlled source of light such as a laser or a coded light in order to recover the 3D information. Passive methods are used by the Computer Vision community and necessitate only the information contained in the images of the scene. In this last kind of approach, the only input data to the algorithm are a set of images, possibly calibrated. Its main advantages are the low cost of the system and the possibility of immediate color. The main disadvantage is the lower precision of the reconstructions compared to the precision obtained with active techniques.

Among the 3D **image-based modeling** techniques, silhouette-based approaches

are very popular since they provide good initial models for further processing in 3D reconstruction algorithms and are efficient enough for real time applications. Other types of information are also used in 3D reconstruction such as stereo or albedo. We propose a new 3D modeling algorithm that is able to efficiently fuse silhouettes and stereo for high quality 3D object reconstruction. This algorithm has been successfully tested in *real life* in the framework of the SCULPTEUR European project IST-2001-35372, where more than 100 objects have been acquired using the proposed technique. This reconstruction technique has been improved in order to simplify the overall acquisition process. As a result, we have simplified one of the critical steps of any Computer Vision technique: camera calibration. We have proposed a new camera calibration algorithm that uses silhouettes as the only input data. This technique makes use of the silhouettes in a novel way in order to recover the camera parameters under circular motion, which is commonly used for image-based 3D object modeling. Assuming that the silhouettes are "*sufficiently*" well extracted, we propose the silhouette coherence as a global quality measure for the calibration of a subset of the camera parameters by maximizing the overall silhouette coherence. Together with the proposed reconstruction method, silhouette-based camera calibration allows us to obtain high quality 3D reconstructions without the use of any calibration pattern.

This manuscript is organized in three chapters, one dedicated to the camera calibration problem, another for the 3D reconstruction algorithm and a final chapter that describes in detail some of the most interesting experimental results.

Chapter 1 addresses the problem of camera calibration, which is fundamental before any attempt of 3D reconstruction. A quick review of classic and auto-calibration methods is first discussed. Then the core of the proposed calibration method is presented with the notion of silhouette coherence that appears as an extension of the epipolar tangency approach. A fast implementation of the silhouette coherence is developed, which allows comparing the proposed algorithm with the epipolar tangency approach using both synthetic and real data.

In **Chapter 2** we describe the proposed 3D object modeling algorithm as a deformable model evolution that allows mixing silhouettes and stereo. We first discuss the main existing techniques for 3D object modeling and how the proposed method compares to them. Then, the complete algorithm pipeline is described. Since the main algorithm is a deformable model, we start by the model initialization, followed by the description of the two external forces that will drive the evolution process: one texture-based force and one silhouette-based force. We end with the regularization forces and the creation of a texture map once the deformable model has converged.

Finally we present in **Chapter 3** some of the most relevant results obtained by combining the algorithms described in the two previous chapters. The chapter is divided into 5 sections. The 3 first sections present reconstruction results using one circular sequence. They are classified depending on how much information we provide to the algorithm: from 3 image sequences (one sequence for the texture, one sequence for the silhouettes and a final sequence of a calibration pattern to use it with a classic calibration technique), to a single color sequence that permits automatically extracting the object silhouettes, and calibrating the system with them. We also discuss the limits

of the proposed technique related to the types of information we use (silhouettes and stereo) and the way they are fused (a deformable model). We finally present additional results that exploit *two* circular motion sequences of a same object and a snapshot of the best reconstructions obtained with the proposed technique.

Chapter 1

Silhouette-based Motion Estimation and Camera Calibration

In this chapter we present the concept of silhouette coherence and its application to motion estimation and camera calibration under circular motion. In Section 1.1 we shortly recall the classical pinhole camera model and we give a brief explanation of classic and auto calibration methods. In Section 1.2 we describe existing techniques that use silhouettes for motion estimation or camera calibration. In Section 1.3 we introduce the new concept of silhouette coherence of a set of silhouettes and propose it as a global quality measure for the calibration of a subset of the camera parameters by maximizing the overall silhouette coherence. In Section 1.4 we show that the silhouette coherence criterion can be seen as an extension to the epipolar tangency criterion. Since we work with circular motion sequences, which is known to be a critical motion for auto-calibration, we discuss in Section 1.5 which subset of the camera parameters can be recovered using silhouette coherence and why. In Section 1.6 we propose a fast implementation of the silhouette coherence criterion and discuss different optimization techniques in Section 1.7. Finally we present experimental results with synthetic and real silhouettes in Sections 1.8 and 1.9.

1.1 Introduction to Camera Calibration

Camera calibration is a very important initial step for Computer Vision, especially for 3D modeling, pattern recognition or robot navigation. By calibration we mean the determination of the full set of parameters of the image formation model. Although camera calibration can be decomposed into radiometric and geometric calibration, we are mainly interested in geometric calibration. The geometric calibration can be defined as finding the mathematical relationship between the 3D coordinates of a point of the scene, and the 2D coordinates of the projected point into the image. There exist many different methods for geometric calibration (see [Hartley and Zisserman, 2000] or [Faugeras and Luong, 2001] for a detailed description of the theory and the algorithms), but they can be classified into two main groups:

- photogrammetric calibration,

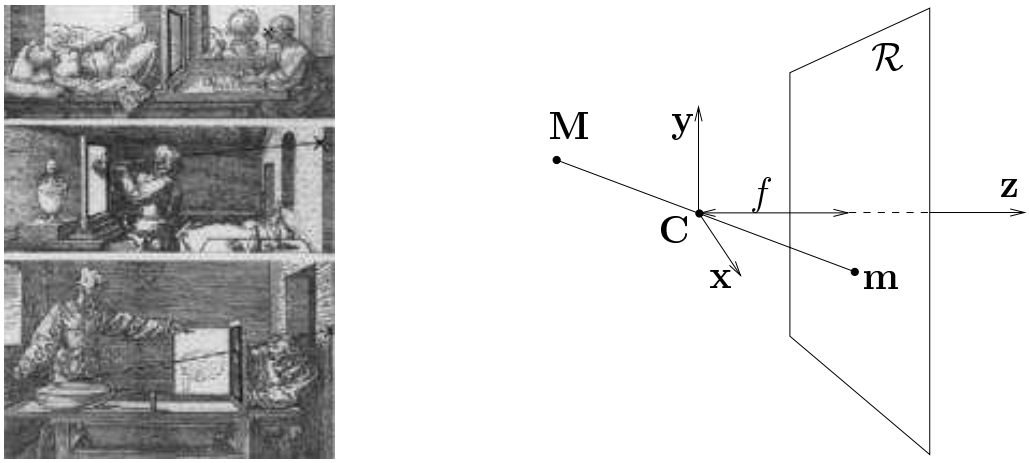


Figure 1.1: *Pinhole camera model. Left: perspective law by the Renaissance painters [Dürer, 1977]. Right: geometrical model.*

- auto calibration (also known as self calibration).

Photogrammetric methods use the known geometry of a calibration pattern to calibrate the cameras. Self calibration methods use only image information and some previous knowledge of the scene to calibrate the system up to a scale factor.

In the following subsections we introduce the classic pinhole camera model and present briefly the basic methods of photogrammetric calibration and self calibration.

1.1.1 Camera Model

A camera can be seen as a device that transforms the Euclidean space of a scene $\mathbb{R}^3$ into the camera image space $\mathbb{R}^2$. This transformation is not linear in the general case. However, If we suppose perfect lenses, the transformation is projective linear. Because the Euclidean space can be seen as a specialization of the projective space, we can embed the Euclidean space into the more general projective space in order to linearize the camera transformation. This embedding action increases the dimension of the space by one (for any point of the Euclidean space, the new last coordinate is 1). The embedding action permits seeing a camera as a device that linearly maps the projective space $\mathbb{P}^3$ into the projective plane $\mathbb{P}^2$.

A simple model to describe the formation of an image is the so called pinhole camera (see Fig.1.1). The model is completely described with the choice of a retina plane $\mathcal{R}$ and a projection center $\mathbf{C}$ (see Fig. 1.1 right). For a given 3D point $\mathbf{M} = (M_x, M_y, M_z)^t$, its corresponding 2D projection $\mathbf{m} = (m_x, m_y)^t$ is simply computed as the intersection of the optic ray defined by the point $\mathbf{M}$ and the projection center $\mathbf{C}$ with the retinal plane $\mathcal{R}$. For the simplest case, the camera projection center is located at the origin and the retinal plane is located at a distance f along the $\mathbf{z}$ axis, which gives the following projection equations for the point $\mathbf{M}$:

$$m_x = fM_x/M_z,\ m_y = fM_y/M_z. \tag{1.1}$$

If we use homogeneous coordinates, we get:

$$\begin{bmatrix} m_x \\ m_y \\ 1 \end{bmatrix} \simeq \begin{bmatrix} fM_x \\ fM_y \\ M_z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} M_x \\ M_y \\ M_z \\ 1 \end{bmatrix}, \tag{1.2}$$

where "$\simeq$" means "*equality up to a scale*". If we generalize the equations for a general camera position $\mathbf{t}$ and a general viewing direction represented by the rotation matrix R, we obtain:

$$\mathbf{m} \simeq \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}_3^t & 1 \end{bmatrix} \mathbf{M}. \tag{1.3}$$

The last step to completely describe the camera model is that, in general, the 2D coordinates suffer a last transformation that depends only on the camera itself. This transformation allows passing from the world units, e.g. millimeters, to the camera units: the pixel. Although this transformation is not always linear due to imperfect lenses, there is a more simple model called the intrinsic matrix K that is an affine model composed of 5 parameters. Since the focal distance f appears as a scale factor in the projection equation, it can be incorporated in the intrinsic matrix, which gives:

$$K = \begin{bmatrix} f & -f\cot\theta & u_0 \\ 0 & \frac{\alpha f}{\sin\theta} & v_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{1.4}$$

The new 2D axes have a different scale along the $\mathbf{x}$ axis and the $\mathbf{y}$ axis that is controlled by the aspect ratio parameter α, and may not be orthogonal, the angle between both axes being θ. Finally, there is also a pixel offset called the principal point: $(u_0, v_0)^t$.

The complete projection model is defined as:

$$\mathbf{m} \simeq K[I_3\ \mathbf{0}_3] \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}_3^t & 1 \end{bmatrix} \mathbf{M} = \underbrace{K[R\ \mathbf{t}]}_{P} \mathbf{M}, \tag{1.5}$$

where the matrix P is commonly called the camera projection matrix. The decomposition of the projection matrix P into intrinsic parameters (matrix K) and extrinsic parameters (matrix $[R\ \mathbf{t}]$) is known as the Euclidean camera model. Since the 3×4 projection matrix P is defined up to a scale, it has a total of 11 degrees of freedom (dof), just like the Euclidean decomposition (5 for K + 6 for $[R\ \mathbf{t}]$). In fact, for any projection matrix P, we can always find an Euclidean decomposition, and what is even better, this decomposition is unique. This argument can be easily demonstrated by using a QR decomposition on the most left 3×3 submatrix of P, which corresponds to the product KR.

We have just described the most complete linear model of a pinhole camera. However, depending on the real conditions, simpler models can also produce very good results. In the following subsection we discuss some of the most common simplifications that will be useful in the rest of the chapter.

1.1.2 Simplifications of the projective camera model

One of the first simplifications consists of using ideal values for the aspect ratio α and the skew factor θ, i.e., $\alpha = 1$ and $\theta = \frac{\pi}{2}$. This approximation is of special interest for CCD or CMOS cameras, since the deviation from the ideal values is very small. It just considers the pixels as perfectly square and the CCD matrix (the retinal plane) aligned with the **xy** plane. This gives us a simplified intrinsic matrix K with only 3 dof:

$$K = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{1.6}$$

The second simplification concerns the camera projection model itself. As we have seen in Eqn. 1.1, the 2D projection $\mathbf{m}$ of a 3D point $\mathbf{M}$ depends on the depth from the point to the camera, which is why the model is projective linear. However, under the assumption that the depth variation of the scene is small compared to the distance to the camera, there exist several linear camera models known as *affine camera* models that approximate the perspective camera model. The relation between the world coordinates and the pixel coordinates is affine. The general form of an affine projection matrix is as follows:

$$P \simeq \begin{bmatrix} P_A & p_A \\ 0_3^t & 1 \end{bmatrix}, \tag{1.7}$$

where P_A is a 2×3 matrix of rank 2 and p_A is a 2x1 vector. There exist some well known affine camera models such as the orthographic camera, the weak perspective camera or the para-perspective camera. Of particular interest is the weak perspective model, whose general form is:

$$\begin{aligned} P &\simeq \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & d \end{bmatrix} \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}_3^t & 1 \end{bmatrix}, \\ &\simeq \begin{bmatrix} f & 0 & u_0 + \frac{f}{d} t_x \\ 0 & f & v_0 + \frac{f}{d} t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{r}_1^t & 0 \\ \mathbf{r}_2^t & 0 \\ \mathbf{0}_3^t & d \end{bmatrix}, \end{aligned} \tag{1.8}$$

where d is the mean depth to the scene, the rotation matrix being decomposed as $R = [\mathbf{r}_1 \mathbf{r}_2 \mathbf{r}_3]^t$. If $\overline{\mathbf{M}}$ is the centroid of the scene, the mean depth d can be found as $d = \mathbf{r}_3^t \cdot \overline{\mathbf{M}} + t_z$. This model simplifies the projection equations by replacing the individual depth to the camera by the mean depth to the camera, i.e., all the points are projected as if they were at the same depth from the camera. As shown in Eqn. 1.7, there is one interesting consequence of this simplification: we cannot distinguish between the pixel offset $\frac{f}{d}(t_x, t_y)^t$ and the principal point $(u_0, v_0)^t$ when using a weak perspective model. This is a very interesting point since, as we will see in the rest of this chapter, it is very difficult to recover at the same time the principal point and the translation of a circular motion using only silhouettes. This is caused by the fact that the depth variation along the silhouette contour is in general very small

compared to the mean depth to the object, i.e, $\Delta d \ll d$. This would suggest employing a weak perspective camera model when calibrating with silhouettes. However, there is another similar simplification that we can use for the camera model under the assumption $\Delta d \ll d$ under circular motion: using the center of the image as principal point. It allows calibrating with silhouettes while using a perspective camera model. This results from the assumption of $\Delta d \ll d$, which validates the approximation of the principal point by the center of the image under circular motion, since the principal point error can be compensated by the translation. This will be shown in Section 1.5. For silhouette-based calibration, we have then preferred to keep using a perspective model while fixing the principal point to the center of the image. However, it would be useful to compare the principal point approximation with the weak perspective approximation in terms of calibration results, which we have not done yet.

1.1.3 Classic Camera Pose and Calibration Methods

Once we have chosen a camera model, the camera calibration process simply consists of recovering the 11 parameters that define the camera projection matrix P in Eqn. 1.5. Classic calibration methods are issued from the photogrammetry community, where a calibration target with a known geometry is needed in order to completely calibrate the camera. Although the use of a calibration pattern imposes more constraints on the acquisition process, the resulting calibration is very accurate. The basic input of the calibration algorithm is the set of 3D points of the pattern and their corresponding 2D projections. All the methods are based on the optimization of a cost criterion and can be classified into two main groups: linear methods and non-linear methods. Linear methods were the first to appear. They allow recovering the projection matrix in a direct way. However, the calibration results are not very accurate. This is mainly due to the rotation parameterization and to the fact that the optimized criterion has no geometrical meaning. In order to solve these problems, non-linear methods allow decomposing the projection matrix into its Euclidean form, parameterizing the rotation (e.g. Euler angles), using more complex non-linear camera models such as those including lenses distortion, and using more complex cost functions.

Bellow we give a brief description of the different classic calibration techniques.

Linear Methods

The basic linear method, also known as DLT (Direct Linear Transform), was first proposed by [Abdel-Aziz and Karara, 1971], but the same basic idea was already developed by the photogrammetry community about 20 years before [Das, 1949]. This method permits formulating the calibration problem as the solution of a system of linear equations.

The first step is to write down the so called collinearity equations between a 3D point $\mathbf{M}$ and its projection $\mathbf{m}$. This is accomplished by simply developing the projec-

tion equation Eqn. 1.5:

$$\begin{bmatrix} m_x \\ m_y \\ 1 \end{bmatrix} \simeq \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} M_x \\ M_y \\ M_z \\ 1 \end{bmatrix}, \quad \begin{cases} m_x = \frac{p_{11}M_x+p_{12}M_y+p_{13}M_z+p_{14}}{p_{31}M_x+p_{32}M_y+p_{33}M_z+p_{34}} \\ m_y = \frac{p_{21}M_x+p_{22}M_y+p_{23}M_z+p_{24}}{p_{31}M_x+p_{32}M_y+p_{33}M_z+p_{34}} \end{cases} \tag{1.9}$$

If we dispose of a calibration pattern defined by n 3D Points $\mathbf{M}^i$, and their corresponding observed 2D projections $\widetilde{\mathbf{m}}^i$, each observation provides two equations as in Eqn. 1.9, one per coordinate. The problem now is to find a linear error criterion. [Abdel-Aziz and Karara, 1971] propose to use the following error criterion:

$$\begin{aligned} \epsilon_x^i &= p_{11}M_x^i + p_{12}M_y^i + p_{13}M_z^i + p_{14} - \widetilde{m}_x^i(p_{31}M_x^i + p_{32}M_y^i + p_{33}M_z^i + p_{34}) \\ \epsilon_y^i &= p_{21}M_x^i + p_{22}M_y^i + p_{23}M_z^i + p_{24} - \widetilde{m}_y^i(p_{31}M_x^i + p_{32}M_y^i + p_{33}M_z^i + p_{34}). \end{aligned} \tag{1.10}$$

Although this error criterion has not a geometric meaning (it cannot be seen as a 2D geometric measure), the equations are linear and so they can written as a linear system of the type $A\mathbf{p} = 0$:

$$\begin{bmatrix} & & & & & & & \vdots & & & & \\ M_x^i & M_y^i & M_z^i & 1 & 0 & 0 & 0 & 0 & -\widetilde{m}_x^i M_x^i & -\widetilde{m}_x^i M_y^i & -\widetilde{m}_x^i M_z^i & -\widetilde{m}_x^i \\ 0 & 0 & 0 & 0 & M_x^i & M_y^i & M_z^i & 1 & -\widetilde{m}_y^i M_x^i & -\widetilde{m}_y^i M_y^i & -\widetilde{m}_y^i M_z^i & -\widetilde{m}_y^i \\ & & & & & & & \vdots & & & & \end{bmatrix} \cdot \begin{bmatrix} p_{11} \\ p_{12} \\ \cdot \\ \cdot \\ \cdot \\ p_{33} \\ p_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \\ 0 \end{bmatrix} \tag{1.11}$$

This system can be solved using the SVD algorithm (Singular Value Decomposition), which imposes the additional constraint $||\mathbf{p}|| = 1$.

The last step concerns the passage from the recovered vector of parameters to the projection matrix. Since the projection matrix is defined up to a scale factor, we need an additional constraint in order to recover the Euclidean camera parameters. Different constraints have been proposed by different authors in the literature: the authors of [Abdel-Aziz and Karara, 1971] use the constraint $p_{34} = 1$, [Faugeras and Toscani, 1986] propose the most robust constraint $||(p_{31}, p_{32}, p_{33})^t|| = 1$, and finally [Melen, 1994] proposes a QR matrix decomposition.

The DLT algorithm needs a minimum of 6 points (12 equations) in order to solve the 11 dof of the projection matrix. Starting from this basic method, different authors have proposed algorithms using a lower number of points at the cost of not recovering all the parameters. Typically, if we only want to estimate the camera pose, only 3 points suffice [Haralick et al., 1991], but the solution is not unique and additional information is needed. [Quan and Lan, 1998] propose a linear algorithm using 4 or more points that provides a unique solution. Recovering some of the internal parameters is also possible with fewer than 6 points. [Triggs, 1999] proposes a 4-point algorithm to recover the pose and the focal length, and a 5-point algorithm to recover the pose, the focal length and the principal point.

Non-Linear Methods

The main advantages of non-linear methods against linear ones are that they have no restrictions regarding the complexity of the camera model, the cost function or the rotation parameterization. The main disadvantage is that, since non-linear methods are iterative methods, computation time is higher and they need an initial solution that has to be close enough to the real solution to avoid local minima. This is why linear methods are still useful in order to be used as an initial solution for the non-linear problem.

The first application of a non-linear calibration method for Computer Vision is owed to [Tsai, 1987]. He addresses all the previous problems of linear methods:

- separation of the intrinsic and extrinsic parameters according to the Euclidean model,
- parameterization of the rotation matrix (Euler angles),
- minimization of a true 2D distance,
- inclusion of radial and tangential distortion terms due to imperfect lenses.

The collinearity equations become non-linear: $m_x = F(\Phi, \mathbf{M})$, $m_y = G(\Phi, \mathbf{M})$, where F and G are non-linear functions and Φ is the vector of parameters: rotation Euler angles, translation, intrinsic parameters and lenses distortion parameters. The cost criterion is simply defined as the SSE (Sum of Squared error): $\sum_i (m_x^i - \widetilde{m}_x^i)^2 + (m_y^i - \widetilde{m}_y^i)^2$. Although the problem is non-linear, we can easily compute the analytic derivative equations, which allows using derivative-based optimization techniques such as the Levenberg-Marquardt algorithm.

Starting from Tsai's method, non-linear camera calibration has been extensively treated in the literature. Among the different extensions, there is one that has special interest: the recovery of the 3D structure at the same time as the camera parameters. This method was first developed in the photogrammetry community [Brown, 1976]. It consists of refining at the same time the 3D structure and the camera parameters. Basically it introduces the calibration pattern $\mathbf{M}^i$ inside the optimization algorithm in order to improve the accuracy even more. In this case, the parameter vector Φ also contains the 3D coordinates of the calibration pattern. The only problem with this algorithm is that, since we increase the number of parameters to optimize, we need to be sure there is enough redundancy in the system to be able to recover all the parameters. Redundancy is typically obtained by adding multiple views of the same calibration pattern inside the optimization algorithm. If we dispose of a calibration pattern with n points, seen from m different views, and the intrinsic parameters remain constant, the total number of dof will be $5 + 5 + 6m + 3n$, i.e., for a typical model of 5 classic intrinsic parameters, plus 5 parameters for the lens distortion model, plus $6m$ parameters for the m different camera poses and $3n$ parameters describing the calibration pattern geometry. For typical calibration targets of tens of points, 3 images suffice to over-determine the system. An excellent survey of the theory of bundle adjustment can be found in [Triggs et al., 2000], where they discuss different

parameterizations, cost functions and optimization algorithms. Something important to note is that, since we optimize also the geometry of the calibration pattern, we lose the metric information, since the system cannot recover the global scale. This information has to be introduced externally, e.g., with the exact distance between two given points of the calibration pattern.

A good example of a camera pose and calibration algorithm that recovers the 3D structure is the algorithm developed by [Lavest et al., 1998], where they study the calibration accuracy relative to the calibration pattern accuracy. Since they optimize the geometry of the calibration pattern as well, they show that we can obtain a good calibration accuracy with a relatively low accuracy on the construction of the calibration pattern. In fact, the main bottleneck to obtain a good calibration is the accuracy of the 2D feature detector. For a good calibration, typical detection errors need to be smaller than 0.05 pixels, which generally needs the use of an intensity-model-based feature detector.

The method described in [Lavest et al., 1998] can be considered to be between classic calibration methods and auto-calibration methods: it needs a calibration pattern but it is able to optimize it as well. The following section describes a completely different type of calibration methods, where only the uncalibrated images and the rigidity of the scene are required.

1.1.4 Auto-Calibration

Auto-Calibration (or Self-Calibration) is the process of recovering internal camera parameters directly from multiple uncalibrated images. Compared to the conventional calibration methods, where a higher knowledge of the scene is required, auto-calibration methods use only the rigidity of the scene and constraints on the intrinsic or extrinsic parameters, e.g., the intrinsic parameters are unknown but constant for all the views. Although there exist different ways of imposing the auto-calibration constraints, the general approach to auto-calibration is almost always the same:

- obtain the projective camera matrices,
- update the projective camera matrices to Euclidean matrices using auto-calibration constraints.

The projective camera matrices are computed using epipolar geometry from the correspondence of the same features detected in different views, the most common features being points. Updating the camera matrices from projective to Euclidean can be done in a "*direct*" way or in "*stratified*" way. Direct methods pass directly from the projective form to the Euclidean form. Stratified methods first update the projective camera matrices to affine camera matrices (i.e., find the plane at the infinity) and, from affine camera matrices to Euclidean camera matrices. Stratified methods are in general more robust and simplify the passage from the affine stratum to the Euclidean one.

Direct Methods

- **Kruppa equations**. The first auto-calibration method is historically due to [Faugeras et al., 1992], [Maybank and Faugeras, 1992]. They proposed an approach based on the Kruppa equations [Kruppa, 1914] and established the relation between the camera intrinsic parameters and the absolute conic. The Kruppa equations use only epipolar geometry, which has two advantages over other methods: they do not need either to compute the plane at the infinity or to relate all the projective cameras into a single coordinate system. The cons are the robustness and the number of possible solutions, which makes this approach only practical for very few images (3 or 4) since the number of solutions grows very fast with the number of views. Another problem is that there exist special critical motions only for the Kruppa equations [Sturm, 2000]. An interesting point about the Kruppa equations is the fact that, as shown by [Luong and Faugeras, 1997], they are equivalent to the Trivedi constraints [Trivedi, 1988] and the Huang and Faugeras constraints [Huang and Faugeras, 1989].

- QR **decomposition**. Hartley proposed a different technique to self-calibration based on the QR decomposition of the projection matrix [Hartley, 1994]. This solution is more robust than using the Kruppa equations and it can be applied to any number of views.

- **Absolute quadric**. [Triggs, 1997] proposed two algorithms (one linear and one non-linear) to directly estimate the absolute quadric from a set of images (the absolute quadric is the dual of the absolute conic). Similar equations had already been used by [Heyden and Aström, 1996] but without relating them to the absolute quadric concept.

Stratified Methods

Even if the direct methods upgrade from projective to Euclidean in a direct way, the plane at the infinity is still computed either implicitly (e.g. the absolute quadric definition includes the plane at the infinity) or explicitly (see [Hartley, 1994]). The only method that actually does not use at all the plane at the infinity is the Kruppa equations; it manages to eliminate its dependency at the expense of robustness.

Stratified methods, as opposed to direct ones, first recover the plane at the infinity and then find the intrinsic parameters. The idea of separating the computation of the plane at the infinity from the intrinsic parameters appears already in the methods of [Hartley, 1994], [Armstrong et al., 1994] and [Faugeras, 1995]. One of the best results on stratified methods is due to [Pollefeys and Van-Gool, 1997], who developed a complete stratified auto-calibration approach based on the recovery of the plane at the infinity using the"*modulus constraint*" [Pollefeys et al., 1996].

1.2 Related Work

We present a new approach to silhouette-based calibration that deals with the notion of silhouette coherence. Roughly speaking, we exploit the rigidity property of 3D objects to impose the only possible constraint between the silhouettes of the same 3D object: they must be coherent, meaning that it must exist a 3D object that might have generated these silhouettes. Bottino and Laurentini study the same problem in [Bottino and Laurentini, 2003] but with a different point of view. They provide the notion of *silhouette compatibility* to state if a given set of silhouettes of the same 3D object are possible or not. They give some rules to state if a set of silhouettes is compatible or not, but only for the special case of orthographic cameras and without providing a means to compute their amount of *incompatibility*.

In his PhD thesis [Cheung, 2003], Cheung uses the term of *consistent alignment* for the registration of two visual hulls. However, he discards using it in an optimization algorithm because he considers it too computational expensive to be used in practice.

We describe next a criterion of *silhouette coherence* that allows us to measure a degree of coherence and thus offers us the possibility of using optimization methods to recover some of the camera parameters. The silhouette coherence approach is related to three different kinds of techniques: i) camera calibration and motion estimation, ii) texture registration between a set of images and a 3D model, and iii) visual hull computation and registration (see Eqn. 1.13 for the mathematical definition of the visual hull concept [Laurentini, 1994]).

- A large collection of methods for **camera auto-calibration** and motion estimation exists [Faugeras and Luong, 2001]. They rely on correspondences between the same primitives detected in different images. For the particular case of circular motion, the methods of [Fitzgibbon et al., 1998] and [Jiang et al., 2002] work well when the images contain enough texture to make a robust detection of the primitives. Otherwise, silhouettes can be used instead. For the special case of surfaces of revolution, an auto-calibration method using only silhouettes has been proposed by [Wong et al., 2003]. For general surfaces, silhouettes have been mainly used for camera motion estimation. For general surfaces, silhouettes have been mainly used for camera motion estimation. In [Mendonça et al., 2001], the circular motion of a general object is recovered by considering the symmetry properties of the surface of revolution induced by the rotation of the object. In [Wong and Cipolla, 2001] motion recovery is achieved by using the notion of epipolar tangencies [Rieger, 1986], [Porrill and Pollard, 1991], i.e., points on the silhouette contours that belong to an epipolar line tangent to the silhouette (see Fig. 1.2). More recently, [Furukawa et al., 2004] propose recovering the general motion of an affine camera using also epipolar tangencies. Although these methods give good results, their main drawback is the limited number of epipolar tangencies per pair of images, generally only two: one at the top and one at the bottom of the silhouette. When additional epipolar tangencies are available, the problem is to match them between different views and handle their visibility, as proposed in [Furukawa et al., 2004].

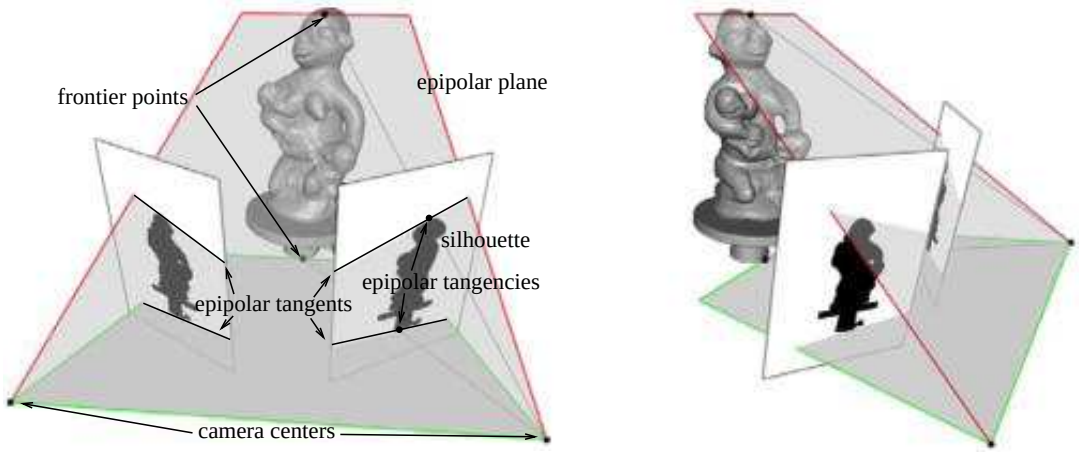


Figure 1.2: *Two views of the two epipolar planes (red and green) that are tangent to the surface. The epipolar tangents correspond to the intersection between the epipolar planes tangent to the surface and the retinal planes. The epipolar tangencies are defined as the tangent points between the silhouettes and the epipolar tangents. By construction, an epipolar tangency is the projection of the corresponding frontier point.*

- Concerning **texture registration**, there exists a number of algorithms that try to register a 3D representation of an object with a set of textured views of the same object using silhouettes. Calibration is accomplished by minimizing the error between the contours of the silhouettes and the contours of the projected object. In [Matsushita and Kanedo, 1999] and [Neugebauer and Klein, 1999] the error is defined as the sum of the distances between a number of sample points on one contour and their nearest points on the other. In [Lensch et al., 2001] a hardware-accelerated method is used to compute the similarity between two silhouettes as the area of their intersection.

- **Visual hull computation and registration**. Visual hull computation is a very active area since it is one of the fastest and most robust ways of obtaining an initial estimation of a 3D object. It can be precise enough for real time rendering applications such as the one proposed by [Li et al., 2003] or used as an initial estimation for further processing in 3D reconstruction algorithms such as the one that we have developed and which is described in chapter 2. When several unregistered silhouette sequences of the same object are available, the problem of registering the corresponding visual hulls appears. Although [Sullivan and Ponce, 1998] do not explicitly propose the registration of two different sequences of silhouettes, they propose a way of estimating the pose of a 3D model relative to a set of silhouettes, so silhouette registration can in fact be achieved by first reconstructing a 3D model using one of the sequences followed by a pose estimation relative to the second sequence of silhouettes. In [Park and Subbarao, 2002] two different reconstructions are obtained from each silhouette sequence. Then, the two 3D reconstructed models are matched using tangent planes and stability constraints. The authors of [Cheung et al., 2003] also start with two different

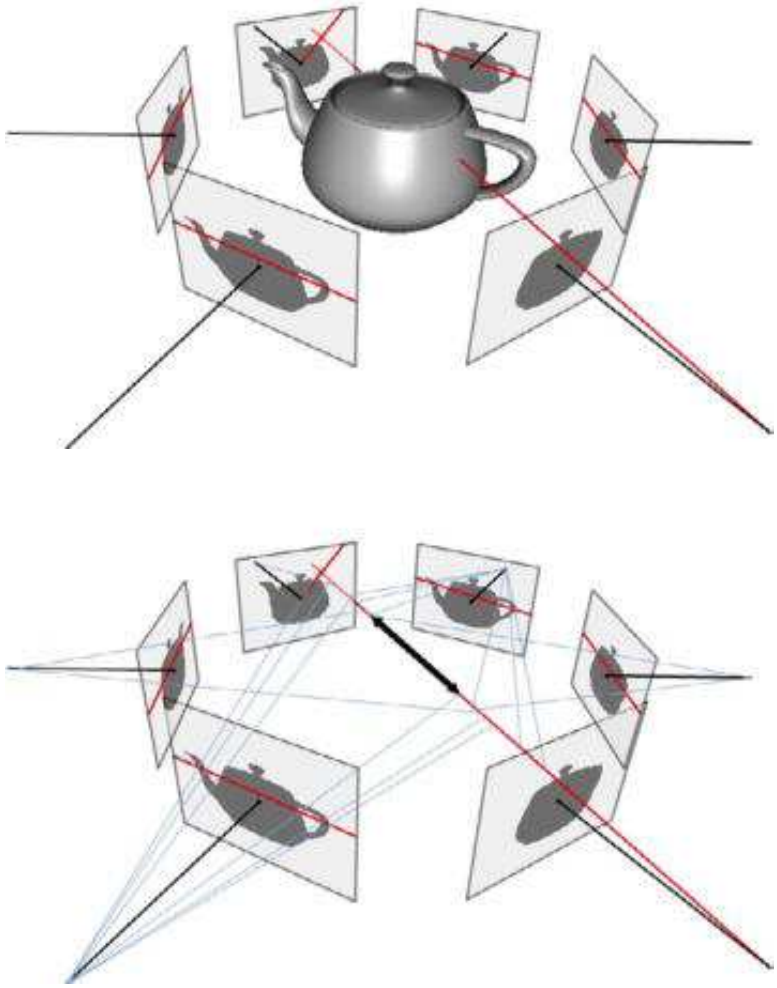
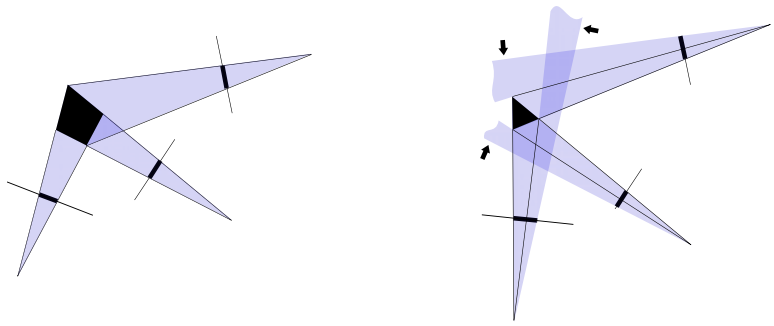


Figure 1.3: *Computation of the 3D optic ray intersection by back projection of the silhouette intervals. Top: intersection of the optic ray with the real object. Bottom: intersection of an optic ray with the visual hull using silhouette intervals.*

reconstructions. They use them to speed up a stereo matching algorithm to locate 3D surface points, which are then used to find the rigid motion between the two sequences.

The proposed silhouette coherence criterion is inspired from the texture registration approach proposed by [Lensch et al., 2001]. But the main difference is that we do not need a known 3D model corresponding to the real object. The 3D model is *implicitly* reconstructed from the silhouettes at the same time as the camera calibration by a visual hull method. Specifically, the use of the technique described by [Matusik et al., 2000] allows all the computations to be done in the image domain, which overcomes the need for a 3D representation as in [Sullivan and Ponce, 1998], [Park and Subbarao, 2002] or [Cheung et al., 2003].



a b

Figure 1.4: *2D examples of different silhouette coherences. The reconstructed visual hulls are the black polygons. a) Perfectly coherent silhouette set. b) Set of 3 silhouettes with low silhouette coherence. The arrows indicate incoherent optic ray pencils.*

1.3 A Measure of Silhouette Coherence

Given a set of silhouettes of the same 3D object taken from different points of view, and the corresponding set of camera projection matrices, we would like to measure the coherence of both the silhouette segmentation and the camera projection matrices. The only information contained in a silhouette is a classification into two categories of all the optic rays that go through the optic center of the associated view: those that intersect the object and those that do not, depending if the pixel that defines the optic ray belongs to the silhouette or not. Let us consider an optic ray defined by a silhouette pixel and thus intersecting the object. If the silhouettes and the camera projection matrices are perfect, it is obvious that the projection of the optic ray into any other view *must* intersect the corresponding silhouette. The back projection of this intersection onto the 3D optic ray defines one or several 3D intervals where we know that the optic ray intersects the real object surface (see Fig.1.3). In the case of only two views, the corresponding silhouettes will not be coherent if there exists at least *one* optic ray classified as intersecting the object by one of the silhouettes and whose projection does not intersect the other silhouette. In the case of n views, the lack of coherence is defined by the existence of at least one optic ray where the depth intervals defined by the $n-1$ other silhouettes have an empty intersection. This lack of coherence can be measured simply by counting how many optic rays in each silhouette are not coherent with the other silhouettes. Two examples of coherent and non-coherent silhouettes in the 2D case are shown in Fig.1.4. The optic ray pencils that are not coherent with the other silhouettes are shown by an arrow in Fig.1.4.b.

A first way of computing a coherence measure is as follows:

- compute the reconstructed visual hull defined by the silhouettes,
- project the reconstructed visual hull back into the cameras, and
- compare the reconstructed visual hull silhouettes to the original ones.

In the situation of ideal data, i.e., perfect segmentation and perfect projection matrices, the reconstructed visual hull silhouettes and the original silhouettes will be exactly the same (see Fig. 1.4.a). With real data, both the silhouettes and the projection matrices will not be perfect. As a consequence, the original silhouettes and the reconstructed visual hull silhouettes will not be the same, the reconstructed visual hull silhouettes being **always contained** in the original ones. This can be explained mathematically in the following way:

Let S_i be the i^{th} image silhouette and P_i the corresponding camera projection matrix. We can define the cone C_i generated by the silhouette S_i as the set of 3D points $\mathbf{M}$ that verifies:

$$C_i = \{\mathbf{M} \in \mathbb{R}^3 \ : \ P_i\mathbf{M} \in S_i\}. \tag{1.12}$$

The reconstructed visual hull $\mathcal{V}$ defined by the silhouette set S_i, $i = 1, \ldots, n$ can be written as the following cone intersection:

$$\mathcal{V} = \bigcap_{i=1,\ldots,n} C_i = \{\mathbf{M} \in \mathbb{R}^3 \ : \ P_i\mathbf{M} \in S_i \ \forall i\}. \tag{1.13}$$

The silhouette of $\mathcal{V}$ into the i^{th} image, noted $S_i^{\mathcal{V}}$, is defined as the set of 2D image points $\mathbf{m}$ such as:

$$S_i^{\mathcal{V}} = \{\mathbf{m} = P_i\mathbf{M} \ : \ \mathbf{M} \in \bigcap_{i=1,\ldots,n} C_i\}. \tag{1.14}$$

According to Eqn. 1.12, we can separate the contribution of silhouette S_i to $S_i^{\mathcal{V}}$ as follows:

$$S_i^{\mathcal{V}} = \{\mathbf{m} = P_i\mathbf{M} \ : \ \mathbf{M} \in \bigcap_{j \neq i} C_j\} \cap S_i. \tag{1.15}$$

Hence, by construction $S_i^{\mathcal{V}} \subseteq S_i \ \forall i$. If the silhouettes and the projection matrices are perfect, then $S_i^{\mathcal{V}} = S_i \ \forall i$.

The next question is how to measure the similarity $\mathcal{C}$ between a given silhouette S_i and its corresponding reconstructed visual hull silhouette $S_i^{\mathcal{V}}$. A first quick answer would be to use the ratio of areas between the two silhouettes:

$$\mathcal{C}(S_i, S_i^{\mathcal{V}}) = \frac{\int S_i^{\mathcal{V}}}{\int S_i} \in [0, 1]. \tag{1.16}$$

However, this measure has two important drawbacks: the dynamic range and the implementation cost. The dynamic range is a problem because the silhouette areas can be very large for big images and, for small differences between silhouettes, the dynamic range of this measure will be very small and not accurate enough for the applications we are considering. As we will see in the implementation of the algorithm, there is also a performance reason for this choice since we will need to discretize the evaluation of the measure, and the computation time will be proportional to the size of the silhouette area to evaluate.

To solve these two important issues we propose using the ratio of the silhouette contour lengths rather than the ratio of the silhouette areas. Let ∂_i denote the contour

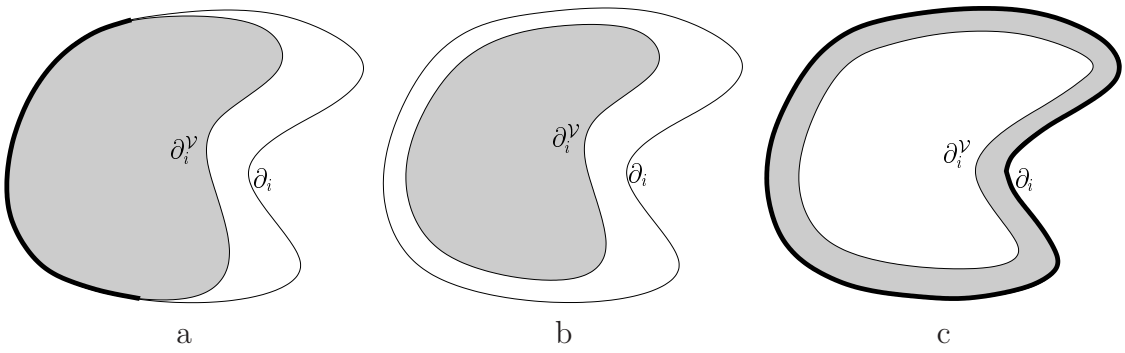


Figure 1.5: *Different scenarios for silhouette comparison. The visual hull silhouette is shown in gray. The common contour $\partial_i^{\mathcal{V}} \cap \partial_i$ is drawn with a thick stroke. (a) Normal scenario. (b) Special case where the coherence computed using only contours is zero whereas the coherence using areas is much greater. (c) Special case where the coherence using contours is 100% whereas the coherence using areas is very low.*

of the original silhouette S_i, and $\partial_i^{\mathcal{V}}$ the contour of the reconstructed visual hull silhouette $S_i^{\mathcal{V}}$. A measure $\mathcal{C}_{sc}$ of coherence between these two silhouettes can be defined as the ratio between the length of their common contours $\partial_i^{\mathcal{V}} \cap \partial_i$ and the total length of ∂_i:

$$\mathcal{C}_{sc}(S_i, S_i^{\mathcal{V}}) = \frac{\int(\partial_i^{\mathcal{V}} \cap \partial_i)}{\int \partial_i} \in [0, 1]. \tag{1.17}$$

An interesting question about using the contours rather than areas is the fact that both measures might greatly differ for some special cases as shown in Fig.1.5.b and Fig.1.5.c. Using the contour-based measure will penalize scenarios such as in Fig.1.5.b while encouraging scenarios such as in Fig.1.5.c. But it happens that case b is much more common than case c for the problem of silhouette coherence. In fact, if none of the silhouettes has interior holes, case c is impossible by construction of the visual hull. As discussed in section 1.6, we use only the exterior contour of the silhouettes, which implies that case c will never be encountered in practice. However, case b can be easily reproduced by simply having an erroneous smaller focal length for one of the views. The weakness of using contours instead of areas in case Fig.1.5.b can be avoided by using a multi-resolution approach (see Section 1.6.2) that allows a better handling of the scenario of Fig.1.5.b.

The measure $\mathcal{C}_{sc}(S_i, S_i^{\mathcal{V}})$ evaluates the *coherence* between the silhouette S_i and all the other silhouettes $S_{j,j\neq i}$ that contributed to the reconstructed visual hull. To compute the total coherence between all the silhouettes, we can just compute the mean coherence of each silhouette with the $n-1$ other silhouettes:

$$\mathcal{C}_{sc}(S_1, \ldots, S_n) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{C}_{sc}(S_i, S_i^{\mathcal{V}}) \tag{1.18}$$

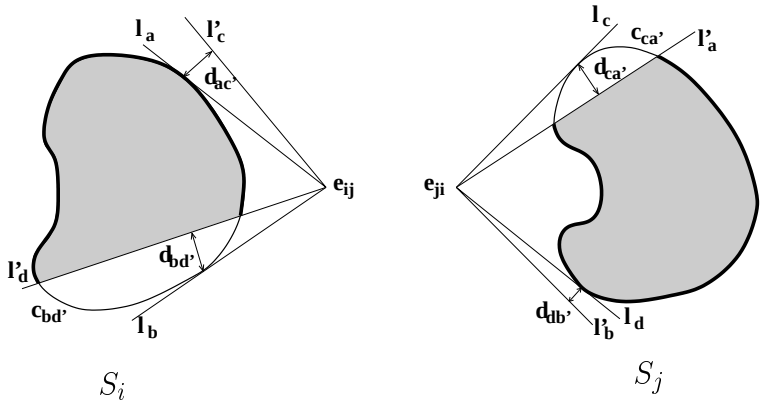


Figure 1.6: *Epipolar tangency and silhouette coherence criteria for a pair of silhouettes.*

1.4 Silhouette Coherence: an Extension to the Epipolar Tangency Criterion

In the case of a sequence of cameras under circular motion with constant intrinsic parameters, [Wong and Cipolla, 2001] demonstrate the feasibility of recovering the motion using epipolar tangencies. We show below that the proposed silhouette coherence criterion is a more general approach that extends the criterion of epipolar tangencies.

For a given pair of views, as shown in Fig.1.6, the epipolar tangency approach minimizes the square distance between epipolar tangents of one view (l_a and l_b in view i, l_c and l_d in view j) and the transformed epipolar tangents of the other view (l'_c and l'_d in view i, l'_a and l'_b in view j). That is, it minimizes $\mathcal{C}_{et} = d^2_{ac'} + d^2_{bd'} + d^2_{ca'} + d^2_{db'}$. For the same pair of silhouettes, the optimization of the coherence criterion corresponds to minimizing the length of the contours $c_{ca'}$ and $d_{bd'}$. So we can see that, except for pathological configurations, both criteria try to minimize the sectors defined by the epipolar tangents in one view and their corresponding epipolar tangents in the other view. This implies that, if we optimize our coherence criterion on a set of silhouettes by pairs, we get the same behavior as [Wong and Cipolla, 2001], and thus we are also able to recover the camera motion using the silhouette coherence. Furthermore, the epipolar tangency criterion can actually also recover some intrinsic parameters such as the focal length, even if [Wong and Cipolla, 2001] only use this criterion to estimate the camera motion.

When using the proposed silhouette coherence, silhouettes are not just taken by pairs but all at the same time. This makes that the information we exploit is not only on the outer epipolar tangencies but all over the silhouette contour. As a result, even if we dispose of silhouettes where the outer epipolar tangencies are not available, the silhouette coherence criterion is still valid. We present an example in Section 3.4.1 where we do not dispose of the bottom of the silhouettes but for which we are still able to estimate the camera parameters with very good accuracy. This scenario is quite common in practice, since sometimes it is very difficult to correctly separate the object from its support or from the turntable.

Although we have the theoretical possibility of recovering some of the intrinsic parameters, it is well known that the circular motion is a critical motion for self-calibration [Sturm, 1997a, Sturm, 1997b]. The question is which parameters can be theoretically recovered and if in practice they affect the coherence criterion enough to be retrieved by optimization. This point is addressed in the next Section.

1.5 Validity of the Coherence Measure

To be able to efficiently exploit the coherence measure we need first to know its application limits. As we stated above, if we have perfect silhouettes and perfect camera matrices, then $\mathcal{C}_{sc}(S_1, \ldots, S_n) = 1$. This never happens in practice. Let us assume that the silhouettes are perfectly segmented. We can maximize the silhouette coherence by adjusting the camera parameters in order to reduce mismatches between silhouettes. But maximizing coherence between silhouettes does not mean finding the right camera parameters [Cheung, 2003]. This depends on:

- **the object shape**. The worst case corresponds to a sphere turning around its center. In this particular case all the silhouettes are the same and therefore there is no unique solution to the problem of silhouette coherence maximization because of the sphere symmetry. Thus, in general, we can not guarantee the uniqueness of the solution.

- **the number of silhouettes**. If we use only a small number of silhouettes, then the coherence criterion is less accurate and may be maximized for a large class of solutions. However, if we take a sufficient number of pictures, real objects are generally asymmetric enough to guarantee a unique solution as will be shown in the practical examples.

- **the interdependence between parameters of the camera**. Some parameters can affect the coherence criterion in a similar way such that it is impossible to distinguish between them during the coherence maximization.

So the coherence criterion is not the ultimate criterion for recovering all the parameters in a global optimization, but it can work quite well for some particular scenarios where the number of parameters to recover is not very high and silhouettes provide enough information to optimize them.

Recovering the camera principal point $(u_0, v_0)^t$ and translation vector $\mathbf{t} = (t_x, t_y, t_z)^t$ under circular motion at the same time is a very difficult task. This is due to the fact that the translation introduces in the projection equation a pixel offset similar to the principal point. If we look at the projection equation for circular motion of a 3D point $\mathbf{M}$ into a pixel point $\mathbf{m}$ (see Section 1.6.3 for a discussion of the different parameterizations), we have:

$$m \simeq K[R(\theta)|\mathbf{t}] \begin{pmatrix} \mathbf{M} \\ 1 \end{pmatrix}, \tag{1.19}$$

where K is the intrinsic matrix, $\mathbf{t}$ the translation vector and $R(\theta)$ the rotation matrix around the rotation axis, θ being the angle of rotation. The only parameter that changes between two cameras under circular motion is the angle of rotation θ. Let $R(\theta) = [\mathbf{r}_1\mathbf{r}_2\mathbf{r}_3]$, then the projective equation can be written in the following way:

$$\begin{aligned} m_x &= f\frac{\mathbf{r}_1^t\mathbf{M} + t_x}{\mathbf{r}_3^t\mathbf{M} + t_z} + u_0 = f\frac{\mathbf{r}_1^t\mathbf{M} + t_x}{\mathbf{r}_3^t\Delta\mathbf{M} + \mathbf{r}_3^t\overline{\mathbf{M}} + t_z} + u_0, \\ m_y &= f\frac{\mathbf{r}_2^t\mathbf{M} + t_y}{\mathbf{r}_3^t\mathbf{M} + t_z} + v_0 = f\frac{\mathbf{r}_2^t\mathbf{M} + t_y}{\mathbf{r}_3^t\Delta\mathbf{M} + \mathbf{r}_3^t\overline{\mathbf{M}} + t_z} + v_0, \end{aligned}$$

where $d = \mathbf{r}_3^t\overline{\mathbf{M}} + t_z$ is the average depth of the scene along the line of sight and $\Delta d = \mathbf{r}_3^t\Delta\mathbf{M}$ is the deviation from the average depth for a particular 3D point $\mathbf{M}$, i.e. $\mathbf{M} = \Delta\mathbf{M} + \overline{\mathbf{M}}$ for any 3D point $\mathbf{M}$. If we are projecting a 3D object, $\mathbf{r}_3^t\overline{\mathbf{M}} + t_z$ will correspond to the average depth of the 3D object relative to the camera. After developing the Taylor series:

$$\frac{1}{\Delta d + d} = \frac{1}{d}\left(1 - \frac{\Delta d}{d} + \left(\frac{\Delta d}{d}\right)^2 - \cdots\right),$$

we obtain:

$$\begin{aligned} m_x &= f\frac{\mathbf{r}_1^t\mathbf{M}}{d} + f\frac{t_x}{d} + u_0 + \mathcal{O}(\frac{\Delta d}{d}), \\ m_y &= f\frac{\mathbf{r}_2^t\mathbf{M}}{d} + f\frac{t_y}{d} + v_0 + \mathcal{O}(\frac{\Delta d}{d}). \end{aligned}$$

We can notice that if Δd is small compared to d, then the effect of the translation $\mathbf{t}$ can be interpreted as a pixel offset of value $(f_x\frac{t_x}{d}, f_y\frac{t_y}{d})^t$ up to an error $\mathcal{O}(\frac{\Delta d}{d})$. If instead of projecting the entire object, we project only the contour generators (to construct silhouettes), then the condition to hold the above statement is more easily achieved. It suffices that the depth variation along the contour generators remains small compared to the distance between the object and the camera, which happens rather commonly in practice. This explains why the translation and the principal point are very difficult to recover at the same time using silhouettes and why it is so difficult to separate their contribution in the silhouette coherence measure.

The focal length is a different case. Its main problem is that very large variations of the focal length can produce only small variations of the silhouette coherence. Even with an infinite focal length (orthographic projection) high values of coherence can still be obtained (see Fig. 1.7 left). For this reason, it is recommendable to initialize the focal length with a lower bound of its expected value or to use another parameterization such as the logarithm of the focal length (Fig. 1.7 center) or the field of view (fov) (Fig. 1.7 right).

1.6 Coherence Criterion Implementation

The silhouette coherence criterion $\mathcal{C}$ being defined, a first implementation, and probably the simplest, would be the following:

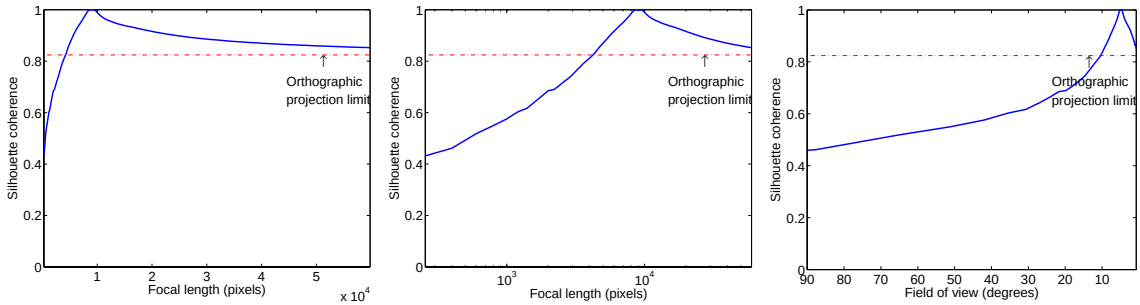


Figure 1.7: *Silhouette coherence variation with focal length. Discontinuous line represents the coherence for orthographic projection: focal length=∞, (fov=0). Left: plot of silhouette coherence as a function of the focal length in pixels. Center: same plot using a logarithmic axis. Right: plot of the silhouette coherence as a function of the field of view:* $fov = 2 \cdot atan(size_{retina}/(2 \cdot focal))$*, with* $size_{retina} = 768$ *pixels.*

- for a given set of cameras and silhouettes, compute the reconstructed 3D visual hull by using any of the multiple existing techniques,
- project the reconstructed visual hull into the cameras,
- compute the coherence criterion.

This approach has two drawbacks: computation time and volume sampling. The former may be a problem since it may take several minutes to compute a 3D visual hull, and the latter because constructing a 3D visual hull usually needs a discrete 3D representation, such as a volume grid or an octree. If we want a highly accurate visual hull, we need a high resolution 3D model of the visual hull, which is computationally very expensive and not appropriate for an iterative optimization process. In addition, we are not interested in a 3D representation in itself but in comparing 2D views of it with the original silhouettes. Therefore, it seems a waste of time to reconstruct the visual hull completely when only some views of it are required. The imaged-based visual hull (IBVH) technique [Matusik et al., 2000] does not compute a 3D representation of the reconstructed visual hull but only 2D views of it. The key idea is as follows: for any 2D point in an image, we can compute the intersection between its corresponding optic ray and the visual hull by a ray-casting approach. According to the definition of the visual hull, this is equivalent to (see Fig. 1.3):

- projecting the optic ray into each silhouette,
- computing the 2D intersection intervals between the projected ray and each silhouette,
- back projecting all the 2D intervals onto the original 3D optic ray,
- computing the intersection on the 3D optic ray of all the intervals of all the silhouettes.

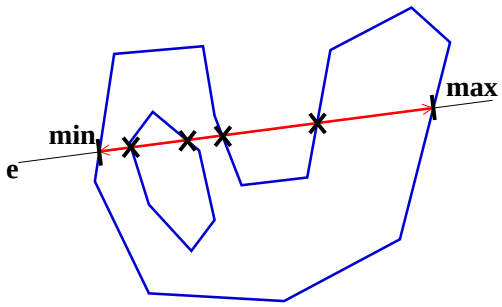


Figure 1.8: *Simplified intersection algorithm.*

For any optic ray, we have a set of remaining depth intervals, possibly empty, which represent the intersection between the optic rays and the implicit visual hull.

To compute the 2D intersection between the projected ray and a silhouette, we use an interval intersection algorithm. With the approach of interval intersection, the silhouette contours are coded as a closed sequence of segments. For a given projected ray, we compute the intersection of the half line with the sequence of segments, which gives us the 2D intervals directly.

In order to accelerate computations, we use a simplified version of the algorithm described in [Matusik et al., 2000]. The first simplification is that we do not take into account contours inside the silhouettes, i.e., we only consider genus-0 silhouettes. Furthermore, we do not compute all the intersecting intervals for a given optic ray. We just compute the min and max of the interval intersection with each silhouette (see Fig. 1.8). This is a conservative approximation of the real coherence, i.e., the coherence score that we obtain by storing only the min and max depth is always equal to or greater than the real one. However, in practice, the deviation from the coherence computed with all the intervals is small.

1.6.1 Interval Intersection Algorithm

Let us consider a closed polygonal silhouette contour P defined by a sequence of N points $\mathbf{p}_{i=\{0,\ldots,N-1\}}$ and an epipolar line l (half-line having its origin at the epipole e). The goal is to compute the intersection of the epipolar line with the polygonal contour as a set of intervals, possibly empty. The greedy algorithm consists of testing the intersection against each edge of the polygon $[\mathbf{p}_i, \mathbf{p}_{i+1}]$, with $i = \{0, \ldots, N-1\}$, assuming $\mathbf{p}_N = \mathbf{p}_0$. The resulting intersection points are then sorted along their distance to the epipole e. This allows recovering the ordered list of intersections which define the intersection intervals. Since the polygon is closed, the number of intersections will be odd if the epipole e is outside the polygon and even if it is inside the polygon, excluding the special tangent case. This algorithm runs in linear time $\mathcal{O}(N)$. If we have M epipolar lines to intersect with the same contour P, the complexity of the problem is $\mathcal{O}(MN)$, which can be very large for big values of M and N. [Matusik et al., 2000] propose breaking this complexity using epipolar geometry constraints. They exploit, in particular, the fact that all the epipolar lines have the same origin, i.e., the epipole e. They propose *sorting* the contour points $\mathbf{p}_i$ according to their slope

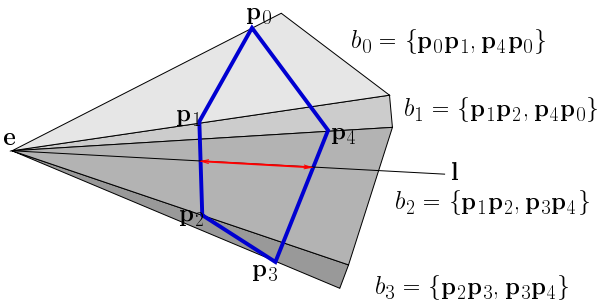


Figure 1.9: *Polygon decomposition into bins for a given epipole e.*

relative to the epipole e. This sorting divides the contour into $N-1$ *bins*. They store inside each bin all the edges that traverse it and sort them along their distance to the epipole. In the example of figure 1.9, we have a contour of $N = 5$ points. Given an epipole e, the contour can be decomposed into 4 different bins $b_{i=\{0,1,2,3\}}$. Now, if we wanted to compute all the interval intersections as in [Matusik et al., 2000], we would need to attach to each bin b_i the ordered list of segments $[\mathbf{p}_j, \mathbf{p}_{j+1}]$ that traverse it. However, because we only need the min/max intervals (see Fig.1.8), we actually just need to attach the closest and farthest segments from the epipole that traverse the bin. For a given epipolar line l, we compute its slope and recover the bin to which the epipolar line belongs to (b_2 in the example of Fig. 1.9). The last operation is the computation of the intersection between the line and the edges contained in that bin. The crucial point to accelerate this computation is to sort the bins. The bin-retrieval operation can then be done only in $\mathcal{O}(\log(N))$. Moreover, if epipolar lines are generated in an ordered way, we can *cache* the last bin-retrieval operation and linearly search the next bin starting from the old one, which requires in general a smaller number of operations than $\log(N)$. As a consequence, the algorithm is basically running in $\mathcal{O}(M\log(N))$ plus the additional overhead due to the bin creation. The bin creation computation time corresponds to two sorting operations over N elements, one to create the bins (slope sorting) and another to sort the list of segments that traverse it (depth sorting). If we assume the complexity of the sorting algorithm to be $N\log(N)$[1] , and the number of sampling points per silhouette equal to the number of points defining the silhouette ($M \approx N$), the total complexity of the algorithm will be $\mathcal{O}(N\log(N))$. If we dispose of n silhouettes, the complexity of computing the coherence between a silhouette and its corresponding reconstructed visual hull silhouette is $\mathcal{O}(nN\log(N))$.

The pseudo code of the coherence algorithm $\mathcal{C}_{sc}(S_i, S_i^{\mathcal{V}})$ between a silhouette S_i and its corresponding reconstructed visual hull silhouette $S_i^{\mathcal{V}}$ is as follows:

[1]Radix Sort or Byte Sort algorithms have a complexity of only $\mathcal{O}((k+1)N)$ with k the number of radices of the input values (e.g. k=4 for floats and k=8 for doubles). Therefore, when working with floats, radix sort has a better performance when $N > 32$ (excluding the sorting overhead).

pseudo code of the silhouette coherence algorithm

```
1  FLOAT Coherence (SILHOUETTE silRef,SILLIST silList)
2  INTEGER emptySamples = 0
3  INTEGER totalSamples = 0
4  For each VEC2D p in Contours(silRef)
5     INTERVAL interval = [0,inf]
6     VEC3D ray = InverseProjection(silRef,p)
7     For each SILHOUETTE sil in silList, sil ≠ silRef
8        VEC2D epipole = Projection(sil,CameraOrigin(silRef))
9        VEC2D direction = Projection(sil,ray)
10       INTERVAL int2D = Intersect(sil,epipole,direction)
11       INTERVAL int3D = InverseProjection(sil,int2D)
12       interval = Intersect(interval,int3D)
13       If interval = void, then emptySamples++; break
14    end
15    totalSamples++
16 end
17 return (totalSamples-emptySamples)/totalSamples
```

1.6.2 δ-Offset Silhouette Contour Approach

In the two last subsections we have explained how to compute the silhouette coherence on a given set of 2D points. In the formulation, these 2D points do not need to be actually on a grid (like pixels). However, the discrete nature of the algorithm appears in two ways: i) we need a finite set of 2D points where to compute the coherence, ii) the silhouettes actually need to be coded with a finite number of segments in order to benefit from the low complexity of the algorithm described in [Matusik et al., 2000]. Silhouettes can be obtained by any segmentation method and delivered in various representations such as images or closed spline curves. However, if we want to be fast, in the end, we need to convert them into polygons. Concerning the 2D points where to sample the coherence criterion, we recall that we want to test the coherence only along the contours of the silhouette, as stated in Section 1.3. Since all the silhouettes are defined as polygons, a first solution would be to select the 2D sampling points equally spaced along each silhouette polygon. But this is not such a good idea. The main reason is that the resulting coherence measure is very noisy. The depth of the intervals on the silhouette contours is small and the floating point computation error can be enough to change a valid interval into an empty interval. A more interesting way to generate the sampling points is to choose them also equally spaced but at a given distance δ from the silhouette contour (see Fig. 1.10).

Sampling at a given distance δ from the contours has the advantage of making the coherence measure more robust to silhouette segmentation noise. Although the distance δ is limited by the thickness of the silhouette itself (see Fig. 1.10 right), this is not a problem in practice because the structures we encounter in real silhouettes have a width of at least a dozen pixels, which is enough for the δ values we consider. Since silhouettes are usually issued from an image segmentation algorithm, we can

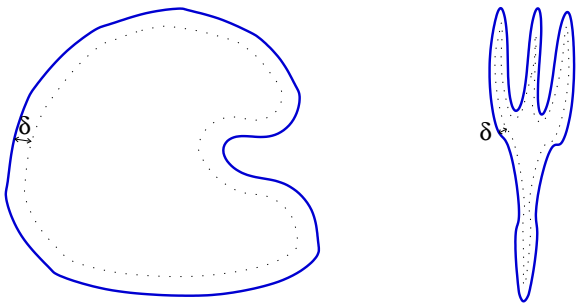


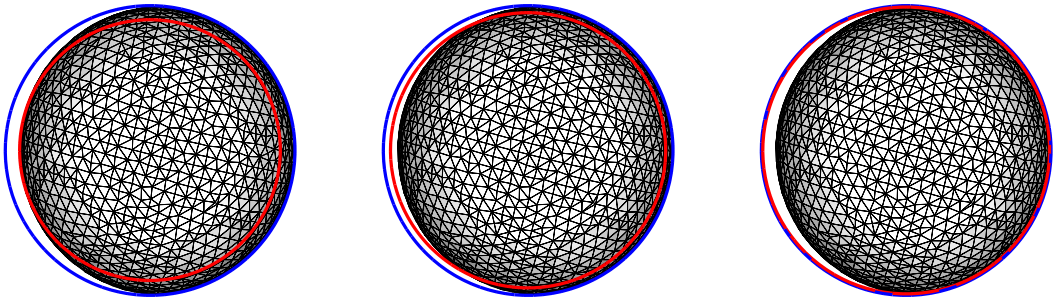
Figure 1.10: *Contour sampling for coherence computation.*

expect to have a segmentation precision of about 0.5 pixels, which is a good δ value for an accurate coherence measure. Moreover, we can select δ to take into account the noise of the silhouette segmentation algorithm. The larger δ is, the more robust the algorithm is against bad segmentation. But robustness is obtained at the expense of accuracy. For a given δ, the silhouette coherence will not distinguish between the original silhouettes and the reconstructed visual hull silhouettes that have an error smaller than δ (see Fig. 1.11).

In order to improve the convergence properties of an optimization algorithm using the coherence measure, we can link together a **multi-resolution algorithm** with a **decreasing δ approach**. The idea is to optimize the coherence measure in a hierarchical way. Instead of using the original silhouettes, we start with a subsampled version of the silhouettes. Upon convergence, we move forward to the next higher level of resolution and we iterate until we use the original silhouettes. Using different levels of resolution allows an automatic scaling of the three parameters that define how the coherence measure is computed: number of silhouette segments, number of sampling points and distance δ between silhouette segments and sampling points. If we consider 4 levels of resolution, with a scale factor of 2 between consecutive levels, the first level will use silhouettes subsampled by a factor of 8. Next levels will use a factor of 4, 2 and 1 respectively. In practice, 2 or 3 different resolution levels suffice. This way of proceeding has two advantages over using the original silhouettes without subsampling:

- **Better convergence properties**. Using different levels of resolution allows using different δ distances and filtering the original silhouettes at the same time. It smoothes the coherence measure and helps to avoid possible local minima.

- **Faster computation time**. Reducing the resolution of the silhouettes by a factor of 2 reduces also by a factor of 2 the number of silhouette segments N and sampling points M. Since the computation time is $\mathcal{O}(M \log(N))$, each level of resolution improves the computation time by a factor a little bit better than 2.



coherence of 100% coherence of 70% coherence of 50%

Figure 1.11: *Comparison of the coherence measure for different δ values. The original silhouette is shown in blue. We show at the same time the projection of the reconstructed visual hull in black. The sample points are chosen along the red contour for a decreasing δ distance from left to right.*

1.6.3 Circular Motion Parameterization

There exist different parameterizations of the circular motion characterized by the camera rotating around a fixed axis (see Fig. 1.12). The degrees of freedom of this motion are the fixed distance t of the camera to the rotation axis $\mathbf{a}$ (1 dof), the camera rotation R_{cam} (3 dof) and the relative angle ω to the (arbitrary) initial position. If we dispose of k cameras, the total number of extrinsic dof will be $1+3+(k-1)=k+3$ dof. The reason why the translation direction has only 1 dof, and not 2, is that we have the freedom to choose the origin $\mathbf{o}$ of the world coordinate system ($\mathbf{xyz}$) anywhere on the rotation axis. The choice of the origin cancels out one degree of the translation direction.

Axis-based Parameterization

We give here a brief description of the circular motion parameterization used in [Fitzgibbon et al., 1998]. Given any arbitrary initial position of the camera, we can choose the origin of the world coordinate system $\mathbf{o}$ as the projection of the camera center $\mathbf{c}$ onto the rotation axis $\mathbf{a}$, the $\mathbf{y}$ world axis aligned with the rotation axis $\mathbf{a}$ and the $\mathbf{z}$ world axis aligned with the vector $\mathbf{o}-\mathbf{c}$. With this parameterization, the camera pose is computed as:

$$[R_{cam}R_y(\omega)|R_{cam}t\mathbf{z}], \tag{1.20}$$

where

$$R_y(\omega) = \begin{bmatrix} cos(\omega) & 0 & sin(\omega) \\ 0 & 1 & 0 \\ -sin(\omega) & 0 & cos(\omega) \end{bmatrix}. \tag{1.21}$$

The rotation matrix R_{cam} captures both the rotation axis (2 dof) and the translation direction (1 dof). But the separability between the rotation axis parameterization and the translation parameterization is highly desirable since, as we will see in the experiments, the sensibility of the silhouette coherence to the translation is much higher than to the rotation axis.

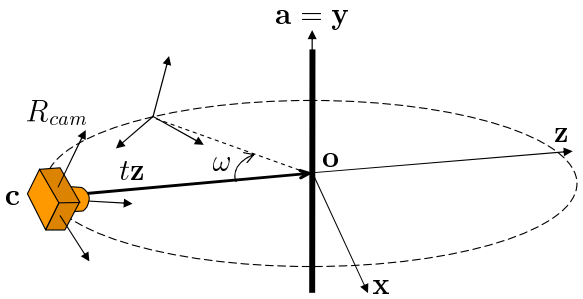


Figure 1.12: *Circular motion configuration.*

Camera-based Parameterization

We propose a different parameterization that has the advantage of separating the rotation axis parameterization and the translation parameterization. Instead of parameterizing the rotation axis and the translation with a single rotation matrix R_{cam}, we choose another type of parameterization where the first camera defines the world axes. The camera pose is described as:

$$[R_{\mathbf{a}}(\omega)|\mathbf{t}], \tag{1.22}$$

where $R_{\mathbf{a}}(\omega)$ is written as a function of ω and $\mathbf{a} = (a_x, a_y, a_z)^t$ as follows:

$$R_{\mathbf{a}}(\omega) = (1-\cos(\omega)) \begin{bmatrix} a_x^2 & a_x a_y & a_x a_z \\ a_x a_y & a_y^2 & a_y a_z \\ a_x a_z & a_y a_z & a_z^2 \end{bmatrix} + \begin{bmatrix} \cos(\omega) & -a_z \sin(\omega) & a_y \sin(\omega) \\ a_z \sin(\omega) & \cos(\omega) & -a_x \sin(\omega) \\ -a_y \sin(\omega) & a_x \sin(\omega) & \cos(\omega) \end{bmatrix}. \tag{1.23}$$

The rotation axis is no longer aligned with the **y** axis and is coded using the spherical coordinates $(\theta_{\mathbf{a}}, \phi_{\mathbf{a}})$:

$$\mathbf{a}(\theta_{\mathbf{a}}, \phi_{\mathbf{a}}) = (\sin(\theta_{\mathbf{a}})\cos(\phi_{\mathbf{a}}), \sin(\theta_{\mathbf{a}})\sin(\phi_{\mathbf{a}}), \cos(\theta_{\mathbf{a}}))^t. \tag{1.24}$$

The translation is coded with a single angle $\alpha_{\mathbf{t}}$ that simply accounts for the deviation between the camera viewing direction (the **z** axis) and the rotation axis (see Fig. 1.13). If the angle $\alpha_{\mathbf{t}}$ is zero, the camera viewing direction intersects the rotation axis. Different parameterizations can be found for the translation **t** depending where the origin of the world is chosen on the rotation axis **a**. If the origin is located at the intersection of the **xz** plane with the rotation axis **a**, we have (see Fig. 1.13.a):

$$\mathbf{t}(\alpha_{\mathbf{t}}) = t(\sin(\alpha_{\mathbf{t}}), 0, \cos(\alpha_{\mathbf{t}}))^t. \tag{1.25}$$

The only drawback of this new parameterization is that, in order to compute $\alpha_{\mathbf{t}}$ accurately, the projection of the rotation axis needs to be close to vertical (see Fig. 1.13.a). To solve this problem, another parameterization is proposed in figure 1.13.b. The world origin is located at the intersection of the rotation axis with the plane

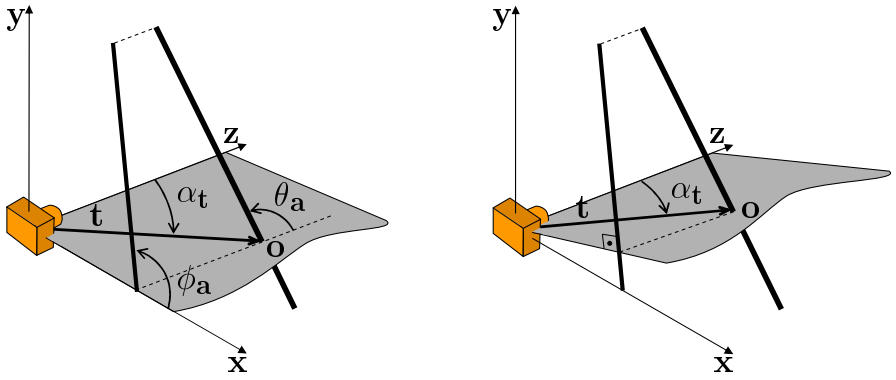


Figure 1.13: *Different parameterizations of* $\mathbf{t}$ *as a function of* $\alpha_{\mathbf{t}}$.

defined by the $\mathbf{z}$ axis and perpendicular to the projection of the rotation axis $\mathbf{a}$ into the $\mathbf{xy}$ plane, which gives the following equation for the translation:

$$\mathbf{t}(\alpha_{\mathbf{t}}) = t(\sin(\alpha_{\mathbf{t}})\sin(\phi_{\mathbf{a}}), -\sin(\alpha_{\mathbf{t}})\cos(\phi_{\mathbf{a}}), \cos(\alpha_{\mathbf{t}}))^t. \tag{1.26}$$

Since, in practice, the projection of the rotation axis is not far from being vertical, the parameterization in Eqn. 1.26 is not necessary and we can use equation 1.25, which is simpler.

The initialization is simple because the coordinate system is related to the camera. Since the camera $\mathbf{y}$ axis is usually aligned with the rotation axis $\mathbf{a}$, initializing the rotation axis with the $\mathbf{y}$ axis is a good approximation, i.e., $\theta_{\mathbf{a}} = \phi_{\mathbf{a}} = 90$ degrees. Translation is also simple to initialize since objects are in general centered around the rotation axis, with the camera pointing to it, so an initial value of $\alpha_{\mathbf{t}} = 0$ is not very bad.

In addition to the extrinsic parameters, we also have the intrinsic parameters K (see Eqn. 1.6). The complete projection matrix P of a particular camera under circular motion has the following form:

$$P(\omega) = K\ [R_{\mathbf{a}}(\omega)|\mathbf{t}]. \tag{1.27}$$

1.7 Optimization of the Cost Criterion

In order to maximize our silhouette coherence criterion we need to use a numerical optimization algorithm. In this section we give a rapid overview of the different available non-linear optimization techniques. We need to minimize a cost function $f(\mathbf{x})$ over parameters $\mathbf{x}$, starting from some given initial estimate $\mathbf{x}$ of the minimum. We try to find a displacement $\mathbf{x} \rightarrow \mathbf{x} + \delta\mathbf{x}$ that locally minimizes the cost function. Although this does not usually give the exact minimum, it will improve the initial parameter estimate and allow us to iterate until convergence.

Historically, most approaches to optimization take advantage of a familiar technique of classical analysis: the Taylor's series expansion of the objective function. One

can classify most methods for numerical optimization according to how many terms of the expansion are exploited. Newton's method, which assumes the availability of first and second derivatives and uses the second-order Taylor polynomial to construct local quadratic approximations of f, is a second-order method. Steepest descent, which assumes the availability of first derivatives and uses the first-order Taylor polynomial to construct local linear approximations of f, is a first-order method. In this taxonomy, zero-order methods do not require derivative information and do not construct approximations of f. They are direct search methods, which indeed are often called zero-order methods in the engineering optimization community.

We have tested both direct search methods and derivative-based methods for the optimization of the silhouette coherence criterion. Although we have not the possibility of computing the analytic derivatives of the silhouette coherence criterion, this does not prevent us from using derivative-based methods to optimize it, since we can estimate the derivatives numerically [Griewank, 2000].

Based on the results that we have obtained, we can make some remarks about the different tested methods. Concerning the direct search methods, we have tested simplex and Powell methods from [Press et al., 1992] and the pattern-based optimization method APPS described by [Hough et al., 2001], whose software is available at `software.sandia.gov/appspack`. Both Powell's and APPS provide the best results of the three of them. Powell's has slightly better convergence properties while APPS is much more efficient in the number of total function evaluations. Something remarkable about Powell's algorithm is its ability to work "*out of the box*", without the need of any other information, whereas APPS is a constrained optimization technique and requires the definition of both the scale and the limits of the search space.

Concerning the derivative-based methods, their implementation is much trickier than for direct search ones. A very important aspect is the need of variable scaling and more generally preconditioning, since derivative-based methods are very sensitive to a bad scaling of the variables. We have tested the conjugate gradient and BFGS from [Press et al., 1992] and the Levenberg-Marquardt from netlib at `www.netlib.org`. The general conclusion is that they actually reach better convergence rates than direct methods when they are close enough to the optimum. However, as we will see in Section 1.8.4, the cost functions that we are considering are noisy, which implies that derivative-based methods are not well adapted when starting far from the optimum: we need to filter the function to obtain a reliable gradient. The implicit filtering algorithm IFFCO has also been tested [Gilmore and Kelley, 1995], but results are comparable to the ones obtained with Powell's method, and many more parameters need to be adjusted. In fact, we do not discard obtaining better results with a better parameter selection, but this is not an easy task since they are not very intuitive.

In the following we present a rapid sketch of both direct search and derivative-based methods.

1.7.1 Direct Search Methods

The first direct search methods were proposed during the 50s and the 60s (see [Fletcher, 1965] for a review). The optimization community scorned direct search methods due

to the lack of convergence results. A regained interest in direct search methods appeared after the thesis of [Torczon, 1989], where she proposed a multi-directional search method and the associated convergence proof [Torczon, 1997], formalized the concept of pattern search methods, and proposed a convergence analysis.

A historical perspective of different direct search methods is presented by [Wright, 1996] or more recently by [Kolda et al., 2003]. Direct search methods can be generally classified into 3 main groups, namely *pattern search methods*, *simplex methods*, and methods with *adaptive sets of search directions*.

Pattern search methods

Pattern search methods are characterized by a series of exploratory moves that consider the behavior of the objective function at a pattern of points, all of which lie on a rational lattice. Methods proposed previously by [Hooke and Jeeves, 1961] or [Polak, 1971] can be categorized as pattern search methods. They are based on a smart exploration of a grid based on a predefined geometrical pattern. Recent pattern-based optimization methods such as [Hough et al., 2001] allow taking advantage of parallel computing interfaces as MPI (Message Passing Interface) to accelerate computations.

Simplex search methods

Simplex search methods are based on the vertices of a simplex, which is updated to reflect the local geometry of the objective function. The first simplex method was proposed by [Spendley et al., 1962], but the most popular was proposed by [Nelder and Mead, 1965], even if it was proved not to converge in some circumstances [Mckinnon, 1998].

Methods with adaptive sets of search directions

This family includes Rosenbrock's method [Rosenbrock, 1960] and Powell's method [Powell, 1964]. These algorithms attempt to accelerate the search by constructing directions designed to use information about the curvature of the objective obtained during the course of the search.

1.7.2 Derivative-based Methods

There exist a great variety of optimization algorithms that use function derivatives (see [Fletcher, 1987] or [Nocedal and Wright, 1999] for more details). As mentioned previously, they can be classified according to how many terms of the Taylor decomposition they use. If we develop the quadratic Taylor series, we get:

$$f(\mathbf{x}+\delta\mathbf{x}) \approx f(\mathbf{x}) + \mathbf{g}^t\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^t H\delta\mathbf{x}, \tag{1.28}$$

where $\mathbf{g}$ is the gradient vector and H is the Hessian matrix. The simplest methods use only first derivatives, which gives the greedy steepest descent method [Morse and Feshbach, 1953] or the more advanced conjugate gradient method [Fletcher and Reeves,

1964]. They basically follow a direction related to the gradient vector. More advanced methods require the use of the second order derivatives (the Hessian matrix). When using second order derivatives, the local model of the objective function (see Eqn. 1.28) is a simple quadric with a unique global minimum that can be found explicitly by setting $df(\mathbf{x}+\delta\mathbf{x}) \approx H\delta\mathbf{x}+\mathbf{g}$ to zero, which gives the so called Newton step:

$$\delta\mathbf{x} = -H^{-1}\mathbf{g}. \tag{1.29}$$

Iterating the Newton step gives the Newton's method. Since computing the real Hessian matrix can be hard for complex cost functions, there exists a family of algorithms called quasi-Newton methods (such as BFGS), that actually estimate H from the variations of $\mathbf{g}$ during the iteration. Newton's methods may have convergence problems (e.g., convergence to a saddle point). In order to improve the convergence, it is preferable to use the Newton step as a descent direction for a line search algorithm rather than as the exact displacement at each iteration. Finally, in order to best exploit the convergence properties of first order and second order methods, the Damped Newton methods use as descent direction a combination of the Newton and gradient directions:

$$(H+\lambda W)\delta\mathbf{x} = -\mathbf{g}, \tag{1.30}$$

where λ is a weighting factor and W is a weighting matrix (often the identity) between the gradient and Newton directions: a large λ gives the gradient direction while $\lambda = 0$ gives us the original Newton step. A very well known example of a Damped Newton method is the Levenberg-Marquardt method [Levenberg, 1944], [Marquardt, 1963].

1.8 Experiments with One Rotation Sequence

We have tested and compared the proposed discrete measure of silhouette coherence with the epipolar tangency criterion using synthetic exact silhouettes, synthetic noisy silhouettes and real silhouettes.

The implemented epipolar tangency criterion has the following form:

$$\mathcal{C}_{et} = \frac{1}{\sum_{i=1}^{n}\sum_{j\in\mathcal{N}(i)} K_{ij}} \sum_{i=1}^{n}\sum_{j\in\mathcal{N}(i)} K_{ij}\mathcal{C}_{et}(S_i,S_j), \tag{1.31}$$

$$K_{ij} = \begin{cases} 0 & if\ e_{ij}\in S_i\ or\ e_{ji}\in S_j \\ 1 & else \end{cases},$$

where n is the number of available silhouettes, $\mathcal{N}(i)$ is the subset of peer silhouettes associated to the silhouette i, $\mathcal{C}_{et}(S_i,S_j)$ is the epipolar tangency coherence between two silhouettes as defined in section 1.3, and K_{ij} takes into account whether we can compute $\mathcal{C}_{et}(S_i,S_j)$ or not. In the particular case where the epipole is inside the silhouette, $\mathcal{C}_{et}(S_i,S_j)$ cannot be computed, so $\mathcal{C}_{et}$ is weighted accordingly. We have to precise the meaning of $\mathcal{N}$ in Eqn. 1.31. If we want to compare each silhouette with the $n-1$ others, then $\mathcal{N}_{all}(i) = \{i+1,\cdots,n\}$. However, this is not the original definition given in [Wong and Cipolla, 2001]. They use $\mathcal{N}_3(i) = \{i+1,\cdots,\min(i+3,n)\}$.

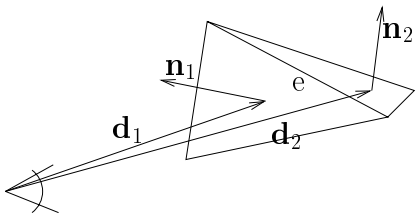


Figure 1.14: *Contour generator detection.*

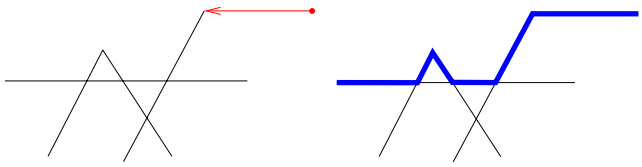


Figure 1.15: *Computation of the outer silhouette polygon.*

Although they do not justify this choice, it is probably due to the resulting computating time saving ($\mathcal{N}_3$ is faster than $\mathcal{N}_{all}$) and to the convergence properties of $\mathcal{C}_{et}$: using a small number of silhouettes around the current silhouette should help to smooth the energy shape. But this leads to a worse accuracy than when using all the silhouettes. Furthermore, in the experiments described in subsection 1.8.4, both $\mathcal{N}_3$ and $\mathcal{N}_{all}$ exhibit very similar convergence properties. Then, to speed up the algorithm, we have first used $\mathcal{N}_3(i)$ to get close to the optimum and then $\mathcal{N}_{all}$ to reach a better accuracy.The optimization method used with both the epipolar tangency criterion and the silhouette coherence criterion is Powell's conjugate directions algorithm [Powell, 1964].

For all the examples, we have used single axis rotation sequences with constant intrinsic parameters. Although at first sight this can seem a little bit constraining, it is a really useful configuration commonly used in practice. In addition, using single axis rotation sequences does not mean using only one rotation sequence. We can in fact use several rotation sequences from the same object, which improves the 3D reconstruction results as we will see in Section 1.9.

1.8.1 Synthetic Exact Silhouettes

In this experiment, we dispose of a synthetic Teapot represented by a triangle mesh. We want to obtain an exact set of silhouettes by computing the 2D polygons generated by the projection of the 3D mesh. This is accomplished in the following two steps:

- detection and projection of the edges that are contour generators,
- extraction of the minimal polygon that contains all the 2D projected segments.

Edges belonging to the contour generators are defined as those that share two triangles with different visibility, i.e., edges that share one front triangle and one back triangle. Mathematically, an edge e is a contour generator if, given the two face normals $\mathbf{n}_1$ and $\mathbf{n}_2$, and the two corresponding viewing directions $\mathbf{d}_1$ and $\mathbf{d}_2$, the following relation holds: $s_1 s_2 < 0$, with $s_1 = \mathbf{n}_1 \cdot \mathbf{d}_1$ and $s_2 = \mathbf{n}_2 \cdot \mathbf{d}_2$ (see Fig. 1.14).

Once we have the 2D segment soup, we need to construct the silhouette polygon defined by an ordered list of segments. This can be done by using an automate that iteratively marches along the oriented edges, cutting and pruning if necessary. We can see in Fig. 1.15 left an example of the input to the state machine algorithm and the

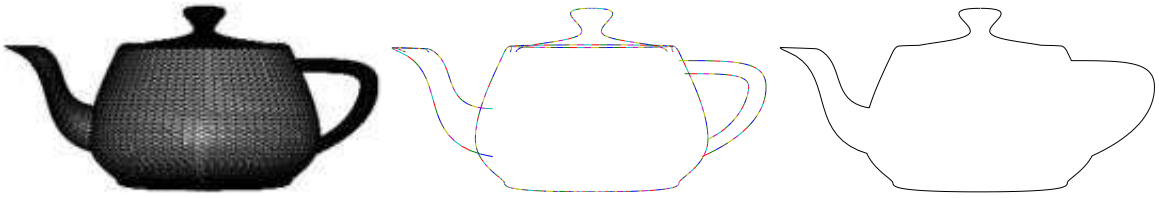


Figure 1.16: *The two steps to compute exact silhouettes from a 3D mesh. From left to right, original Teapot mesh, apparent contours and extracted outer contour.*

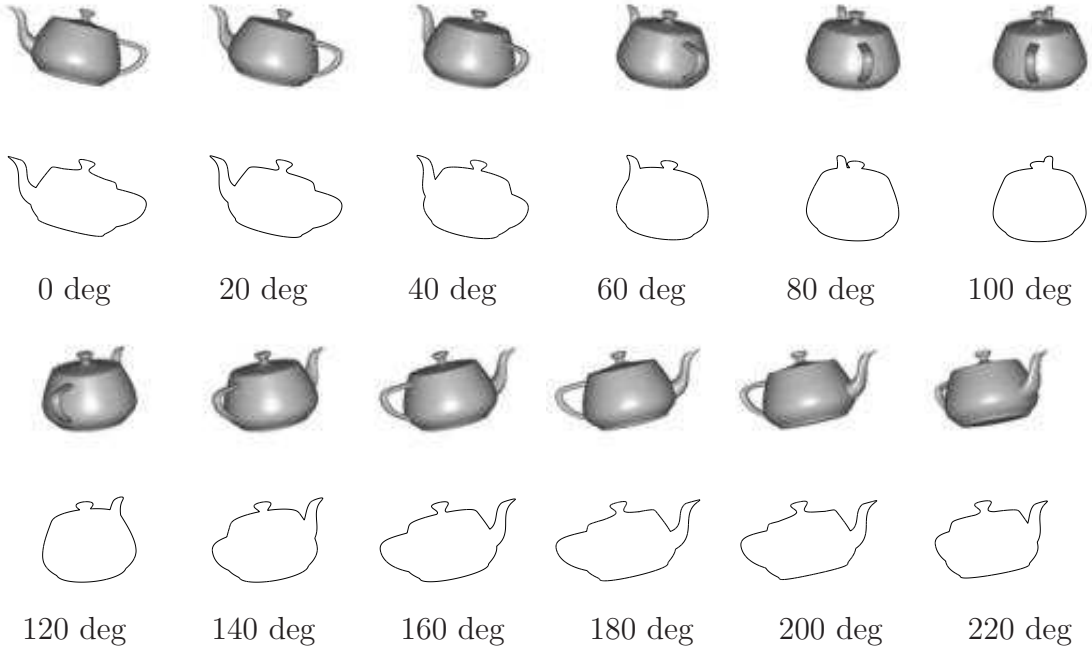


Figure 1.17: *Example of a sequence of the Teapot with 12 views, their corresponding exact silhouettes and their absolute camera angles.*

output we want in Fig. 1.15 right. The algorithm is initialized with a segment (in red) that belongs to the outside contour of the silhouette. This initialization segment can be found simply by a scan line approach. Starting from the origin of the current oriented segment, the state machine looks for the closest intersection or contact point and updates the old segment (cuts out the used parts). We iterate this procedure until the state machine comes back to the initialization segment. We can see in Fig. 1.16 an example of the entire procedure, with the original synthetic Teapot on the left, the projection of the contour generator edges in the middle, and the final silhouette contour on the right. We show in Fig. 1.17 a sequence of 12 exact silhouettes generated with the synthetic Teapot. The retinal plane is 1024x768 pixels size and the focal length is $f = 9000$ pixels, as defined in equation 1.6 ($fov = 4.89$ degrees).

Next we present an experiment using the exact silhouettes of Fig. 1.17. We compare the accuracy of the motion and focal length estimation for the epipolar tangency and the silhouette coherence criterion. We assume that the rest of the intrinsic parameters are known. Since we dispose of 12 silhouettes, the total dof is 15: the relative angles

Teapot		rotation axis (degrees)		translation (degrees)	focal (pixels)
		$\theta_{\mathbf{a}}$	$\phi_{\mathbf{a}}$	$\alpha_{\mathbf{t}}$	f
initial		106.0000	110.0000	1.4	6000
real		86.6265	90.5757	0.0	9000
$\mathcal{C}_{et}$	recovered	86.6265	90.5757	0.0	9000
	error	$< 10^{-6}$	$< 10^{-6}$	$< 10^{-6}$	0.02
$\mathcal{C}_{sc}$	recovered	86.6255	90.5757	$2.7 \cdot 10^{-5}$	8997
	error	$9.7 \cdot 10^{-4}$	$< 10^{-6}$	$2.7 \cdot 10^{-5}$	3.14

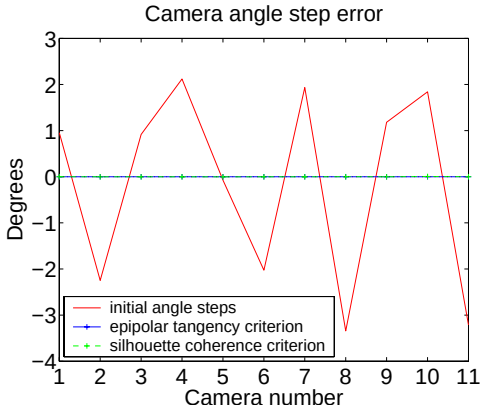


Table 1.1: *Motion and focal estimation with **exact silhouettes**. Mean camera step error of $3.2 \cdot 10^{-5}$ degrees with the **epipolar tangency criterion** (solid blue) and $3.9 \cdot 10^{-3}$ degrees with the **silhouette coherence criterion** (dashed green), $\delta = 10^{-3}$ pixels.*

$\Delta\omega_i$ (11 dof) plus the rotation axis $(\theta_{\mathbf{a}}, \phi_{\mathbf{a}})$ (2 dof), the translation direction $\alpha_{\mathbf{t}}$ (1 dof) and the focal length f (1 dof). Obviously, since we only have image information, we cannot recover the distance to the rotation axis t. It is worth noting that, because the original synthetic Teapot is a triangle mesh, coding the silhouettes as a polygons does not introduce any representation noise since silhouettes *really* are polygons. As a result, due to the absence of noise, the epipolar tangency measure performs perfectly (see Table 1.1).

Since silhouettes are exact, we have used a very small offset of $\delta = 10^{-3}$ pixels in order to maximize the accuracy of the silhouette coherence criterion (see Section 1.6.2). Although the proposed silhouette coherence measure performs well too (see Table 1.1), the epipolar tangency criterion achieves better results (mean camera step error of $3.2 \cdot 10^{-5}$ against $3.9 \cdot 10^{-3}$ with the silhouette coherence criterion). This is simply due to the fact that, when silhouettes are exact, the epipolar tangency criterion is a much more accurate measure of the silhouette coherence between two silhouettes than the discrete silhouette coherence measure we use, the silhouette criterion being computed by sampling points along the silhouette contours.

1.8.2 Synthetic Noisy Silhouettes

In the next experiment we add some noise to the silhouettes. This is not an easy task since there is no simple model of the silhouette segmentation process. However, in practice, there is one type of noise that is easy to reproduce and is present in real images: the camera sampling. Even if we manage to obtain very accurate silhouettes of an object, we always have the limitation of the camera resolution. Because we use pixels to capture the scene, we implicitly introduce a silhouette representation error of a maximum of 0.5 pixels (see Fig. 1.18).

We have performed two types of experiments with noisy silhouettes. In the first experiment we have used pixelized silhouettes that simulate the camera sampling process. In the second kind of experiments we have tested the exact silhouettes with

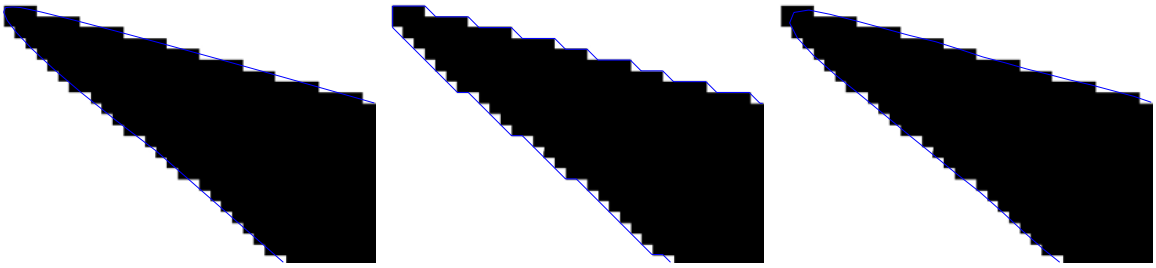


Figure 1.18: *Silhouette contour discretization. Left: original exact contour (blue) superposed to the binary silhouette. Middle: extracted contour passing through the border of the contour pixels. Right: extracted contour after a curvature-driven regularization.*

additive random noise in order to better study the precision of the algorithms in a statistical way.

Pixelized Silhouettes

We can use many different techniques to extract a *continuous* contour from a binary image, e.g. snakes [Kass et al., 1988] or splines [Plass and Stone, 1983]. However, since we want to test the accuracy in the presence of noise, we have tested both pixelized and regularized silhouette contours. In figure 1.18 left we can see the original contour (in blue) and the corresponding pixelized silhouette (in black). In Fig. 1.18 middle we have the contour extracted from the pixelized silhouette and on the right we have the same contour regularized by a curvature-driven snake. We can compare the quality of the extracted contours with the exact ones using simple statistics such as the Root Mean Square Error (RMSE) or the Mean Absolute Error (MAE). If we want to compare a given polygon $P = \{\mathbf{p}_i,\ i = 0, \cdots, N-1\}$ with a reference polygon $P^{ref} = \{p_i^{ref},\ i = 0 \cdots, N^{ref} - 1\}$, we can define the $RMSE$ and MAE as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} d(\mathbf{p}_i, P^{ref})^2}, \tag{1.32}$$

$$MAE = \frac{1}{N}\sum_{i=0}^{N-1} d(\mathbf{p}_i, P^{ref}), \tag{1.33}$$

where the distance between a 2D point and the reference polygon is the minimum distance between the 2D point and the 2D segments defining the polygon:

$$d(p, P^{ref}) = \min_i d(p, [p_i^{ref}, p_{i+1}^{ref}]). \tag{1.34}$$

For the example of Fig. 1.18, the pixelized segmentation gives a MAE of 0.33 pixels and a $RMSE$ of 0.39 pixels. The regularized version achieves a MAE of only 0.1 pixels and a $RMSE$ of 0.13 pixels.

When using the pixelized silhouettes (see Fig. 1.18 middle), the epipolar tangency criterion does not perform very well, especially for the camera angle steps where it

Teapot		rotation axis (degrees)		translation (degrees)	focal (pixels)
		$\theta_{\mathbf{a}}$	$\phi_{\mathbf{a}}$	$\alpha_{\mathbf{t}}$	f
initial		106.0000	110.0000	1.4	6000
real		86.6265	90.5757	0.0	9000
$\mathcal{C}_{et}$	recovered	86.6052	90.5604	0.0003	9055
	error	0.0212	0.0153	0.0003	55
$\mathcal{C}_{sc}$	recovered	86.6104	90.5718	0.0006	9057
	error	0.0161	0.004	0.0006	57

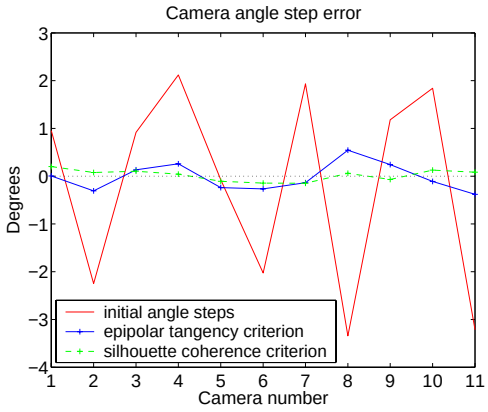


Table 1.2: *Motion and focal estimation with* ***pixelized silhouettes****. Mean camera step error of 0.24 degrees with the* ***epipolar tangency criterion*** *and 0.11 degrees with the* ***silhouette coherence criterion****,* $\delta = 0.5$ *pixels.*

gets a mean error of 0.24 degrees (see Table 1.2). The silhouette coherence criterion behaves much better, obtaining a mean camera step error of only 0.11 degrees (see Table 1.2).

Regularized Silhouettes

Concerning the regularized silhouettes (see Fig. 1.18 right), we could expect a better precision of both criteria since the $RMSE$ and MAE are smaller. It turns out that this not true fot the epipolar tangency criterion. The error is simply "*distributed*" differently. The estimation of the rotation axis is slightly improved while the translation and the focal length are worse and the camera angles remain almost the same (see Table 1.3). Somehow, this distribution of the error is not completely unexpected since the amount of information contained in the pixelized silhouettes and in the regularized silhouettes is the same: the regularized silhouettes are obtained from the pixelized silhouettes, no other information being added.

The silhouette coherence criterion behaves in a different way. It has better results when using regularized silhouettes, and this is mainly due to a less noisy computation of the depth intervals than when using pixelized silhouettes. Regularizing the silhouette contours allows using a small contour offset $\delta = 0.25$ pixels, while the offset used for the pixelized silhouettes is $\delta = 0.5$ pixels. We show in subsection 1.8.4 (see Fig. 1.29) the effect of regularizing the silhouette contours when computing the depth intervals. For both cases, the sampling points are chosen equally spaced, with a distance of 1 pixel between samples, which gives approximately 2000 sampling points per silhouette.

Exact Silhouettes + Random Noise

In this experiment we have added noise to the exact silhouettes in order to statistically measure the precision of the epipolar tangency criterion and the discrete silhouette coherence criterion. The noise has been simply added along the normal of the contours. The amplitude of the noise is computed in the same way as in [Wong et al., 2003],

Teapot		rotation axis (degrees)		translation (degrees)	focal (pixels)
		$\theta_{\mathbf{a}}$	$\phi_{\mathbf{a}}$	$\alpha_{\mathbf{t}}$	f
initial		106.0000	110.0000	1.4	6000
real		86.6265	90.5757	0.0	9000
$\mathcal{C}_{et}$	recovered	86.6067	90.5617	0.0018	9060
	error	0.0197	0.0140	0.0018	60
$\mathcal{C}_{sc}$	recovered	86.6040	90.5776	0.0001	9013
	error	0.0224	0.0019	0.0001	13

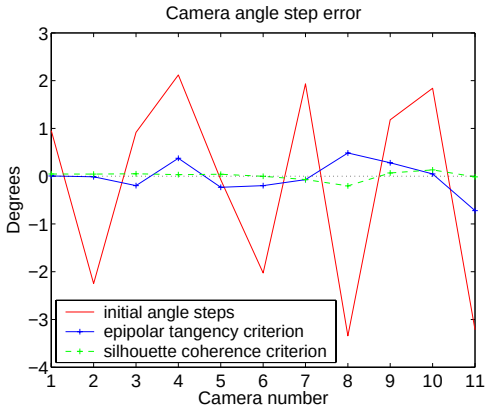


Table 1.3: *Motion and focal estimation with* ***regularized silhouettes****. Mean camera step error of 0.24 degrees with the* ***epipolar tangency criterion*** *and 0.06 degrees with the* ***silhouette coherence criterion****,* $\delta = 0.25$ *pixels.*

i.e., as a uniform noise smoothed by a Gaussian filter, which avoids unrealistic jaggedness along the silhouette contour. The noise variance is computed after the Gaussian filtering. For each noise variance, we have computed 200 samples in order to obtain reliable results. We have estimated both the motion and the focal length. Due to the non-linear energy shapes, we have supposed the camera angle steps known to avoid too many outliers due to local minima. The estimated motion is therefore composed of the rotation axis and the translation direction. The criterion used to measure the error between the recovered rotation axis $\mathbf{a}_{recovered}$ and the real axis $\mathbf{a}$ is the angle between both axes: $\Delta\mathbf{a} = acos(\mathbf{a}_{recovered} \cdot \mathbf{a})$.

We show a total of 4 figures, Fig. 1.19 for the rotation axis estimation, figures 1.20 and 1.21 for the translation angle recovery results, and Fig. 1.22 for the focal length estimation results. We have tested the epipolar tangency criterion with 4 different sets of peer silhouettes $\mathcal{N}$ in Eqn. 1.31: 1 silhouette $\mathcal{N}_1(i) = \{\min(i+1, n)\}$, 3 silhouettes $\mathcal{N}_3(i) = \{i+1, \cdots, \min(i+3, n)\}$, 6 silhouettes $\mathcal{N}_6(i) = \{i+1, \cdots, \min(i+6, n)\}$ and all the silhouettes $\mathcal{N}_{all}(i) = \{i+1, \cdots, n\}$. Note that, although the total number of silhouettes is 12, $\mathcal{N}_6$ and $\mathcal{N}_{all}$ are not the same: silhouette 1 and silhouette 12 will never be compared using $\mathcal{N}_6$ but they will using $\mathcal{N}_{all}$. Concerning the silhouette coherence criterion, we have tested two different values of δ (see Section 1.6.2): $\delta = 0.5$ pixels and $\delta = 1$ pixels.

According to the estimation results of the camera motion (rotation axis and translation direction) shown in figures 1.19 and 1.20, the silhouette coherence performs better than any of the epipolar tangency criteria for large noise variances, and for both δ distances. This is justified by the fact that the silhouette coherence criterion uses the entire contours for the computation while the epipolar tangency criterion is actually using only the epipolar tangent points. However, the behavior of the silhouette coherence changes with low noise, where the epipolar tangency criterion performs better and, what is even surprising, the silhouette coherence curves are not monotone with the noise standard deviation: e.g., for $\delta = 1$, we obtain better accuracy with a noise of 0.3 than with a noise of 0.1. This unexpected behavior is caused by the rapid

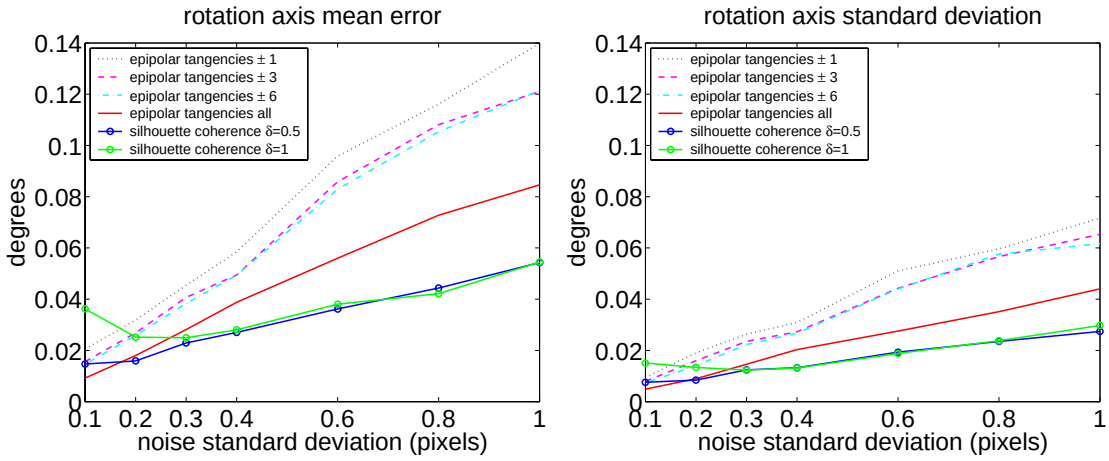


Figure 1.19: *Rotation axis recovery precision as a function of the noise standard deviation. Left: mean error. Right: standard deviation.*

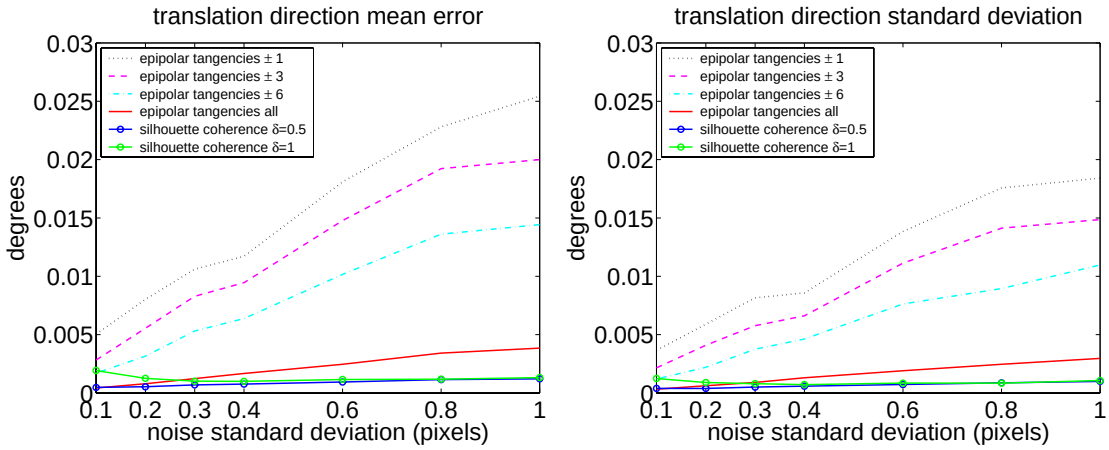


Figure 1.20: *Translation direction recovery precision as a function of the noise standard deviation. Left: mean error. Right: standard deviation.*

saturation of the silhouette coherence criterion. For low noise, the silhouette coherence is very easily maximized if δ is not small enough. As a result, the silhouette coherence criterion shows a constant platform around the optimum whose size depends on the value of δ and the noise of the silhouettes.

If we compare the results for the translation estimation, i.e., the $\alpha_{\mathbf{t}}$ parameter, we can appreciate a strong dependence of the epipolar tangency criterion on the number of silhouettes used (see figure 1.20). As we will discuss in the next Section 1.8.4, the epipolar tangency criterion needs a large baseline to estimate $\alpha_{\mathbf{t}}$ with accuracy. This is justified by the fact that, when the baseline is small, the epipolar tangencies change very little with large variations of $\alpha_{\mathbf{t}}$. Even when using all the possible silhouette pairs for the epipolar tangency criterion ($\mathcal{N} = \mathcal{N}_{all}$), the silhouette coherence criterion performs far better than the epipolar tangency one (see Fig. 1.21).

Finally, we show the focal length estimation results in Fig. 1.22. Both the epipolar tangency criterion (with $\mathcal{N}_{all}$) and the silhouette coherence criterion perform very well,

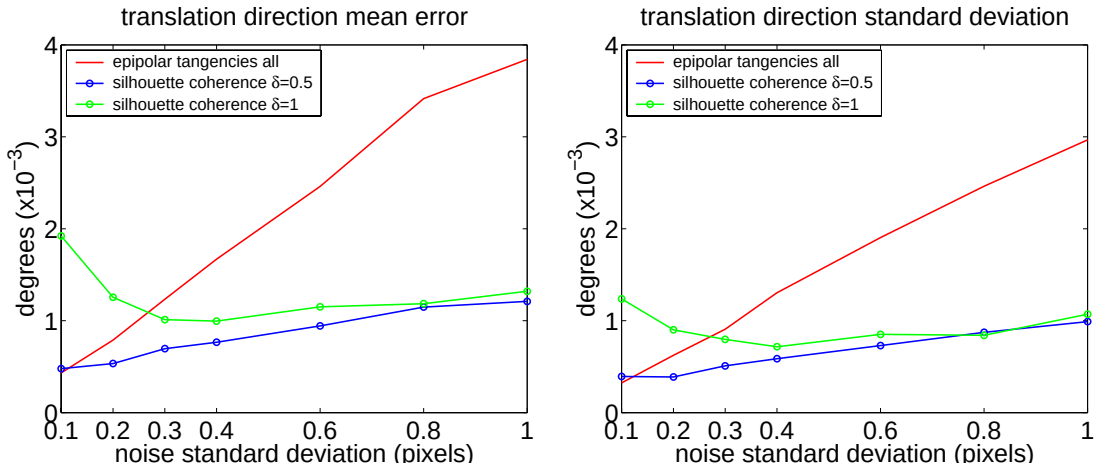


Figure 1.21: *Detail of the translation direction recovery precision as a function of the noise standard deviation. Left: mean error. Right: standard deviation.*

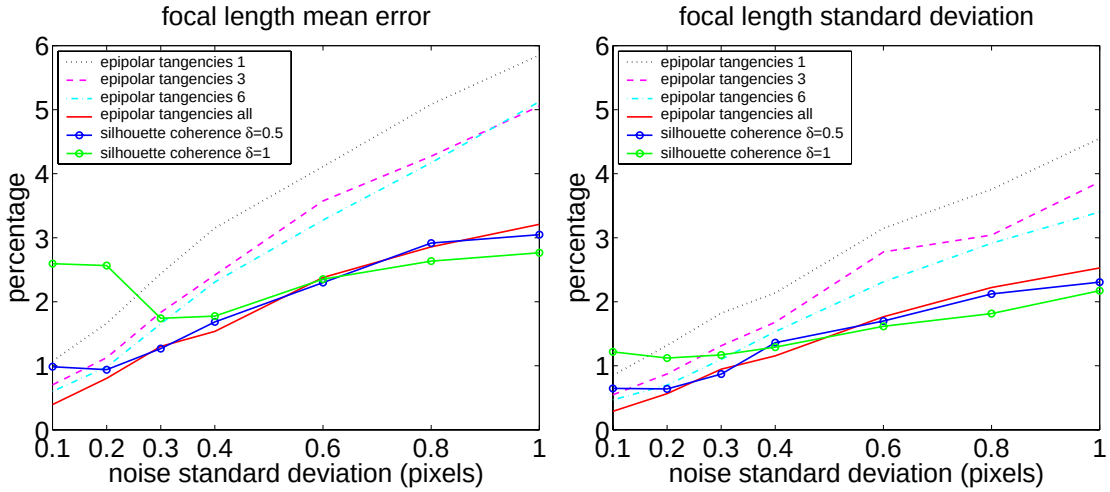


Figure 1.22: *Focal length recovery precision as a function of the noise standard deviation. Left: mean error. Right: standard deviation.*

with less than 3% of error for a noise standard deviation of 1 pixel. An interesting remark about the silhouette coherence criterion is that, with $\delta = 1$ pixel, the focal length error is almost the same with a noise standard deviation of 0.1 and 1.0 pixels. As explained before, it shows the saturation effect of the silhouette coherence criterion when the values of δ are large compared to the noise of the silhouettes.

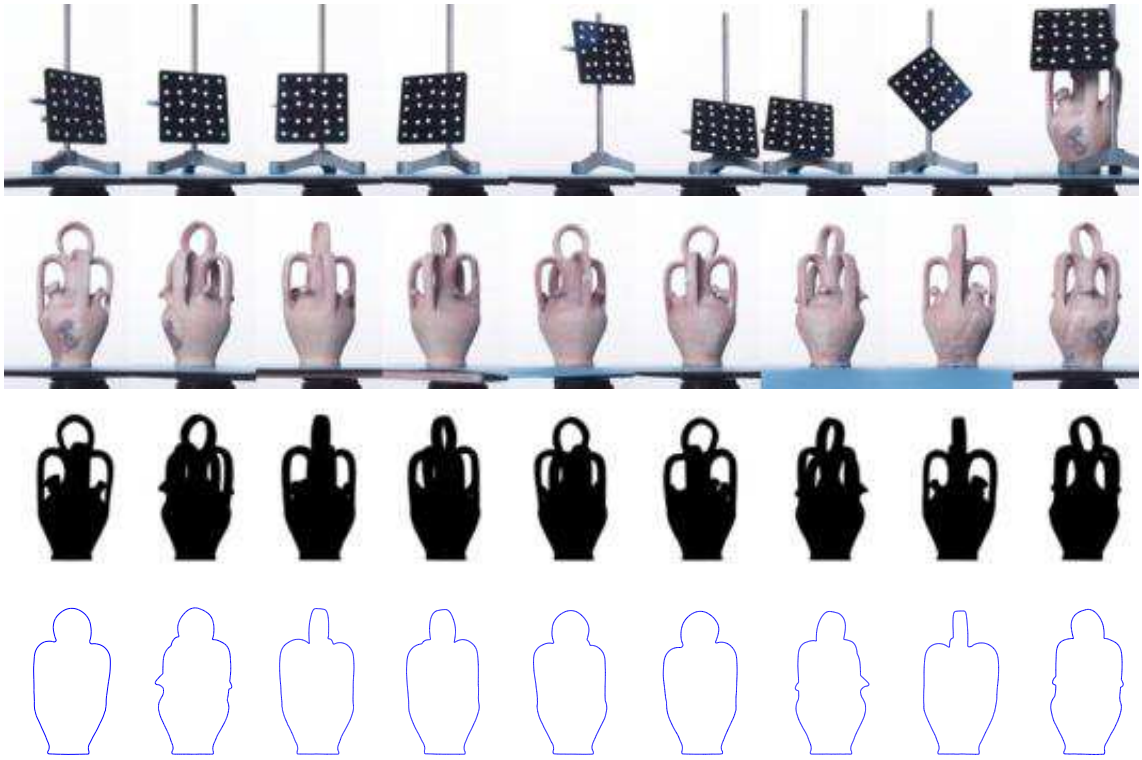


Figure 1.23: *Some of the original images of the Pitcher sequence (of a total of 36 images). From top to bottom: calibration target images, color images, binarized silhouettes and extracted smooth polygon silhouettes.*

1.8.3 Real Silhouettes

In this subsection we present a real case of a Pitcher sequence. We do not present any of the multiple objects that will be discussed in Chapter 3 due to the uncertainty on the accuracy of their classic calibration step. Obtaining very good accuracy requires perfectly controlling the acquisition environment and most of the reconstructed objects have been acquired by professional photographers that are not necessarily computer vision experts. This is important since the precision we obtain for some parameters using silhouettes (in particular, for the translation direction $\alpha_{\mathbf{t}}$) has the same order of magnitude as with the classic calibration method.

We dispose of a sequence of 36 color images of 2008x3040 pixels that have been binarized by an automatic color-segmentation algorithm (see Fig. 1.23). We also dispose of a sequence of a calibration pattern in order to accurately recover the intrinsic parameters and the circular motion using [Lavest et al., 1998]. In particular, the 4 first images of the calibration pattern (see Fig. Fig. 1.23 top) belong to a longer sequence under pure circular motion that allows recovering the rotation axis and the translation direction. The remaining images provide redundancy and accuracy to the estimation by exploring the entire 3D volume. In order to extract smooth contours from the silhouettes, we have used a GVF 2D snake [Xu and Prince, 1998], where the GVF has been computed from the gradient contours of the color images. The sampling distance

Pitcher		rotation axis (degrees)		translation (degrees)	focal (pixels)
		$\theta_{\mathbf{a}}$	$\phi_{\mathbf{a}}$	$\alpha_{\mathbf{t}}$	f
initial		90.000	90.000	0.0	2000
calibrated		99.671	90.343	0.4266	6606
$\mathcal{C}_{et}$	recovered	99.639	90.319	0.4290	6589
	error	0.032	0.024	0.0024	17
$\mathcal{C}_{sc}$	recovered	99.654	90.335	0.4286	6580
	error	0.017	0.008	0.0020	26

Table 1.4: *Rotation axis, translation direction and focal length recovery for the Pitcher sequence.*

used for the silhouette coherence criterion is $\delta = 0.25$ pixels.

We have performed two different tests with the silhouette coherence and the epipolar tangency criteria. The first one consists of estimating only the rotation axis, the translation and the focal length with all the silhouettes available. In the second test we have taken 9 views regularly spaced of 20 degrees and we have computed the full circular motion, i.e., rotation axis, translation direction and camera angles by fixing the focal length to its calibrated value.

We show in Table 1.4 the results of the estimation of the rotation axis, the translation direction and the focal length. The results are good for both criteria. The silhouette coherence criterion performs better than the epipolar tangency criterion when computing the rotation axis and the translation direction. This is especially true for the rotation axis, with an overall axis error $\Delta\mathbf{a}$ of 0.018 degrees for the silhouette coherence and 0.040 degrees for the epipolar tangency. The epipolar tangency criterion is slightly better when computing the focal length.

If we compute the complete circular motion using 9 views, the results using the silhouette coherence are very satisfactory as the mean angle step error is only of 0.13 degrees, which is a good result compared with the mean angle step error of 0.26 degrees obtained with the epipolar tangency criterion (see Table 1.5). Although these results are good, they confirm an important aspect about the expected accuracy of the camera angles recovery, for which silhouette-based criteria are not very accurate. This is mainly due to two reasons: the sensivity to silhouette noise, and the weak angle information contained in the silhouettes. This second point is important point and depends on how much the silhouette changes when turning the object. As we can see in Fig. 1.23 bottom, the silhouettes of the Pitcher sequence are pretty similar when the object turns. It is easy to deduce that, if the silhouette does not change when the angle varies, we will not be able to recover the angles. As a consequence, if we want to have better accuracy when estimating the camera angles, the image sequence should ensure that silhouettes change a maximum from one silhouette to another. In the case of the Pitcher sequence, this could be done by separating the object from the rotation axis, which would produce a silhouette that "*walks*" all around the image when the object turns.

Pitcher		rotation axis (degrees)		translation (degrees)
		$\theta_{\mathbf{a}}$	$\phi_{\mathbf{a}}$	$\alpha_{\mathbf{t}}$
initial		90.000	90.0000	0.0
calibrated		99.671	90.3431	0.4266
$\mathcal{C}_{et}$	recovered	99.596	90.2799	0.4307
	error	0.075	0.0632	0.0041
$\mathcal{C}_{sc}$	recovered	99.714	90.3434	0.4259
	error	0.043	0.0003	0.0007

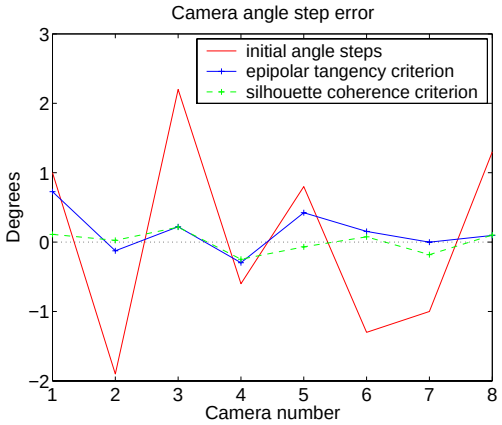


Table 1.5: *Complete circular motion estimation with known focal for the Pitcher sequence. Mean camera step error of 0.26 degrees with the epipolar tangency criterion and 0.13 degrees with the silhouette coherence criterion.*

1.8.4 Energy Shape

One of the critical points of the optimization procedure is the existence of local minima. Both the epipolar tangency method and the silhouette coherence method suffer from non-convex energy maps. In order to visualize the energy shape of the epipolar tangency and the silhouette coherence criteria, we have computed some 2D slices of the energy shape for the camera motion problem with two cases of noisy silhouettes (the synthetic Teapot with a noise standard deviation of 1 pixel, and real Pitcher sequence). In Fig. 1.24 we show a rendered view of the energy corresponding to the rotation axis estimation $(\theta_{\mathbf{a}}, \phi_{\mathbf{a}})$ for the silhouette coherence criterion with $\delta = 0.5$ pixels (left) and the epipolar tangency criterion with $\mathcal{N}_{all}$ (right). We can clearly distinguish two main optima that correspond to the correct direction but with opposite signs. The highest peak roughly corresponds to $\mathbf{a} = (0, 1, 0)^t$ while the second peak corresponds to $\mathbf{a} = (0, -1, 0)^t$. A first conclusion about the behavior of both criteria is that they are in fact quite similar at a global scale. However, once it is near the optimum, the energy shape of the epipolar tangency around the optimum is perfectly convex, which is not completely true for the silhouette coherence criterion.

In 1.25 we compare the energy of the epipolar tangency criterion for the rotation axis estimation with different sets of peer silhouettes $\mathcal{N}$. The energy shape does not change very much when using different $\mathcal{N}$. In particular, the energies computed with $\mathcal{N}_3$ and $\mathcal{N}_{all}$ look very similar, which indicates similar convergence properties of the optimization algorithm for both criteria. We present in Fig. 1.26 a detail around the maximum peak of the Teapot energy shape, for different sets $\mathcal{N}$, and for the couple of parameters $(\alpha_{\mathbf{t}}, \theta_{\mathbf{a}})$, i.e, the translation direction and one of the rotation axis parameters. The shape of the energy function confirms and explains the results of the translation estimation that already appear in Fig. 1.20: the epipolar tangency criterion is extremely non-sensitive to variations of $\alpha_{\mathbf{t}}$ compared to the silhouette coherence criterion shown in Fig. 1.27. However, this result is expected if we consider that, when varying $\alpha_{\mathbf{t}}$, the epipoles move along a line that is almost perpendicular to the projection of the rotation axis. This implies that the epipolar tangency criterion

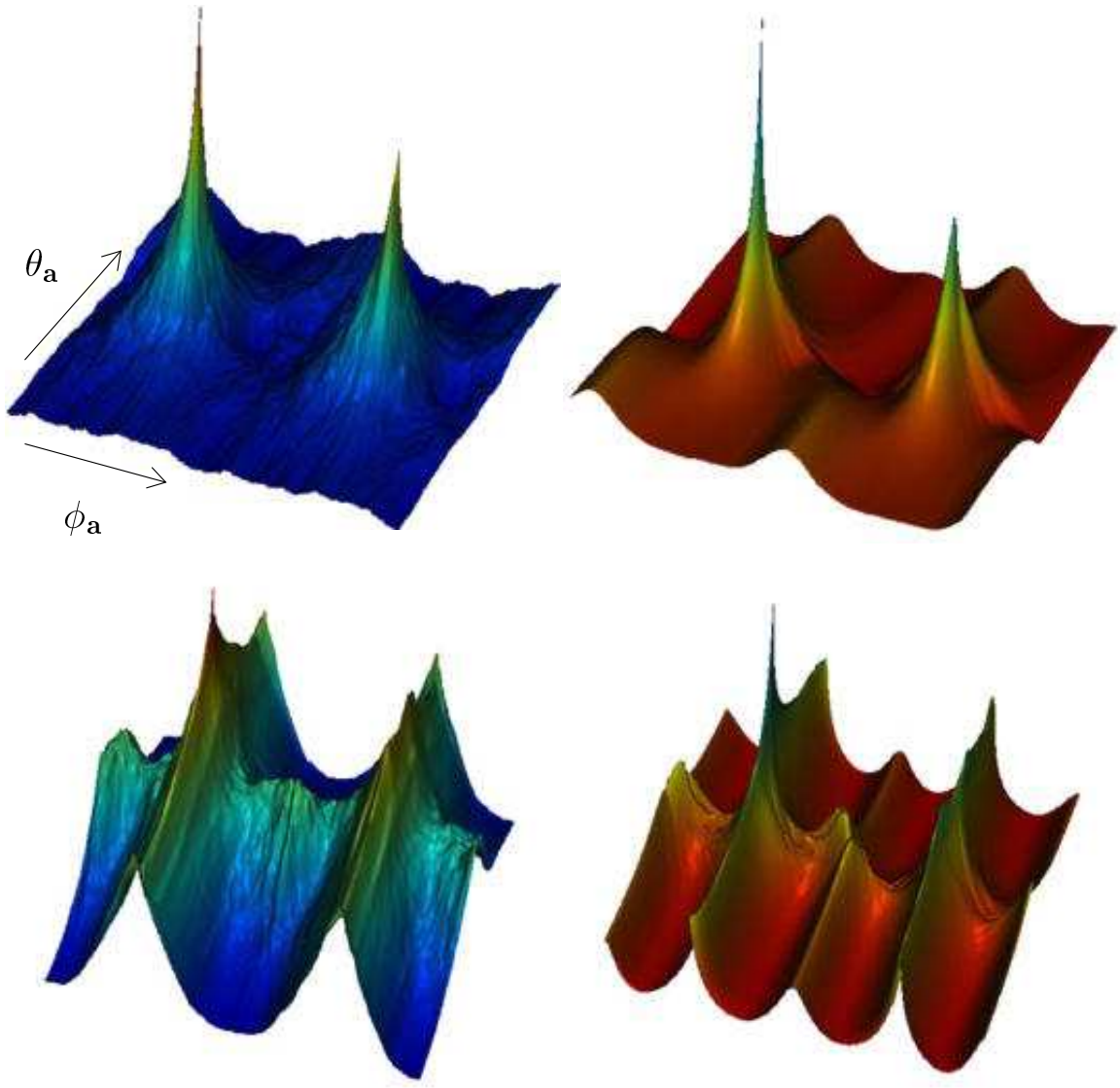


Figure 1.24: *Energy shape of the* ***silhouette coherence criterion*** *with* $\delta = 0.5$ *pixels (left) and the* ***epipolar tangency criterion*** *with* $\mathcal{N}_{all}$ *(right) for the rotation axis estimation of the Teapot (top) and the Pitcher (bottom). The energy domain corresponds to the entire Gauss sphere, i.e.,* $\theta_{\mathbf{a}} \in [0, \pi]$, $\phi_{\mathbf{a}} \in [0, 2\pi]$.

will not be sensitive to $\alpha_{\mathbf{t}}$ if the epipoles are very far away, since the epipolar tangencies will change very little. The criterion will be more sensitive if the epipoles are near the silhouettes, which corresponds to large camera angles between pairs of cameras. This explains why the epipolar tangency criterion is more sensitive when using all the possible silhouette pairs $\mathcal{N}_{all}$ (Fig. 1.26 right).

The discrete silhouette coherence behaves in a different way depending on the scale. At a global scale, the criterion is similar to the epipolar tangency criterion, as shown in Fig. 1.24. However, if we look carefully, the energy surface is not completely smooth, there is a noise due to the discrete nature of the current silhouette coherence

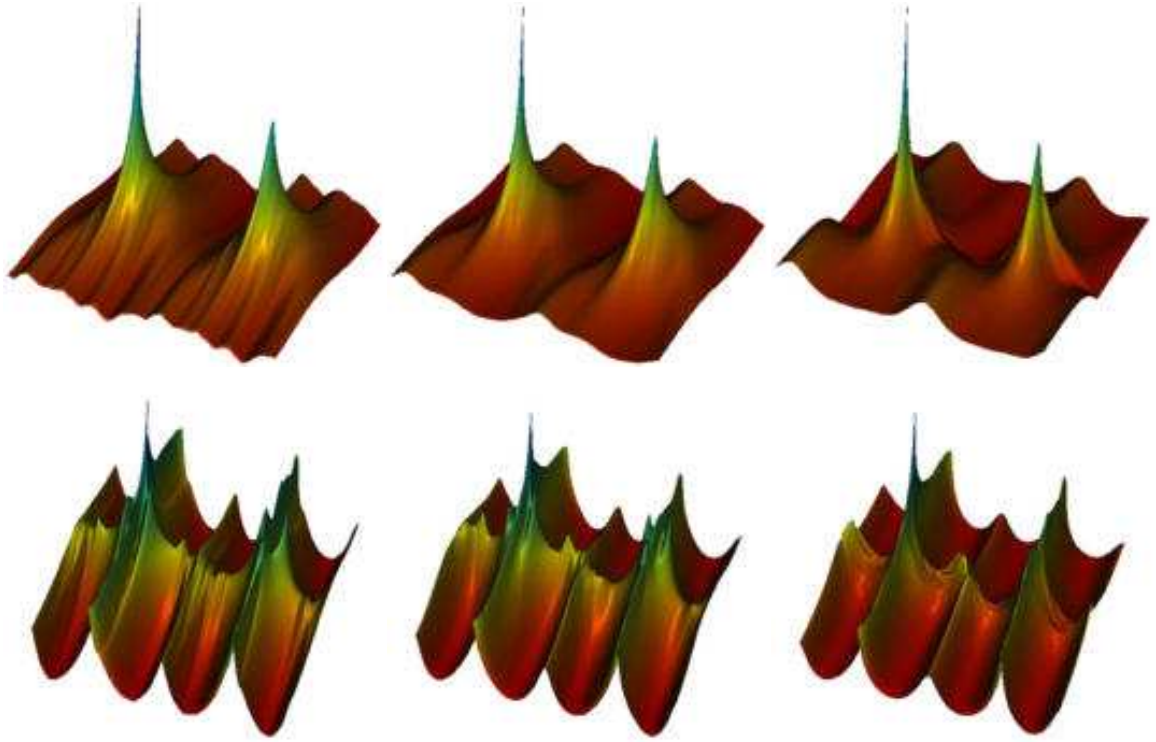


Figure 1.25: *Energy shape of the* ***epipolar tangency criterion*** *for the rotation axis estimation of the Teapot (top) and the Pitcher (bottom). The energy domain corresponds to the entire Gauss sphere, i.e.,* $\theta_{\mathbf{a}} \in [0, \pi]$, $\phi_{\mathbf{a}} \in [0, 2\pi]$. *From left to right, energy shapes for 3 different sets of peer silhouettes* $\mathcal{N}_1$, $\mathcal{N}_3$, *and* $\mathcal{N}_{all}$ *respectively (see Eqn. 1.31).*

implementation: the silhouette coherence is computed as a ratio between the number of sample points with non empty intervals among a finite set of sample points. In fact, the noise of the silhouette coherence criterion greatly depends on the number of sample points per silhouette that we use (see Fig. 1.27). At this point, the discrete nature of the approach limits its accuracy. This discretization problem has been already quoted in the example using synthetic exact data in Section 1.8.1, where the epipolar tangency criterion performs really well while the silhouette coherence criterion remains limited.

In Fig. 1.28 we show a detail of the energy function of Fig. 1.24 top for the couple of parameters $(\theta_{\mathbf{a}}, \phi_{\mathbf{a}})$. We observe how the silhouette coherence is smoothed when adding more samples (see Fig. 1.28 left and middle). This suggests that the number of samples should be adapted to the desired accuracy, i.e., we should use a small number of samples to accelerate computations at a global scale, and then, once we are near the optimum, we should increase the number of samples to improve the accuracy. This technique is implicitly used in the multi-resolution approach proposed in section 1.6.2. Also, the optimization method should be adapted to the number of samples: a derivative-based method will hardly optimize an energy as the one shown in Fig. 1.28 left, while a direct search method such as Powell's one will be more efficient. Concerning the epipolar tangency criterion, it shows a very smooth and convex energy function since we are near the optimum (see Fig. 1.28 right).

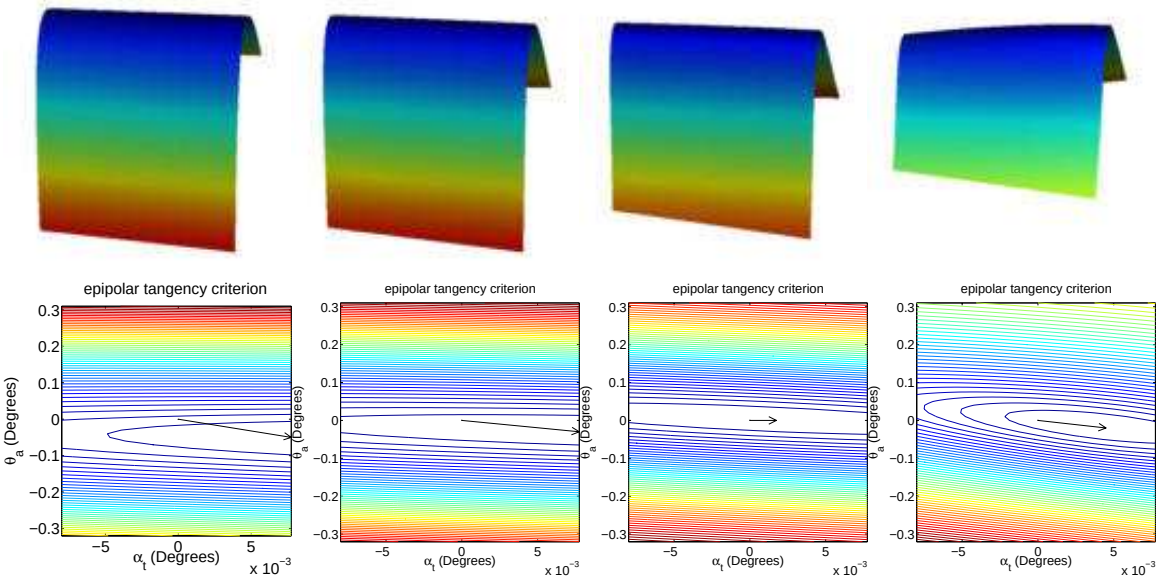


Figure 1.26: *Detail around the maximum peak of the energy shape of the* ***epipolar tangency criterion*** *for the couple of parameters* $(\alpha_{\mathbf{t}}, \theta_{\mathbf{a}})$*, Teapot sequence. From left to right, energy shapes for 4 different sets of peer silhouettes* $\mathcal{N}_1$*,* $\mathcal{N}_3$*,* $\mathcal{N}_6$*, and* $\mathcal{N}_{all}$ *respectively. Top: rendered views. Bottom: corresponding isoconstours of the energy functions.*

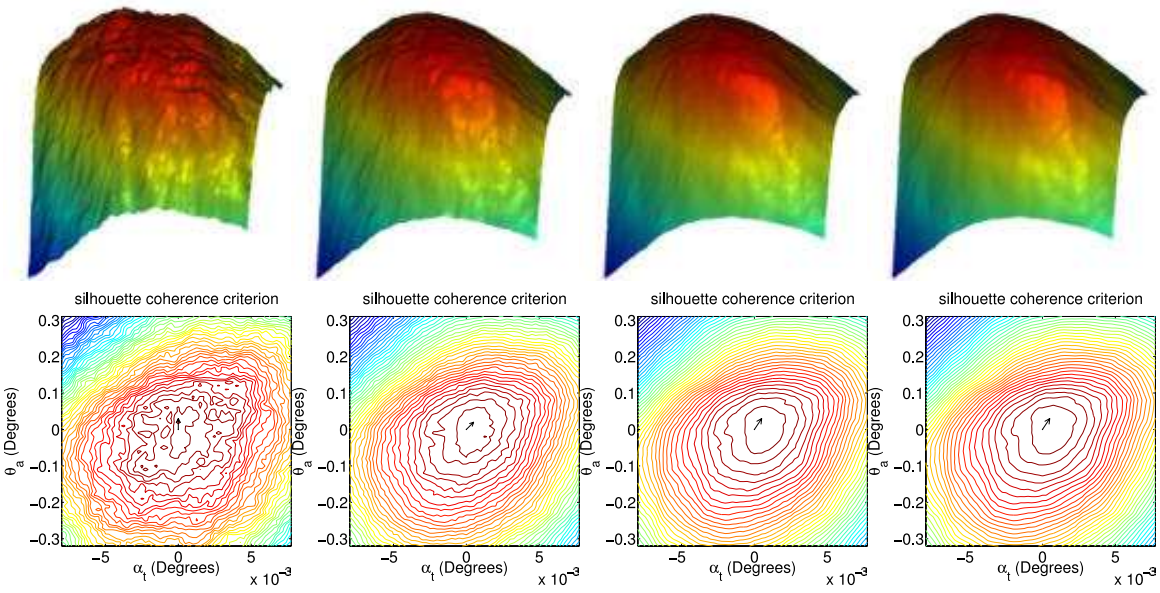


Figure 1.27: *Detail of the energy shape of the* ***silhouette coherence criterion*** *for the couple of parameters* $(\alpha_{\mathbf{t}}, \theta_{\mathbf{a}})$*. Teapot sequence with* $\delta = 0.5$ *pixels. From left to right, energy shapes for 350, 1400, 5600 and 11200 samples per silhouette respectively. Top: rendered views. Bottom: corresponding isocontours of the energy functions.*

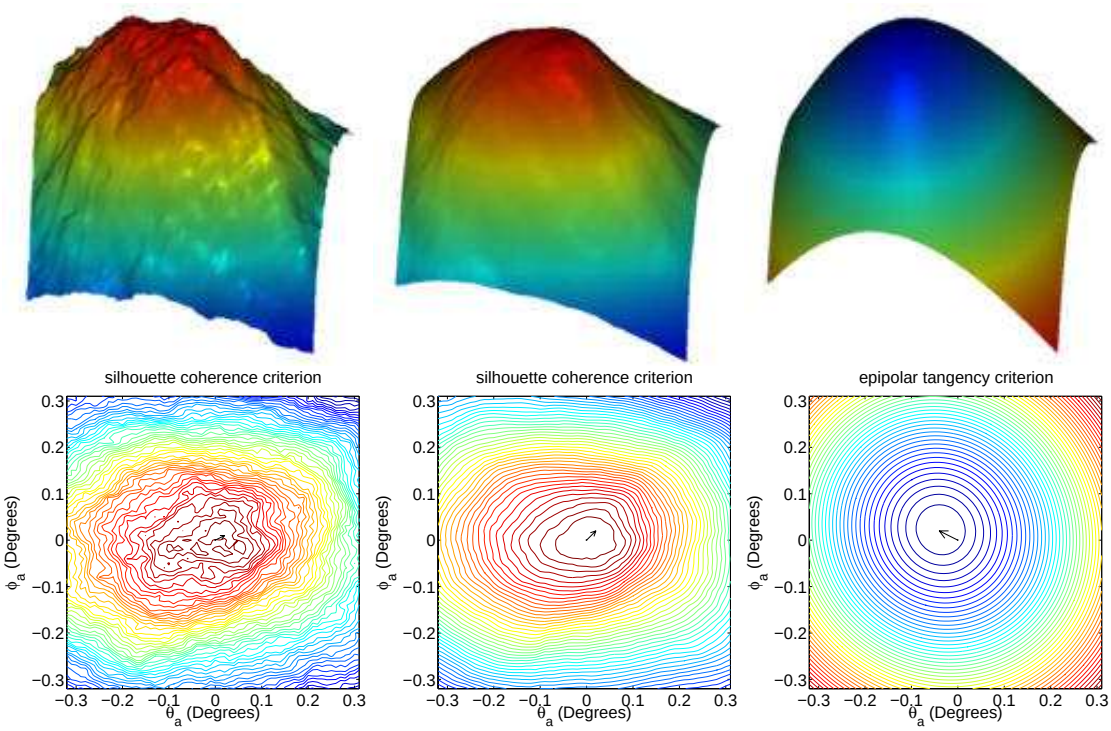


Figure 1.28: *Detail of the energy shapes and corresponding isocontours for the couple of parameters* $(\theta_{\mathbf{a}}, \phi_{\mathbf{a}})$. *Left and middle:* ***silhouette coherence criterion*** *computed with 350 and 11200 samples per silhouette respectively. Right:* ***epipolar tangency criterion****.*

Another important factor that contributes to the shape of the silhouette coherence criterion is the quality of the silhouette extraction. This can be easily illustrated with the synthetic Teapot sequence by drawing the depth intervals computed along the first silhouette S_1, i.e., the depth intervals used to compute $\mathcal{C}_{sc}(S_1, S_1^{\mathcal{V}})$ in subsection 1.6.1. We show in Fig. 1.29 the depth intervals corresponding to the exact, pixelized and regularized silhouettes. We can appreciate how noisy the depth intervals are for the pixelized silhouettes (see Fig. 1.29 middle) in comparison with the exact ones (see Fig. 1.29 left). In figure 1.29 right we observe that much of the noise is filtered when the regularized silhouettes are used.

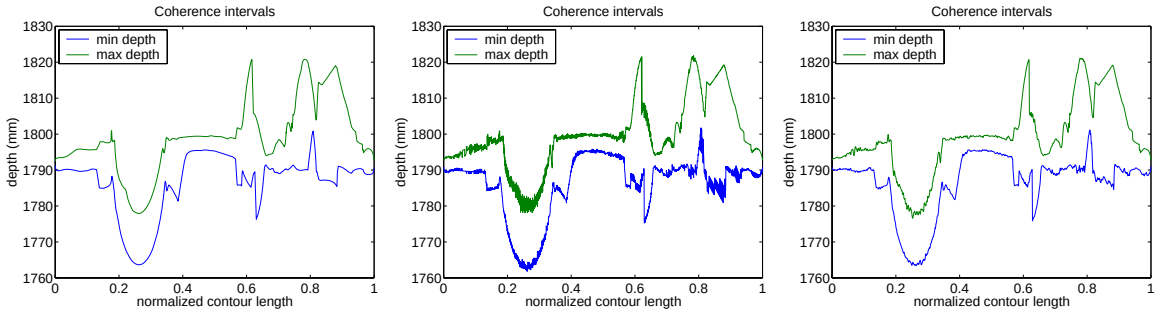


Figure 1.29: *Depth intervals used in the computation of the silhouette coherence with* $\delta = 0.5$ *pixels. From left to right, exact silhouettes, pixelized silhouettes and regularized silhouettes.*

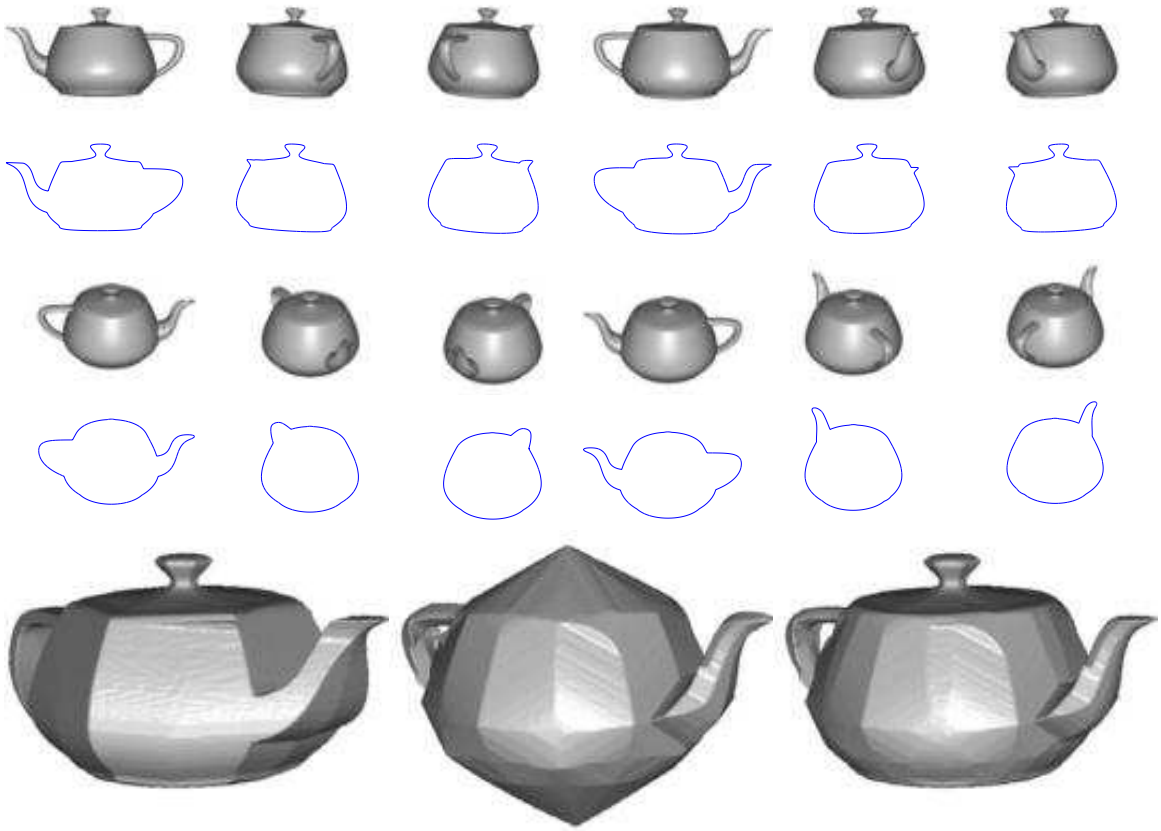


Figure 1.30: *Two different sequences of the Teapot and their corresponding visual hulls. Top: first sequence and the corresponding silhouettes. Middle: second sequence and the corresponding silhouettes. Bottom: from left to right, visual hull of the first sequence, visual hull of the second sequence and visual hull defined by both sequences (intersection of the two first visual hulls).*

1.9 Registration of two Calibrated Sequences

We consider the silhouettes of two different rotation sequences $\mathcal{S}^1$ and $\mathcal{S}^2$ of the same object, each sequence being calibrated independently. We would like to register both sequences in order to reconstruct the object using all the views available. The two sequences are related by: a rotation (Euler angles (α, β, γ)), a translation (t_x, t_y, t_z), and a scaling factor s. This makes 7 parameters to optimize $v = (\alpha, \beta, \gamma, t_x, t_y, t_z, s)$. The mutual silhouette coherence function can be defined as:

$$\mathcal{C}_{sc}(\mathcal{S}^1, \mathcal{S}^2) = \frac{1}{2}\left(\frac{1}{N_1}\sum_{i=1}^{N_1}\mathcal{C}(S_i^1, \mathcal{S}^2) + \frac{1}{N_2}\sum_{j=1}^{N_2}\mathcal{C}(S_j^2, \mathcal{S}^1)\right). \tag{1.35}$$

We present here the results for two different objects: the synthetic Teapot, which we use to evaluate the algorithm (Fig. 1.30), and the Pitcher object (Fig. 1.34).

1.9.1 Synthetic Exact Silhouettes

The Teapot object allows us to measure the theoretical precision of the algorithm. Although we could have used sequences of 36 silhouettes, we have preferred to use only sequences of 6 silhouettes with an angle step of 60 degrees (see Fig. 1.30 top and middle), which gives the corresponding visual hulls shown in Fig. 1.30 bottom, where the rightmost visual hull is defined by the two silhouette sequences, i.e., it is the intersection of the two first visual hulls. left and middle. It is worth noting that, although the original silhouettes have holes (and so do the visual hulls), the silhouettes shown in Fig. 1.30 have only the external contour of the silhouette. Using a small number of silhouettes allows us to show how different the two registered visual hulls can be and how, even if the number of silhouettes is not very high, we can get already good registration accuracy.

As already described in subsection 1.8.2, we have added random noise to the exact silhouettes in order to statistically measure the precision of the discrete silhouette coherence criterion. For each noise variance, we have computed 200 samples in order to obtain reliable results.

Concerning the registration results, we have performed two tests: recovering the Euler angles and the translation (6 parameters), and recovering also the scale (7 parameters).

The results for the Euler angles and translation estimation are shown in Fig. 1.31. For small noise variances, the curves show the same behaviour as when estimating the circular motion, i.e., they are not monotone. As already pointed out in subsection 1.8.2, this is explained by the saturation of the silhouette coherence criterion for a δ value that is large compared with the noise standard deviation. This is confirmed by the shape of the energy near the optimum in Fig. 1.32. When the noise standard deviation is small (0.1 pixels), the silhouette coherence saturates near the optimum (see Fig. 1.32.b). This saturation effect makes that the standard deviation of the recovered parameters is directly related to the size of the saturated plateau in Fig. 1.32.b.

In figure 1.33 we show the registration results when we recover simultaneously the Euler angles, the translation and the scale. The results are slightly worse than when estimating only the Euler angles and the translation. However, the accuracy is still very good. The results of the Euler angles estimation are even improved while the translation is a little bit more penalized by the inclusion of the scale in the set of parameters to optimize.

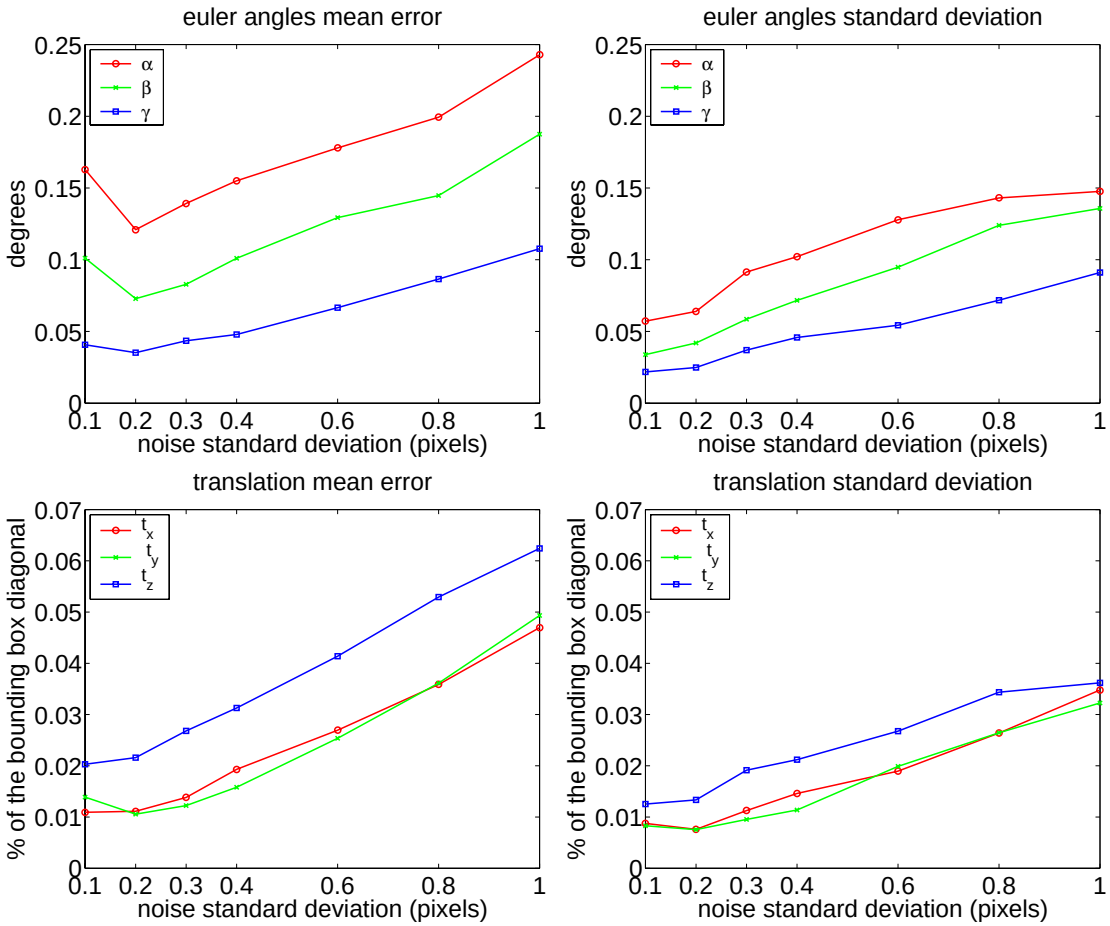
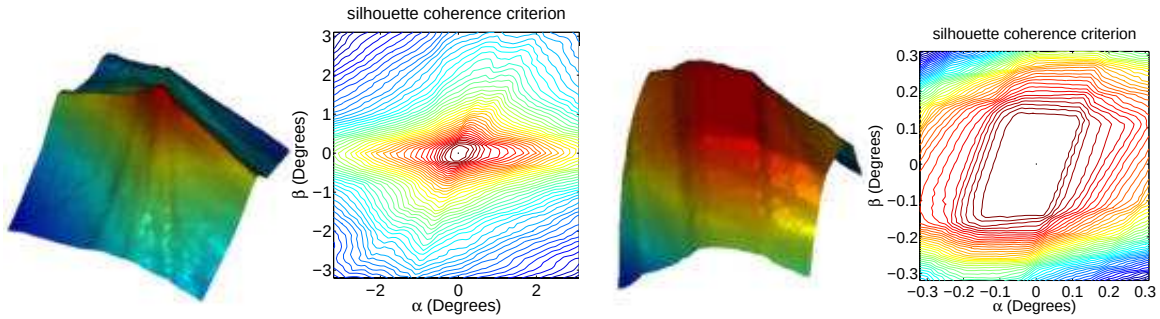


Figure 1.31: *Registration of the Teapot sequences as a function of the noise standard deviation, $\delta = 0.5$ pixels. Top: Euler angles error. Bottom: translation error (percentage of the bounding box diagonal, equal to 161 mm). Left: mean error. Right: standard deviation.*



(a) (b)

Figure 1.32: *Energy shapes of the Teapot sequences for parameters α and β with noisy exact silhouettes (standard deviation of 0.1 pixels), $\delta = 0.5$ pixels. a) Left: silhouette coherence energy shape, right: corresponding isocontours. b) Detail around the optimum.*

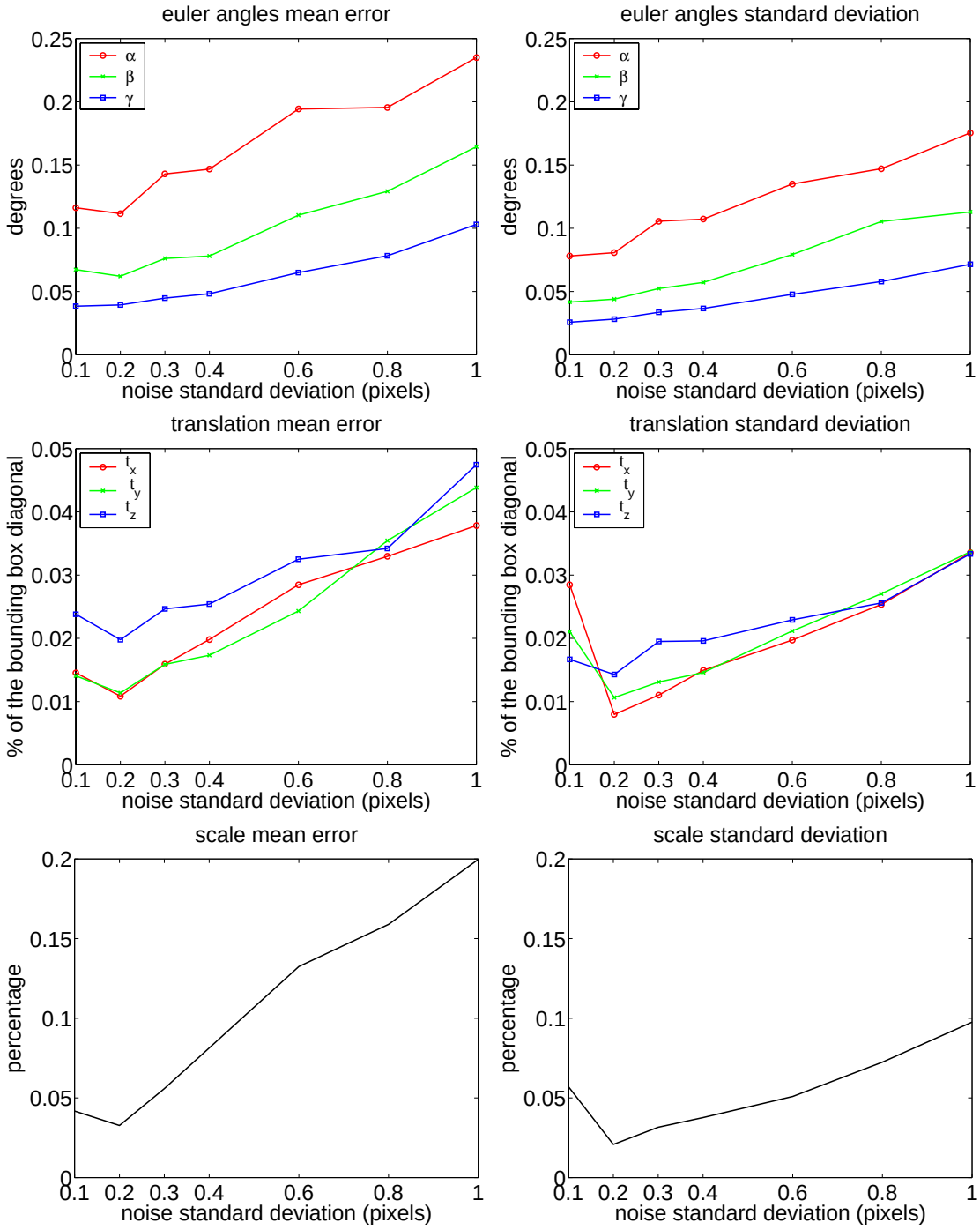


Figure 1.33: *Registration of the Teapot sequences as a function of the noise standard deviation, $\delta = 0.5$ pixels. Top: Euler angles error. Middle: translation error (percentage of the bounding box diagonal, equal to 161 mm). Bottom: scale error. Left: mean error. Right: standard deviation.*

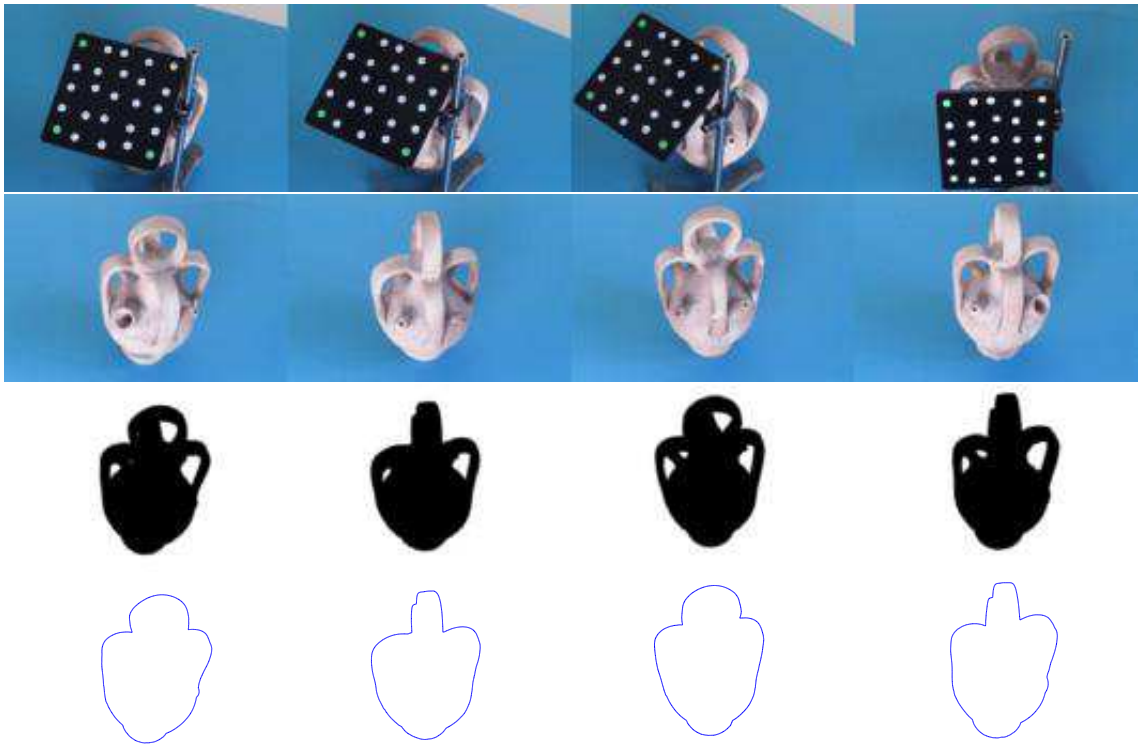


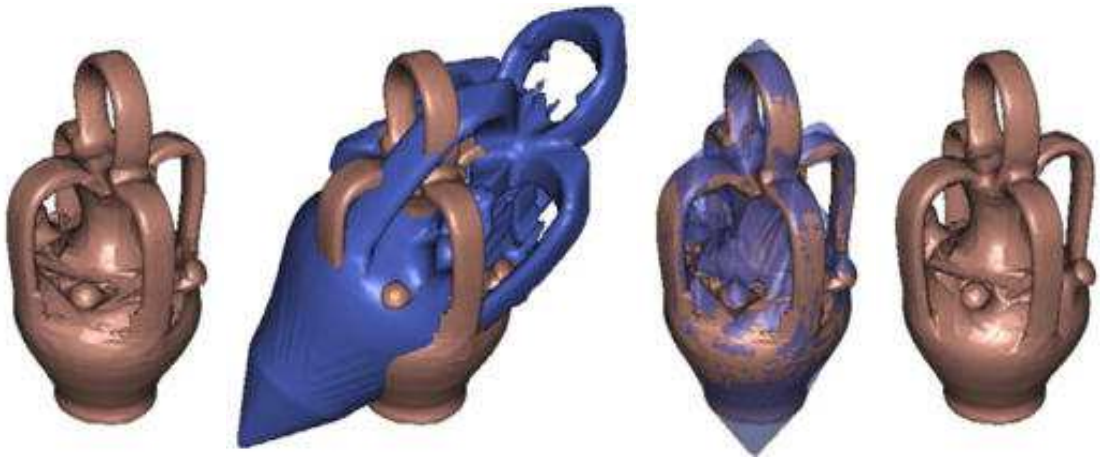
Figure 1.34: *Some of the original images of the second Pitcher sequence (of a total of 36 images). From top to bottom: calibration target images, color images, binarized silhouettes and extracted smooth polygon silhouettes. The first Pitcher sequence is shown in Fig. 1.23.*

1.9.2 Real Silhouettes

We have used the Pitcher sequence shown in Fig. 1.23 and a second sequence of the Pitcher object (see Fig. 1.34). The second sequence is composed of 36 images of 3040x2008 pixels and the silhouettes have been binarized with the same automatic color-segmentation technique as for the first sequence. The object has not been moved between both acquisitions, only the camera. In fact, to recover the relative motion between both sequences, two pictures of a fixed calibration pattern (one with the first camera position and another with the new camera position) have been used in the calibration algorithm. These two pictures correspond to the rightmost calibration image in Fig. 1.23 top, and the rightmost calibration image in Fig. 1.34 top. Note that the relative position of the Pitcher to the calibration pattern is the same in both images, so we can recover the transformation between the sequences and compare it with the one obtained by maximizing the silhouette coherence criterion. We observe the difference between the silhouette coherences obtained for the two sequences: 98.19% and 90.99% respectively (see Fig. 1.35.a and Fig. 1.35.b. The lower score of the second sequence is due to the worse quality of its extracted silhouettes. Despite these segmentation errors, the accuracy obtained after optimization is rather good (see table 1.6). As we can appreciate in figure 1.35.b, the huge disparity of the reconstructed visual hulls makes the use of other registration methods such as the ICP [Besl and

Pitcher	rotation (degrees)			translation (mm)			scale
	α	β	γ	t_x	t_y	t_z	s
initial	103.000	6.000	8.000	-10.000	460.000	650.000	1.20000
calibrated	63.776	-4.605	-2.497	-36.680	478.329	656.895	1.00000
recovered	63.813	-4.560	-2.487	-36.492	478.125	656.915	1.00014
error	0.037	0.045	0.009	0.188	0.204	0.020	0.00014

Table 1.6: *Registration results for the Pitcher sequence.*



a b c d

Figure 1.35: *Pitcher of 30 cm bounding size. (a) First reconstructed visual hull; coherence of 98.19%. (b) Unregistered second reconstructed visual hull; coherence of 90.99%; mutual coherence of 54.46%. (c) Reconstructed visual hulls after registration; mutual coherence of 95.66%. (d) Resulting reconstructed visual hull after registration.*

McKay, 1992] very difficult.

Once we have registered the two sequences, we can compute a more accurate visual hull using the silhouettes of both sequences, which improves the quality of the reconstructed models see (Fig. 1.35.d).

1.10 Conclusions

A new approach to silhouette-based camera motion and focal length estimation under circular motion has been developed. It is based on the definition of the silhouette coherence concept, defined as a similarity between a set of silhouettes and the silhouettes of their visual hull. This approach has been successfully tested for different estimation problems such as motion and focal recovery from a single rotation sequence or independent sequence registration. The high precision of the estimation results are due to the use of the full silhouette contour in the computation, whereas classic methods based on epipolar tangencies use a few tangent points at most. We have validated the method using real sequences, which enables us to reconstruct 3D objects using the 3D object modeling technique developed in the next chapter. It allows us to completely

reconstruct a 3D object only from a sequence of images under circular motion, without the need of a calibration target.

Our main contributions are:

- the definition and implementation of the concept of silhouette coherence,
- the detailed comparison between the epipolar tangency criterion and the proposed silhouette coherence criterion,
- the study of the accuracy of the epipolar tangency criterion as a function of the number of pairs of silhouettes we use in the criterion.

Concerning the **epipolar tangency criterion**, we show that a fast implementation based on a polygonal representation of the silhouettes can provide very good results. Also, although in the original paper of [Wong and Cipolla, 2001] the criterion is only used to estimate camera motion, we show that it is in fact also possible to estimate the focal length, as demonstrated by the numerous experiments with synthetic and real silhouettes. Finally, as a difference with the original formulation of [Wong and Cipolla, 2001], we show that it is worth using all the available pairs of silhouettes to compute the criterion rather than using the closest pairs, but at the price of a more expensive computation.

As a perspective, the proposed **silhouette coherence criterion** can still benefit from several improvements. An important issue is to get rid of the discrete nature of the current implementation of the silhouette coherence criterion. The main idea would be to compute the exact visual hull silhouettes as closed polygons. It would allow us to define a continuous silhouette coherence by just comparing the two polygons defining the original silhouette and the visual hull silhouette. To compute the exact silhouette of the visual hull, we can proceed as in [Franco and Boyer, 2003], using the existing technique of ray casting. The greedy algorithm would be:

- compute the 3D visual hull patches generated by intersecting each silhouette with the other $n-1$ silhouettes,
- project the edges of the 3D patches into the desired view,
- compute the minimal polygon that contains all the 2D projected edges.

Finally, based on the accuracy results for the different variables in the optimization process, we should be able to improve the optimization algorithm by using a derivative-based optimization algorithm with a robust estimation of the derivatives adapted to this problem. Also, we will further investigate the robustness of this technique for more complicated scenarios than circular motion.

Chapter 2

Silhouette and Stereo Fusion for 3D Object Modeling

In this chapter we present the proposed 3D object modeling technique. Since the method is based on a deformable model, each section of the chapter addresses one basic block of the deformable model framework applied to our problem. In Section 2.1 we describe the related work and how our proposed technique compares to it. In Section 2.2 we give a fast overview of the basis of deformable models and justify the choice of a classic deformable model instead of a more recent and popular level-set method. In Section 2.3 we describe how to initialize the deformable model while in Sections 2.4 and 2.5 we develop the two external forces that will drive the deformable model: a texture-based force and a silhouette-based force. In Section 2.6 we explain the regularization forces and how to formulate them in the case of a triangle mesh. The complete iteration process is summarized in Section 2.7. The last step consists of computing a texture map from a given set of images and a 3D model, which is described in Section 2.8.

2.1 Related Work

Acquiring 3D models is not an easy task and abundant literature exists on this subject. There are three major approaches to the problem of 3D real model representation: pure image-based rendering techniques, hybrid image-based techniques, and 3D modeling techniques.

Pure image-based rendering techniques as [Chen and Williams, 1993] and [McMillan and Bishop, 1995] try to generate synthetic views from a given set of original images. They do not estimate the real 3D structure behind the images, they only *interpolate* the given set of images to generate a synthetic view.

Hybrid methods make a rough estimation of the 3D geometry and mix it with a traditional image-based rendering algorithm in order to obtain more accurate results. The authors of [Debevec et al., 1996] propose the FAÇADE algorithm that allows modeling architectural scenes from a sparse set of still photographs. They reconstruct a rough estimation of the geometry and use it to produce renderings of the scene with a view-dependent texture mapping technique. In [Matusik et al., 2000], an image-based

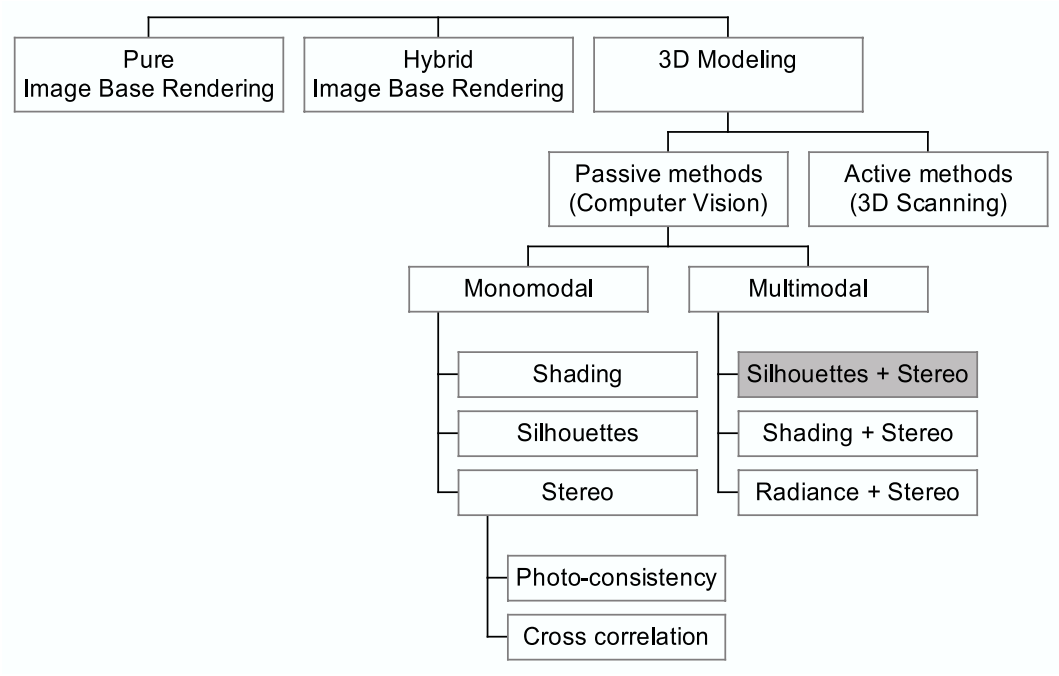


Figure 2.1: *Overview of existing 3D real model representation techniques.*

rendering technique is proposed to generate additional views starting from a set of images based on the concept of visual hull without an explicit 3D reconstruction. This method has been extended by [Slabaugh et al., 2002] and [Li et al., 2002] by improving the visual hull with stereo information. For both pure and hybrid IBR methods, the goal is to generate coherent views of the real scene, rather than obtain metric measures of it.

In opposition to these techniques, the third class of algorithms tries to recover the full 3D structure. Among the 3D modeling techniques, two main groups are to be distinguished: active methods and passive ones. Active methods use a controlled source of light such as a laser or a coded light in order to recover the 3D information [Schmitt et al., 1986, Curless and Levoy, 1996, Levoy et al., 2000]. Passive methods use only the information contained in the images of the scene [Slabaugh et al., 2001]. They can be classified according to the type of information they use. A first class consists of the shape from silhouette methods. The first time that silhouettes were used for 3D modeling is the Ph.D. thesis of [Baumgart, 1974], where he proposed to construct the surface defined by a set of silhouettes using polyhedra. Starting from that, there has been a wide variety of methods to construct 3D models from silhouettes, ranging from octrees [Potmesil, 1987], differential analysis [Vaillant and Faugeras, 1992], 3d grid [Niem and Wingbermuhle, 1997, Matsumoto et al., 1997] or splines [Sullivan and Ponce, 1998]. However, the theoretical properties of the surface defined by a set of silhouettes were first studied by [Laurentini, 1994] with the definition of the visual hull concept. It is shown that the visual hull is an upper bound of the real object and that it offers a better approximation than the convex hull, i.e., the visual hull is contained between the convex hull and the real surface. Methods based on silhouettes are fast and

robust but, because of the type of information used, they are limited to simple shaped objects. We can find commercial products based on this technique. Another approach includes the shape from shading methods [Horn and Brooks, 1989]. They are based on the diffusing properties of Lambertian surfaces. They mainly work for 2.5D surfaces and need very constraining hypotheses such as the use of orthographic cameras or punctual light sources. In fact, the strength of the constraints required for this kind of methods to work makes them useless under realistic conditions. Recent improvements on the algorithms, in particular, the case of projective cameras as in [Prados and Faugeras, 2003] or [Courteille et al., 2004], open some new possibilities for practical applications. A third class of methods uses the color information of the scene. The color information can be used in different ways, depending on the type of scene we try to reconstruct. A first way is to measure color consistency to carve a voxel volume [Seitz and Dyer, 1997], [Seitz and Kutulakos, 1998] or [Matsumoto et al., 1999]. But they only provide an output model composed of a set of voxels, which makes it difficult to obtain a good 3D mesh representation. In order to solve this problem, the authors of [Zhang and Seitz, 2001] and [Yezzi et al., 2002] propose using the color consistency measure to guide a deformable model. An additional problem of color consistency algorithms is that they compare absolute color values, which makes them sensitive to light condition variations. A different way of exploiting color is to compare local variations of the texture, as done in cross-correlation methods [Faugeras and Keriven, 1998, Sarti and Tubaro, 2002]. As a specialization of the color-based group, there are specific methods that try to use, at the same time, another type of information such as silhouettes [Liedtke et al., 1991, Fua and Leclerc, 1996, Matsumoto et al., 1999, Cross and Zisserman, 2000, Isidoro and Sclaroff, 2003], shading [Fua and Leclerc, 1995, Jin et al., 2000] or radiance [Yezzi and Soatto, 2001, Soatto et al., 2003]. Although very good results are obtained, the quality is still limited, the two main problems being the extraction of the 3D data from the images and the way the fusion of different data is done. The authors of [Matsumoto et al., 1999] use a 3D volume that they first carve with silhouettes and then use a photo-consistency measure to carve away additional voxels. The same procedure is used by [Cross and Zisserman, 2000], but the photo-consistency measure used is in fact a cross-correlation measure, which increases the robustness. These two last algorithms use a volume grid for the fusion of data. In [Liedtke et al., 1991], a 3D model is initialized with the visual hull and then each vertex is displaced to maximize a photo-consistency measure. The main problems of this method are the lack of robustness of the photo-consistency measure employed and the poor quality of the deformed mesh. In [Isidoro and Sclaroff, 2003] the 3D model is also initialized with the visual hull. Then the model is iteratively deformed, where the force at each vertex is computed using the shortest displacement that maximizes a photo-consistency criterion along several epipolar rays. Concerning the fusion of silhouettes and stereo, the authors [Ilic and Fua, 2003a, Ilic and Fua, 2003b, Plänkers and Fua, 2003] have recently proposed an interesting fusion framework using implicit surfaces. The authors [Fua and Leclerc, 1995] use a deformable model to combine stereo and shading. The fusion is performed by weighting two complementary terms: a stereo-based force and a shading-based force. In [Soatto et al., 2003] the approach

is extended to deal with the more general concept of radiance, which gives quite good results.

According to the presented taxonomy, the algorithm of 3D reconstruction that we present in this chapter can be classified as a passive multimodal method that uses silhouettes and stereo information. We perform the fusion of both silhouettes and texture information by a deformable model evolution. The main difference with the methods mentioned above is the way the fusion is accomplished, which enables us to obtain very high quality reconstructions. A similar approach to our work has been recently proposed by [Nobuhara and Matsuyama, 2003]. A deformable model is also used to fuse texture and silhouette information. However, the objectives of their work are not the same as ours. They are interested in dynamic 3D shape reconstruction of moving persons while our specific aim is high quality 3D and color reconstructions of museological objects. In their case they need fast reconstructions, which implies low resolution. In our case the reconstruction can be made off-line, so we can afford high quality models.

2.2 Algorithm Overview

The goal of the system is to be able to reconstruct a 3D object from a sequence of geometrically calibrated images. To do so, we dispose of several types of information contained in the images. Among all the information available, shading, silhouettes and features of the object are the most useful for shape retrieval. Shading information needs a calibration of the light sources, which implies an even more controlled environment for the acquisition. The use of the silhouettes requires a good extraction of the object from the background, which is not always easy to accomplish. Finally, of all the features available from an object, such as texture, points, contours, or more complicated forms, we are mainly interested in texture, whenever it exists. Since exploiting shading imposes heavy constraints in the acquisition process, the information we will use consists of silhouettes and texture. The next step is to decide how to mix these two types of information to work together. As we will see, this is not an easy task because those types of information are very different, almost "*orthogonal*": when we consider the normal to the surface, silhouettes give a maximum of information about the surface points whose normal is orthogonal to the viewing direction, whereas image texture is more useful in regions where the surface normal is parallel to the viewing direction.

2.2.1 Classical Deformable Models vs. Geometric Deformable Models

Deformable models offer a well-known framework to optimize a surface under several kinds of information. Two different related techniques can be used depending on the way the problem is posed: a classical deformable approach, such as snakes [Kass et al., 1988] or balloons [Cohen, 1991], or a geometric deformable model approach using level-sets [Caselles et al., 1993], [Malladi et al., 1995]. The main advantage of the classical

deformable approach is its simplicity of implementation and parameter tuning. Its main drawback is the constant topology constraint, i.e., the fact that the genus of the surface cannot change during the evolution. Level-set based algorithms have the advantage of an intrinsic capability to overcome this problem but its main disadvantages are the computation time and the difficulty to control the topology. Computation time can be addressed using a narrow band implementation [Adalsteinsson and Sethian, 1995]. Controlling the topology is a more difficult problem but, the authors [Han et al., 2003] have recently proposed an interesting way of avoiding topology changes in level set methods. Despite these improvements, level-set methods remain complex and expensive when dealing with high resolution deformable models (volume grids of 9 to 11 levels, i.e., volumes of 512x512x512 to 2048x2048x2048). Since this is our main objective, we have chosen to use the classical 3D deformable model [Terzopoulos et al., 1987, Cohen et al., 1992] as framework for the fusion of silhouette and stereo data. This implies that the topology has to be completely recovered before the deformable model evolution occurs as discussed in Section 4. Since the proposed way to recover the right topology is the visual hull concept, the topology recovery will depend on the intrinsic limitations of the visual hull. This implies that there exist objects for which we are unable to recover the correct topology (no silhouettes seeing a hole) that could be potentially reconstructed using a level-set method (the correct topology being recovered with the stereo information). However, we observe that, in practice, if we dispose of enough views, the visual hull provides the correct topology for most of the common objects; therefore, this is not a severe handicap.

2.2.2 The Classical Deformable Model Approach

The deformable model framework allows us to formulate our problem as a global energy minimization problem. The total energy term $\mathcal{E}$ is composed of two different types of energy: an internal energy $\mathcal{E}_{int}$, and an external energy $\mathcal{E}_{ext}$. The internal energy measures how *regular* the surface is while the external energy depends only on the external data related to the particular problem we try to solve. In general, this energy will be non-convex with possible local minima. The internal energy helps to "convexify" the total energy term and to obtain a final well-shaped surface. Since, as discussed previously, we want to exploit silhouettes and texture, the external energy for this particular problem will be composed of two terms, one related to the silhouettes $\mathcal{E}_{sil}$ and another that takes texture into account $\mathcal{E}_{tex}$. The minimization problem is posed as finding the surface S of $\mathbb{R}^3$ that minimizes the energy $\mathcal{E}(S)$ defined as follows:

$$\mathcal{E}(S) = \mathcal{E}_{ext}(S) + \mathcal{E}_{int}(S) = \mathcal{E}_{tex}(S) + \mathcal{E}_{sil}(S) + \mathcal{E}_{int}(S), \tag{2.1}$$

Minimizing equation 2.1 means finding S_{opt} so that:

$$\begin{aligned} \nabla\mathcal{E}(S_{opt}) &= \nabla\mathcal{E}_{tex}(S_{opt}) + \nabla\mathcal{E}_{sil}(S_{opt}) + \nabla\mathcal{E}_{int}(S_{opt}) = 0, \\ &= \mathcal{F}_{tex}(S_{opt}) + \mathcal{F}_{sil}(S_{opt}) + \mathcal{F}_{int}(S_{opt}) = 0, \end{aligned} \tag{2.2}$$

where ∇ is the gradient operator, and $\mathcal{F}_{tex}$, $\mathcal{F}_{sil}$ and $\mathcal{F}_{int}$ represent the forces that drive the snake. Equation 2.2 establishes the equilibrium condition for an optimal

Figure 2.2: *Different object initializations. From left to right: bounding box+convex hull, convex hull+visual hull, visual hull + real object.*

solution, where the three forces cancel each other out. A solution to equation 2.2 can be found by introducing a time variable t for the surface S and solving the following differential equation:

$$\frac{\partial S}{\partial t} = \mathcal{F}_{tex}(S) + \mathcal{F}_{sil}(S) + \mathcal{F}_{int}(S). \tag{2.3}$$

The discrete version becomes:

$$S^{k+1} = S^k + \Delta t(\mathcal{F}_{tex}(S^k) + \mathcal{F}_{sil}(S^k) + \mathcal{F}_{int}(S^k)). \tag{2.4}$$

Once we have sketched the energies that will drive the process, we need to make a choice for the representation of the surface S. This representation defines the way the deformation of the snake is done at each iteration. Among all the possible surface representations, the triangular mesh and the simplex mesh [Delingette, 1994] are very common due to their simple implementation and well known properties. Triangular meshes dispose of powerful operators to add and delete edges, points and triangles. These operators are very useful to control the deformable surface evolution and keep the triangular surface within the desired limits of edge size and triangle shape. Simplex meshes allow computing surface properties more easily since the connectivity of any point of the mesh is always the same (the simplex mesh is the dual graph of the triangle mesh: in a triangle mesh, a face has always 3 neighbor faces while in a simplex mesh, a point has always 3 neighbor points). Simplex meshes are useful for computing the internal force $\mathcal{F}_{int}$, however, they are particularly painful to do operations like adding or deleting points and edges. This is the reason why we prefer to use a traditional triangle mesh representation and use an internal force whose computation is a little more complicated, but not in excess as we will show in Section 2.6.

To completely define the deformation framework, we need an initial value of S, i.e., an initial surface S_0 that will evolve under the different energies until convergence.

We describe the snake initialization in Section 2.3, the force driven by the texture of the object in Section 2.4 and the force driven by the silhouettes in Section 2.5. The internal force is detailed in Section 2.6, the mesh evolution in Section 2.7 and the texture mapping procedure in Section 2.8.

Figure 2.3: *Example of objects whose topology cannot be correctly captured by the visual hull concept.*

2.3 Snake Initialization

The first step in our minimization problem is to find an initial surface *close enough* to the object surface in order to guarantee a good convergence of the algorithm. This step is not trivial since a deformable model can easily get stuck into a local minima [Berger and Mohr, 1990]. *Close* has to be considered in a geometrical and topological sense. The geometric distance between the initial and the object surfaces has to be reduced in order to limit the number of iterations in the surface mesh evolution process and thereby the computation time. The topology of the initial surface is also very important since classical deformable models maintain the topology of the mesh during its evolution. On the one hand, this imposes a strong constraint that makes the initialization a very important step since the initial surface must capture the topology of the object surface. On the other hand, the topology-constant property of a classical snake provides more robustness to the evolution process.

If we make a list of possible initializations (see Fig. 2.2), we can establish an ordered list, where the first and simplest initialization is the bounding box of the object. The next simplest surface is the convex hull of the object. Both the bounding box and the convex hull are unable to represent surfaces with a genus greater than 0. A more refined initialization, which lies between the convex hull and the real object surface is the visual hull [Laurentini, 1994]. The visual hull can be defined as the intersection of all the possible cones containing the object. In practice, a discrete version is usually obtained by intersecting the cones generated by back projecting the object silhouettes of a given set of views (see Section 2.3.1). As a difference with the convex hull, it can represent surfaces with an arbitrary number of holes. However, this does not imply that it is able to completely recover the topology of the object and, what is even worse, the topology of the visual hull depends on the discretization of the views (see Fig. 2.4).

Computing the visual hull from a sequence of images is a very well known problem of computer vision and computer graphics [Martin and Aggarwal, 1983, Potmesil, 1987, Niem and Wingbermuhle, 1997, Matusik et al., 2000]. Different approaches exist, depending on the type of output, way of representation and fidelity to the theoretical visual hull. In our case, we are interested in methods producing good quality meshes

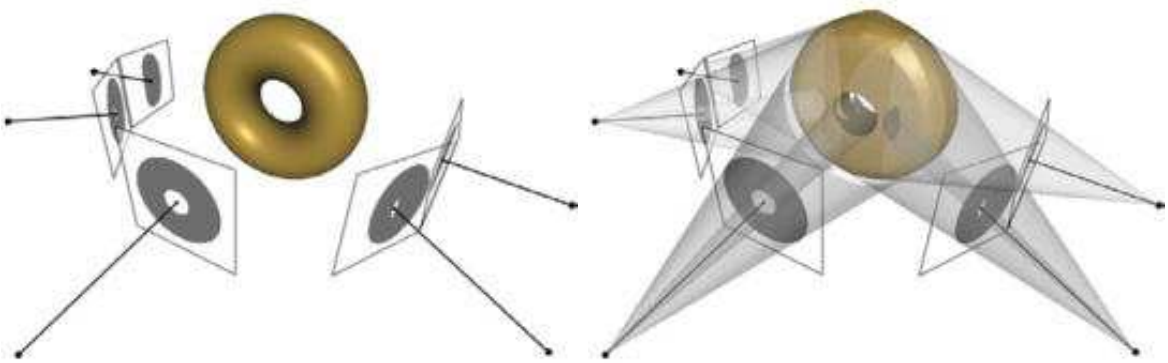


Figure 2.4: *Example of topological problem arising with a finite number of cameras. The original torus topology on the left is not correctly recovered with the discrete visual hull on the right.*

(manifold, smooth, triangles with aspect ratio[1]close to 1), even if the fidelity is not very high. In addition to a good quality mesh, another primary requirement is to obtain the right topology. Volume carving methods are a good choice because of the high quality output meshes that we can obtain through a marching cube [Lorensen and Cline, 1987] or marching tetrahedron algorithm. The degree of precision is fixed by the resolution of the volume grid, which can be adapted according to the required output resolution. But this adaptability can also generate additional problems of topology: if the resolution of the grid is low compared to the size of the visual hull structures, the aliasing produced by the sub-sampling may produce topological artifacts that the theoretic visual hull does not have. To sum up, three different sources of deviation may arise between the real object topology and the computed visual hull topology:

- **Errors due to the nature of the visual hull** (see Fig. 2.3). Real objects may have holes that cannot be seen as a silhouette hole from any point of view. The visual hull will then fail to represent the correct topology for this kind of object.

- **Errors due to the use of a finite number of views** (see Fig. 2.4). They can be solved by having the adequate points of view that allow recovering the right topology of the real object.

- **Errors due to the implementation algorithm** (see Fig. 2.5). They are caused by the numerical precision or the sub-sampling of the silhouettes. They can be avoided by increasing the precision of the algorithm or by filtering the silhouettes.

[1]The aspect ratio of a triangular element is defined as the ratio of the circumradius of the triangle to twice its inradius. Hence the aspect ratio of an equilateral triangle is exactly 1.

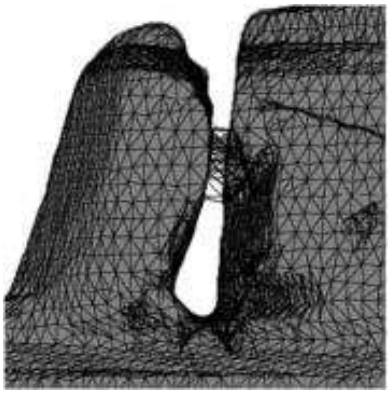


Figure 2.5: *Bad topology caused by an insufficient resolution of the visual hull construction algorithm. We show in gray the original silhouette and in black wireframe the reconstructed visual hull mesh.*

2.3.1 Visual Hull Computation

As stated previously, we are mainly interested in computing good quality visual hull meshes rather than exact visual hull meshes as in [Lazebnik et al., 2001]. In practice, one of the best ways to obtain good meshes is to first construct a discrete 3D volume and then mesh it. The main disadvantage of volume-based methods is the sampling and possible aliasing artifacts that may arise if the volume resolution is not adapted to the size of the structures we want to recover. Since we do not know the size of the 3D structures in advance, this 3D constraint has to be translated into an image constraint: the projection of a voxel into the images gives the size in pixels of the silhouette structures we can recover. This means that if the voxel size after projection is 10 pixels, we cannot expect to recover holes or silhouette details smaller than 10 pixels. Moreover, to avoid aliasing artifacts such as the one shown in Fig. 2.5 right, for a given voxel size, silhouettes have to be filtered using mathematical morphology operators.

Among all the possible implementations, using an octree for the volume representation such as in [Potmesil, 1987] provides good computation times and low memory storage space.

We dispose of a set of n silhouettes S_i and their corresponding projection matrices P_i. In order to construct the octree volume, the algorithm needs two additional input data: the level of detail, e.g., the size of the voxel, and an initial bounding box. The level of detail is in general always fixed for a given image size, depending on the desired resolution and the size of the silhouettes. As an example, a common choice for 2008x3040 images is 8 to 9 levels of depth, i.e., the equivalent of a voxel grid of size 256x256x256 or 512x512x512. More difficult is the initial guess of the bounding box. Since we do not dispose of any 3D information, we have to infer an initial bounding box only from the set of silhouettes. This can be done by considering the 2D bounding boxes of each silhouette. The 3D back projection of a 2D bounding box can be seen as the volume defined by 4 3D planes (see Fig. 2.6). As a consequence, the back projection of n 2D bounding boxes defines a 3D convex hull formed by $4n$

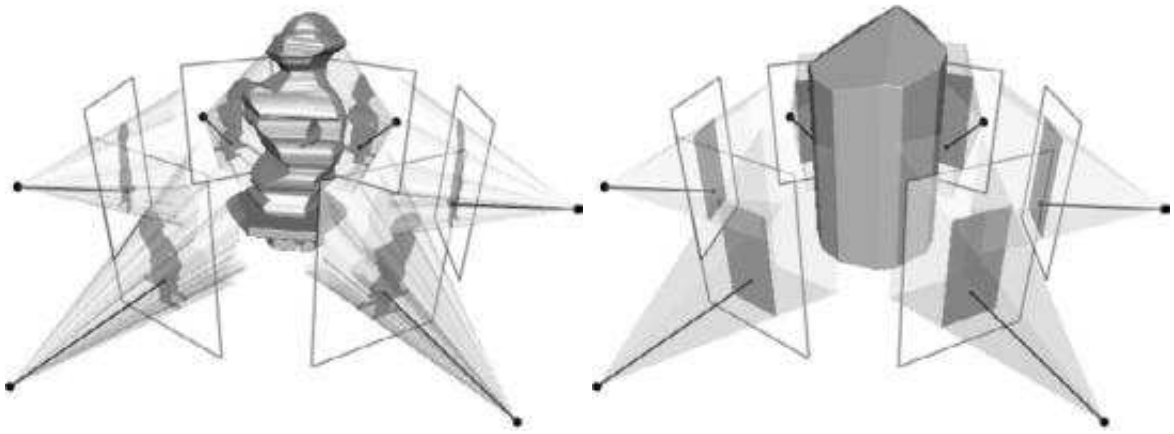


Figure 2.6: *Visual hull of a set of 6 silhouettes (left) and the corresponding convex/visual hull of the silhouette 2D bounding boxes (right). The bounding box of the convex hull defined by the silhouette 2D bounding boxes can be analytically computed using a simplex algorithm.*

planes. The bounding box of the convex hull can be analytically computed by a simplex optimization method [Press et al., 1992] for each of the 6 variables defining the bounding box. Each plane $\mathbf{p}^i = (p^i_x, p^i_y, p^i_z, p^i_w)$ induces a linear constraint of the type:

$$\mathbf{p}^i\mathbf{x} \leq 0, \tag{2.5}$$

where the objective function to maximize depends on which variable of the bounding box we want to recover. If, for example, we want to compute the largest x coordinate of the bounding box, the linear problem will consist of maximizing the objective function "$f_{obj} = x$" subjected to the $4n$ linear constraints:

$$p^i_x \cdot x + p^i_y \cdot y + p^i_z \cdot z + p^i_w \leq 0, \;\; i = \{1, \cdots, 4n\}. \tag{2.6}$$

Starting from the 8 children of the cube defined by the bounding box, the octree approach subdivides a cube into 8 children whenever it is *on* the isosurface, and iterates the process recursively until the maximum level of depth is reached. In the case of a visual hull construction, we have to define an isosurface function that corresponds to the visual hull surface. As we have seen in Chapter 1, a silhouette is only capable of telling us which part of the volume *is not* the object. As a result, if we code a silhouette image with -1 for the foreground, i.e., the silhouette itself, and 1 for the background, we can construct a visual hull isolevel function for a given 3D point v as:

$$f_{iso}(v) = \max_i S_i(P_i v), i = \{1, \cdots, n\}, \tag{2.7}$$

the visual hull being defined as the 0-isolevel of f_{iso}. Based on this definition, a given cube can be tagged with 3 different states according to the f_{iso} value of the corners $c_{i=\{1,\cdots,8\}}$ that define the cube[2]:

[2]In order to reduce aliasing effects, the current algorithm oversamples the edges of the cube up to the maximum octree level such that the cube tag decision is based also on the f_{iso} values of the intermediate sampled points along the edges and not only on the corner values.

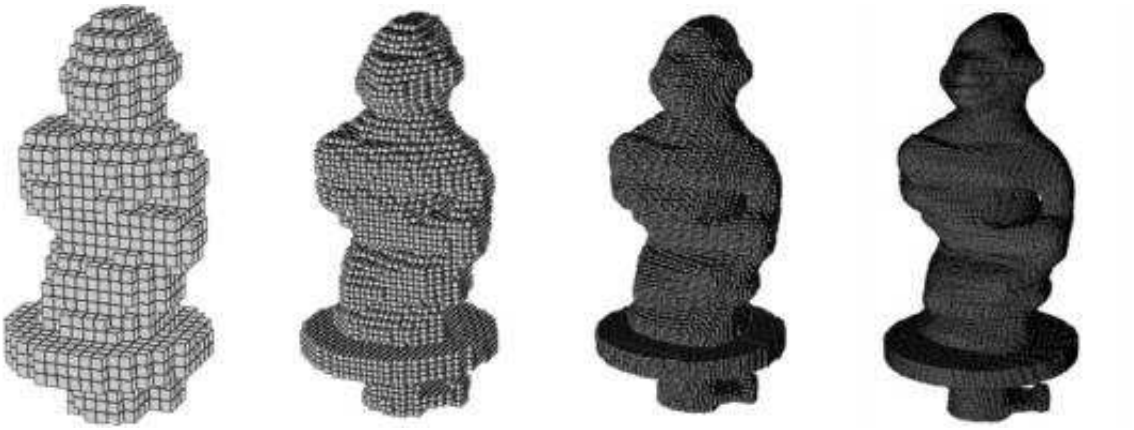


Figure 2.7: *Octree construction of a visual hull. From left to right, the level of detail ranges from 5 to 8 depth levels.*

in	$f_{iso}(c_i) < 0 \; \forall i$	all the silhouettes see it in.
on	$\exists i, j$ $f_{iso}(c_i) < 0, f_{iso}(c_j) > 0$	no silhouette sees it completely out but at least one silhouette sees it partially out.
out	$f_{iso}(c_i) > 0 \; \forall i$	at least one silhouette sees it completely out.

To evaluate a given cube, we project it into all the silhouettes to assign it one of the 3 available tags. If the cube is *on* and the maximum depth is not still reached, we subdivide it and recursively test its children. At the end, only the cubes that are on the visual hull surface have been subdivided. We can see the result of this step for different levels of resolution in figure 2.7.

Once we have constructed the octree, the next step is to mesh it. The basic algorithm of marching cubes consists of creating a signature for each cube that defines its state, i.e, which corners c_i are *in* and which are *out*, and computing the exact points where the isosurface cuts the cube edges. Then, based on a look-up table that contains all the possible configurations ($2^8 = 256$ different configurations), each cube is meshed individually and all the individual patches are fused to create a complete mesh. The only problem with this procedure is that there exist some ambiguous configurations for which the cube mesh is not unique, and, if we do not dispose of more information, we cannot select the right one. The consequence of this ambiguity is the possibility of generating meshes that are not always manifold, which is very bad for the application we are considering. The problem resides in the fact that the 12 edges that compose a cube do not suffice to characterize the surface that traverses it. The best solution to this problem is a variant of the marching cubes that uses a tetrahedron decomposition. Each cube is decomposed into 5 tetrahedra and each tetrahedron is meshed individually (see Fig. 2.8). In practice, the tetrahedron decomposition algorithm is the same as the classic marching cubes one but, instead of computing only 12 edge intersections, we need to compute also 6 additional diagonals on the cube faces, which makes a total of 16 edges. In fact, it is the diagonals that provide enough information to resolve the ambiguity about where the surface is. However, dealing with diagonals introduces more complexity in the algorithm since the diagonals are not the same

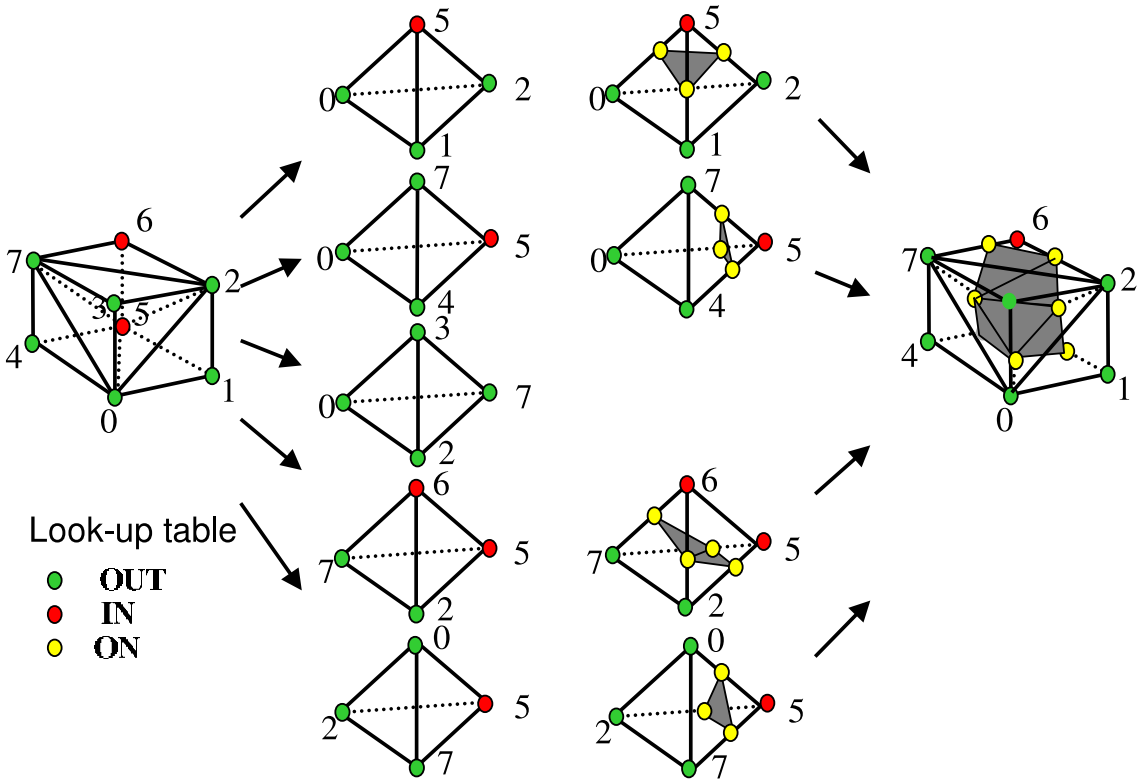


Figure 2.8: *Marching cubes algorithm using tetrahedron decomposition. The zero level intersection (yellow points) are found using a dichotomy approach.*

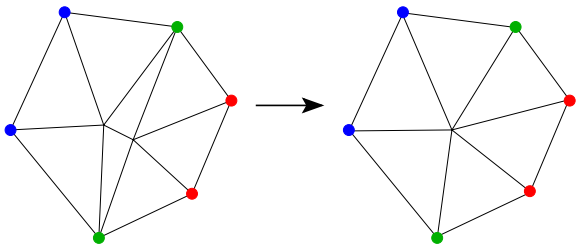


Figure 2.9: *Edge collapse operation. It can be executed only if the only shared neighbors between the two vertices of the edge to be collapsed belong to the faces attached to the edge (in green), i.e., no blue neighbor is a red neighbor and vice versa.*

for two adjacent cubes, but mirrored. Another issue is the fact that this algorithm generates many more triangles than the classic marching cubes algorithm, which makes it even more necessary to use a decimation step. The decimation is based on the edge collapse operator [Hoppe et al., 1993] (see Fig. 2.9). It is basically a loop where, for a desired minimum edge size, we select the smallest edge of the mesh and compare it with the minimum size. If smaller, we collapse the edge (see Fig. 2.9), whenever it is possible[3], and loop until there is no more edges to collapse.

[3]We need to guarantee that the resulting surface after the edge collapse operation is still manifold.

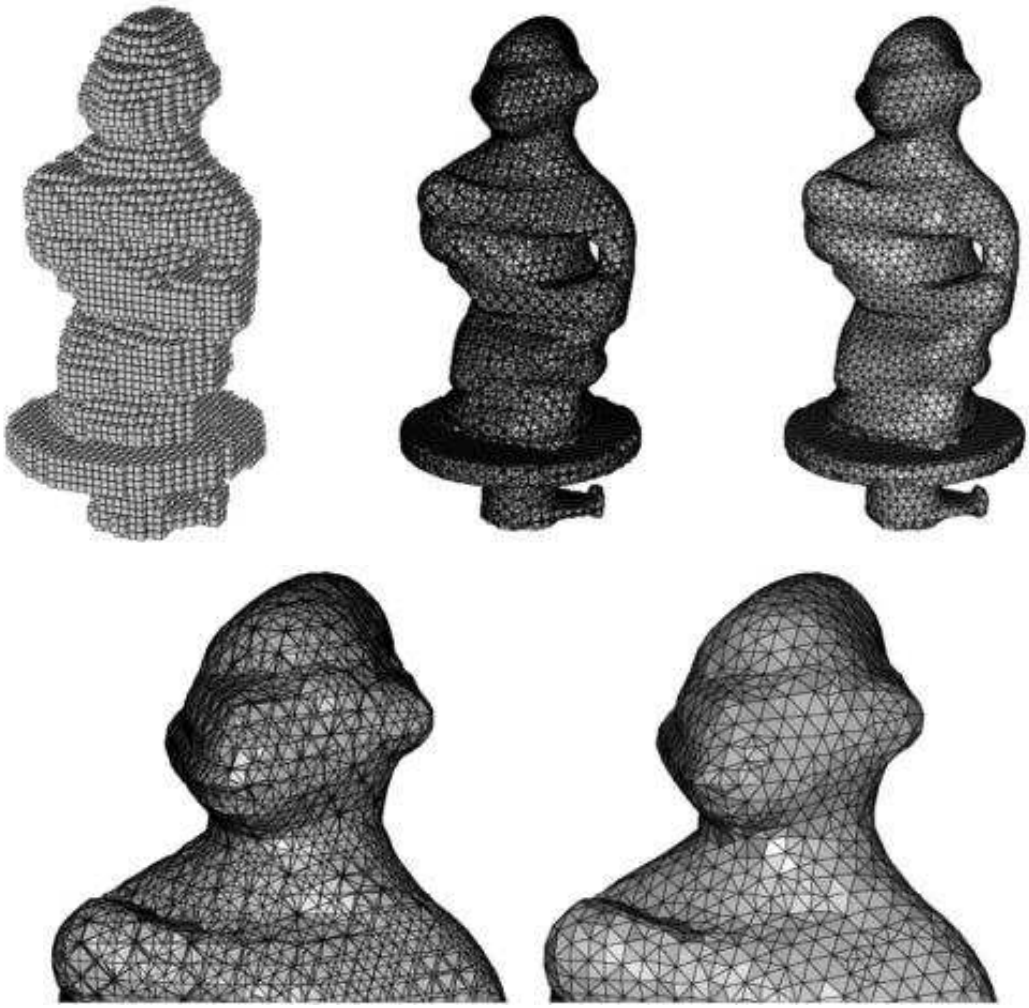


Figure 2.10: *Main steps in the visual hull mesh construction for 6 levels of resolution. Top: from left to right, octree construction, marching cubes meshing and decimation. Bottom: detailed view of the marching cubes mesh before and after decimation.*

In Fig. 2.10 we present the main steps in the visual hull mesh construction for 6 levels of resolution. We note the existance of a great quantity of very small edges in the output of the marching cubes (see Fig. 2.10 bottom left), and how they are eliminated after the decimation step (see Fig. 2.10 bottom right).

2.4 Texture Driven Force

In this section we further develop the texture force $\mathcal{F}_{tex}$ appearing in equation 2.2. This force contributes to recovering the 3D object shape by exploiting the texture of the object. We want this force to maximize the image coherence of all the cameras that see the same part of the object. It is based on the following projective geometry property: if two cameras see the same surface, then the two images are related by a geometric transformation that depends only on the 3D geometry of the object. This

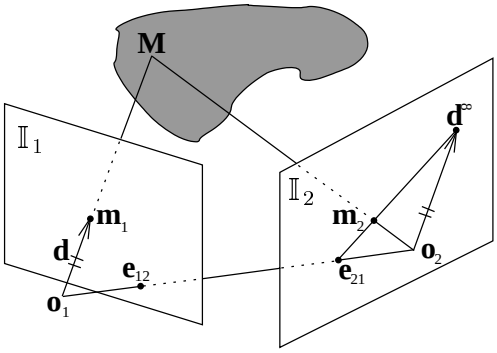


Figure 2.11: *Epipolar geometry.*

property is only fully valid under the common hypothesis of perfect projective cameras, perfect Lambertian surface and same lighting conditions. Different approaches exist to measure the coherence of a set of images, but they can be classified into two main groups whether they make a local radiometric comparison (e.g. photo-consistency measures as in voxel coloring [Seitz and Dyer, 2000]) or a spatial comparison of relative radiometric distributions (e.g. cross-correlation measures). For a taxonomy of different stereo algorithms, see [Scharstein and Szeliski, 2002]. We have chosen the normalized cross-correlation because of its simplicity and robustness in the presence of highlights and changes of the lighting conditions. Using the example of Fig. 2.11, the normalized cross-correlation $C(\mathbf{m}_1, \mathbf{m}_2)$ between pixels $\mathbf{m}_1$ and $\mathbf{m}_2$ is defined as follows:

$$C(\mathbf{m}_1, \mathbf{m}_2) = \mathbf{n}_1 \cdot \mathbf{n}_2, \quad \mathbf{n}_j = \frac{\mathbb{I}_j(N(\mathbf{m}_j)) - \overline{\mathbb{I}_j(N(\mathbf{m}_j))}}{||\mathbb{I}_j(N(\mathbf{m}_j)) - \overline{\mathbb{I}_j(N(\mathbf{m}_j))}||}, j = 1, 2, \tag{2.8}$$

where $N(\mathbf{m}_j)$ is a neighborhood around $\mathbf{m}_j$ in image $\mathbb{I}_j$, and $\mathbb{I}_j(N(\mathbf{m}_j))$ is the vector of the image values in this neighborhood. This measure compares the intensity distributions inside the two neighborhoods. It is invariant to changes of the mean intensity value and of the dynamic range inside the neighborhoods. Concerning the shape of the neighborhood N, there exist different types of "windows" depending on the complexity and knowledge of shape information. The first and simplest neighborhood is the point itself as in [Liedtke et al., 1991] but, obviously, this is not a cross-correlation measure but rather a photo-consistency measure. If the window size is bigger than one pixel, the next simplest neighborhood is the square window, which has been extensively used in computer vision. Another useful correlation window is the discrete disk, which presents fewer corner artifacts than the square window. Both the disk and the square window are the same for any image and for any point, since we do not dispose of any additional information. However, if we dispose of 3D shape information, e.g., we want to evaluate a 3D surface, then neighborhoods can integrate the anamorphism induced by projective geometry. The first way to integrate 3D shape information is to locally estimate the tangent plane to the surface. The tangent plane induces a homography between different images and between the tangent plane itself and any image. This allows defining the neighborhood in the tangent plane domain and "project" it to the different images. If the reference neighborhood is square, then

the projected neighborhoods will be quadrangular while, if the original window is a disk, the projections will be ellipses. Finally, we can fit even more complicated primitives to the surface, such as quadrics [Cross and Zisserman, 2000], which allows using even more complicated neighborhoods, e.g., quadric patches.

The availability of so many different ways of computing correlations arises an interesting question about which one is the best. In fact, the question can be generalized to the choice between model-dependent similarity measures (such as tangent plane or quadric based correlations) or model-independent similarity measures (such as square window correlation). At first glance, it may seem that using model-dependent correlations will give better results than the greedy square windows. This is true if the current shape is close to the real surface and thus, the tangent plane or the quadric have a geometric meaning. However, if the shape is far away from the real surface, the surface derivatives, such as the tangent plane, have no meaning at all. Another interesting point is about the choice of more complex models such as quadrics. This is clearly related to the size of the correlation windows. If the correlation window is big enough to contain significant variations of the shape, then it is interesting to use quadrics in order to better model the shape. But current digital cameras have very high resolutions (from 6 Mpixels to 24 Mpixels), which, even with correlation windows of tens of pixels, still permits considering the surface as locally flat. Moreover, as shown in Fig. 2.12, the difference between a square window and a plane-based window is negligible for small baselines around the reference image (image 0) while it is very strong for large baselines. We show in figure 2.13 the resulting normalized windows. It allows better appreciating the deformation induced by the use of a square window instead of a plane-based window. When using a square window (Fig. 2.13 left), images close to the reference image have a little deformation. However, for large baselines, the difference is enormous (compare for instance image 0 with image 6 in both the square case and the plane case).

The choice of the correlation method is intimately related to the algorithm used to optimize the correlation score. Two different types of approaches for this optimization have been proposed in the literature:

- In the first type, the texture similarity is computed using the current shape estimation. It permits explicitly computing visibility and using it in the texture similarity computation. If the measure is improved by deforming the model locally, then the model is updated and the process iterated as in [Fua and Leclerc, 1995, Isidoro and Sclaroff, 2003, Nobuhara and Matsuyama, 2003]. Level-set based methods as in [Faugeras and Keriven, 1998, Sarti and Tubaro, 2002] explore a volumetric band around the current model but they still remain locally dependent on the current model shape. Since the exploration does not test all the possible configurations, the algorithm can fail because of local maxima of the texture coherence criterion. Therefore, computing visibility with the current shape does not always imply having a more accurate texture criterion. If the current model is far away from the real shape, the visibility can be wrong and so can the texture criterion. However, if the current model is close to the real shape, taking visibility into account can improve the final result.

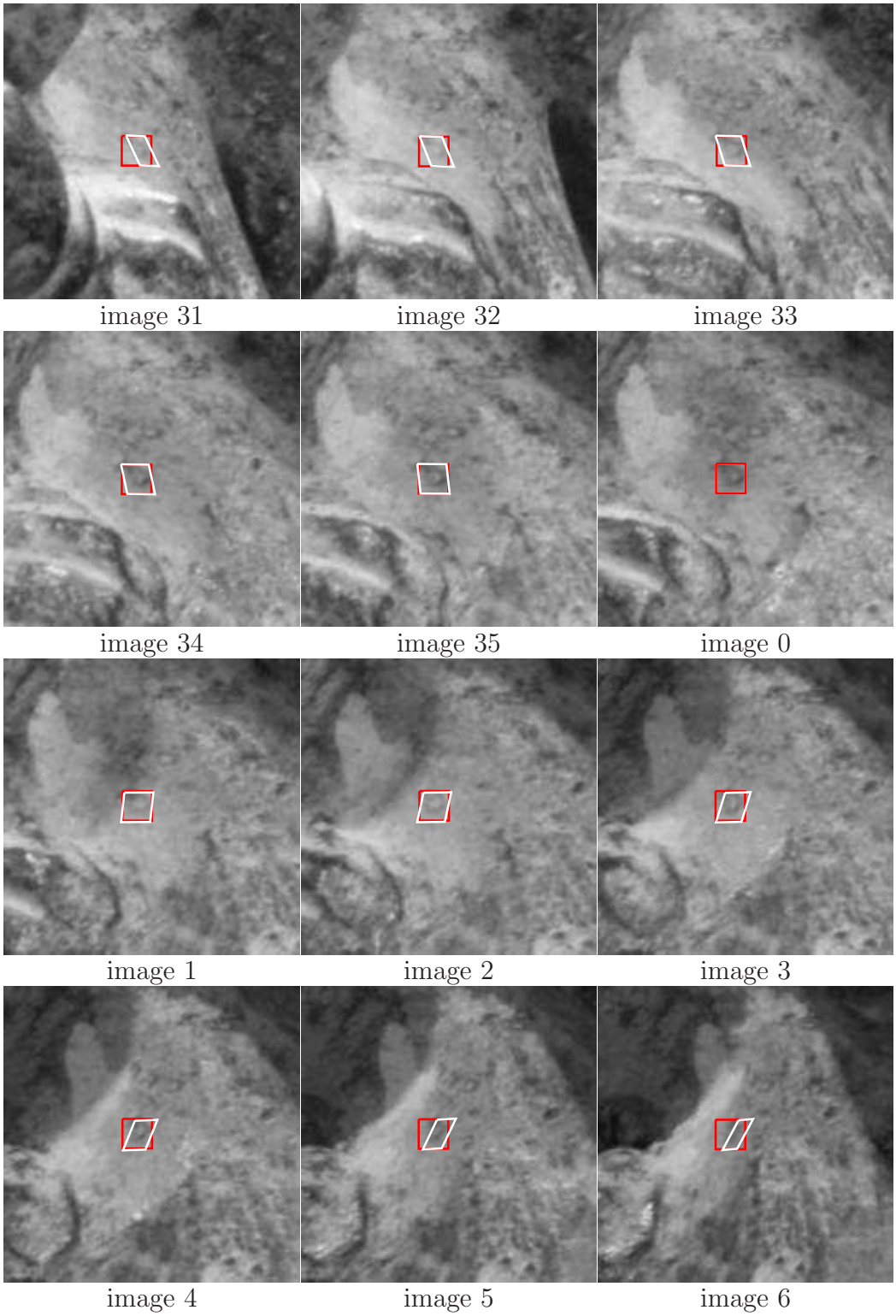


Figure 2.12: *Some snapshots of the Twins object superposed with a square window (red) and a tangent plane window (white). The windows size is 21x21 pixels and the baseline between adjacent images is* 10 *degrees. Both the 3D position and the surface normal estimate are supposed to be correct.*

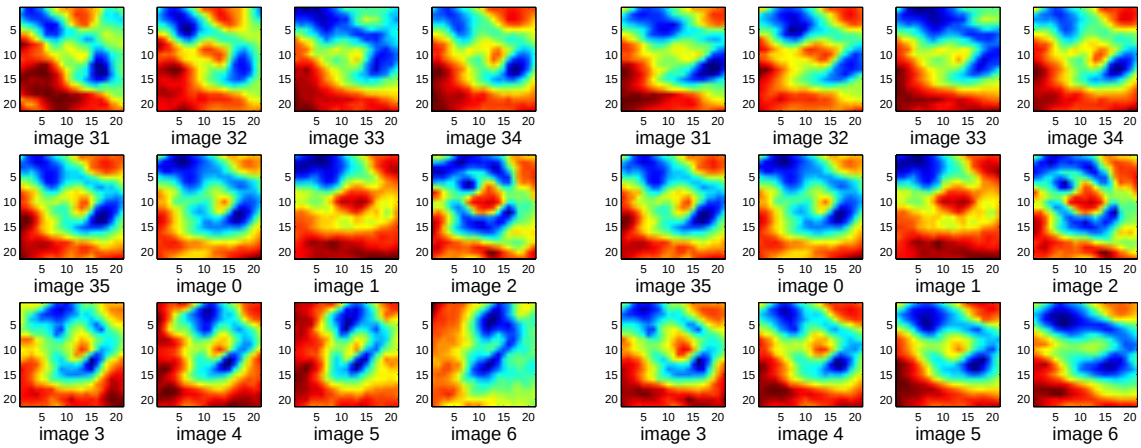


Figure 2.13: *Normalized correlation windows of Fig. 2.12 for the square window (left) and the plane-based window (right).*

- The second type of approaches refers to the use of a model-independent texture criterion to test all the possible configurations. In order to improve the robustness even more, we can accumulate the criterion values into a 3D grid by using a voting approach as in [Matsumoto et al., 1999, Medioni et al., 2000].

Since our initialization (visual hull) can be far away from the real surface, we believe that model-dependent criteria may have local minima problems. For this reason, we prefer to use a model-independent voting approach. Voting favors robustness in the presence of highlights and passing from the image information to a more usable information of the sort "probability of finding a surface" [Broadhurst et al., 2001] or "surface density". In order to completely define a texture-based force, we compute a gradient vector flow on the surface density volume, which allows us to define a texture force in any point of the volume.

We introduce some notions of projective geometry between multiple images in section 2.4.1, the voting approach in section 2.4.3 and the computation of the gradient vector flow in section 2.4.4.

2.4.1 Geometric Relation between Multiple Images

We first define the projective geometry relation between any 3D point $\mathbf{M}$ and its projections $\mathbf{m}_1$ and $\mathbf{m}_2$ in the two images $\mathbb{I}_1$ and $\mathbb{I}_2$(see Fig. 2.11). Let $\mathbf{M}(\delta)$ be the optic ray generated by the optical center $\mathbf{O}_1$ and the direction $\mathbf{d}$ defined by a pixel $\mathbf{m}_1$ of $\mathbb{I}_1$ as follows:

$$\mathbf{M}(\delta) = \mathbf{O}_1 + \delta\mathbf{d}, \quad \delta \in [0, \infty). \tag{2.9}$$

Let P_2 be the projection matrix of the second camera, the projection of the optic ray into $\mathbb{I}_2$ can be computed as:

$$\mathbf{m}_2(\delta) \sim P_2\mathbf{M}(\delta). \tag{2.10}$$

Let $\mathbf{d}^\infty$ and $\mathbf{e}_{21}$ be the projections in $\mathbb{I}_2$ of the infinity point $\mathbf{M}(\delta \to \infty)$ and the origin $\mathbf{O}_1$ of the optic ray respectively:

$$\mathbf{e}_{21} \sim P_2\mathbf{O}_1,\ \mathbf{d}^\infty \sim P_2\mathbf{M}(\delta \to \infty). \tag{2.11}$$

We have then the following relations:

$$\mathbf{m}_2(\delta) \sim \mathbf{e}_{21} + \delta\mathbf{d}^\infty \sim \begin{bmatrix} e_{21_x} + \delta d_x^\infty \\ e_{21_y} + \delta d_y^\infty \\ e_{21_z} + \delta d_z^\infty \end{bmatrix} \sim \begin{bmatrix} \frac{e_{21_x}+\delta d_x^\infty}{e_{21_z}+\delta d_z^\infty} \\ \frac{e_{21_y}+\delta d_y^\infty}{e_{21_z}+\delta d_z^\infty} \\ 1 \end{bmatrix}. \tag{2.12}$$

For a given 3D depth δ, its relationship with the 2D distance between the epipole $\mathbf{e}_{21}$ and $\mathbf{m}_2(\delta)$ can be obtained as follows:

$$||\mathbf{m}_2(\delta) - \mathbf{e}_{21}|| = \sqrt{\left(\frac{e_{21_x} + \delta d_x^\infty}{e_{21_z} + \delta d_z^\infty} - \frac{e_{21_x}}{e_{21_z}}\right)^2 + \left(\frac{e_{21_y} + \delta d_y^\infty}{e_{21_z} + \delta d_z^\infty} - \frac{e_{21_y}}{e_{21_z}}\right)^2}, \tag{2.13}$$

and after simplification:

$$||\mathbf{m}_2(\delta) - \mathbf{e}_{21}|| = \frac{\delta}{e_{21_z} + \delta d_z^\infty}\sqrt{a^2 + b^2}, \tag{2.14}$$

where

$$\begin{aligned} a &= d_x^\infty - d_z^\infty e_{21_x}/e_{21_z}, \\ b &= d_y^\infty - d_z^\infty e_{21_y}/e_{21_z}. \end{aligned} \tag{2.15}$$

This simple formula allows the passage from the 2D pixel distance $||\mathbf{m}_2(\delta) - \mathbf{e}_{21}||$ to the 3D metric depth δ. It applies also for any other view $\mathbb{I}_j$ seeing the same optic ray, and links together all the corresponding 2D distances $||\mathbf{m}_j(\delta) - \mathbf{e}_{j1}||$.

Let us consider our problem of 3D recovery from texture. We want to optimize, for a given pixel in one image, the texture coherence with the other images. An optic ray can be defined by the pixel, and we search for the 3D point $\mathbf{M}$ belonging to the optic ray that maximizes the normalized cross-correlation with the other images. This can be done by sampling the projection of the optic ray in every image. In practice, the knowledge of the visual hull, which is an upper bound of the object, allows us to accelerate computations. For a given pixel, the 3D depth to scan is fixed by the intersection of its optic ray with the visual hull. This intersection gives a depth interval in which we know that the object is contained. According to equation 2.14, this depth interval can be translated into pixel intervals for the correlation computation.

Equation 2.14 enables multi-correlation algorithms to stay in the pixel space for all the computations. For the same optic ray, individual correlations are computed with the different images and merged into a unique pixel abscissa. Using several correlation curves for the same optic ray allows us to make a more robust decision. We can see in Fig. 2.14 the corresponding correlation curves of the example of figures 2.12 and 2.13. We can appreciate in particular that, for a maximum baseline of $\pm 20°$, (left column of Fig. 2.14) the correlation curves are quite similar for both square window correlations and plane-based correlations. Even if the knowledge of the tangent plane provides

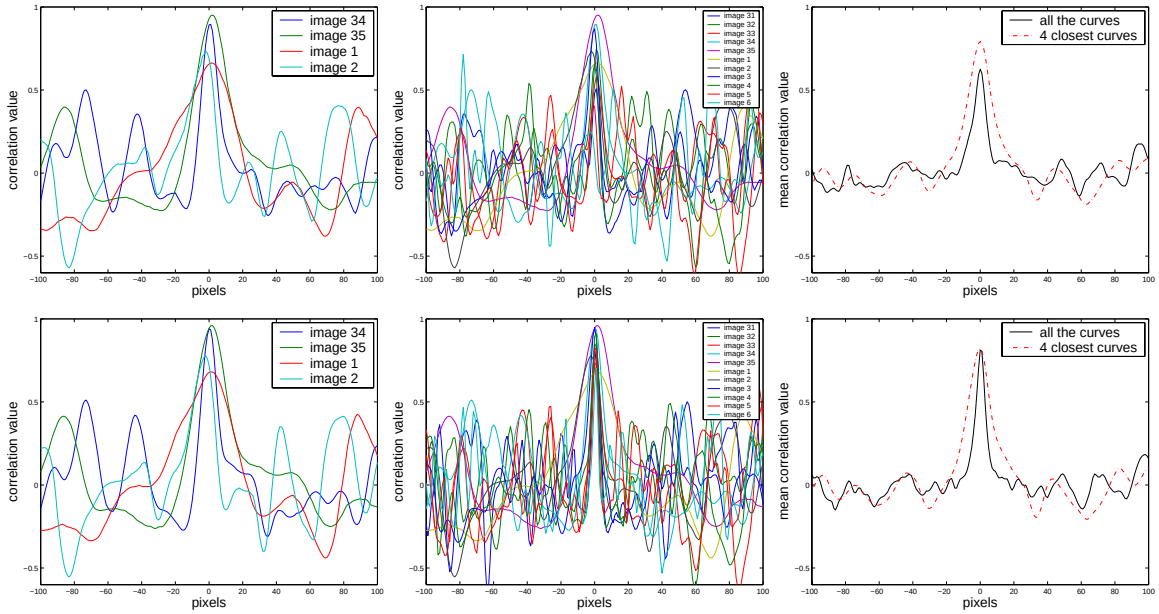


Figure 2.14: *Cross correlation curves of the sequence of Fig. 2.12 with a window size of 11x11 pixels for both a square window (top) and a plane-based window (bottom).*

additional information, it is mainly translated into a higher correlation score. The shape of the correlation curves remains quite the same and so does their precision. In the middle column, we dispose of all the available curves (maximum baseline of $\pm 60°$) for both the square window (top) and the plane-based window (bottom). On the right column we have computed the mean correlation curve for the $\pm 20°$ baseline (dashed red) and the same result for the $\pm 60°$ baseline (solid black). We notice that, when using square windows, adding new curves improves localization (the peak is narrower) but it greatly penalizes the correlation score. This is expected since images with a large baseline are strongly deformed (see images 31, 32, 5, and 6 in Fig. 2.13 left). Using the plane-based windows with all the curves gives a very good localization and the correlation score is still very high (almost no difference between 4 or 12 images in Fig. 2.13 bottom right). Based on this behavior, the conclusion is that using tangent plane correlations is worth only if we dispose of large baseline images and the surface is already very close to the real one. Otherwise, it is preferable to use square windows since they do not need an estimate of the real surface and provide also good accuracy, at the expense of a limited maximum baseline, to avoid too low correlations.

Although we have used in Fig. 2.14 the mean correlation as a way to combine the individual correlation curves, this is only valid for well textured surfaces that give nice curves. If the material properties are less Lambertian, correlation curves can be much noisier and it may be difficult to locate the maximum correctly. Instead, a more robust way to proceed is to compute the local maxima of the correlation curves and cumulate them into a 1-D grid. We can then take the bin that contains the largest number of votes and average the curves that have a local maximum greater than a threshold inside that bin. Obviously, the size of the bin is important since a step size of

1 pixel does not allow voting while a very big step has the same result as the standard mean. In practice, a bin step of a ten of pixels performs well. To accept the hit, two conditions must be satisfied: a minimum number of curves n_{bin} must have voted for that bin and the total score inside the bin must be greater than a fixed threshold t_{bin}. The value of n_{bin} depends only on the number of correlation curves that we fuse. Typically, for 4 correlation curves, $n_{bin} = 2$. The value of t_{bin} remains always fixed and is $t_{bin} = 0.6$.

Finally, an important point about window-based correlations is that, in general, the reference image must be in epipolar correspondence with all the other images with which correlation is computed. After epipolar rectification, a row of pixels in the reference image is in epipolar correspondence with the same row in the other images (epipolar lines are perfectly horizontal after rectification). However, this rectification has not been done in the present work since, for circular motion sequences with a rotation axis close to the y-axis, epipolar lines are close to horizontal. As an example, in Fig. 2.12, the distance to the epipole is more than 100 times the size of the image (2008x3040) for a baseline of 10 degrees and around 50 times for a baseline of 20 degrees. The epipolar lines have a slope angle of 1.25 degrees relative to the horizontal axis.

2.4.2 Local vertex optimization using multi-correlation

Now that we know how to estimate the depth to the surface for a given optic ray using multi-correlation, one may wonder what happens if we apply it in a direct way to the initial model, i.e., to the visual hull. This algorithm would be close to [Liedtke et al., 1991] but, instead of using photo-consistency, we have a more sophisticated multi-correlation criterion. Basically, for a given vertex of the mesh, the algorithm does the following:

1. compute the visibility (using zbuffers),
2. compute the reference view (median camera) from the set of visible cameras,
3. compute the depth along its reference optic ray:
   (a) initial estimation using a small set of cameras (small baseline),
   (b) accurate estimation using a large set of cameras around the initial estimation (large baseline),
4. move the vertex to its new position.

We can see in figures 2.15 and 2.16 the reconstruction results for two different objects. A first conclusion is that, for the well textured regions where the visual hull is close to the real surface, the algorithm works quite well (this is especially visible for the Bighead object in Fig. 2.15). The main problems that we found with this algorithm are twofold: mesh crossings and vertices that did not correlate well. Mesh crossings may happen since there is no collision detection in the individual vertex deformation.

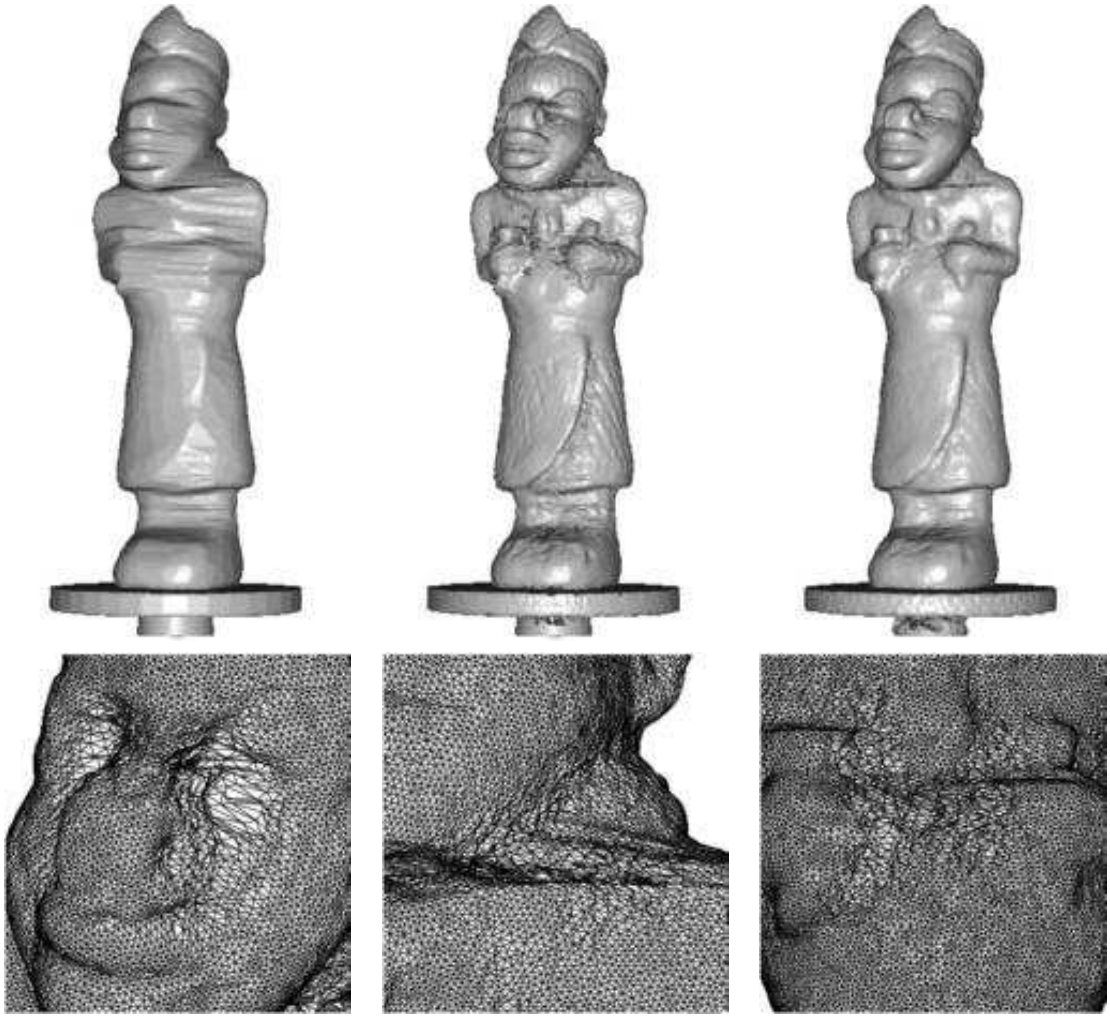


Figure 2.15: *Reconstruction of the Bighead object by local vertex optimization using multi-correlation. From left to right: visual hull, deformed model, filtered model and detail of some reconstruction errors of the filtered model.*

They can easily disappear using a deformable model with regularization. The second problem is the reliability of the coherence measures. Some vertices are moved into wrong positions while others are not moved since correlation is not high enough due to a lack of surface texture. This problem is much more interesting since it is more difficult to resolve and arises an important question about how to extract the 3D shape information from the set of images in a robust way. To improve the extraction of 3D shape information from images, we propose a more robust global voting approach with a model-independent correlation measure: a square window multi-correlation.

In the following section we describe the voting approach we have developed to extract the 3D shape information from the images.

Figure 2.16: *Reconstruction of the Twins object by local vertex optimization using multi-correlation. From left to right: visual hull, deformed model, filtered model and detail of some reconstruction errors of the filtered model.*

2.4.3 Proposed Voting Approach

The simplest way to exploit the stereo information of the images is to compute a depth map for every image and merge them into a single coordinate system such as 3D volume. A first algorithm is to simply estimate a 3D point on the surface for every pixel in every image:

pseudo code of the greedy correlation algorithm

```
1  For each image in imageList
2    For each pixel in image
3      Compute the depth interval from the visual hull
4      Compute the correlation curves
5      Transform all the curves into the same pixel abscissa
6      Find the best candidate depth[4]
7      If candidate is not valid[5], continue with next pixel
8      Compute the 3D position P of the candidate depth
9      Add the correlation value to the voxel grid containing P
```

In our implementation, the correlation for a given pixel is computed with a fixed number of cameras. For a typical sequence of 36 images, a good compromise between computation time, accuracy and robustness is obtained by using a maximum baseline of $\pm 20°$ (which gives a total of 4 correlation curves: $\pm\ 10\ °$ and $\pm\ 20\ °$). This configuration is also used in [Matsumoto et al., 1999], and can resist up to 2 bad correlation curves

[4] using the robust correlation curve selection with the 1-D bin approach explained at the end of Section 2.4.1

[5] minimum values for n_{bin} and t_{bin}

(see Fig. 2.14) due to highlights, occlusions or non-coherent textures occurring in the corresponding images. Although this algorithm does not explicitly handle occlusions, they are implicitly detected as bad correlations, in the same way as a highlight or a lack of texture. Besides, any other stereo algorithm can be plugged in instead of the proposed one. However, the problem is the computation time. For large images (2000 x 3000), the computation time can reach 16 hours on a fast machine. This time can be strongly reduced with almost no loss because of the redundancy of the computation. The redundancy can be classified into two main groups: redundancy inside an image and redundancy between different images.

- The **redundancy between images** is caused by the fact that several images see at the same time the same piece of surface. If we have already computed a surface estimation using one image, we can back project the 3D points into the next image, giving an initial estimation of the distance to the surface. The problem is that if the previous image did not correlate well, errors may propagate and prevent the following images from attenuating it.

- The **redundancy inside an image** can be exploited using the previous knowledge of the content of the image. In our case, it is a picture of an object and we can expect it to be locally continuous. This implies that, if the surface is correctly seen and if there is no occlusion, the depth values of neighboring pixels should not be very different.

In order not to degrade the quality of the correlations too much, we have only exploited the redundancy inside images. It allows us to reduce the depth interval for the correlation criterion. In the greedy algorithm, for each pixel, we test the entire depth interval defined by the visual hull without taking into account if its neighbors have already found a coherent surface. To be able to benefit from already computed correlations, the image can be partitioned into different resolution layers as shown in Fig. 2.17.

The greedy algorithm is first run on the lowest resolution layer (black pixels in Fig. 2.17), with the depth intervals defined by the visual hull. For consecutive layers, the depth intervals are computed using the results of the precedent layer. To estimate the depth interval of a pixel based on the results of the previous layer, a record of the correlation values is maintained in order to control the reliability of the estimation. The theoretical maximum improvement that we can reach with this method in the case of 3 layers as illustrated in Fig. 2.17 is 16 times as fast as the greedy method. This case corresponds to a black layer computation time much higher than the grey and white ones. In practice, the improvement is around 5 or 6 times faster for well-textured images. The worst case corresponds to non-textured images where correlations become unreliable. The depth interval estimation fails, necessitating the use of the greedy method.

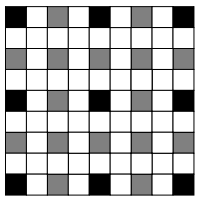


Figure 2.17: *Example of an image partition into 3 different resolution layers.*

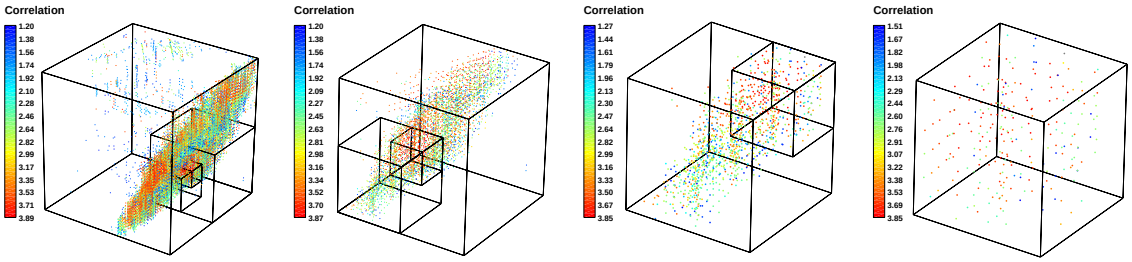
Besides the improvement in computation time, an important improvement in storage space is to substitute the 3D volume grid by a more compact octree structure. The result of the correlation step will be a 3D octree containing the accumulated hits of all the pixel estimations. The new algorithm can be coded as:

```
pseudo code of the redundancy-based correlation algorithm
For each image in imageList
  For each layer in image
    For each pixel in layer
      If layer = first layer
        Compute the depth interval from the visual hull
      Else
        Compute the depth interval from the previous layer
      Compute the correlation curves
      Transform all the curves into the same pixel abscissa
      Find the best candidate depth
      If candidate is not valid, continue with next pixel
      Compute the 3D position P of the candidate depth
      Add the correlation value to the octree voxel containing P
```

The resolution of the octree used to contain the correlation hits depends both on the maximum baseline of the stereo algorithm but also on the calibration quality. Basically, we can consider that the stereo algorithm has a resolution of one pixel at most. Since we are using images between 3000x2000 and 4000x4000 pixels, the maximum octree resolution will be between around 11 and 12 levels. But in that case, there would be too few correlation hits per voxel for voting to be useful. Besides, practical accuracy is lower than in theory due to the loss of image space when taking photographs (the object image being smaller than the image size), loss of resolution caused by the size of the correlation window, by the maximum camera baseline and by blurred images due to insufficient depth of field or wrong focus. Typical resolutions are between 10 and 11 levels, which is already very high. We show in Fig. 2.18 the correlation hits for different voxel sizes of a well textured zone of the Twins sequence (roughly the same region than the one shown in Fig. 2.12).

We observe in particular that, even if the image sequence is 3000x2000 pixels, a 9-level voxel (Fig. 2.18 right) has already the same size as the uncertainty of the estimated surface. This can be better appreciated in Fig. 2.19, where we see the same voxel from a side view to evaluate the thickness of the cloud of correlation hits. The thickness of the cloud depends on the image resolution and system calibration, but also on the texture of the surface and the number of images used to correlate. We note in Fig. 2.19 that the width of the surface is equal to the edge size of a 9-level voxel.

The output of the voting step is an octree volume that contains, for each voxel, the sum of the individual correlation scores contained in that voxel. This volume can be seen as a volume of surface probability where a voxel with a high score is very probable to contain the real object surface. Low score voxels may indicate a false match (due for example to highlights) *or* the presence of a surface with no texture or noisy texture. Figure 2.20 shows the voting volume for the Twins sequence. Since it is very difficult



6^{th} level (total score=51526) | 7^{th} level (total score=13453) | 8^{th} level (total score=3222) | 9^{th} level (total score=798)

Figure 2.18: *Voxel size as a function of the octree depth. The accumulated correlation score inside the voxel decreases by a factor of 4 with each new level, as expected. Individual correlation scores range from 1.2 to 4 since: i) the 4 closest cameras were used to correlate ($\pm 10°$ and $\pm 20°$), with a minimum curve correlation score $t_{bin} = 0.6$, and ii) a minimum of 2 coherent curves are needed $n_{bin} = 2$.*

to visualize the voting volume, we present two slices on the octree in Fig. 2.20 top left, and different rendered views of the volume after binarization using different thresholds. As we can see for the case of no threshold, the voting volume contains voxels that do not belong to the object surface. These voxels have relatively low score values and thresholding the correlation octree permits their elimination. This is the main utility of the voting approach: to add robustness to the failure of the correlation approach (mainly due to large deviations from the Lambertian hypothesis).

This volume by itself cannot be used as a force to drive a deformable model. A possible force could be the gradient of the correlation volume. The objection is that it is a very local force defined only in the vicinity of the object surface. The proposed solution to this problem is using a gradient vector flow (GVF) field to drive the snake.

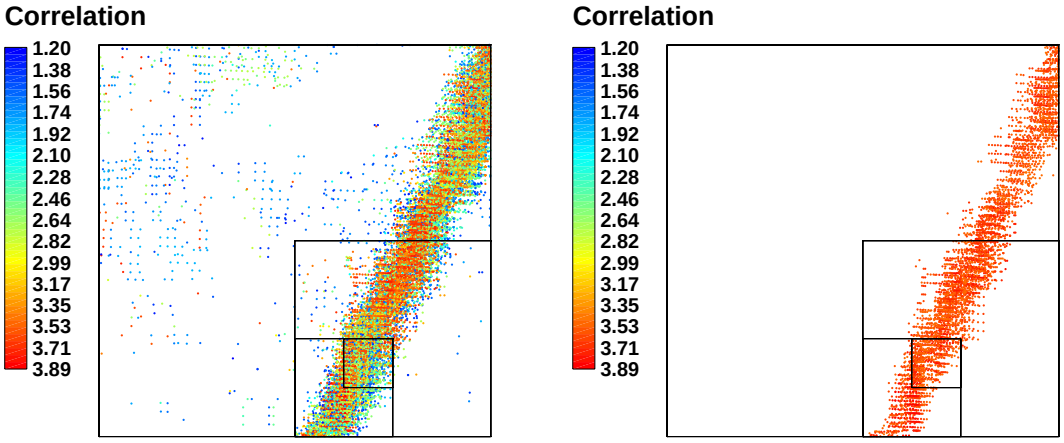


Figure 2.19: *Correlation voting with different voxel sizes from 6 to 9 levels. Left: all the accumulated correlation hits. Right: high score hits with a correlation greater than 3.0 are shown.*

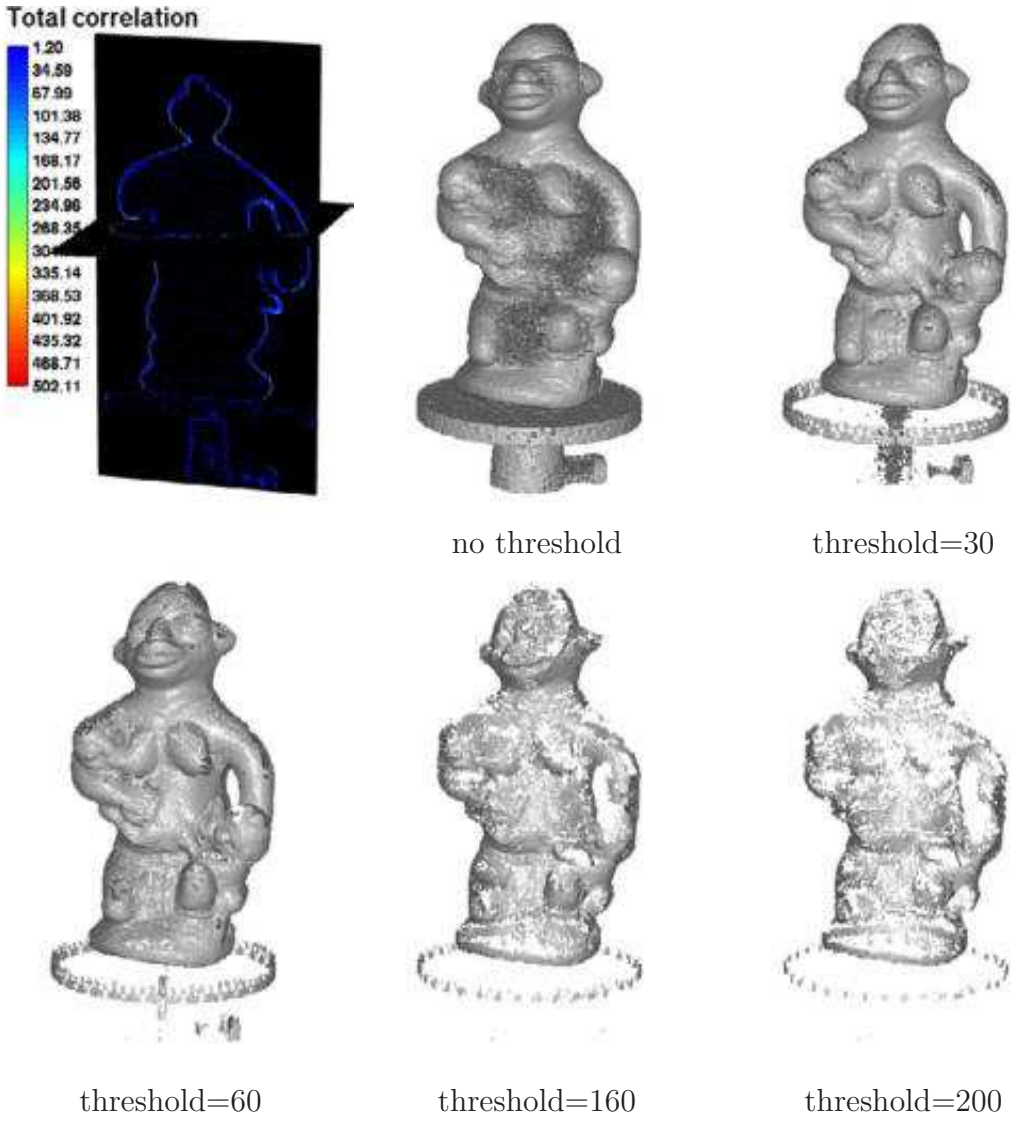


Figure 2.20: *Correlation voting octree for the Twins sequence (36 images) with 10 levels of depth and a maximum baseline of* $\pm 20°$*. Top left: two slices of the octree volume. From top to bottom in scanline order: rendered views of the octree volume after binarization for increasing threshold values of the accumulated correlation.*

2.4.4 Octree-based Gradient Vector Flow

The GVF field was introduced by [Xu and Prince, 1998] as a way to overcome a difficult problem of traditional external forces: the capture range of the force. This problem is caused by the local definition of the force, and the absence of an information propagation mechanism. To eliminate this drawback, and for all the forces derived from the gradient of a scalar field, they proposed generating a vector field force that propagates the gradient information. The GVF of a scalar field $f(x,y,z) : \Omega \subset \mathbb{R}^3 \mapsto \mathbb{R}$ is defined as the vector field $\mathbf{F} = (u(x,y,z), v(x,y,z), w(x,y,z)) : \Omega \subset \mathbb{R}^3 \mapsto \mathbb{R}^3$ that minimizes the following energy functional $\mathcal{E}_{GVF}$:

$$\mathcal{E}_{GVF} = \int_\Omega \mu(u_x^2+u_y^2+u_z^2+v_x^2+v_y^2+v_z^2+w_x^2+w_y^2+w_z^2)+||\mathbf{F}-\nabla f||^2||\nabla f||^2 dxdydz, \quad (2.16)$$

where μ is the weight of the regularization term and ∇ is the gradient operator. The solution to this minimization problem has to satisfy the following Euler-Lagrange equations:

$$\begin{aligned} \mu\nabla^2 u - (u-f_x)(f_x^2+f_y^2+f_z^2) &= 0, \\ \mu\nabla^2 v - (v-f_y)(f_x^2+f_y^2+f_z^2) &= 0, \\ \mu\nabla^2 w - (w-f_z)(f_x^2+f_y^2+f_z^2) &= 0, \end{aligned} \quad (2.17)$$

where ∇^2 is the Laplacian operator. A numerical solution can be found by introducing a time variable t and solving the following three differential equations:

$$\begin{aligned} \tfrac{\partial u}{\partial t} &= \mu\nabla^2 u - (u-f_x)(f_x^2+f_y^2+f_z^2), \\ \tfrac{\partial v}{\partial t} &= \mu\nabla^2 v - (v-f_y)(f_x^2+f_y^2+f_z^2), \\ \tfrac{\partial w}{\partial t} &= \mu\nabla^2 w - (w-f_z)(f_x^2+f_y^2+f_z^2). \end{aligned} \quad (2.18)$$

We can note that the three equations are not coupled, which allows solving each equation separately as a scalar differential equation with partial derivatives in u, v and w respectively. These equations are known as the *generalized diffusion equations*, and have been used in different fields such as heat conduction, reactor physics or fluid theory.

The GVF can be seen as the original gradient smoothed by the action of a Laplacian operator. This smoothing action allows eliminating strong variations of the gradient and, at the same time, propagating it. The degree of smoothing/propagation is controlled by μ. If μ is zero, the GVF will be the original gradient, if μ is very large, the GVF will be a constant field whose components are the mean of the gradient components.

Since our data have been stored in an octree structure, the GVF has to be computed on a multi-resolution grid. For this, we need to be able to:

- define the gradient operator and the Laplacian operator in the octree grid;
- define how to interpolate between voxels with different sizes.

In three dimensions, the gradient and Laplacian operators are defined as:

$$\nabla f = (f_x, f_y, f_z), \qquad \nabla^2 f = f_{xx} + f_{yy} + f_{zz}. \quad (2.19)$$

In the case of a regular grid with a spacing of $[\Delta x, \Delta y, \Delta z]$, the first and second derivatives can be approached by central finite differences:

$$\begin{array}{rcl} f_x & \approx & \frac{f(x+\Delta x,y,z)-f(x-\Delta x,y,z)}{2\Delta x}, \\ f_{xx} & \approx & \frac{f(x+\Delta x,y,z)-2f(x,y,z)+f(x-\Delta x,y,z)}{\Delta x^2}. \end{array} \tag{2.20}$$

If the grid is not regular, then the finite differences will not be centered. An easy way to find the equivalent formulas for a non-regular grid is to estimate the parabolic curve ax^2+bx+c that passes through 3 points (Fig. 2.21), and compute the derivatives of the estimated curve [Fornberg, 1988]. After solving the equation system, we find:

$$\begin{array}{rcl} f_x(x_0) \approx 2ax_0+b & = & \frac{1}{(\delta+\Delta)}\left(\frac{f(x_0+\Delta)-f(x_0)}{\Delta/\delta} - \frac{f(x_0-\delta)-f(x_0)}{\delta/\Delta}\right), \\ f_{xx}(x_0) \approx 2a & = & \frac{2}{(\delta+\Delta)}\left(\frac{f(x_0+\Delta)-f(x_0)}{\Delta} + \frac{f(x_0-\delta)-f(x_0)}{\delta}\right). \end{array} \tag{2.21}$$

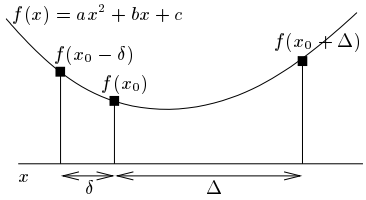


Figure 2.21: *Parabolic curve passing through 3 points.*

As far as the interpolation is concerned, and to simplify the computation, we have to add a constraint to the topology of the multi-resolution grid: the difference of resolution in the neighborhood of a voxel, including the voxel itself, cannot be greater than one level. This is not a strong constraint since the resolution of the octree needs to change slowly if we want good numerical results in the computation of the GVF.

There exist three different scenarios in the multi-resolution numerical algorithm. The first one is when the current voxel and all its neighbors have the same size (see Fig. 2.22(a)). In this case, computations are done as with a mono-resolution grid. The second one is when the current voxel is bigger than or equal to its neighbors (see Fig. 2.22(b)). For those voxels with the same size, computations are carried out in an ordinary way. For those that are smaller, a mean value is simply used to get the correct value in the scale of the current voxel:

$$f_x(A) \approx \frac{f(EFGH)-f(D)}{2\delta}, \quad f_y(A) \approx \frac{f(B)-f(C)}{2\delta}. \tag{2.22}$$

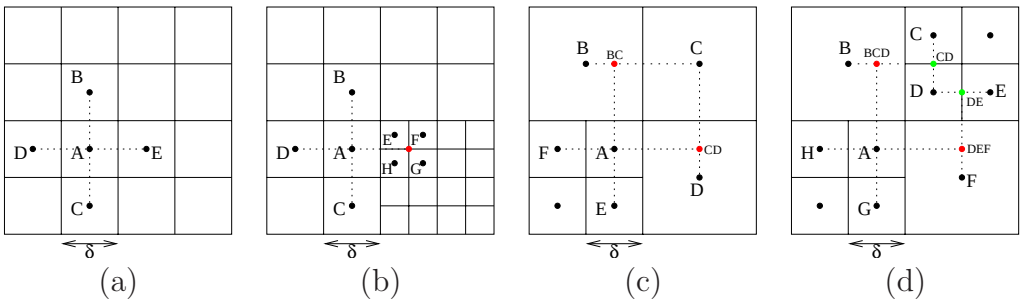


Figure 2.22: *Value interpolations (2D example): (a) Mono-grid case. (b) The current voxel is bigger that its neighbors. (c),(d) The current voxel is smaller than its neighbors.*

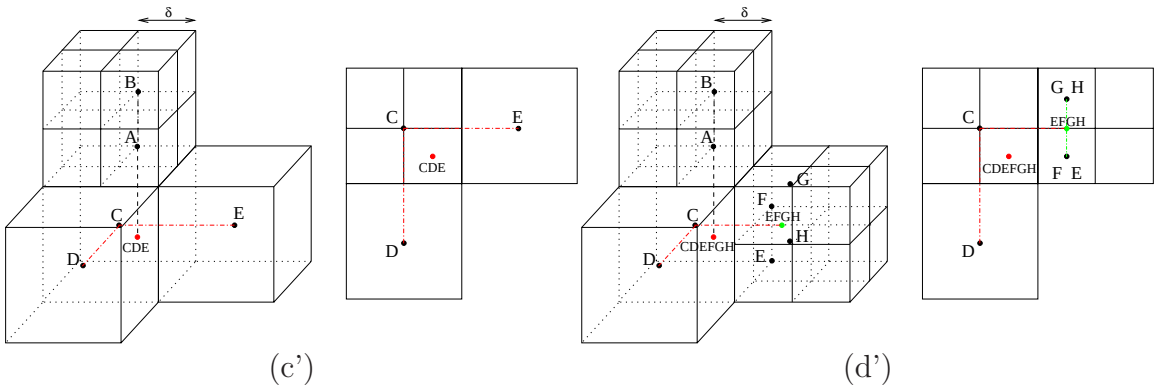


Figure 2.23: *Value interpolations (3D example): (c'),(d') The current voxel is smaller than its neighbors.*

The third case corresponds to the current voxel being smaller than or equal to its neighbors (see Fig. 2.22(c) and (d)). We illustrate two different configurations, and in both we want to compute the gradient at point A. In Fig. 2.22(c) we need the value of the function f at points E, F, BC and CD:

$$\begin{aligned} f_x(A) &\approx \frac{1}{(\delta+1.5\delta)}\left(\frac{f(CD)-f(A)}{1.5} - \frac{f(F)-f(A)}{1/1.5}\right), \\ f_y(A) &\approx \frac{1}{(\delta+1.5\delta)}\left(\frac{f(BC)-f(A)}{1.5} - \frac{f(E)-f(A)}{1/1.5}\right). \end{aligned} \tag{2.23}$$

In the example shown in Fig. 2.22(d) the values BCD and DEF are obtained by interpolating B with CD, and DE with F, respectively. If we translate these examples into 3D, cases (c) and (d) are transformed into cases (c') and (d') in Fig. 2.23 respectively. For the sake of clarity, we only illustrate the interpolation along the z axis. In example (c') we need the value $f(CDE)$, which can be obtained by bilinear interpolation between $f(C)$, $f(D)$ and $f(E)$:

$$\begin{aligned} f(CDE) &= f(C) + \tfrac{1}{4}(f(D) - f(C)) + \tfrac{1}{4}(f(E) - f(C)) \\ &= \tfrac{2}{4}f(C) + \tfrac{1}{4}f(D) + \tfrac{1}{4}f(E). \end{aligned} \tag{2.24}$$

The gradient $f_z(A)$ along the z axis can be computed as:

$$f_z(A) \approx \frac{1}{(\delta + 1.5\delta)}\left(\frac{f(B) - f(A)}{1/1.5} - \frac{f(CDE) - f(A)}{1.5}\right). \tag{2.25}$$

Example in Fig. 2.23(d') is a little bit more complex than 2.23(c'). The gradient $f_z(A)$ is computed in the same way as equation 2.25 so we need to compute the function value at the point $CDEFGH$ as in the previous case. However, we do not directly dispose of all the neighbor values as before. We first need to compute the value at $EFGH$, as a simple mean value, and then use it to compute the function value at the point $CDEFGH$, as in the previous case:

$$\begin{aligned} f(CDEFGH) &= f(C) + \tfrac{1}{4}(f(D) - f(C)) + \tfrac{1}{3}(f(EFGH) - f(C)) \\ &= \tfrac{5}{12}f(C) + \tfrac{3}{12}f(D) + \tfrac{1}{12}f(E) + \tfrac{1}{12}f(F) + \tfrac{1}{12}f(G) + \tfrac{1}{12}f(H). \end{aligned} \tag{2.26}$$

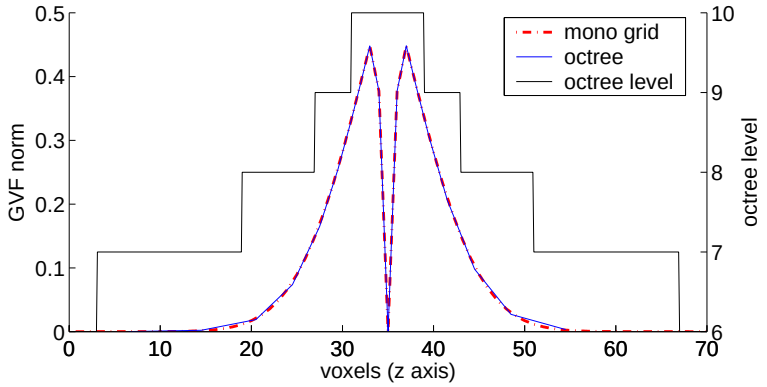


Figure 2.24: *mono grid vs. octree grid GVF with* $\mu = 0.1$.

Although we have only shown some of the most common and interesting configurations, the total number of configurations is much higher. For a given direction (x,y or z), something important to note is that all the available configurations have one neighbor with the same size as the current voxel. This is due to the constraint of at most 1 level of difference in the neighborhood of any voxel. For a given direction, we have thus the following possibilities:

- **equal size**: (1 case) both neighbors have the same resolution as the current voxel,
- **smaller size**: (3 cases) one neighbor is smaller (2 cases), or both neighbors are smaller (1 case)
- **bigger size**: (2×4 cases) one neighbor is bigger, which gives 4 different configurations depending on the size (equal to or bigger than the current voxel) of the "*neighbors of the neighbor*".

This makes a total of 12 configurations per direction. However, the total number of 3D configurations is less than 12^3. The reasons are twofold:

- The configuration along one direction affects the possible configurations of the two other directions, e.g., if a neighbor voxel is smaller along the x direction, then there cannot be any bigger neighbor voxel along the y or z directions.
- For a bigger neighbor voxel (as in Fig. 2.23(c') and (d')), the voxels used to interpolate can be shared between different directions.

In practice, a recursive algorithm has been implemented independently for each direction so it is not necessary to know the exact number of 3D configurations.

In Fig. 2.24 we compare the result of a 3D GVF computation for $\mu = 0.1$ using a regular grid and the octree approach. The scalar field f used in the example is defined as:

$$f(x,y,z) = \begin{cases} 1 & \text{for } z \in [34,36] \\ 0 & else \end{cases} .$$

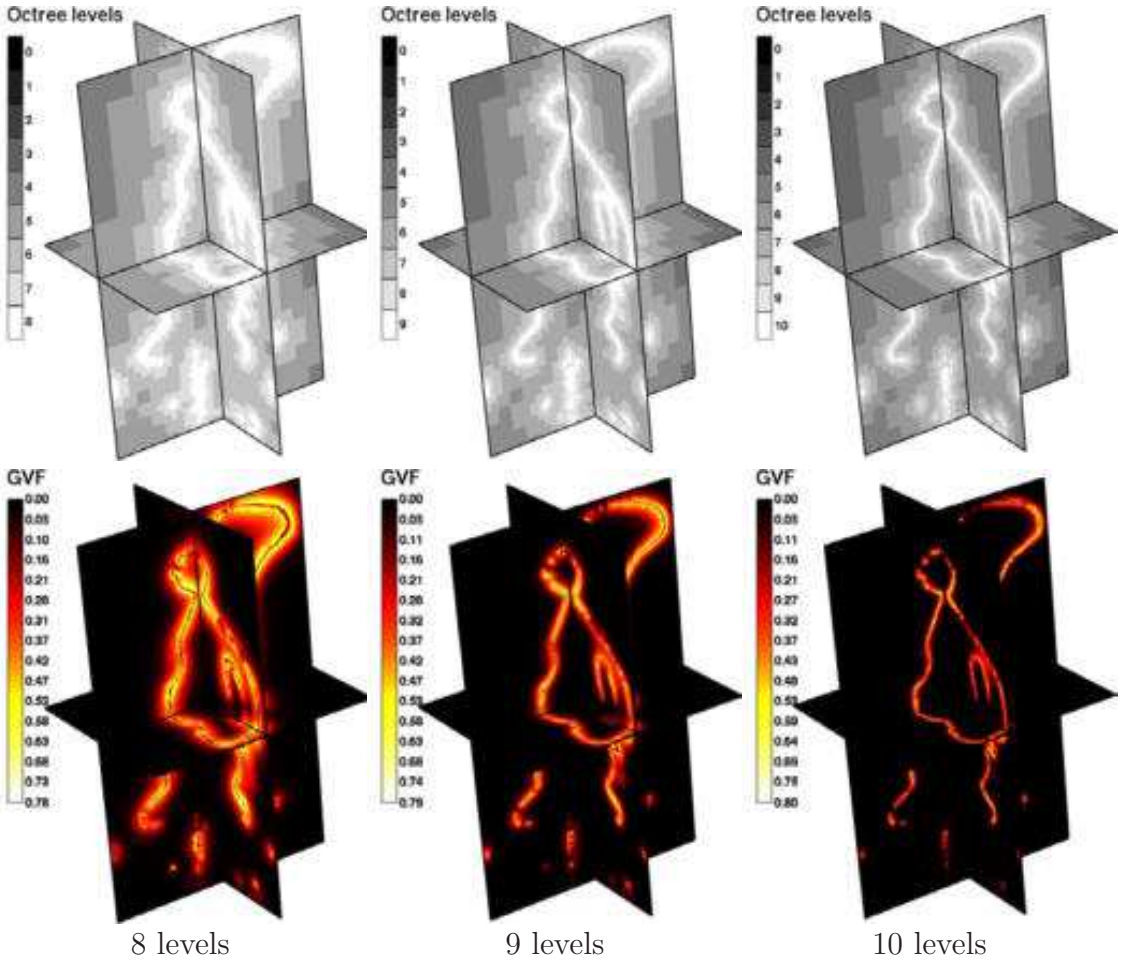


Figure 2.25: *GVF computation ($\mu = 0.1$) for different octree depths of the Twins correlation volume shown in Fig. 2.20. Top: octree partition. Bottom: norm of the GVF.*

We can appreciate the accuracy of the multi-grid computation compared with the mono-grid one. We can hardly see any difference between both curves, only when the octree resolution becomes very low (voxels 20 and 50). Mean values of computation speed up for 10 levels of resolution are between 2 and 3 times as fast as the mono-grid version while storage space is reduced between 10 and 15 times.

In Fig. 2.25 we show the GVF computation for the Twins correlation volume (see Fig. 2.20) with different octree depths and $\mu = 0.1$. If we pay attention to the second row of Fig. 2.25 (the GVF norm), we appreciate the fact that using relatively low octree resolutions allows propagating the GVF much faster than when using high resolutions, where the GVF norm is only significant near the final surface. The effect of using a grid with a lower resolution produces a similar effect to using a greater μ value. However, using a lower resolution grid has a clear advantage over increasing the μ value: the computation time. This suggests using an octree multi-resolution approach where the model is first deformed using a low resolution GVF, the result is the input of another snake evolution with a better GVF resolution, etc. This approach greatly accelerates the deformable model traversal of the empty regions that are far

away from the final surface.

2.5 Silhouette Driven Force

The silhouette force is defined as a force that makes the snake match the original silhouettes of the sequence. If it is the only force of the snake, the model should converge towards the visual hull. Since we are only interested in respecting silhouettes, the force will depend on the self occlusion of the snake. If there is a part of the snake that already matches a particular silhouette, the rest of the snake is not concerned by that silhouette, since the silhouette is already matched. If we compare a visual hull and the real object, we see that the entire real object matches the silhouettes, but not all the points of the object. The object concavities do not obey any silhouette because they are occluded by a part of the object that already matches the silhouettes. The main problem is how to distinguish between points that have to obey the silhouettes (contour generators) and those that do not have to. To solve this problem, the silhouette force can be decomposed into two different components: a component that measures the silhouette fitting, and a component that measures how strongly the silhouette force should be applied. The first component is defined as a distance to the visual hull. For a 3D vertex $\mathbf{v}$ on the mesh of the snake, this component can be implemented by computing the smallest *signed* distance d_{VH} between the silhouette contours and the projection of the point into the silhouette:

$$d_{VH}(\mathbf{v}) = \min_i d(S_i, P_i\mathbf{v}). \tag{2.27}$$

A positive distance means that the projection is inside the silhouette, and a negative distance that the projection is outside the silhouette. Using only this force would make the snake converge towards the visual hull.

The second component measures the occlusion degree of a vertex of the deformable model for a given viewpoint c. The viewpoint is chosen as the camera that defines the distance to the visual hull:

$$\begin{aligned} \alpha(\mathbf{v})_c &= \begin{cases} 1 & \text{for } d_{VH}(\mathbf{v}) \leq 0 \\ \frac{1}{(1+d(S_c^{snake}, P_c\mathbf{v}))^p} & \text{for } d_{VH}(\mathbf{v}) > 0 \end{cases}, \\ c(\mathbf{v}) &= arg\min_i d(S_i, P_i\mathbf{v}). \end{aligned} \tag{2.28}$$

In the definition of α_c, there are two cases. If d_{VH} is negative, it means that the vertex is outside the visual hull. In that case, the force is always the maximum force. For a vertex inside the visual hull, c is the camera that actually defines its distance to the visual hull d_{VH}. S_c^{snake} is the silhouette created by the projection of the snake into the camera c. The power p controls the decreasing ratio of α. This function gives the maximum silhouette force to the vertices that belong to the contour generators. All the other vertices, which are considered as concavities, are weighted inversely to their corresponding snake silhouette distance $d(S_c^{snake}, P_c\mathbf{v})$. This allows the vertices of the deformable model to *detach* themselves from the visual hull. A big value of p allows

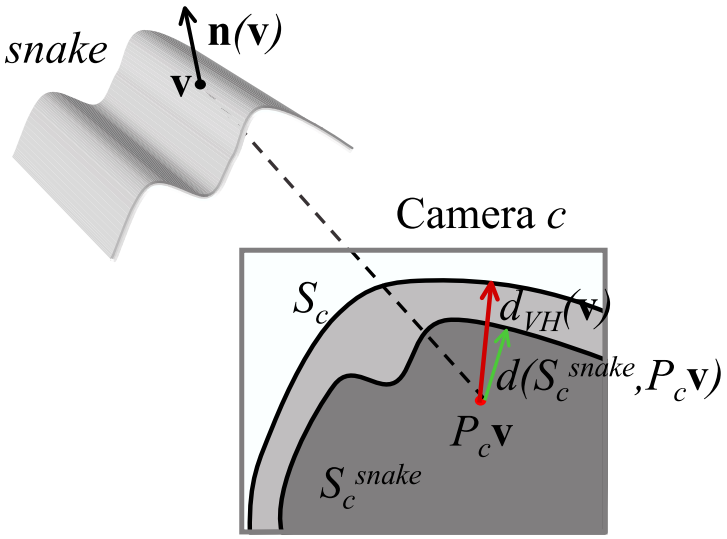


Figure 2.26: *Sketch of the distances involved in the silhouette force computation.*

an easier detachment. But if p is too big, the force becomes too local and does not allow smooth transitions between concavities and contours. The value used in practice is $p = 2$, which is a compromise between smoothness and concavity recovery.

The final silhouette force for a given vertex of the deformable model is a vector directed along the normal to the surface $\mathbf{n}(\mathbf{v})$ and its magnitude is the product of both components:

$$\mathcal{F}_{sil}(\mathbf{v}) = \alpha_c(\mathbf{v}) d_{VH}(\mathbf{v}) \mathbf{n}(\mathbf{v}) \tag{2.29}$$

The silhouette force definition we give here is somewhat similar to the silhouette force defined independently in [Nobuhara and Matsuyama, 2003]. The main difference with our formulation is their binary definition of the $\alpha_c(\mathbf{v})$ function. They only apply the silhouette force to the contour generator vertices, i.e., vertices where $\alpha_c(\mathbf{v}) = 1$. This hinders smooth transitions between contour generators and concavities, whereas in our case, we allow these transitions by using α values that range from 0 to 1. The size of the transition region is controlled by the exponent n in Eq. 2.29. Smooth transitions allow us to better deal with incoherences between silhouettes and stereo.

2.6 Internal Forces

In the classic 3D deformable model formulation [Cohen and Cohen, 1993], the internal energy $\mathcal{E}_{int}$ of a parametric surface $S(s,r)$ is composed of five terms involving the first and second order derivatives of the surface:

$$\mathcal{E}_{int}(S) = \int \gamma_{10} \left\| \frac{\partial S}{\partial s} \right\|^2 + \gamma_{01} \left\| \frac{\partial S}{\partial r} \right\|^2 + 2\gamma_{11} \left\| \frac{\partial^2 S}{\partial s \partial r} \right\|^2 + \gamma_{20} \left\| \frac{\partial^2 S}{\partial s^2} \right\|^2 + \gamma_{02} \left\| \frac{\partial^2 S}{\partial r^2} \right\|^2 ds\, dr, \tag{2.30}$$

where $(\gamma_{10}, \gamma_{01})$ control the elasticity of the surface, $(\gamma_{20}, \gamma_{02})$ its rigidity and γ_{11} its resistance to twist. These parameters are usually constant and grouped into two single

parameters γ_1 and γ_2 that control the elasticity and rigidity of the surface respectively ($\gamma_{10} = \gamma_{01} = \gamma_1$ and $\gamma_{20} = \gamma_{02} = \gamma_{11} = \gamma_2$). Such parameters adjust the model evolution in such as way that a high value of γ_1 tends to shorten the surface and eliminate loops while a high value of γ_2 penalizes curvature variations: makes the surface tend to the sphere (constant curvature). A local minimum of the energy $\mathcal{E}_{int}(S)$ satisfies the associated Euler-Lagrange equation, which gives us the following form for the internal force:

$$\mathcal{F}_{int}(S) = \gamma_1(\frac{\partial^2 S}{\partial s^2} + \frac{\partial^2 S}{\partial r^2}) - \gamma_2(\frac{\partial^4 S}{\partial s^4} + 2\frac{\partial^4 S}{\partial s^2 \partial r^2} + \frac{\partial^4 S}{\partial r^4}) = \gamma_1 \Delta S - \gamma_2 \Delta^2 S, \quad (2.31)$$

where Δ is the Laplacian operator and Δ^2 is the biharmonic operator.

The discrete version of the Laplacian operator $\tilde{\Delta}$ on a triangle mesh can be easily implemented using the umbrella operator, i.e., the operator that tries to move a given vertex $\mathbf{v}$ of the mesh to the center of gravity of its 1-ring neighborhood $\mathcal{N}_1(\mathbf{v})$:

$$\tilde{\Delta}\mathbf{v} = \left(\sum_{i \in \mathcal{N}_1(\mathbf{v})} \frac{\mathbf{v}_i}{m} \right) - \mathbf{v}, \quad (2.32)$$

where $\mathbf{v}_i$ are the neighbors of $\mathbf{v}$ and m is the total number of these neighbors (valence). Concerning the discrete version of the biharmonic operator $\tilde{\Delta}^2$, its implementation on simplex meshes is trivial since we just do $\tilde{\Delta}^2\mathbf{v} = \tilde{\Delta}(\tilde{\Delta}\mathbf{v})$, which can be obtained using the umbrella operator twice [Xu, 2000]. If we try the same formulation on our traditional triangle mesh, we obtain:

$$\begin{aligned}
\tilde{\Delta}(\tilde{\Delta}\mathbf{v}) &= \left(\sum_{i \in \mathcal{N}_1(\mathbf{v})} \frac{\tilde{\Delta}\mathbf{v}_i}{m} \right) - \tilde{\Delta}\mathbf{v}, \\
&= \sum_{i \in \mathcal{N}_1(\mathbf{v})} \left(\left(\sum_{j \in \mathcal{N}_1(\mathbf{v}_i)} \frac{\mathbf{v}_j}{m m_i} \right) - \frac{\mathbf{v}_i}{m} \right) - \left(\sum_{i \in \mathcal{N}_1(\mathbf{v})} \frac{\mathbf{v}_i}{m} \right) + \mathbf{v}, \quad (2.33) \\
&= \left(\sum_{i \in \mathcal{N}_1(\mathbf{v}),\ j \in \mathcal{N}_1(\mathbf{v}_i),\ \mathbf{v}_j \neq \mathbf{v}} \sum \frac{\mathbf{v}_j}{m m_i} \right) - \left(2 \sum_{i \in \mathcal{N}_1(\mathbf{v})} \frac{\mathbf{v}_i}{m} \right) + (1 + \sum_{i \in \mathcal{N}_1(\mathbf{v})} \frac{1}{m m_i})\mathbf{v},
\end{aligned}$$

where m_i is the valence of vertex $\mathbf{v}_i$. The important thing about this equation is that, if used as it is, it will not work for the general case of a triangle mesh, where $m_i \neq m$. The reason is simple: as a difference with the simplex case, the scalar that multiplies the vector $\mathbf{v}$ is not the same for all the vertices, it depends on the local connectivity. This scalar is very important since it gives the sensibility between a change on the vertex position and the corresponding change of the biharmonic operator. For example, when we use the Laplacian as an internal force, we want to apply a displacement to each vertex in order to locally cancel the Laplacian component. By iterating the process,

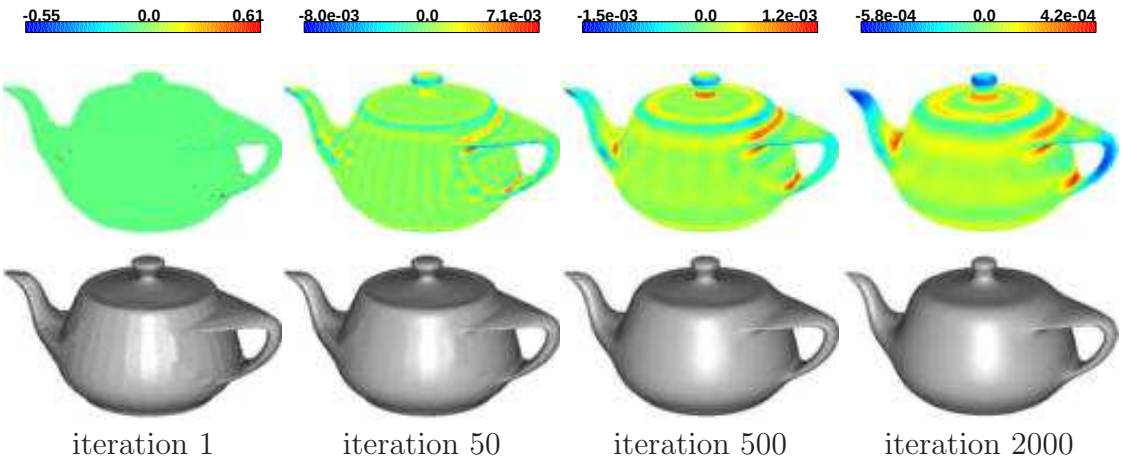


Figure 2.27: *Biharmonic filtering ($\gamma = 1,\ \rho = 1$) for an increasing number of iterations. Top: dot product between the biharmonic force vector ($-\tilde{\Delta}^2\mathbf{v}$) and the surface normal. Bottom: shaded view of the filtered model.*

we hope to globally minimize the Laplacian operator over the entire mesh. It happens that the scalar appearing in front of $\mathbf{v}$ in equation 2.32 is 1, which means that, if we move the vertex $\mathbf{v}$ exactly the Laplacian vector $\tilde{\Delta}\mathbf{v}$, we locally cancel the Laplacian component:

$$\tilde{\Delta}(\mathbf{v}+\tilde{\Delta}\mathbf{v}) = \sum_{i\in\mathcal{N}_1(\mathbf{v})}\frac{\mathbf{v}_i}{m} - (\mathbf{v}+\tilde{\Delta}\mathbf{v}) = \sum_{i\in\mathcal{N}_1(\mathbf{v})}\frac{\mathbf{v}_i}{m} - \mathbf{v} - \sum_{i\in\mathcal{N}_1(\mathbf{v})}\frac{\mathbf{v}_i}{m} + \mathbf{v} = 0. \quad (2.34)$$

To proceed in the same way with the biharmonic operator, according to equation 2.33 we need to scale the biharmonic vector $\tilde{\Delta}(\tilde{\Delta}\mathbf{v})$ by the inverse of $1+\sum_{i\in\mathcal{N}_1(\mathbf{v})}\frac{1}{mm_i}$ and define the following scaled biharmonic operator:

$$\tilde{\Delta}^2\mathbf{v} = \frac{1}{1+\sum_{i\in\mathcal{N}_1(\mathbf{v})}\frac{1}{mm_i}}\tilde{\Delta}(\tilde{\Delta}\mathbf{v}), \quad (2.35)$$

We can then cancel the biharmonic component by moving $\mathbf{v}$ exactly the scaled biharmonic vector $\tilde{\Delta}^2\mathbf{v}$:

$$\tilde{\Delta}^2(\mathbf{v}-\tilde{\Delta}^2\mathbf{v}) = 0. \quad (2.36)$$

The total internal force on a mesh vertex $\mathbf{v}$ as:

$$\mathcal{F}_{int}(\mathbf{v}) = \gamma_1\tilde{\Delta}\mathbf{v} - \gamma_2\tilde{\Delta}^2\mathbf{v}. \quad (2.37)$$

In order to better integrate the internal force into the deformable model, we can reformulate the two internal parameters γ_1, γ_2 as an overall internal weight γ and a rigidity/elasticity ratio $\rho \in [0,1]$, which gives us:

$$\mathcal{F}_{int}(\mathbf{v}) = \gamma((1-\rho)\tilde{\Delta}\mathbf{v} - \rho\tilde{\Delta}^2\mathbf{v}). \quad (2.38)$$

We see in Fig. 2.27 the evolution of a triangle mesh under the single biharmonic force ($\gamma = 1,\ \rho = 1$). The object is filtered in the very first iterations and then it

is almost not deformed, which shows the great numerical stability of the proposed biharmonic operator.

It is worth noting that, because of the definition of the umbrella operator in Eqn. 2.32, the implemented biharmonic operator is not invariant to the mesh sampling, i.e., it requires the edges to have an homogeneous size, as discussed in [Desbrun et al., 1999]. However, since it is going to be used in a deformable model with automatic decimation and refinement, this is not a severe handicap. As a difference with the curvature flow proposed in [Desbrun et al., 1999], the proposed biharmonic operator is obtained as a weighted sum of neighbors in the 2-ring neighborhood, which makes it very easy to implement and with the advantage of giving directly a 3D vector with the right 3D units instead of a curvature tensor.

2.7 Mesh Control

Since the texture force $\mathcal{F}_{tex}$ can sometimes be tangent to the surface of the deformable model, we do not use the force $\mathcal{F}_{tex}$ itself but its projection $\mathcal{F}^{\mathbf{n}}_{tex}$ along the surface normal $\mathbf{n}(\mathbf{v})$:

$$\mathcal{F}^{\mathbf{n}}_{tex}(\mathbf{v}) = (\mathcal{F}_{tex}(\mathbf{v}) \cdot \mathbf{n}(\mathbf{v}))\mathbf{n}(\mathbf{v}). \tag{2.39}$$

This prevents problems of coherence in the force of neighbor vertices and helps the internal force to keep a well-shaped surface.

The deformable model evolution process (equation 2.4) at the k^{th} iteration can then be written as the evolution of all the vertices of the mesh $\mathbf{v}_i$:

$$\mathbf{v}_i^{k+1} = \mathbf{v}_i^k + \Delta t(\mathcal{F}^{\mathbf{n}}_{tex}(\mathbf{v}_i^k) + \beta\mathcal{F}_{sil}(\mathbf{v}_i^k) + \gamma\mathcal{F}_{int}(\mathbf{v}_i^k)), \tag{2.40}$$

where Δt is the time step and β and γ are the weights of the silhouette force and the regularization term, relative to the texture force. Equation 2.40 is iterated until convergence of all the vertices of the mesh is achieved. The time step Δt has to be chosen as a compromise between the stability of the process and the convergence time. An additional remeshing step is done at the end of each iteration in order to maintain a minimum and a maximum distance between neighbor vertices of the mesh. This is obtained by a controlled decimation and refinement of the mesh. The decimation is based on the edge collapse operator [Hoppe et al., 1993] and the refinement is based on the $\sqrt{3}$-subdivision scheme [Kobbelt, 2000].

2.8 Texture Mapping

Once the deformable model has converged, we can map the original set of color images into the 3D object surface to create a texture map. The method used to create the texture map is a particle-based approach such as the one described in [Soucy et al., 1996], [Schmitt and Yemez, 1999] or [Lensch et al., 2001]. This approach has been extended to filter the highlights that may be present in the images and to correctly deal with different triangle texture sizes.

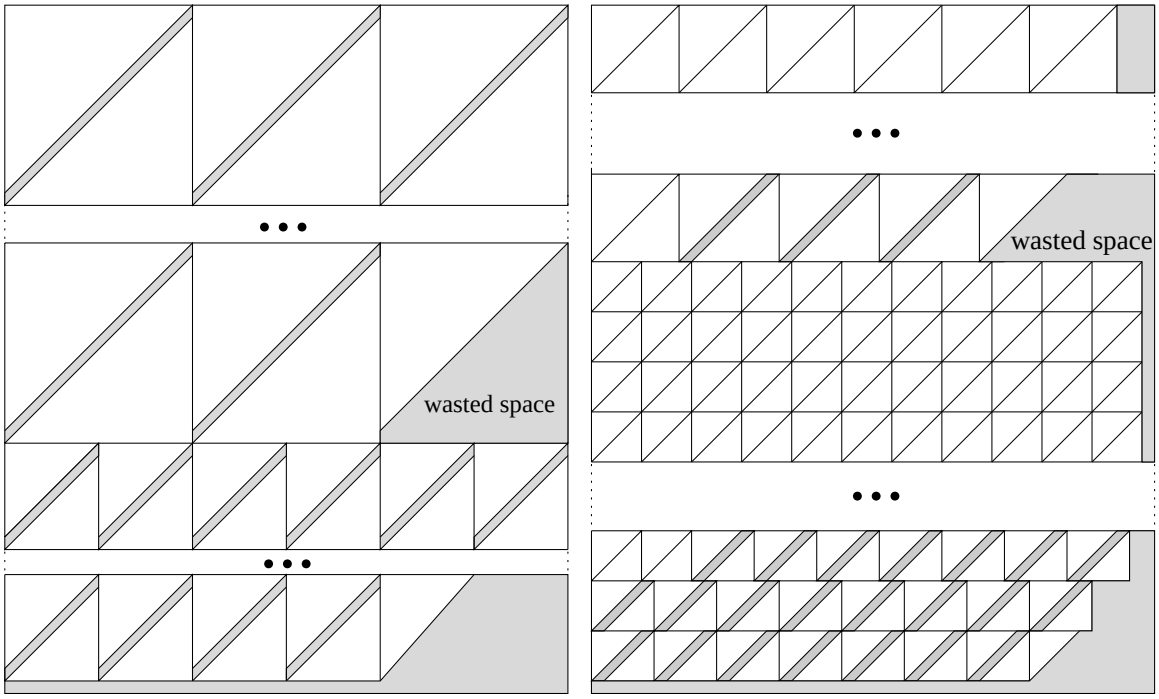


Figure 2.28: *Texture map as a set of texture tiles. Left: partition proposed in [Soucy et al., 1996], the width of the tiles being a power of 2. Right: partition that we propose with most of the triangles grouped into quads, the width of the tiles being an odd number.*

The algorithm to create a texture map is independent of the rest of the reconstruction method. The only required input is a 3D mesh model (no matter how it was obtained) and its corresponding set of calibrated images. The first choice in the texture map computation is the number of texture patches we want. Since we are going to deal with complex topologies and closed surfaces, it is impossible to have only one texture patch for an entire object. One advantage of triangle meshes is that there is an obvious partitioning of the mesh: the triangle. Considering one texture patch per triangle has two advantages: zero rectification distortion and a trivial implementation. The main disadvantage is that because we have so many patches, the redundancy of the frontiers of the patches is high: the same edge appears twice in the texture map and the same vertex appears an average of 6 times[6]. Worse, since texture maps are rectangular images, the greedy method to store the texture of a triangle is to use a square, which wastes half of the space of the square. Using bigger patches (a set of connected triangles) would reduce the length of the frontiers and thus the redundancy inside the texture map. However, just in the same way that a triangle cannot fit into a square, using bigger patches poses also the problem of minimizing the space between patches inside the rectangular image.

We have implemented an improved version of the "patch per triangle" texture map of [Soucy et al., 1996]. The first hypothesis of this algorithm is that triangles have a good aspect ratio (close to equilateral) so the texture patch can be chosen

[6] According to the Eulerian properties of our reconstructed meshes

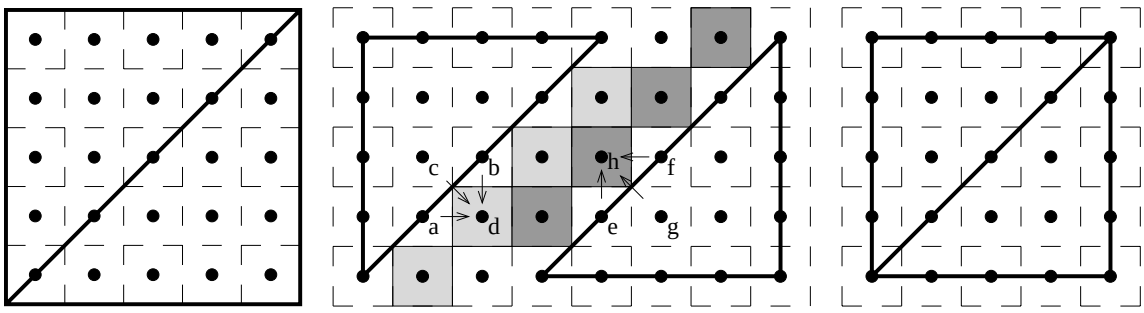


Figure 2.29: *Triangle texture mapping. Left: nearest-neighbor interpolation; the triangle can be mapped to fit the entire half-square. Middle: bilinear interpolation; the interpolation algorithm needs a "border" of at least 0.5 texels, which requires doubling the diagonal, the color of the border pixels being computed as* $d = a + b - c$, $h = e + f - g$. *Right: the diagonal does not need to be doubled when mapping a quad.*

as an equilateral triangle inscribed in a square tile of fixed size, e.g., 5×5 or 8×8 pixels. Since graphic cards prefer texture maps with the size being a power of 2, e.g. 1024 or 2048 pixels, individual triangle textures are also a power of 2 in order to perfectly fit into the texture map. To correctly deal with simplified meshes (with very different triangle sizes), different classes of triangles with increasing power of 2 sizes are provided, e.g., 2×2, 4×4, 8×8, etc. However, since using one square per triangle wastes almost half of the space, they propose to vertically align 2 triangles of the same class into a no longer square tile, which improves the spatial efficiency. This makes tiles of size 2×3, 4×5, 8×9, etc. All the triangle texture tiles with the same size are put in the texture map along rows. When the maximum texture map width is reached, a new row of tiles is started and so on. When the last tile is added, we pass to the following tile size and we start a new row of tiles, wasting an average of half a row (see Fig. 2.28 left). Even though we would like to exploit the left space of the last row by placing new tiles of the next class, this would not be fully satisfactory since tiles are no longer square, which implies a bad alignment between tiles of different sizes. A simple solution to this problem consists of adjusting the number of triangles per class in order to always have an even number of triangles, except for the last class. This ensures the absence of unused space at the last row of every class but the last one.

The algorithm we have just described is the one used in [Soucy et al., 1996]. Now, one important question about this algorithm is that it is designed to work only with a Nearest-Neighbor interpolation for the texture rendering process. This is a capital assumption since it greatly simplifies the constraints to correctly render the triangles. In particular, it does not necessitate to deal with border pixel interpolation issues, as illustrated in Fig. 2.29 left. We have extended this algorithm in order to correctly render the texture using bilinear interpolation. Using bilinear interpolation requires a band around the triangle of at least 0.5 texels in order to correctly compute the bilinear interpolation (see Fig. 2.29 middle). It implies the use of the second diagonal (shown in gray level in Fig. 2.29 middle) for interpolation purposes, which makes that, if we want to align two triangles together into the same tile, we have to add 3 pixels to one of the tile dimensions, as shown in Fig. 2.29 middle. Doubling the diagonal is

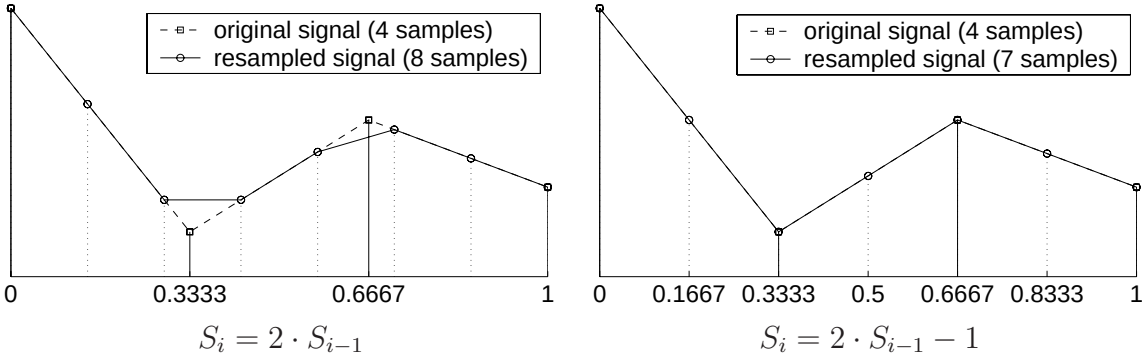


Figure 2.30: *An example of signal resampling using linear interpolation. The original signal has 4 samples (shown by squares). Left: bad interpolation using an* even *number of samples (8 samples). Right: correct interpolation using an* odd *number of samples (7 samples).*

not necessary when mapping a pair of adjacent triangles forming a quad as shown in Fig. 2.29 right.

The second consequence of using bilinear interpolation for the rendering process is that the frontiers between triangles with different resolutions cannot be processed in order to ensure their continuity as in [Soucy et al., 1996]. As shown in Fig. 2.30 left, it is a simple matter of signal sampling to see that it is imposible to perfectly reproduce the low resolution signal (square samples) using the high resolution samples (shown by circles), no matter what their values are. This is simply due to the fact that the new samples are not aligned with the original ones. The solution is to no longer use an *even* series for the triangle sizes but an *odd* series:

$$S_i = 2 \cdot S_{i-1} - 1, \;\; i > 0, \;\; S_0 > 1, \tag{2.41}$$

which is equivalent to

$$S_i = 2^i \cdot S_0 - 1, \;\; i > 0, \;\; S_0 > 1. \tag{2.42}$$

If we generate the different texture triangle sizes using Eqn. 2.42 for $S_0 = 2$, we get the following size sequence: $\{2, 3, 5, 9, 17, 33, 65, 129, ...\}$. To ensure the bilinear continuity between two triangles with different resolution, we just interpolate along the diagonal of the higher resolution triangle in a linear way (as shown in Fig. 2.30 right). We show in Fig. 2.31 the result of rendering two triangles of different class using power of 2 *even* sizes (Fig. 2.31 top), and with power of 2 *odd* sizes (Fig. 2.31 bottom).

Since texture tiles have no longer a power of 2 size, a row of tiles will generally not perfectly fit into the width of the texture map. Consequently, there will be a left space at the end of each row of tiles. Another source of space waste is that aligning two triangles into a rectangular tile requires to increase one of the dimensions of the tile by 3 pixels. This can be avoided by grouping adjacent triangles of the same class into quads in order to map them into a single texture square tile (as in Fig. 2.29 right). In practice, the percentage of triangles with a peer (in our meshes) is better than 90%, which is quite good. The triangles that cannot find a peer are grouped into rectangular tiles, which implies duplicating the diagonal (see Fig. 2.28 right). Using this kind of partition reduces the wasted space to less than 1%.

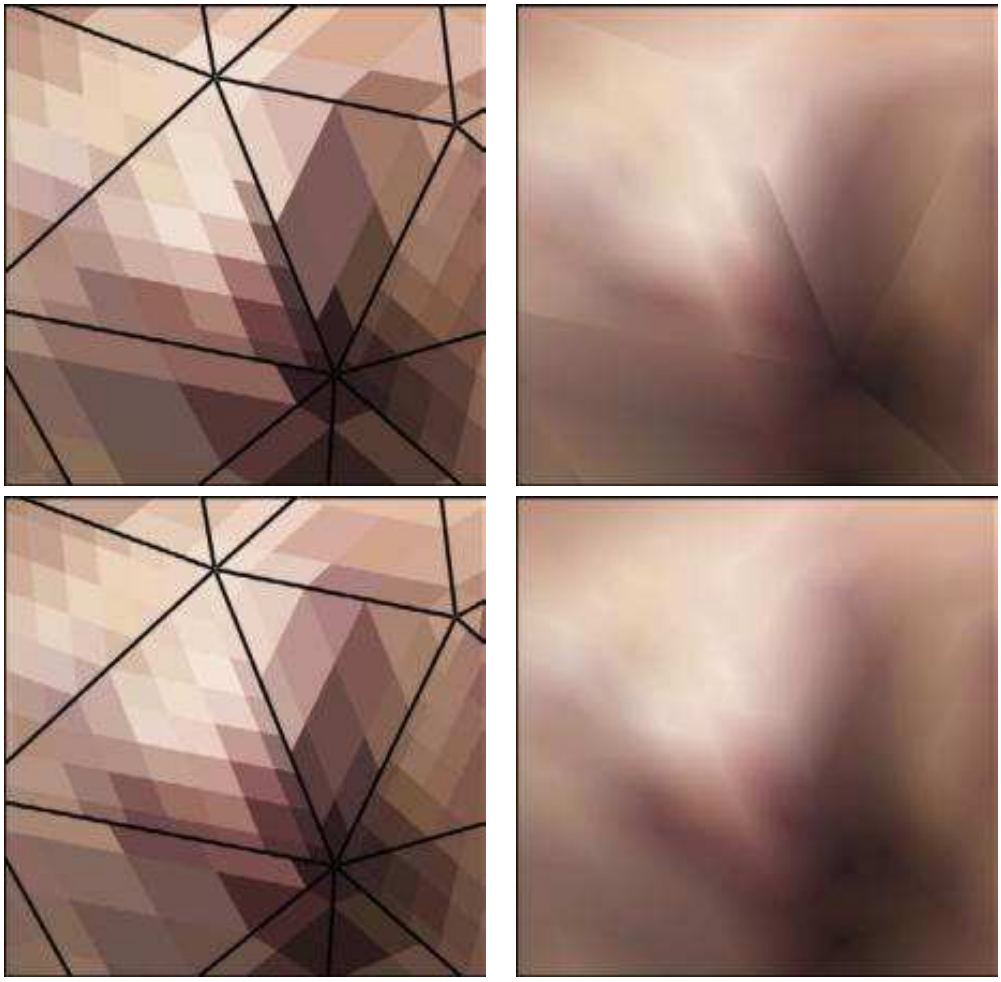


Figure 2.31: *Example of texture rendering using nearest-neighbor interpolation (left column) and bilinear interpolation (right column) for even and odd triangle sizes. Top: texture artifacts between two triangles using even tile sizes (4 and 8 pixels respectively). Bottom: texture rendering without artifacts of the same triangles using odd tile sizes (5 and 9 pixels respectively).*

After having described the algorithm to place a texture tile into the global texture map, the last step to detail is how to compute the texture tile for a given triangle. The algorithm to compute the triangle texture tile can be decomposed into 2 different independent steps: the choice of the triangle texture size, and the filling of the triangle texture tile from the color images.

2.8.1 Choice of the triangle tile size

If we dispose of a given list of possible tile sizes, we want to assign to each triangle one of the available sizes under some user-defined constraints. These user-defined constraints are in general of two types:

- a desired texture quality ratio between the original images and the final texture

map,

- a maximum texture size.

Fixed texture quality ratio

The simplest scenario is to give a desired quality ratio between the original images and the final texture map. For a given triangle, we choose the view that *best* sees the triangle in a geometrical sense (the one that sees it the most fronto-parallel). Then we project the triangle into that view and compute the maximum edge size in pixels. The maximum edge size is multiplied by the desired quality ratio and the final tile size is chosen as the closest available size to the maximum edge size.

Once we have computed the tile size of each triangle, we group the triangles into quads in such a way that only two triangles with the same tile size can be paired together. This is simply done by randomly picking one triangle of the list and checking if it has any neighbor with the same tile size that is not already into any other quad. We iterate this process until no new quads can be created. At the end of the process, we proceed to an additional *upgrading* step where, for all the triangles that do not belong to a quad, we test if any of its neighbors is also alone (which means that they do not have the same tile size), and group them into a quad. The tile size will be the greatest tile size of both triangles, i.e., we always upgrade. All the remaining triangles with the same tile size i are paired together into rectangular tiles of size $S_i \times (S_i + 3)$.

As in [Soucy et al., 1996], a last adjustement step is performed to fill the unused space of the last row of every class. Starting from the class with the largest tile size, if there is still space for n tiles, the algorithm upgrades the n quads with the largest edge size belonging to the nearest inferior class.

To compute the final texture size, we fix the texture width to a power of 2 (typically 1024 or 2048) and we fill the texture map as shown in Fig. 2.28 right.

Fixed texture size

In a more complex scenario, the user fixes both the width and the height of the texture map. For a fixed texture width, the texture height is a function of the texture quality ratio. The goal is to find the best quality ratio such that the final texture height fits into the user defined height. This is achieved using a dichotomy approach starting from a quality ratio of 100%. Although it is worth noting that the quality ratio is not a monotonic function of the texture height (mainly due to the quad grouping process), in practice it is quite monotonic (e.g., see Fig. 2.34), so the algorithm works well.

2.8.2 Filling the triangle texture tile from the color images

In order to fill the texture tile of a given triangle described by its 3 vertices $\mathbf{v}_i$, $\mathbf{v}_j$ and $\mathbf{v}_k$, we sample it into particles as in Fig. 2.32 using the local parameterization (s, t):

$$\mathbf{p}(s,t) = (1-s-t)\mathbf{v}_i + s\mathbf{v}_j + t\mathbf{v}_k, \;\; s \in [0,1], \;\; t \in [0,1], \;\; s+t \leq 1. \tag{2.43}$$

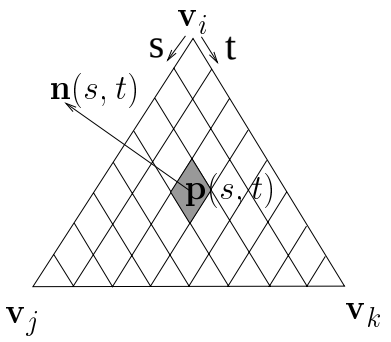


Figure 2.32: *Decomposition of a triangle into particles.*

The color of each particle is stored in a texel of the texture tile. To compute the color of a particle from a particular camera, we simply project the particle into that camera and sample the image at the projected point.

If, as in our case, we dispose of a set of cameras, things are not so easy. For the same triangle, we dispose of several (in some cases many) different views that actually see it. Two problems arise:

- how to ensure smooth transitions between adjacent triangles,
- how to choose the set of cameras that will contribute to the texture.

Color interpolation between different views

Ensuring smooth transitions between adjacent triangles is a very important problem. If we do not ensure the texture continuity in the frontier of the patches, this produces very important artifacts in the final result. The method we propose to solve this problem is choosing the set of cameras *per vertex* and not *per triangle* and then *interpolate* the set of cameras for a given particle inside the triangle. For a given triangle we have then three different sets of cameras $\{C_i\}$, $\{C_j\}$, and $\{C_k\}$. The way to interpolate the camera sets is, for any particle in the triangle, to compute the particle color for each camera set independently and interpolate the resulting colors using the local parameterization (s,t):

$$\begin{array}{rcl} \mathbf{c}(s,t) & = & (1-s-t)\mathbf{c}_i(s,t) + s\mathbf{c}_j(s,t) + t\mathbf{c}_k(s,t), \\ \mathbf{c}_i(s,t) & = & \{C_i\}(\mathbf{p}(s,t), \mathbf{n}(s,t)), \\ \mathbf{c}_j(s,t) & = & \{C_j\}(\mathbf{p}(s,t), \mathbf{n}(s,t)), \\ \mathbf{c}_k(s,t) & = & \{C_k\}(\mathbf{p}(s,t), \mathbf{n}(s,t)), \end{array} \tag{2.44}$$

where $\{C\}(\mathbf{p}, \mathbf{n})$ is the color vector computed with the set of cameras $\{C\}$ for a 3D point $\mathbf{p}$ with a surface normal $\mathbf{n}$.

Using equations 2.44 ensures that the interpolated colors along an edge are the same for the two adjacent triangles, since the color depends only on the camera sets of the two vertices defining the edge. This is true if both triangles have the same tile size, i.e., the same number of samples along the shared edge. If they have not, there will

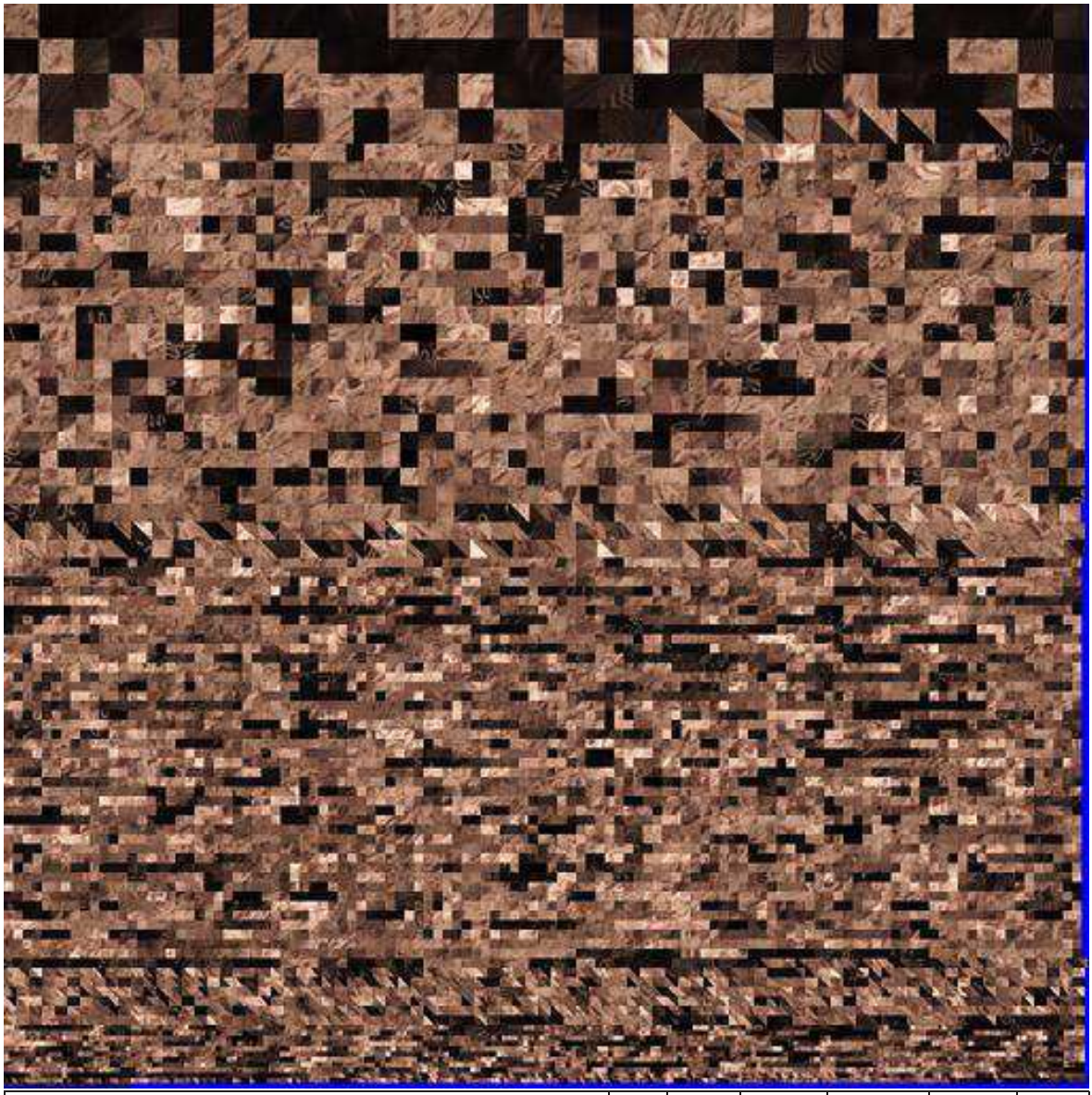
be an aliasing effect (as already shown in Fig. 2.31 top right) when passing from one triangle to another, since the rendering algorithm processes each triangle separately. Since we use an odd tile size series, these artifacts can be simply eliminated (see Fig. 2.31 bottom right) by linear interpolation along the shared edge, as in the one dimensional case of Fig. 2.30.

Choice of the camera set that correctly sees a vertex

The last point to detail is how to choose the camera set $\{C_i\}$ associated to a vertex $\mathbf{v}_i$, and how to obtain the final rgb color for a 3D point $\mathbf{p}$ with a surface normal $\mathbf{n}$. The proposed procedure is similar to [Schmitt and Yemez, 1999]. An initial camera set is constructed with all the cameras that see the vertex without occlusion. Occlusion is computed using a classic z-buffer technique. The resulting camera set is further filtered by testing if any of the images is saturated around the projection of the vertex into that image. The saturation test is implemented by comparing the color value of the vertex on each image with the mean value of the vertex along the whole sequence, and it is done separately on each color channel. The resulting camera set is composed of cameras that see the vertex without highlights. At this point different techniques may be used to obtain a final particle color. We can, for example, compute the weighted mean color of the camera set:

$$\begin{aligned} \{C_i\}(\mathbf{p},\mathbf{n}) &= \frac{1}{\sum_{j\in\{C_i\}} w_j} \sum_{j\in\{C_i\}} w_j I_j(P_j\mathbf{p}), \\ w_j &= -\mathbf{n}\cdot\mathbf{s}_j, \end{aligned} \tag{2.45}$$

where $\mathbf{s}_j$ corresponds to the viewing direction for $\mathbf{p}$ from camera j, P_j is the projection matrix of the camera j, and $I_j(\cdot)$ is the image color value at the given 2D point. More complicated methods can be applied like the weighted median color, where each color component is chosen using a weighted median. However, using so many views blurs the final texture map. Better results are obtained when using only the two views in the selected camera set that best sees the particle in a geometrical sense. Then the color is interpolated as in equation 2.45, but only between the two best views.



tile size (pixels)	2	3	5	9	17	33
total number of triangles	44	289	4073	10718	2628	216
number of triangles grouped into quads	42	254	3652	9646	2370	194

Figure 2.33: *Example of a texture map with a user-defined size of* 1024×1024 *pixels. The mesh has 17968 triangles, 16158 of them being grouped into quads. The wasted space (shown in blue) is of 0.74%.*

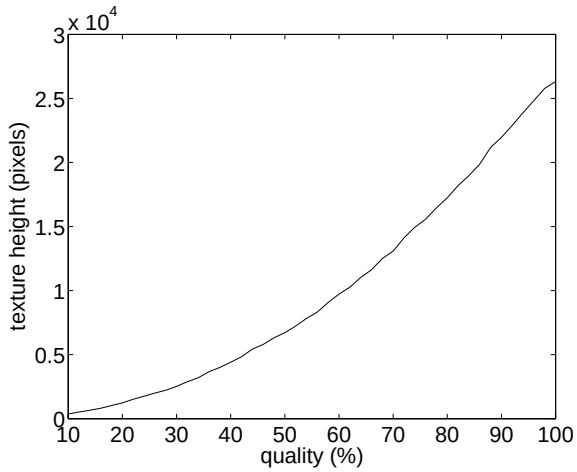


Figure 2.34: *Example of quality ratio estimation used in Fig. 2.33. The original images are 2008x3040 pixels. For a fixed texture width of 1024 pixels, and a desired texture height of 1024 pixels, the corresponding quality ratio is* 18.03%.

Figure 2.35: *Triangle mesh and textured mesh using the texture map of Fig. 2.33. We thank Houman Borouchaki for this simplified version of the bigHead model.*

2.9 Conclusions

We have presented a new approach to 3D object modeling based on the fusion of texture and silhouette information. We propose a full system approach where different known techniques are used and improved to provide high quality results.

Our main contributions are:

- the 1-D bin voting technique, which allows taking a robust decision based on n-stereo correlation curves,
- the implementation of a multi-correlation 3D voting technique, which allows us to extract the shape information from a set of color images in a robust way, improving the performance under real light conditions, especially in the presence of highlights,
- the adaptation of the GVF technique, commonly used in medical imaging, to our specific 3D object modeling problem, which has necessitated a multi-grid implementation that, to our knowledge, has never been used before for 3D object modeling,
- the definition of a deformable model silhouette-based force, which allows integrating the silhouette information into the deformable model evolution,
- the adaptation of the classic biharmonic operator, commonly used in simplex meshes, to a triangular mesh,
- the definition of a practical way to interpolate between different color images in order to create a continuous texture map with filtered highlights,
- the implementation of a texture mapping technique that assigns different texture resolutions to different triangles, while keeping smooth bilinear interpolation transitions between adjacent triangles,
- and finally, the global quality of the entire system, which provides high quality results under real conditions, which is demonstrated by the quantity and quality of the reconstructed objects.

The two main limitations of the algorithm are also its two main sources of robustness: the volume voting approach and the topology constant snake approach. The voting approach allows good reconstructions in the presence of highlights, but it also limits the maximum resolution of the 3D model. A way to overcome this limitation could be introducing the final model into another snake evolution where the texture energy computation would take into account the current shape (visibility and tangent plane or quadric based cross-correlation). Since the initial model is already very close to the real surface, only some iterations would suffice to converge. The second drawback is the topology constant evolution. It allows a guaranteed topology of the final model but it is also a limitation for some kind of objects where the topology cannot be captured by the visual hull. A feasible solution would be to detect self collisions of

the deformable model [Lachaud and Montanvert, 1999], and to launch a local level-set based method in order to recover the correct topology. Also, as discussed in Section 3.3, we need to further investigate the visibility handling of the correlation surface, a more general silhouette force, and a better handling of objects with sharp edges. Additional work includes:

- an improved strategy to detect local convergence of the snake in order to freeze optimized regions and to accelerate the evolution in the empty regions,
- the possible use of the surface curvatures to allow a multi-resolution evolution of the mesh,
- some more advanced work in the generation of the texture. In particular, since we know the geometry of the object, we could apply more advanced signal processing techniques to better remove self-shadows and possibly, under calibrated lighting conditions, partially estimate the color bidirectional reflectance distribution function (BRDF) of the surface elements of the object.

Chapter 3

Results

In this chapter we present a selection of our 3D reconstruction results obtained with the techniques described in chapters 1 and 2. We describe the general setup in section 1 and present the different types of interesting cases that we have encountered in our experiments in the following sections.

3.1 Acquisition Setup

The general acquisition setup consists of a turntable and a fixed digital camera. The object to be scanned is placed on the turn table (see Fig. 3.1) and a sequence of photographs is taken with a fixed angle step: typically 10 degrees, which gives a sequence of 36 images. This is the minimum configuration (1 color sequence) for the reconstruction algorithm. Taking only one sequence of images requires both the automatic extraction of the silhouettes and the auto-calibration of the system. Since the auto-calibration is also based on the silhouettes, it is preferable to have a homogeneous background in order to better binarize the silhouettes. The background color does not need to be colored, very good results are obtained with matte gray. However an important aspect is to have a good focus of the object *on* the contour silhouettes, so that they are not blurred.

In order to simplify the silhouette extraction task, an additional sequence of images can be taken with an active background, which allows recovering very high quality silhouettes. However this second acquisition is only useful if the turntable is accurate enough to repeat the same poses as for the object sequence. A third image sequence of a calibration pattern can be employed to calibrate the entire system with any classic method ([Lavest et al., 1998] in our case). This calibration allows recovering the intrinsic parameters and the camera poses of the *calibration pattern sequence*. Since we are interested in the calibration of the *object sequence*, we need exactly the same setup for both acquisitions. This is quite easy to ensure for the intrinsic parameters of the camera (that are fixed) and the circular motion (rotation axis and translation). However it is more difficult to ensure for the turntable angles, which requires a good turntable precision (better than 0.1 degrees for high resolution images).

Finally, a second sequence of the object in a different position can be taken to

Figure 3.1: *Acquisition setup.*

recover hidden parts of the object. The rigid motion between this second sequence and the first object sequence can be recovered in two ways: using a calibration pattern or using the silhouettes with the technique described in chapter 1. Using a calibration pattern is quite constraining since the pattern in the same position has to appear in at least one photograph of each of the sequences. This is only possible by changing the pose of the camera and leaving unchanged the position of the object on the turntable. The silhouette-based calibration allows us to change the object pose manually while the rest of the setup (turntable and camera) can remain fixed, which is much simpler from a photographer point of view.

In the following sections we present some practical cases with different degrees of acquisition constraints under circular motion, ranging from a perfect controlled environment with different sequences for the color images, the silhouettes and the calibration pattern in section 3.2, to a single color sequence that serves to extract the silhouettes, which are then used to calibrate and reconstruct in section 3.4. Finally, some results are presented using two single axis rotation sequences in section 3.5.

The values of β and γ change very little from one object to another. Because the snake iteration is always done in the voxel coordinate system of the GVF octree, the value of β only depends on the ratio between the image size and the octree size. A typical value is between 0.1 and 0.2. Typical values of γ are between 0.1 and 0.25, depending on the required smoothness, while a good ratio rigidity/elasticity is $\rho = 0.8$.

Computation times are dominated by the correlation voting step: a typical computation time for 36 images of 6 Mpixels is of 3 hours on a Pentium4 1.4GHz machine, although this time strongly depends on the shape of the visual hull and on the size of the object in pixels, i.e., the area of its silhouettes.

3.2 3 Sequences (Color+Silhouette+Calibration)

In this section we dispose of 3 image sequences, one for the object, another for the silhouettes and a last one for the calibration pattern. The sequence of the calibration pattern is used to recover the intrinsic parameters, the rotation axis and the transla-

tion. The angles of the sequence are supposed to be known. Silhouettes are extracted from the silhouette sequence by simple binarization.

3.2.1 BigHead Sequence

Figure 3.2: *Some of the original images (2008x3040 pixels) used in the reconstruction of the BigHead object. Top: 9 samples of a total of 36 color images. Bottom: 9 samples of a total of 36 silhouette images.*

The bigHead sequence is quite favorable for stereo reconstruction since the object is very well textured. The only deviations from the hypothesis of a Lambertian surface are the self-shadows, that are quite visible in Fig. 3.2 top, and the lack of texture of the object support, where the only available textured zone is the tick marks. This can be confirmed looking at the correlation voting volume in Fig. 3.3 left, where we can clearly distinguish the tick marks of the support. Although the octree resolution is already high (10 levels), the correlation values are very high (> 450). This is due to the availability of many texture details that produce strong correlation scores.

We show in Fig. 3.3 top middle the octree partition used for the GVF computation. We can appreciate that, even with the constraint of a slow resolution change, the multi-grid partition succeeds in efficiently filling the volume around the "*surface*" of correlation and computing a high resolution GVF (see Fig. 3.3 bottom middle).

In Fig. 3.4 we illustrate the two components of the silhouette force after convergence, which permits recovering the contour generators of the object (vertices with an α value close to 1 in Fig. 3.4 right).

Starting from the visual hull (Fig. 3.5 left), the algorithm is able to converge to a final solution (Fig. 3.5 middle), which is then used to compute a texture map (Fig. 3.5 right). We can appreciate the quality of both the recovered surface and the computed

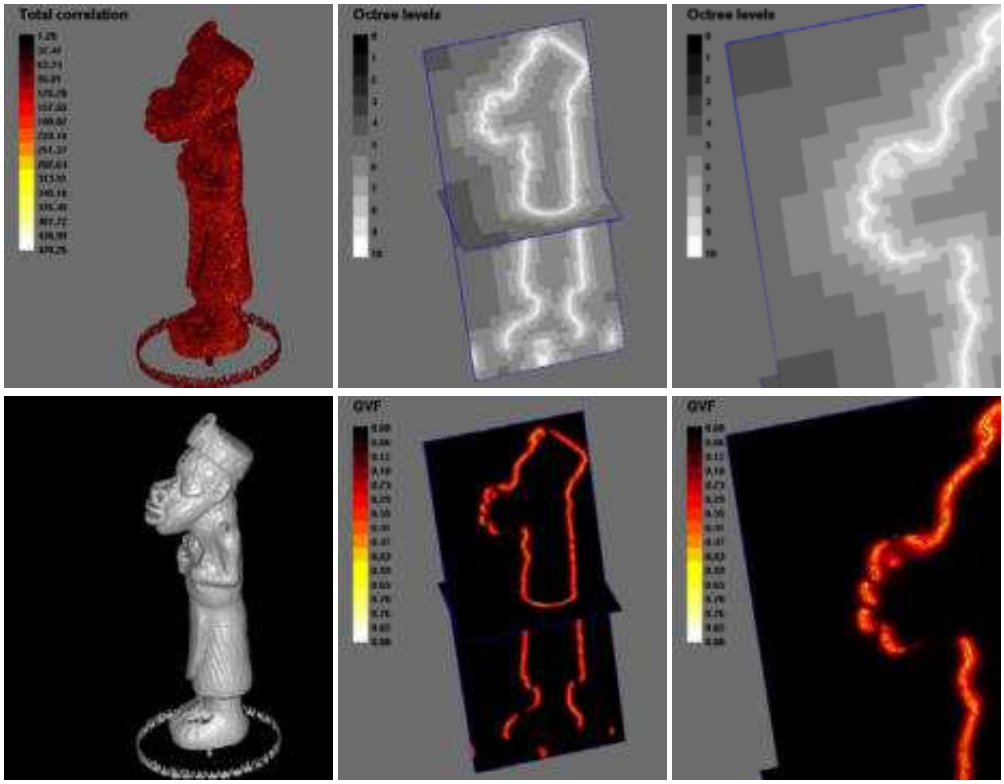


Figure 3.3: *Stereo force used for the reconstruction of the BigHead object. Top left: stereo correlation voting volume. Bottom left: rendered view of the same volume after binarization with a threshold of 30. Top middle: octree partition used for the computation of the GVF (see detail on the right). Bottom middle: GVF norm.*

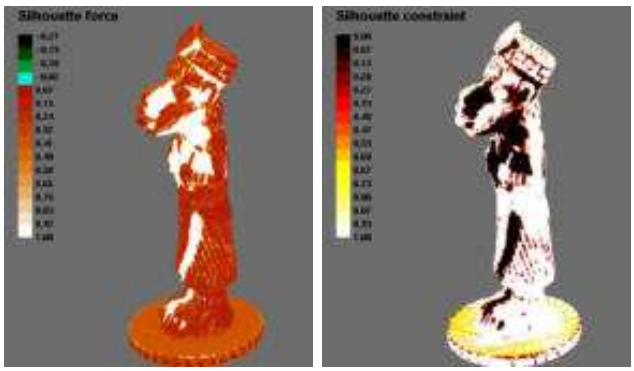


Figure 3.4: *Silhouette force used for the reconstruction of the BigHead object. Left: d_{VH} silhouette force component after convergence. Right: α silhouette force component after convergence.*

texture map. In Fig. 3.6 we dispose of a vector snapshot of the final mesh that allows us to appreciate the great quality of the triangular surface.

Figure 3.5: *Complete reconstruction sequence of the BigHead object. From left to right: visual hull, snake after convergence and texture mapping. The reconstructed model has 114496 vertices.*

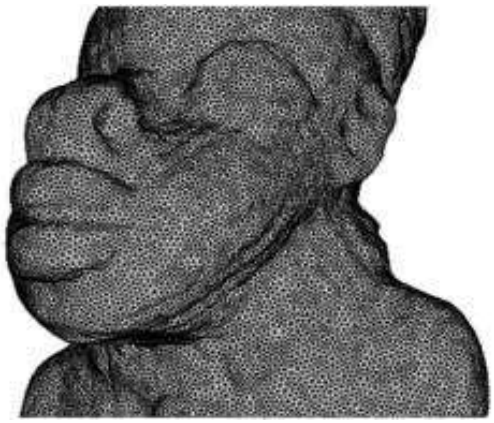


Figure 3.6: *Mesh detail of the BigHead object.*

3.2.2 Twins Sequence

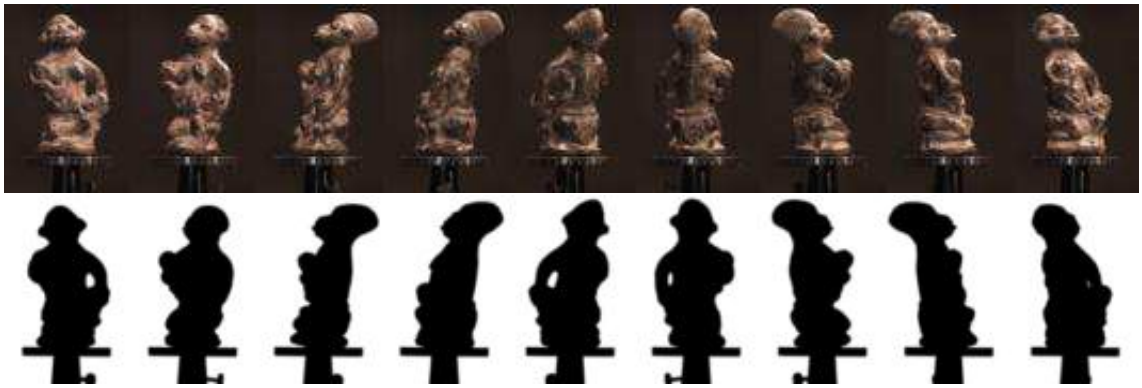


Figure 3.7: *Some of the original images used in the reconstruction of the Twins object. Top: 9 samples of a total of 36 color images. Bottom: 9 samples of a total of 36 silhouette images.*

The Twins model is a more difficult example than the BigHead one. Its topology is more complex (although not really so much: only one hole), and the surface properties are far from being Lambertian (see Fig. 3.7). Furthermore, there are some images which are completely saturated due to strong specular reflections (see Fig. 3.8).

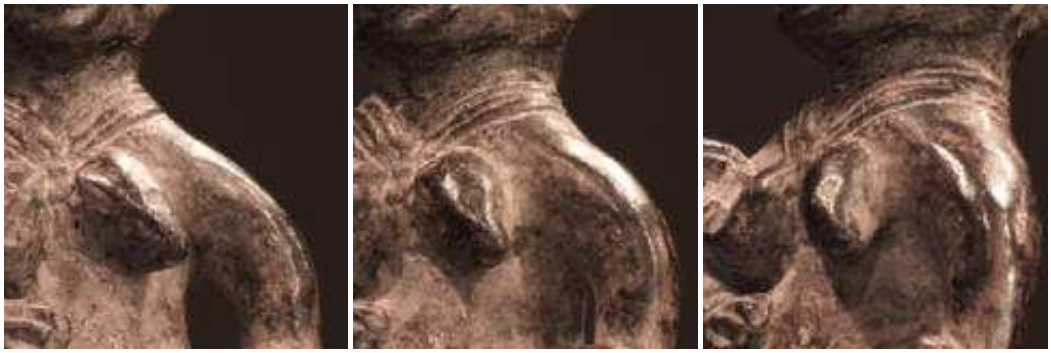


Figure 3.8: *Detail of the original Twins sequence. We can observe the existence of strong highlights on the surface. Since the highlights are not fixed from one image to another, the correlation voting approach allows recovering the right surface, as shown in Fig. 3.10.*

Because the relative position of the object from the lights changes, the highlights "move" all over the object surface, which implies that the saturation zone is not the same for all the views (see Fig. 3.8). The correlation voting approach permits recovering the surface in these regions (see Fig. 3.9) since, even if the stereo does not work in one image, there will be another image that sees the same piece of surface without highlight.

In Fig. 3.10 we present the complete reconstruction steps of the Twins model. We can appreciate the differences between the visual hull initialization (Fig. 3.10 left) and the final reconstruction (Fig. 3.10 middle). In Fig. 3.10 right we show the total force on each vertex after convergence. The jeopardized pattern observed on the

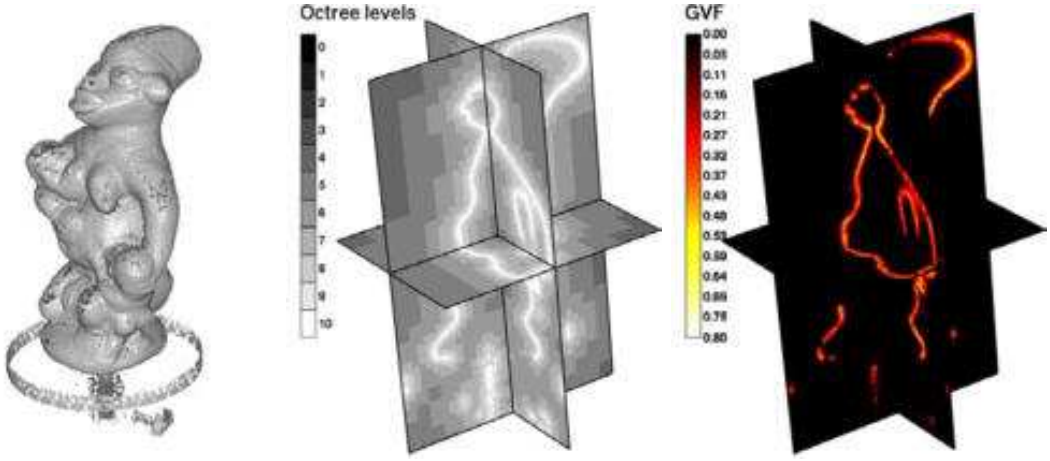


Figure 3.9: *Stereo force computation for the Twins model. From left to right: multi correlation voting volume, octree partition and GVF norm.*

Figure 3.10: *Different steps in the reconstruction process of the Twins object. From left to right, visual hull initialization, final model, texture mapping and total force after convergence. The reconstructed model has 83241 vertices.*

force sign (red for positive sign, blue for negative sign) visually indicates convergence. Figure 3.11 illustrates the influence of the silhouette force. The support of the object does not provide any texture information and cannot be reconstructed using only the texture force (Fig. 3.11 left). Adding the silhouette constraint solves this problem and guarantees the convergence towards the visual hull in this region (Fig. 3.11 middle). As shown in Fig. 3.11 right, the support is completely driven by the silhouette force.

In Fig. 3.12 we present a reconstruction of the same Twins object as in Fig. 3.10 but using only 12 equally spaced images instead of 36. Correlation is computed using only the 2 nearest cameras ($\pm$ 30 deg) and, although artifacts are visible due to the small number of silhouettes, the algorithm performs quite well computing the texture force, which offers a good 3D reconstruction.

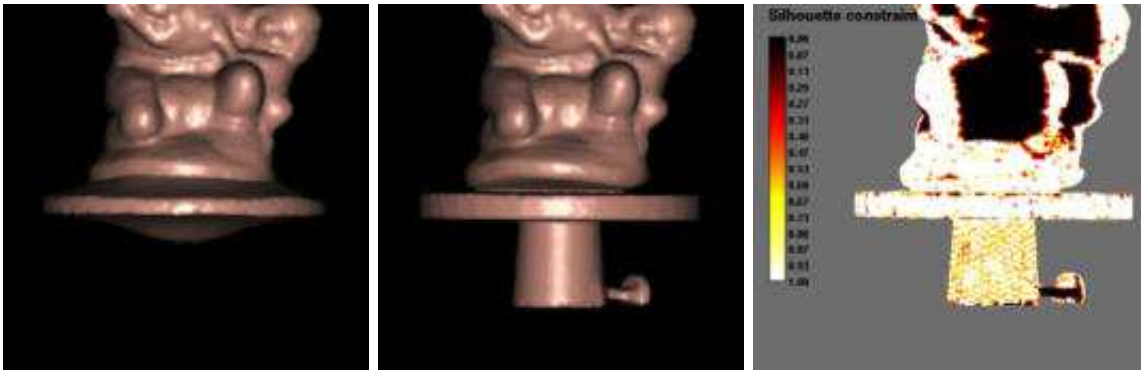


Figure 3.11: *Twins model detail after convergence. Left: evolution under texture force only. Parts of the object with no texture disappear. Middle: evolution under both the texture force and the silhouette force. The parts of the object with no texture follow the silhouette force (visual hull). Right: α component of the silhouette force after convergence.*

Figure 3.12: *Twins model reconstruction using only 12 equally spaced cameras. From left to right: visual hull initialization, final model, texture mapping, and concavity recovery.*

3.2.3 African Sequence

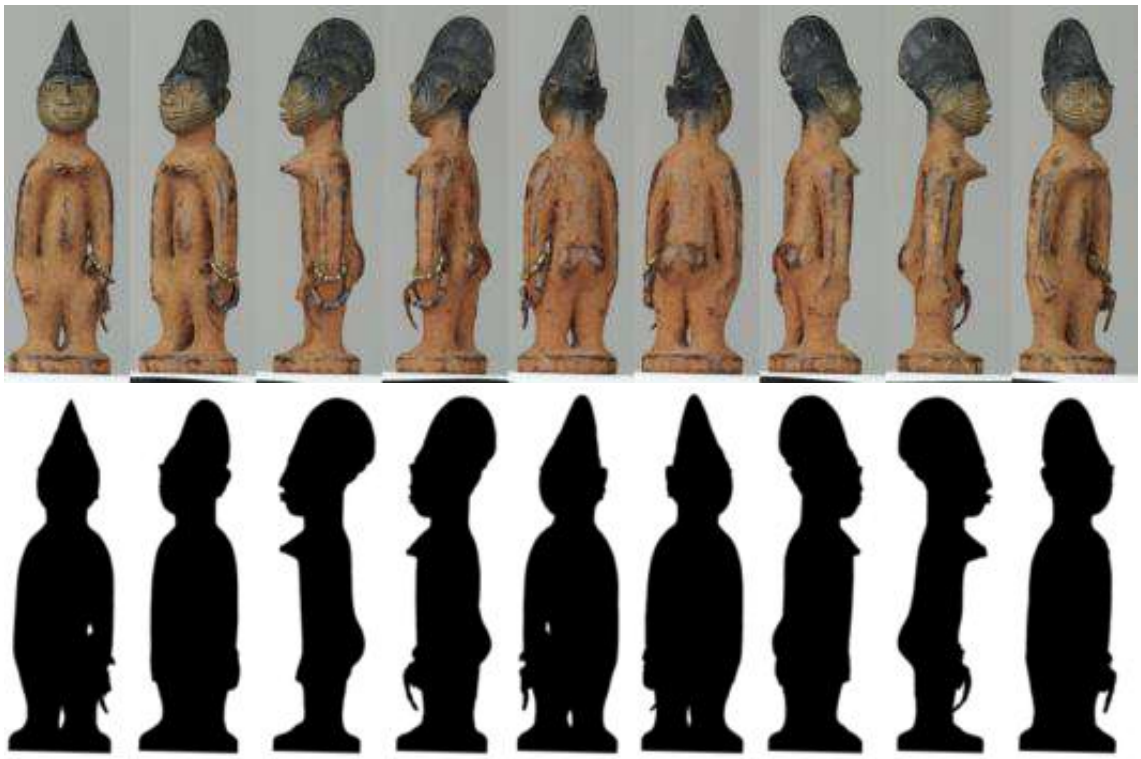


Figure 3.13: *Some of the original images used in the reconstruction of the African object. Top: 9 samples of a total of 36 color images. Bottom: 9 samples of a total of 36 silhouette images.*

A complete reconstruction is presented in Fig. 3.15 using both silhouette and texture information. We are able to recover many details with high accuracy (observe, for instance, the quality of reconstruction of the bracelet and of the rope). In Fig. 3.14 we illustrate the different forces used in the deformable model. Ten octree levels are used in the voting approach (top left), which provides a high precision in the gradient vector flow computation (top middle and top right). At the end of the iterative process, a steady state for the entire mesh is achieved, and concavities are automatically detected (bottom right).

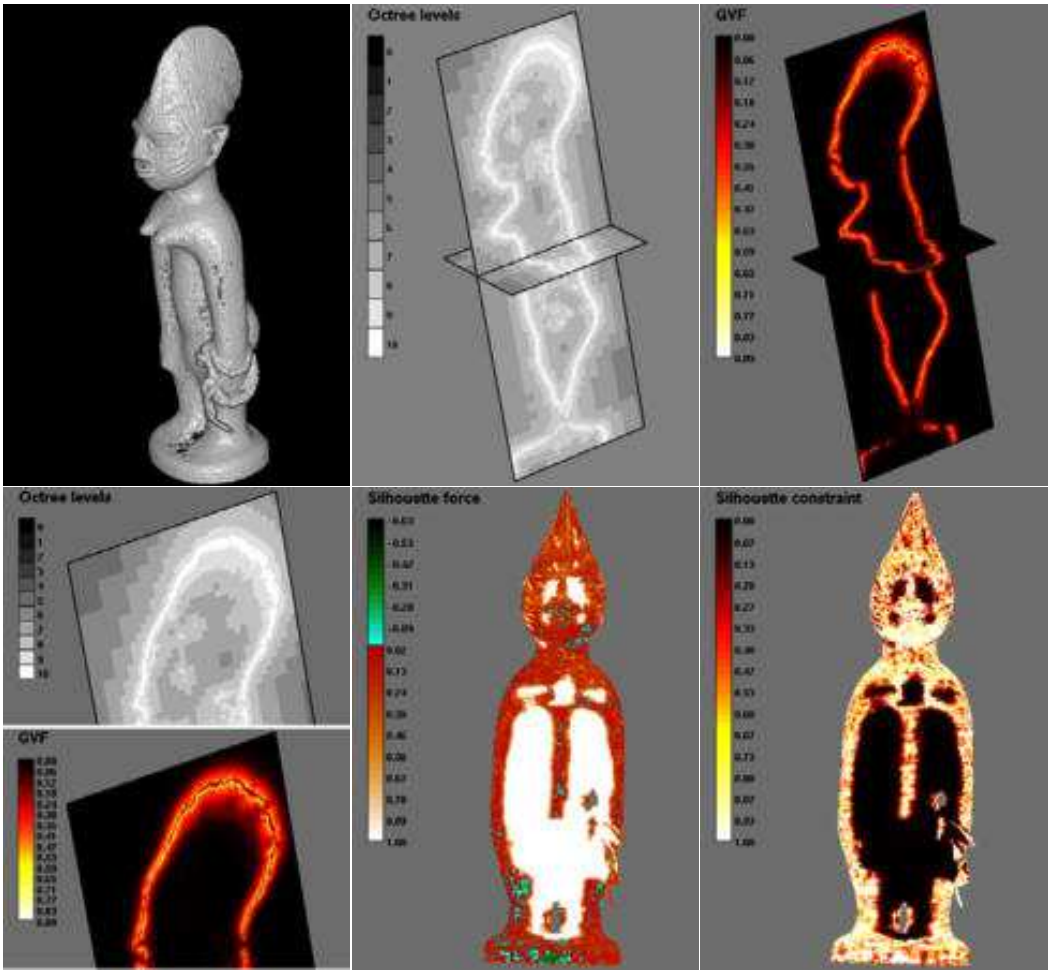


Figure 3.14: *External forces used in the reconstruction of the African model. Top left: volume rendering of the correlation voting volume. Top middle: the octree partition used in the computation of the gradient vector flow field. Top right: norm of the gradient vector flow field. Bottom left: detail of the octree partition and the gradient vector flow volume. Bottom middle:* d_{VH} *silhouette component after convergence. Bottom right:* α *component of the silhouette force after convergence.*

Figure 3.15: *African model after convergence (57639 vertices). Top left: Gouraud shading. Top middle: same view with texture mapping. Top right: lateral view of the textured model. Bottom left: detail of the textured model. Bottom right: same detail in wireframe.*

3.2.4 Tenon Mask Sequence

Figure 3.16: *Some of the original images used in the reconstruction of the Tenon Mask object. Top: 9 samples of a total of 36 color images. Bottom: 9 samples of a total of 36 silhouette images.*

The main challenge of the Tenon object is its topology: it has a total of 9 holes. Six of them are extremely small (around the mask) while the 3 last are medium size (two for the eyes and one for the mouth). The problem is that strong topology artifacts happen due to the visual hull construction as explained in Section 2.3 (see circles in Fig. 3.17 right bottom).

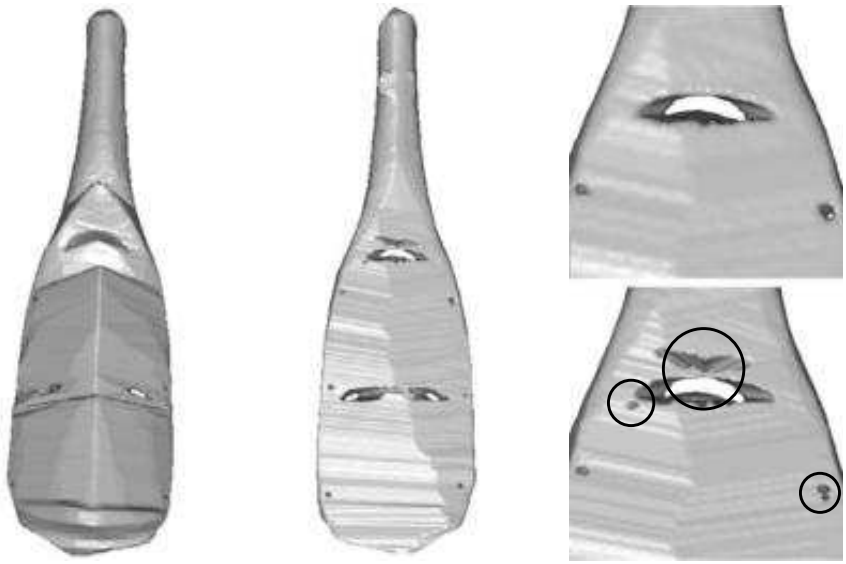


Figure 3.17: *Initial computed visual hull without hole selection. Left: front view. Middle: rear view. Right bottom: detail of topological artifacts present on the rear view. Right top: corrected visual hull.*

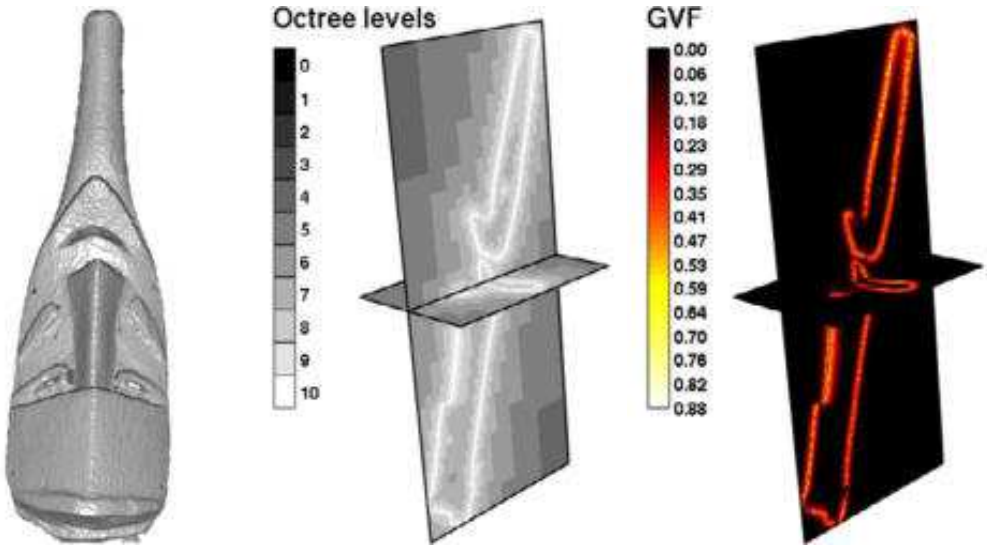


Figure 3.18: *Stereo force computation for the Tenon Mask sequence. From left to right: correlation voting volume after binarization (threshold=30), octree partition and GVF norm.*

These problems can be solved by manually deleting the silhouette holes that produce the "*extra*" 3D holes, which allows recovering a visual hull with the right topology (see Fig. 3.17 right top). Once we have the right topology, we can compute the GVF force (see Fig. 3.18) and iterate the snake until convergence while preserving the original topology. We can appreciate the good results of the final reconstruction in Fig. 3.19. The reconstruction has both the right topology and a high quality geometry.

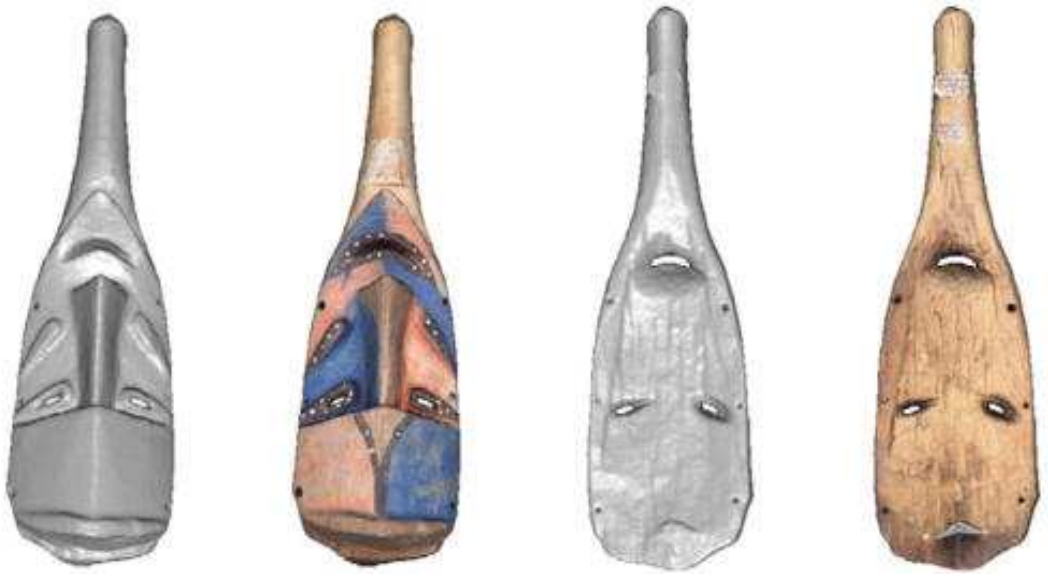


Figure 3.19: *Final Tenon model with 45052 vertices. From left to right: front and rear views with shading and texture.*

3.2.5 Polynesian Sequence

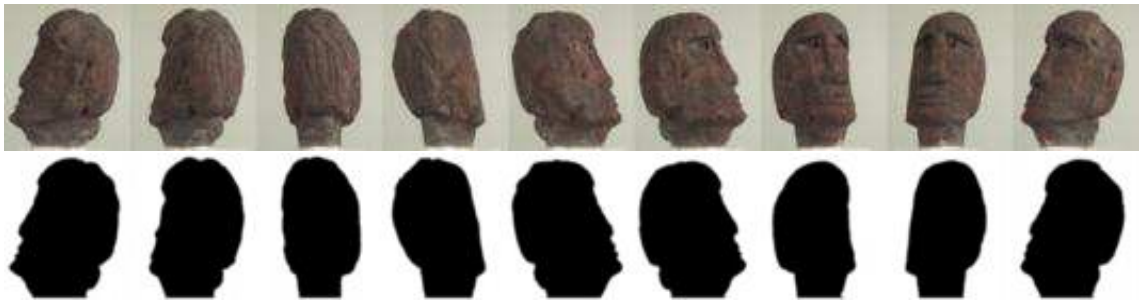


Figure 3.20: *Some of the original images used in the reconstruction of the Polynesian object. Top: 9 samples of a total of 36 color images. Bottom: 9 samples of a total of 36 silhouette images.*

The Polynesian object is a very well textured object (a volcanic stone) that offers us the possibility to compare our reconstruction results with a laser scanner method since the same object was scanned with a Minolta VIVID 910 3D scanner.

We dispose of one sequence of color images, and another sequence of the silhouettes with an active background (a flash with a soft box pointing directly to the camera). However, the original data have a synchronization problem: the silhouettes do not correspond to the color images. This problem is due to the way the images were acquired. A first sequence of color images is taken, then the background is lit on, and another sequence is taken, where the silhouette of the object can be easily extracted. Now, the turntable used is a heavy one that allows very heavy objects on it. This turntable is quite precise whenever the angle steps are always done in the same direction. However, if the turning direction is changed (e.g., to reset the turntable position) there may be a positioning error of up to some degrees. This difference is enough to produce large pixel errors in the image plane (see Fig. 3.21). So, at first glance, the silhouette sequence cannot be exploited. However, if we try to directly extract the silhouettes from the color images, the extracted silhouettes are correct except for the bottom of the object, which has a color similar to the background.

Figure 3.21: *Superposition between the color sequence and the extracted silhouette contour from the silhouette sequence(in blue).*

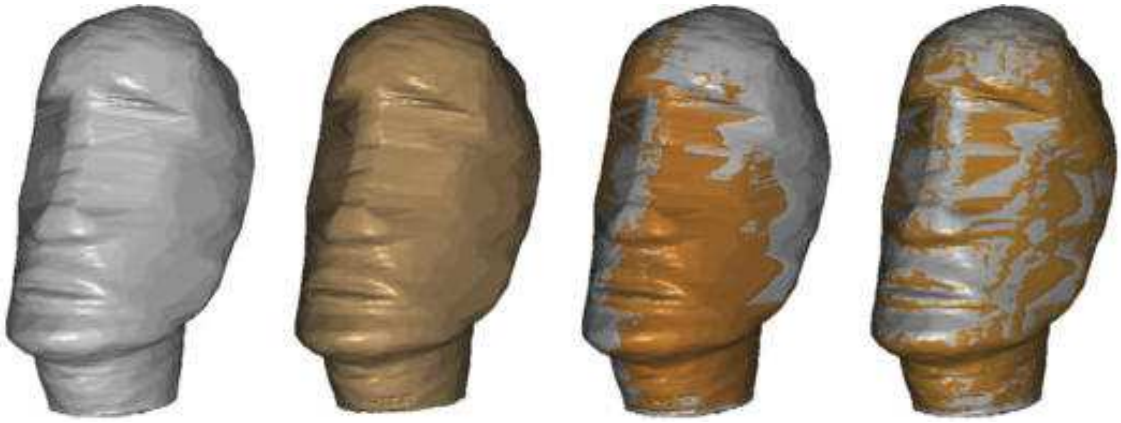


Figure 3.22: *Registration between the visual hull computed with the silhouette sequence and the extracted silhouettes. From left to right: the visual hull computed with the silhouette sequence, the visual hull computed with the extracted silhouettes, the two visual hulls before registration and the two visual hulls after registration.*

In order to exploit the original silhouette sequence, we have registered it with the extracted silhouettes using the silhouette coherence criterion defined in Chapter 1 (see Fig. 3.22). The only parameter to estimate is the angle offset error between the color sequence and the silhouette sequence. After registration, the recovered angle is 1.61 degrees, and the jeopardized pattern of colors in Fig. 3.22 right confirms the success of the registration. An important aspect about the silhouette coherence criterion used in the registration is that, because the bottom of the extracted silhouettes is very noisy and incorrect, the silhouette coherence has been only computed on the medium-top of the silhouettes, i.e., we have only cast optic rays on the top of the silhouettes. This is a very important property compared to the epipolar tangency criterion, since we have the possibility of not using those parts of the silhouettes that we know to be not correct. This happens quite frequently at the bottom of the objects, where the transition between the object and the turntable cannot be easily extracted. As we will see in section 3.4, this property is extremely useful to calibrate some difficult sequences.

In Fig. 3.23 we show the correlation results for the color sequence, and we can appreciate the great resolution of the voxel volume in Fig. 3.23 left.

In Fig. 3.24 we present the comparison between the Minolta scanned object (top left) and the final result after convergence of our method (top middle). As a conclusion, we have a lower resolution due to the stereo limitation and the regularization term. However, the mesh quality is quite good and the main object concavities are well recovered too. In addition, we provide a high quality texture map (top right), which cannot be obtained at that resolution with the used 3D laser scanner.

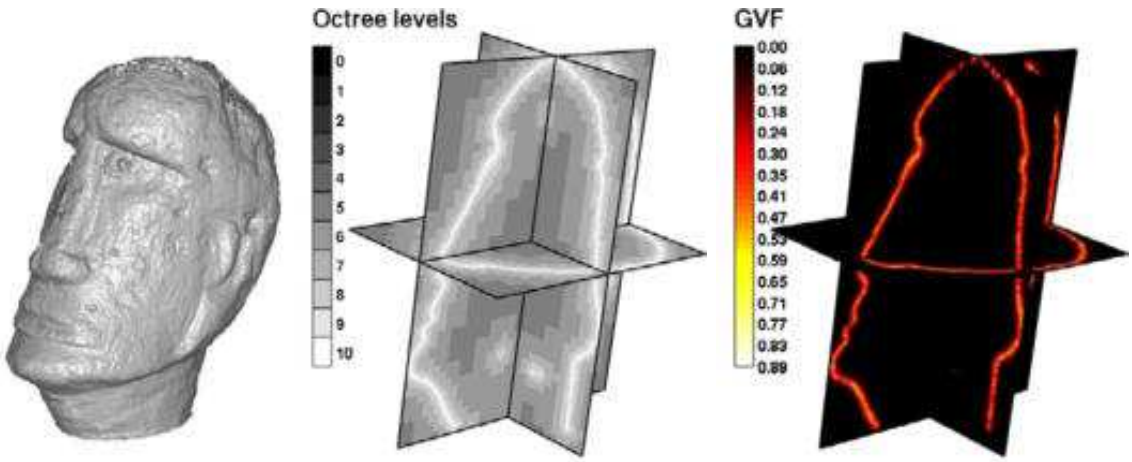


Figure 3.23: *Stereo force computation for the Polynesian sequence. From left to right: correlation voting volume after binarization (threshold=30), octree partition and GVF norm.*

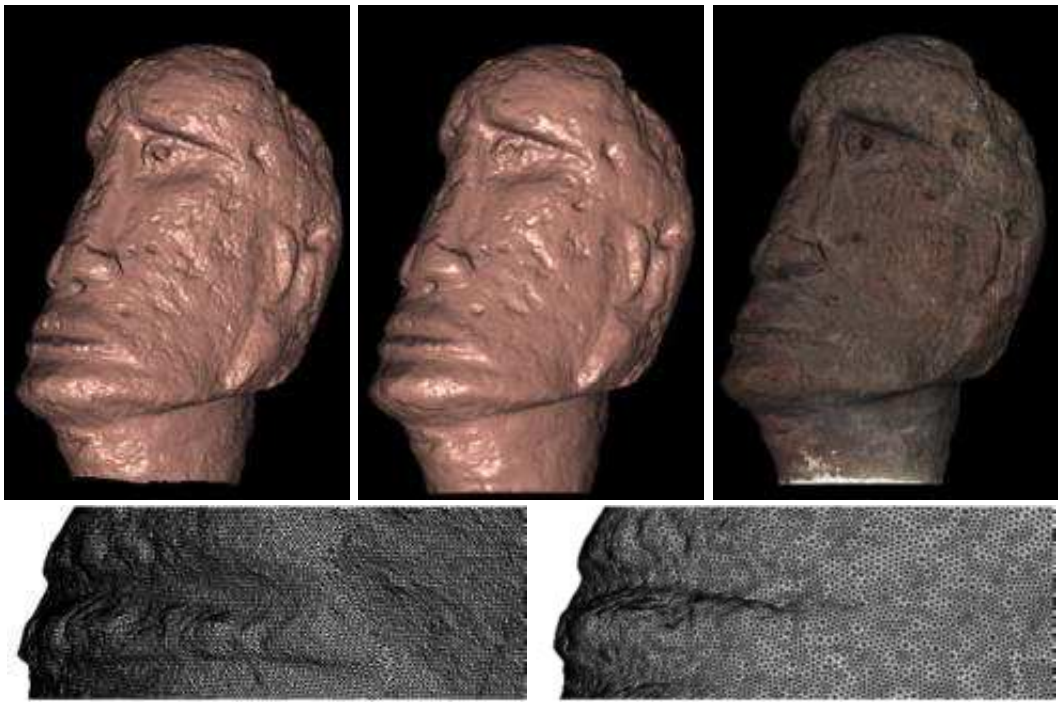


Figure 3.24: *Comparison between the proposed passive method and a laser active method. Top left: laser model of 385355 vertices obtained with a Minolta VIVID 910 3D scanner. Top middle: proposed method after snake convergence (233262 vertices). Top right: textured mesh after convergence. Bottom left: mouth detail of the laser model. Bottom right: same detail of the snake model.*

3.2.6 Guardians Sequence

Figure 3.25: *Some of the original images used in the reconstruction of the Guardians object. Top: 9 samples of a total of 36 color images. Bottom: 9 samples of a total of 36 silhouette images.*

For the Guardians object we dispose of high resolution sequence of 36 images of 4000x4000 pixels. After cropping the region of interest of the object, images are 3000x3800, which is still a very good resolution. We dispose also of the silhouette sequence and the calibration sequence.

This object has a genus of 3, which corresponds to the 3 arms of the statue that create a handle with the statue body. Its particularity is that, even if the computed visual hull has the correct genus, in practice it is very difficult for the deformable model to handle this topology during all the evolution process. In Fig. 3.26 we present the result of the visual hull computation and some detailed views of the generated handles. In Fig. 3.26 right we appreciate the existence of one independent tunnel on the left,

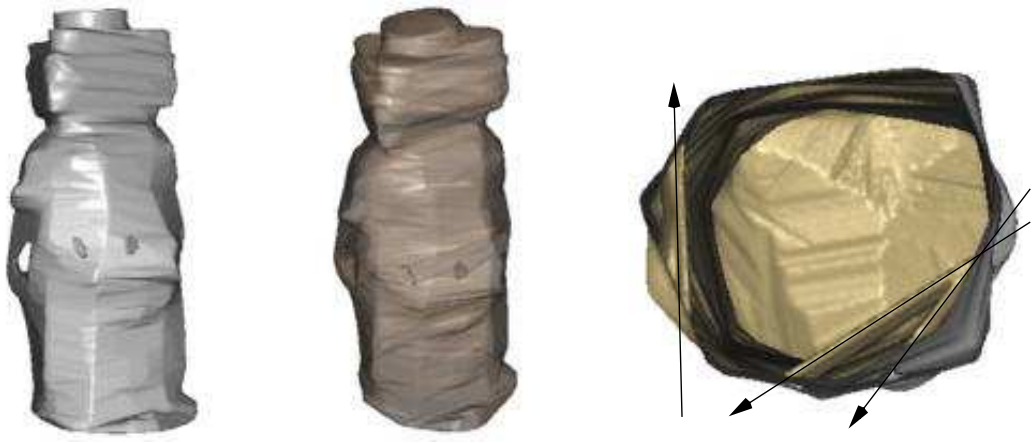


Figure 3.26: *Different views of the Guardians visual hull. From left to right: frontal view, transparent view (we can see the tunnels created by the silhouette holes), and top view of an horizontal cut of the visual hull to better see the tunnels shape.*

Figure 3.27: *Configuration of the deformable model before auto-intersection of the surface.*

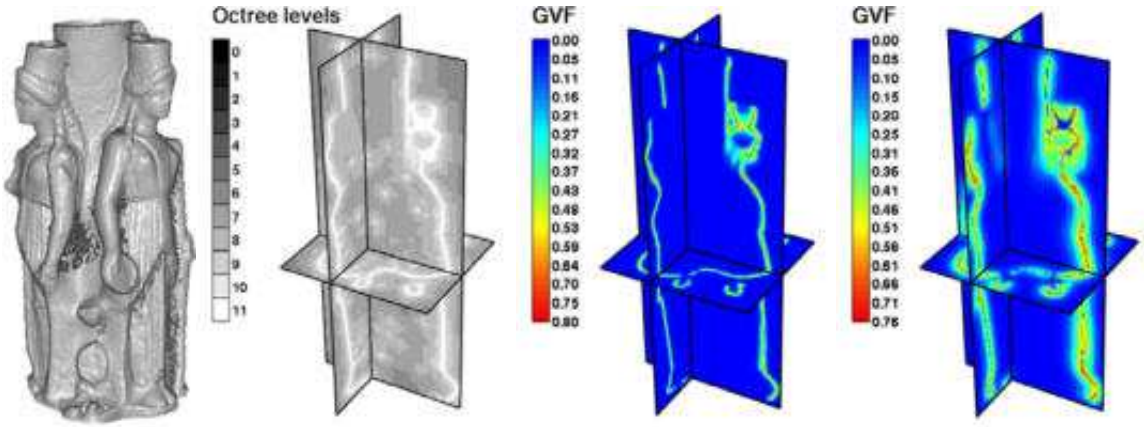


Figure 3.28: *Stereo force computation. From left to right: correlation voting volume after binarization of the accumulated correlation (threshold=20), octree partition of 11 levels, GVF norm for 11 levels and GVF norm for 9 levels.*

and two other tunnels that have one entry in common, i.e., a “T” form with one entry and two exits.

The problem with this topology is that, when the surface evolves, the ”T” junction should move to the left (in Fig. 3.26 right) to allow the surface to traverse that region, since the real surface is behind both tunnels. The problem is that the silhouette will not allow the ”T” junction to slide to the left and the surface deformable model will finish by auto-intersecting with the tunnel (see Fig. 3.27), which is something we do not want to happen.

The only possibility consists of not dealing with holes at this stage. This means that the visual hull has no holes and so will not have the final deformable model. However, after convergence, we can add the holes by embedding the object into an octree, deleting the cubes that are inside the holes and remeshing. By doing this we lose the precision of the deformable model (we have quantized the surface) so we need to reintroduce the new surface (with holes) into the deformable model algorithm for a few more iterations. This method has the advantage of dealing with holes only at the end of the reconstruction algorithm, once the surface is very close to the real object.

For this example we do not need to delete all the holes, but only one of the two tunnels in the ”T” junction.

Figure 3.29: *Different steps in the deformable model evolution. From left to right: low resolution model after convergence (16710 vertices), remesh model with holes (87812 vertices), medium resolution model with holes (69835 vertices), and high resolution model with holes (126195 vertices).*

Next step is the GVF computation. Since images have a very high resolution, the octree used to store the correlation hits has 11 levels of resolution, which is the highest resolution we have handled until now (see Fig. 3.28). To improve convergence, we have first used a low resolution GVF of 9 levels (see Fig. 3.28 right), which gives the low resolution model in Fig. 3.29.a. Then we have increased the resolution of the model and we have done some additional iterations with the 10-level GVF (see the result in Fig. 3.29.b). Finally, we have embedded the medium resolution mesh into an octree, and we have remeshed it using the original silhouettes with holes (see Fig. 3.29.c). The

Figure 3.30: *Four views of the final high resolution textured model.*

Figure 3.31: *Victor Hugo object. From left to right: original image, visual hull without holes and half converged deformable model. Top: front view. Bottom: rear view.*

result has finally been introduced into a high resolution deformable model to obtain the maximum accuracy (see Fig. 3.29.d).

We show in Fig. 3.30 4 views of the resulting high resolution mesh with texture mapping enabled.

3.3 Not So Easy Objects

We present here some objects that push the algorithm to its limits. The limitations are related to the three forces that drive the deformable model: the stereo force, the silhouette force and the internal force. In this section we discuss and illustrate the failure scenario of each force.

Stereo force

The main drawback of the GVF stereo force is its range action. Sometimes we would like a smaller range action and other times we would like it to be stronger.

Concerning the cases where the GVF is too strong, in general it is due to the fact that the GVF is symmetric relative to the voting correlation "surface", but it should *not*. A piece of correlation surface should only attract those regions of the deformable model that lie between that correlation surface, and the cameras that contributed to the correlation surface. This problem is somewhat related to the visibility problem discussed in Section 2.4. However, here we do not discuss if it is more robust to compute

Figure 3.32: *Hippopotamus object. From left to right: one of the original images, correlation volume, model after convergence, and superposition of the deformable mesh with the GVF computed for 10 levels.*

a model-dependent measure (using visibility) or a model-independent measure (such as the proposed correlation voting approach). The problem here is to know, for a given region of the mesh, which part of the correlation surface attracts it. In Fig. 3.31 bottom (see the black ellipsoid) we illustrate a case where the border of the rear mesh has converged to a correlation surface that was not generated from a rear view, but from a front view. The border of the mesh has collapsed into a single skin layer since there was only correlation from the front views, but not from the rear views.

The solution would require storing the set of cameras that voted for a given voxel, and using this information to decide which are the regions of the correlation volume that attract a given part of the deformable model.

On the contrary, sometimes we need a stronger stereo force to traverse more easily the empty regions. With the current GVF implementation, if the surface is very far from the correlation surface, the silhouette force may prevent the deformable model from converging to the correlation surface, such as between the legs in Fig. 3.32 (as shown by the arrow), where the surface cannot converge to the support because the GVF is too weak. For the Hippopotamus object the technique of using a GVF of lower resolution has its limits here. If we reduce the resolution of the GVF, some details such as the eyeteeth will be lost.

This second problem is related to the first problem since, if we know which correlation region is attracting a given vertex, we can accelerate the convergence on the empty regions.

Silhouette force

The silhouette force, as defined in equation 2.29, presumes that for any vertex of the mesh, there is one corresponding silhouette to which it has to obey. If someone else already matches that silhouette, then the vertex is free of going wherever it wants. The "peer" silhouette of a vertex is simply defined as the closest one in pixel distance, i.e., the distance between the projection of the vertex into that silhouette and the silhouette contour is minimized.

The scenario where this force will fail is the following: we select the peer silhouette, we confirm that *that silhouette* is already matched and we decide to free the vertex

Figure 3.33: *Dancer object. From left to right: one of the original images, the corresponding silhouette, superposition of the deformable mesh with one silhouette and superposition of the same mesh with another silhouette.*

Figure 3.34: *Result of applying the silhouette force define in equation 3.1 to the Dancer deformable model. From left to right: the two same silhouettes than in figure 3.33 and a new point of view. The two silhouettes are matched now, however the result is not the desired one, as shown by the red arrow, although silhouettes are matched.*

but, there is another silhouette that is no longer matched because the vertex is no longer constrained by its peer silhouette. This scenario is what has happened in Fig. 3.33. The selected vertex in the left image (shown by the red arrow) does not match the silhouette, but *that* silhouette is *not* its peer silhouette. Its peer silhouette is the silhouette on the most right image in Fig. 3.33, since the distance to its contour is smaller. However, the silhouette on the most right image is already matched by the rest of the mesh, which frees the selected vertex and generates the failure of the silhouette constraint.

A possible solution for this problem is to redefine the silhouette force $\mathcal{F}_{sil}(\mathbf{v})$ to be the maximum of all the possible forces obtained by selecting any silhouette as the peer silhouette. This is equivalent to:

$$||\mathcal{F}_{sil}(\mathbf{v})|| = \max_{c} \alpha_c(\mathbf{v}) d(S_c, P_c\mathbf{v}), \tag{3.1}$$

i.e., we compute the product of the distance to each silhouette with its partial occlusion, and we take the maximum.

If we apply this force to the Dancer (see Fig. 3.34), the results are better and silhouettes are better matched, but the mesh has been eroded even if the silhouettes cannot see it (see Fig. 3.34 right). Another problem of this new silhouette force definition is that it is much more difficult for vertices to detach from silhouettes, which slows the convergence of the deformable model.

This problem is still open, although we think that a better handling of equation 3.1 should solve many of the problems related to the silhouette force.

Internal force

A problem related to the weight of the internal force may appear when modeling surfaces with strong edges as in Fig. 3.31. Although the object suffers also from big topological problems, what is interesting to note is how the model has been eroded in its back (see the red ellipsoid in Fig. 3.31 bottom right). If the width of the surface that we try to model is very thin compared with the edge size, the internal force becomes very strong and the mesh in this area collapses very quickly. Since we perform an additional step of remeshing, the mesh is literally eroded in a hundred of iterations. The proper solution to this problem would be to allow a higher resolution of the mesh in regions of high curvature that *have already converged*. This is important because, if the mesh has not converged, allowing high curvature regions to have more vertices would give a poor final model since the smoothing action is very important to keep a well shaped surface.

3.4 One Color Sequence (No Silhouette nor Calibration Sequence)

3.4.1 Millet Sequence

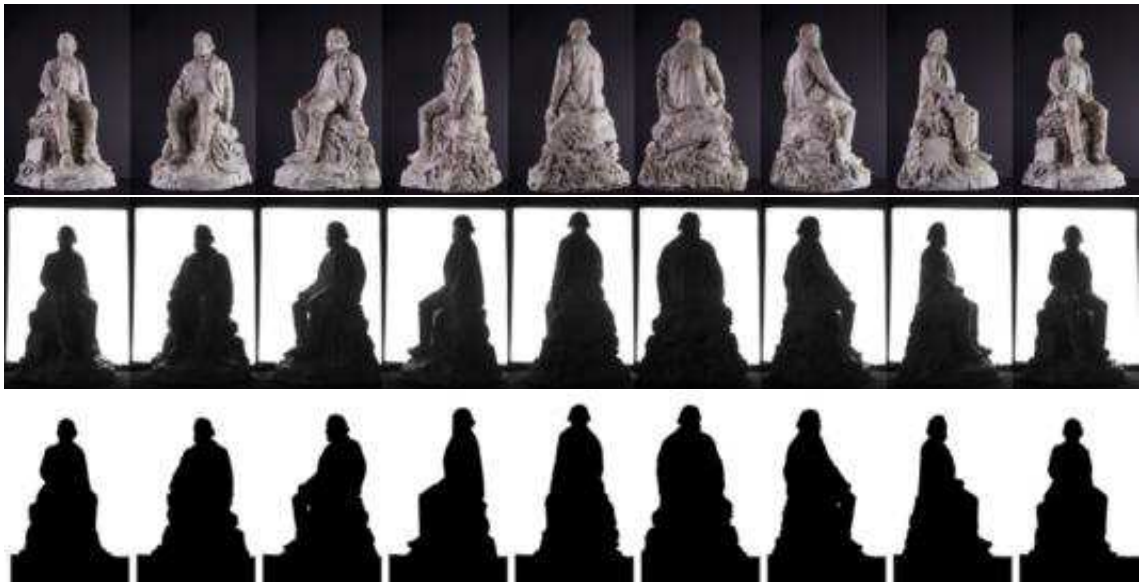


Figure 3.35: *Some of the original images used in the reconstruction of the Millet object. Top: 9 samples of a total of 36 color images. Bottom: 9 samples of a total of 36 silhouette images.*

The Millet object has a very interesting geometry with plenty of details. We dispose of one color sequence and its corresponding silhouette sequence (see Fig. 3.35 middle). However, as we can appreciate in Fig. 3.35 bottom, the acquired silhouettes have a little problem: the silhouette of the bottom of the object is completely lost. Besides, we do not dispose of a calibration sequence so we need to calibrate using silhouettes. The extracted silhouettes pose a problem when computing silhouette coherence since the bottom of the silhouettes is not correct at all. However, we can simply modify the criterion in order to compute the silhouette coherence in the regions of the silhouettes that we know are correct. This is achieved by selecting the sample points, where we test the silhouette coherence, only on the top of the silhouettes, i.e., we can "*clip*" the silhouette coherence computation only on the regions of the silhouettes where we know that it is meaningful to compute the coherence. As a result, we are able to perfectly estimate the rotation axis, the translation and the focal length, with a final coherence score of 99.5%.

Next step is to construct the visual hull. However, as we can see in Fig. 3.36 left, we have two problems with the original visual hull: the bottom of the object and the genus of the visual hull, which is greater than 10! (the original genus of the object is 2, one per leg). To solve the problem of the bottom, we have manually segmented the bottom of 6 silhouettes (spaced of 30 degrees) which helps to carve away much of

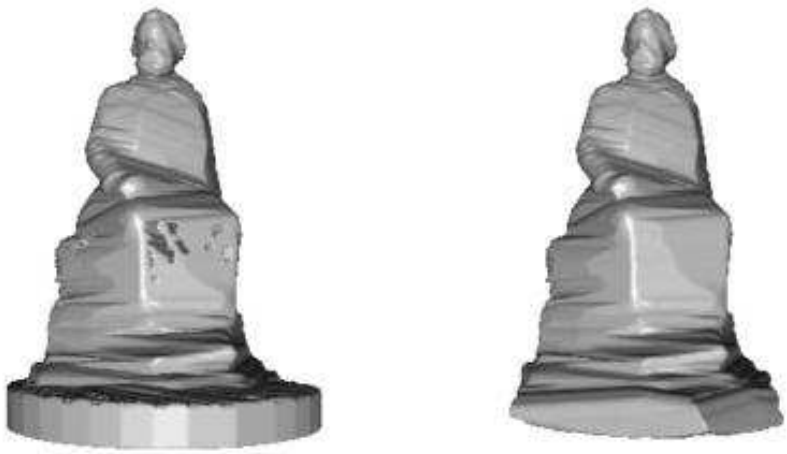


Figure 3.36: *Different computation of the visual hull. Left: visual hull computed with the original silhouettes. Right: visual hull computed with the filled original silhouettes where, for 6 silhouettes of them, the bottom has been extracted manually.*

the wrong space at the bottom of the object. To solve the genus problem, we have just filled all the silhouette holes, so the final visual hull (Fig. 3.36 right) has genus 0. Filling the silhouette holes simplifies the deformable model evolution, while still being able to add the holes *after* convergence.

We present in Fig. 3.37 the result of the correlation voting volume after binarization with a threshold of 30. The recovered surface has very high quality, which confirms the excellent result of the silhouette-based calibration procedure.

Figure 3.37: *Four rendered views of the correlation voting volume for the Millet sequence. The volume has been binarized with an accumulated correlation threshold of 30.*

After computation of the GVF (see Fig. 3.38), we iterate the deformable model with a relative low resolution (see Fig. 3.39.a). After convergence, we increase the resolution to capture smaller details (see Fig. 3.39.b). At this point we would have finished it the topology of the object was correct. However, since we filled all the silhouettes, we still need to "add" the silhouette holes. The reason why it is going to work is that, we look at figure 3.36 left, all the false holes between the legs are caused by a bad surface position of the surface between the legs, which generates all

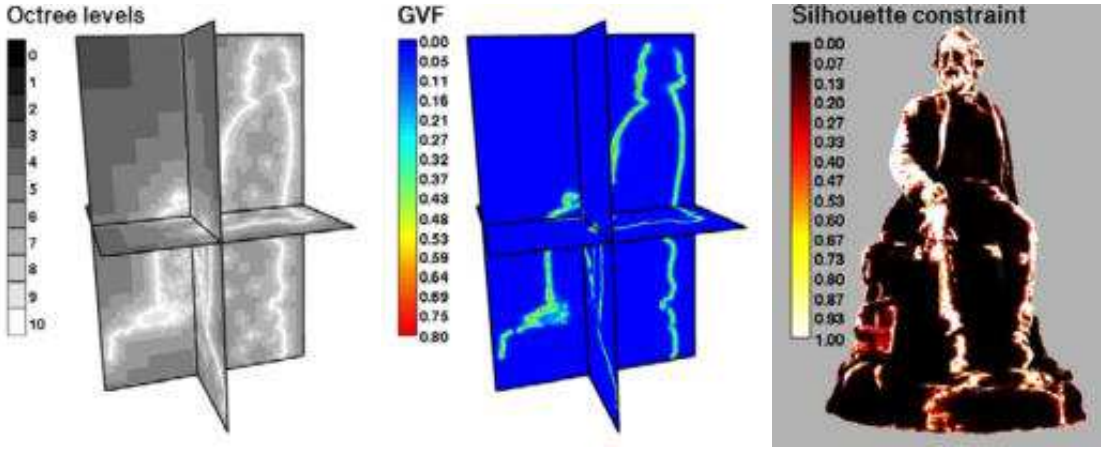


Figure 3.38: *External forces for the Millet sequence. Left: octree partition with 10 levels. Middle: GVF norm. Right: contour generators detected by the silhouette force (α component) after convergence.*

the unnecessary holes. However, after convergence, the surface is correct now, so if we add the silhouette holes, they will generate a correct 3D hole. In practice, the mesh has been embedded into a high resolution octree, which allows us to define a 3D isolevel function where the inside of the object is coded with -1 and the outside with 1. Then we have just computed an iso-surface in the same way as in Section 2.3, where the global isolevel function is the maximum of the 3D isolevel function defined by the mesh, and the silhouette isolevel function of equation 2.7 (see Fig. 3.39.c). After extracting the iso-surface, we reintroduce the mesh into the deformable model step for some additional iterations to recover the precision lost by the octree embedding action (see Fig. 3.39.d).

Finally, we can compute a texture map from the set of color images (see Fig. 3.40). Compared with the original images in Fig. 3.35, the texture map has smoothed the lighting changes in regions where there exist self-shadows (e.g., the legs), although there still persist some lighting artifacts.

Figure 3.39: *Main steps of the evolution of the mesh for the Millet sequence. From left to right: medium resolution model after convergence (74406 vertices), high resolution model after convergence (188680 vertices), remeshed model with holes (395903 vertices) and final high resolution model with holes after convergence (197857 vertices). Top: global view of the Millet object. Bottom: detailed view of the legs.*

Figure 3.40: *Four rendered views of the reconstructed Millet object with texture mapping.*

3.5 Two Color Sequences

3.5.1 Anyi Sequence

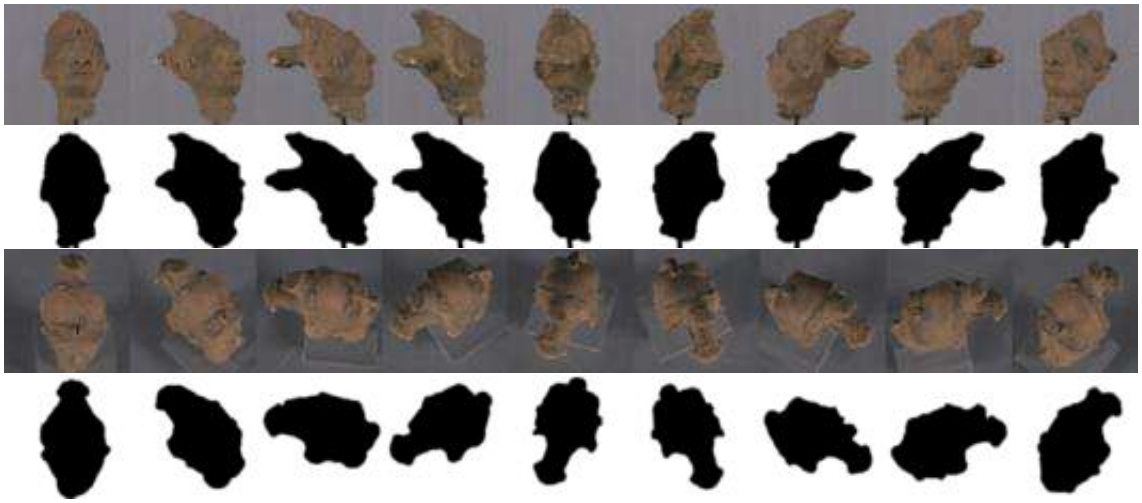
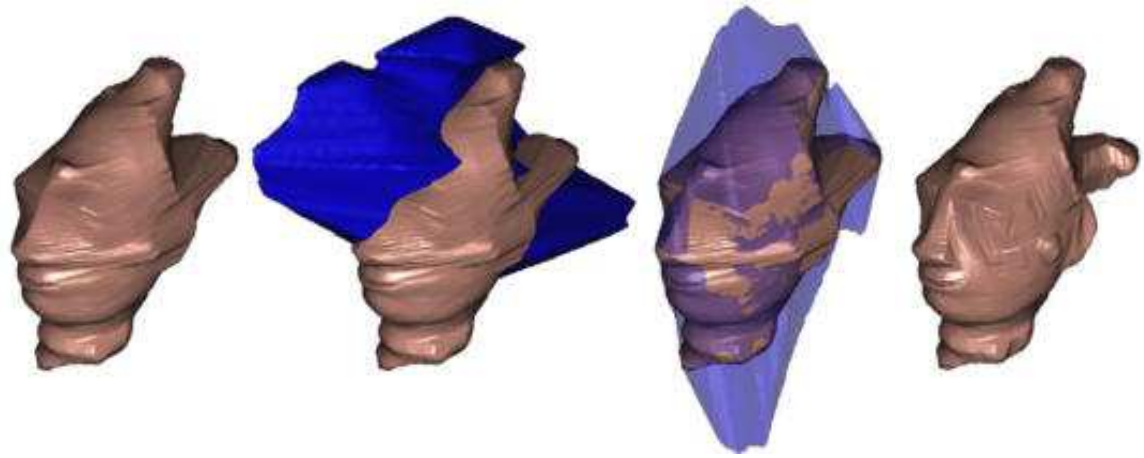


Figure 3.41: *Some of the original images used in the reconstruction of the Anyi object. Although the original images where 2008x3040, the cropped images are only 1200x1200. Top: first color sequence with 9 samples of a total of 72 color images with the corresponding silhouettes. Bottom: second color sequence with 9 samples of a total of 36 silhouette images with the corresponding silhouettes.*

For the Anyi object we dispose of two sequences of color images that have been acquired independently. Although the silhouettes have been extracted automatically,



a) b) c) d)

Figure 3.42: *Statuette (a) First sequence visual hull; silhouette coherence of 94.84%. (b) Unregistered second sequence visual hull; silhouette coherence of 93.73%; mutual coherence of 41.27%. (c) Visual hulls after registration; mutual coherence of 98.18%. (d) Resulting reconstructed visual hull after registration.*

Figure 3.43: *Rectification of two images with a baseline of 20 degrees coming from the two Anyi sequences. Top: two original images from the first sequence (left) and their corresponding rectified ones (right). There is almost no deformation introduced by the rectification since epipolar lines are already almost horizontal. Bottom: two original images from the second sequence and their corresponding rectified ones (right). The rectification greatly modifies the original images due to a rotation axis that is far from the y camera axis (68.7 degrees, to be more precise).*

the recovered silhouette coherences are quite good: 94.84% and 93.73% for the first and second sequence respectively.

Using the mutual silhouette coherence criterion of 1.35 we can register both sequences (see Fig. 3.42). If Fig. 3.42.a and 3.42.b we present the visual hulls before registration, with a mutual coherence of only 41.27%. We can appreciate how different the visual hulls are. After registration, the mutual coherence is of 98.18%. It may seem strange that, after convergence, the mutual coherence is higher than the individual silhouette coherence. This is simply due to the fact that the two measures do not compare the same silhouettes. Once we have registered the two sequences, we can compute the visual hull with the silhouettes of both sequences, which improves the quality of the visual hull (see Fig. 3.42.d).

Next step is to compute the correlation voting octree. In general, two octrees could be computed, one for each color sequence, and then fused into a single octree volume. However, the second color sequence has a particularly bad configuration since the epipolar lines are far from being horizontal. This means that the assumption we made in Chapter 2 of not having to rectify the images does no longer hold. We can corroborate this fact by rectifying two images with a baseline of 20 degrees for the first sequence and for the second sequence using the method of [Fusiello et al., 2000] (see Fig. 3.43). It confirms the fact that the first sequence does not require any rectification since the rotation axis is close to the y axis (1.15 degrees of deviation). However, the second sequence rotation axis is very far from the y axis (68.7 degrees),

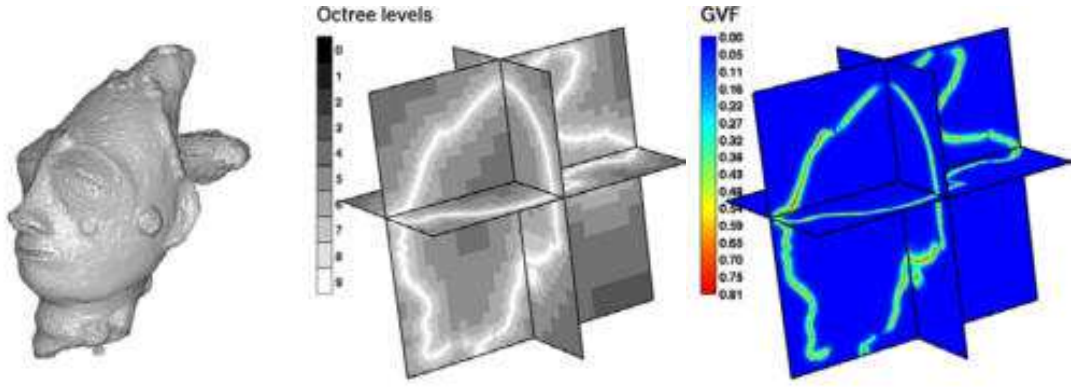
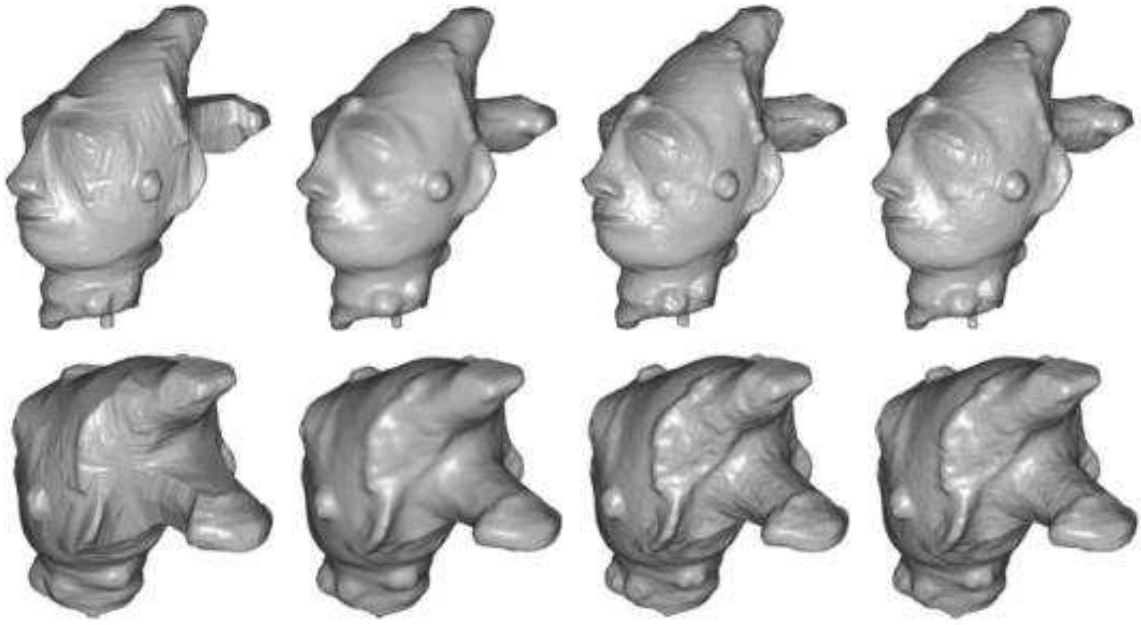


Figure 3.44: *Stereo force computation for the Anyi sequence. From left to right, view of the correlation voting volume after binarization of the accumulated correlation (threshold=15), octree partition and GVF norm.*



98283 vertices 23651 vertices 63420 vertices 217072 vertices

Figure 3.45: *Deformable model evolution while increasing the resolution. From left to right: visual hull, low resolution, medium resolution and high resolution models after convergence. Top: front view. Bottom: rear view.*

which justifies the deformation introduced by the rectification process. Nevertheless, this is not very important in the reconstruction process of the Anyi object since the first sequence already recovers the right object surface. The second sequence will then be useful to compute a more accurate texture map.

In Fig. 3.44 we show the multi-correlation voting volume and the GVF computation for the first Anyi sequence. We can appreciate the fact that the octree has only 9 levels. This is due to the low resolution of the images, where only 1/6 of the total pixel area is effectively used for the object (the rest is background).

Figure 3.46: *Two views of the final textured Anyi model.*

Figure 3.47: *Comparison between using the two color sequences to generate the texture map (left) or using only the first color sequence (right).*

In Fig. 3.45 we show the evolution of the deformable model for different mesh resolutions. We first let the mesh converge with a low resolution and increase it progressively until the desired resolution is obtained. However, we can note that there is too little difference between the medium resolution mesh and the high resolution one, although the latter has 4 times more vertices. This shows the limit of the original images, which is translated into a "low" resolution octree volume.

Despite the low resolution of the images, the final model has many details and allows a good representation of the original object (see Fig. 3.46). In particular, we appreciate in Fig. 3.47 the improvement of computing the texture map with all the available color images (2 sequences) instead of using only the first color sequence (see Fig. 3.47 right).

3.5.2 Toro Sequence

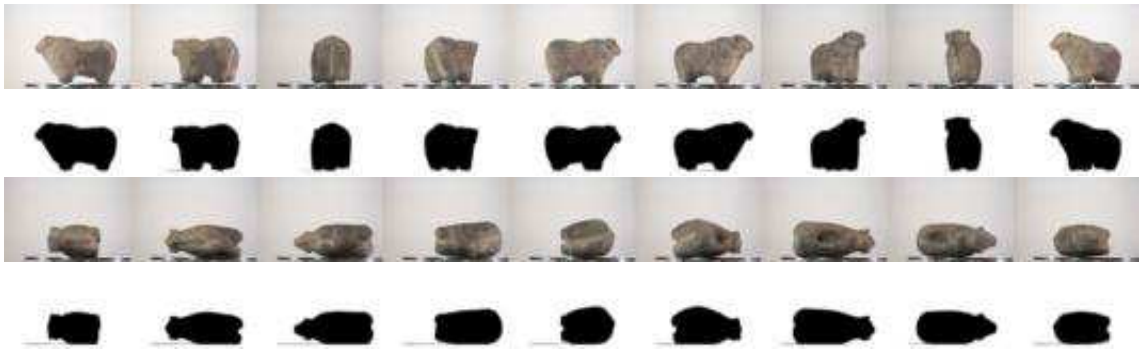


Figure 3.48: *Some of the original images used in the reconstruction of the Toro object. The original images are 4500x3000 pixels. Top: first color sequence with 9 samples of a total of 36 color images with the corresponding silhouettes. Bottom: second color sequence with 9 samples of a total of 36 silhouette images with the corresponding silhouettes.*

For the Toro object we dispose of two sequences of 36 images of 4500x3000 pixels. The object has been manually moved between both sequences, while the camera stays fixed, as seen in Fig. 3.48. This object is special for 3 reasons: we have manually moved the object, we do not dispose of the silhouettes and it has a big concavity on its back, small and deep.

The silhouettes themselves are not very difficult to extract using the color histogram algorithm described in appendix A.4. However, two problems degrade the final quality of the silhouette extraction: blurring and the turntable.

Blurring is a problem for both sequences and its effect is really bad for the silhouette extraction. This can be observed if we compute the gradient of one color image as in Fig.3.49, where we clearly distinguish the imprecision on the contour extraction in the parts of the object that are outside the field of view of the camera.

Figure 3.49: *Strong Blurring distortion due to a small field of view of the Toro sequence. Left: original image. Right: corresponding intensity gradient.*

Figure 3.50: *Separation of the turntable silhouette from the object silhouette by line clipping. Left: original extracted silhouette. Right: clipped silhouette (the clipped region is shown by the red ellipse).*

The second problem is how to separate the object silhouette from the turntable silhouette. The turntable is easy to remove for the first sequence by just clipping all the silhouettes with a horizontal line. The same task is more complicated for the second sequence (see Fig. 3.50) so we need to be a little more conservative to avoid eroding the object silhouette.

As a consequence, although silhouettes are well extracted in an "*image processing*" sense, in practice they are not very accurate. Despite the silhouette inaccuracy, we are able to register both silhouette sequences using the silhouette coherence criterion, but the final mutual coherence score indicates the overall quality of the silhouettes: after registration, the mutual coherence is only of 87%. Nevertheless, the registration quality is enough to reconstruct using both sequences (see Fig. 3.51).

In Fig. 3.52 we show the correlation volume and the GVF computation using only the first color sequence. We appreciate the fact that we are not able to recover the top of the object since the first sequence never sees it. If we compute the second correlation volume and add it to the first one (see Fig. 3.53 top), we obtain a much more complete correlation volume (see Fig. 3.53 top right). In particular, we recover the convavity on the top of the object, which was invisible for the first sequence color sequence.

In order to accelerate the recovery of the concavity on the top the object, we have first iterated the deformable model with a low resolution GVF with only 8 levels (see Fig. 3.54). Upon convergence, we have computed a more refined mesh with the 10-level GVF.

Figure 3.51: *Visual hulls computed with both sequences after registration. From left to right: visual hull of the first sequence, visual hull of the second sequence, superposition of both visual hulls and intersection of both visual hulls.*

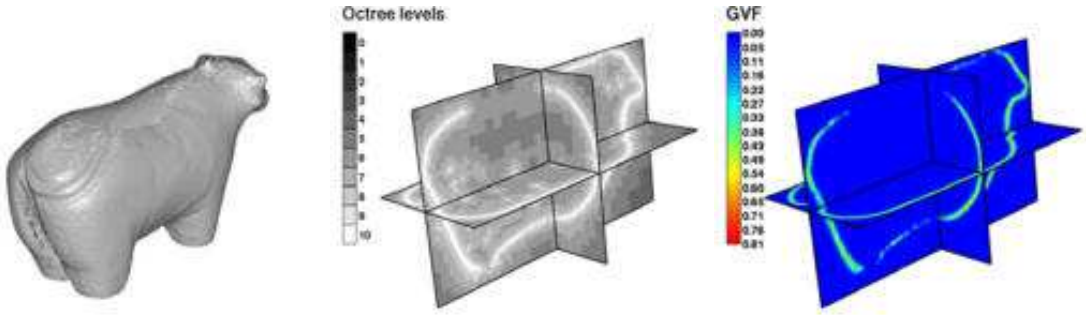


Figure 3.52: *GVF computation for the first Toro sequence. From left to right: rendered view of the correlation volume after binarization of the accumulated correlation (threshold=30), octree partition and GVF norm.*

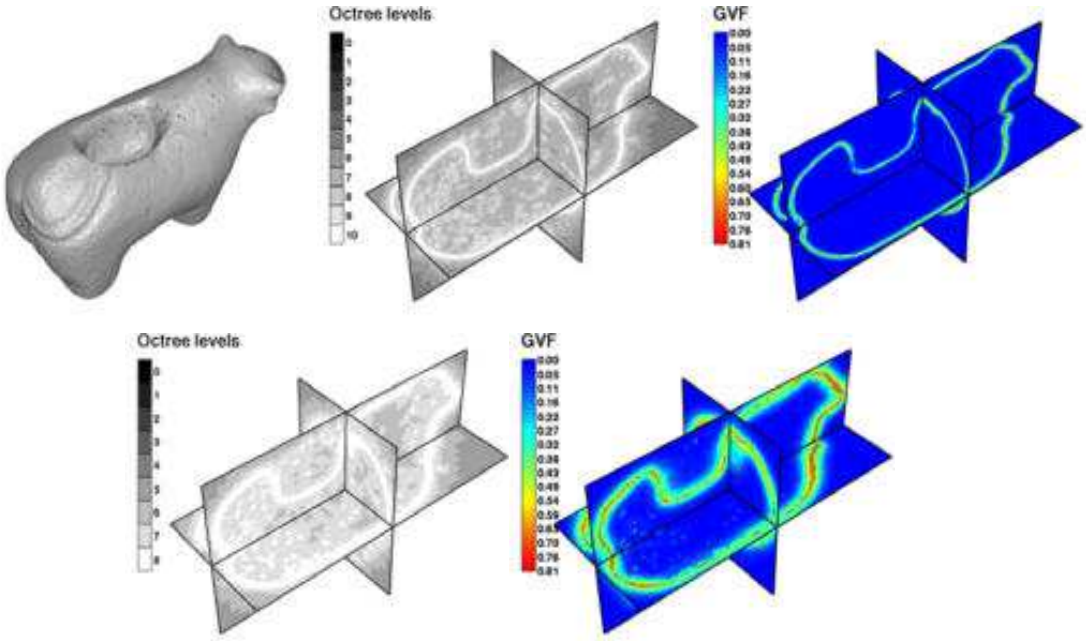


Figure 3.53: *GVF computation using the two Toro sequences. Top: from left to right, rendered view of the mixed correlation volumes after binarization (threshold=30), octree partition and GVF norm. Bottom: octree partition and GVF norm using only 8 levels of depth.*

Finally, we present in Fig. 3.55 two views of the final model with a texture map created using the two color sequences.

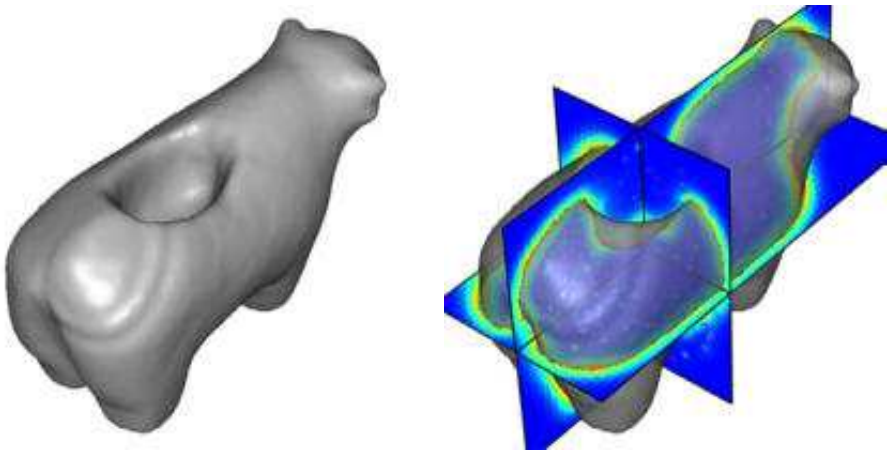


Figure 3.54: *Superposition between the mesh after convergence with a 8-level GVF and 3 slices of the 8-level GVF.*

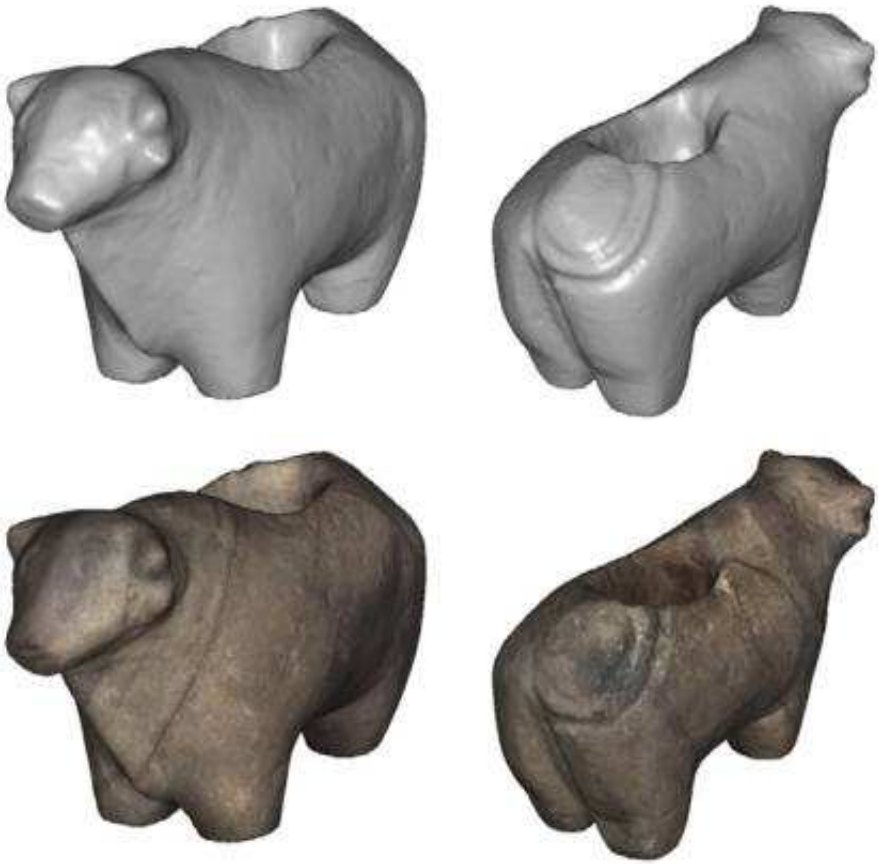


Figure 3.55: *Two views of the final Toro model with 54469 vertices. Top: shaded views. Bottom: same views with texture mapping.*

3.5.3 Buffle Sequence

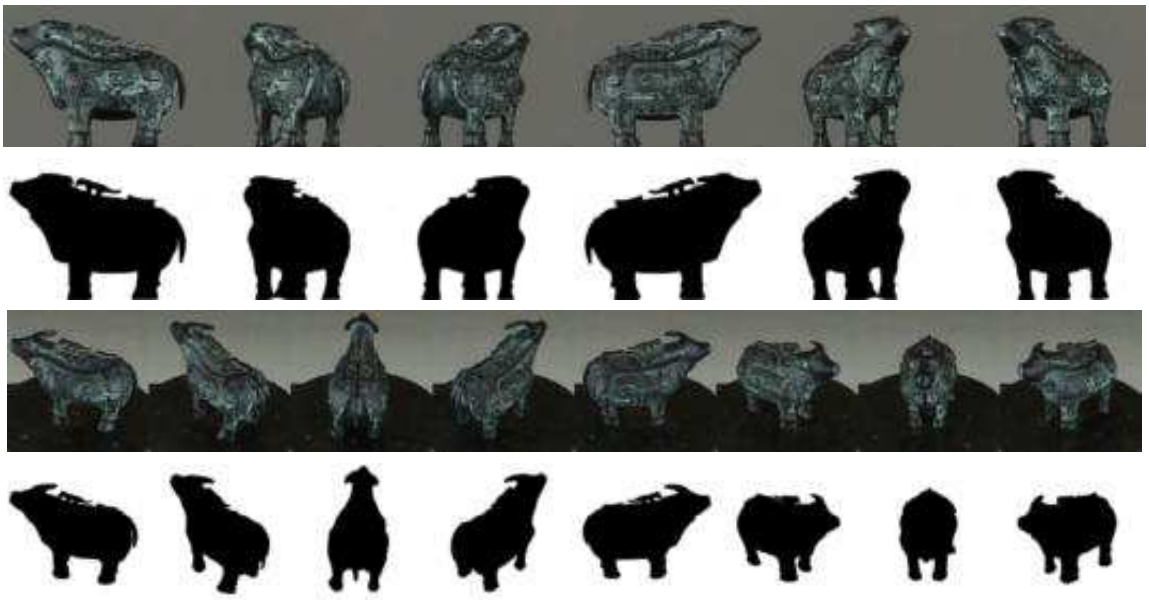


Figure 3.56: *Some of the original images used in the reconstruction of the Buffle object. Although the original images are 3040x2008 pixels, after cropping they are reduced to 1800x1400 for the first sequence and 1600x1600 pixels for the second sequence. Top: first color sequence with 6 samples of a total of 36 color images with the corresponding silhouettes. Bottom: 8 samples of a total of 36 silhouette images with the corresponding silhouettes.*

We dispose of two sequences of 36 images independently calibrated (see Fig. 3.56) and an additional silhouette sequence corresponding to the silhouettes of the first color sequence. Again, we do not know the transformation between the two color sequences. However, the difficulty to register both sequences is the fact that it is really difficult to extract all the silhouettes from the second color sequence. As a result, we have only manually extracted some of them (all the extracted silhouettes from the second sequence are shown in Fig. 3.56 bottom).

Once we have the two sequences of silhouettes, we can register them using the silhouette coherence criterion. In Fig. 3.57 we present the two visual hulls after registration and its intersection, i.e., the visual hull defined by the silhouettes of the first sequence and the second sequence. The mutual coherence after registration is of 95%, which a very good result.

It is worth noting that, once we have registered the two sequences, we can compute the correlation volume inside the visual hull defined by *both* sequences. The more refined the visual hull is, the faster the correlation volume computation will be, since the correlation algorithm will search inside a smaller 3D interval. In Fig. 3.58 we present the correlation volumes of both sequences and the resulting total correlation volume. Although the second sequence rotation axis is far from the y-axis (28.8 degrees of deviation), the correlation algorithm still performs quite well (Fig. 3.58 middle).

The volume of correlation on Fig. 3.58 right can then be used to compute a high

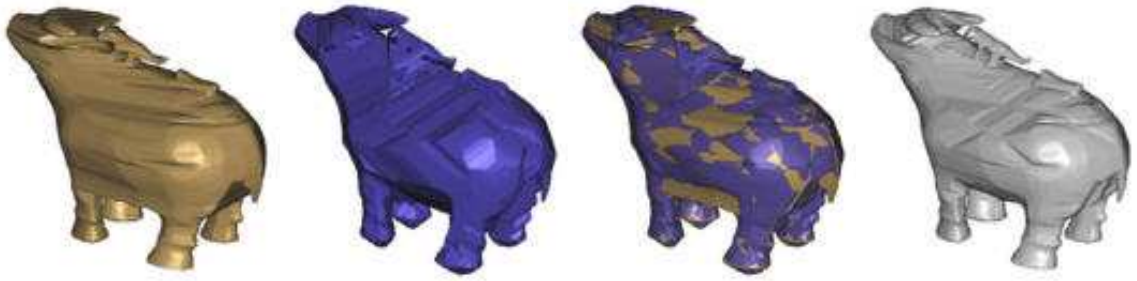


Figure 3.57: *Visual hulls of the two silhouette sequences after registration. The final mutual coherence is of 95%. From left to right: visual hull of the first sequence using 36 silhouettes, visual hull of the second sequence using 8 silhouettes and both visual hulls superposed.*

Figure 3.58: *From left to right, correlation volumes corresponding to the first sequence, the second sequence and the two volumes mixed. All the volumes have been binarized with a threshold of 30.*

resolution GVF (Fig. 3.59). After convergence of the deformable model, we recover the contour generators (α component of $\mathcal{F}_{sil}$) of the model (see Fig. 3.60) as an additional output of the algorithm.

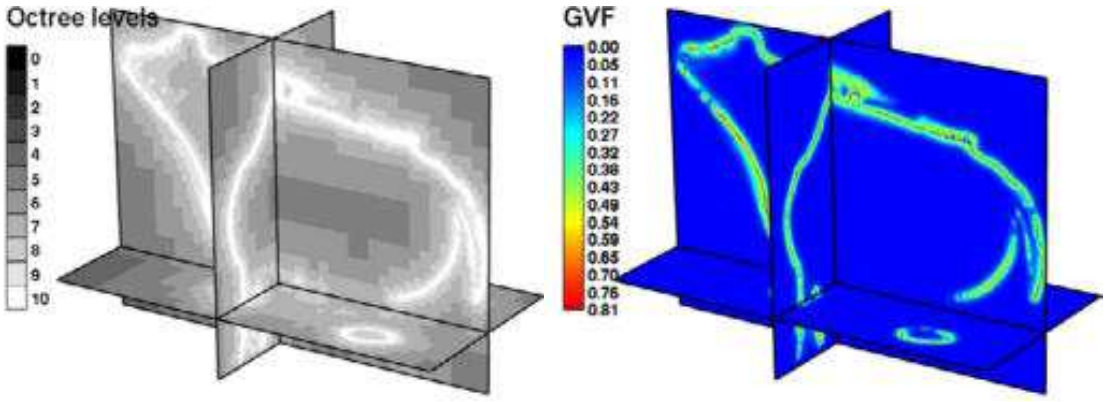


Figure 3.59: *GVF computation for the Buffle model. Left: octree partition. Right: GVF norm.*

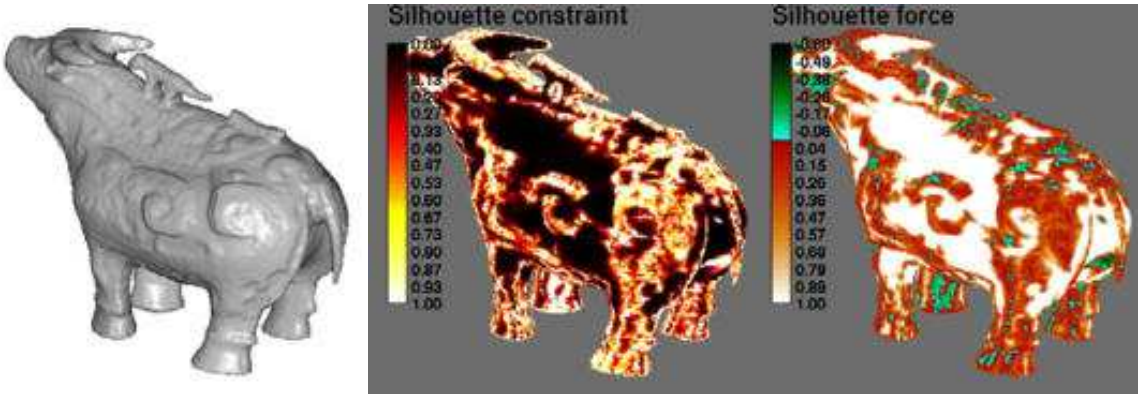


Figure 3.60: *Silhouette force after convergence of the deformable model. Left: mesh after convergence. Middle: detected contour generators (α component). Right: distance to the visual hull (d_{VH} component).*

We can appreciate in Fig. 3.61 the high quality of both the recovered 3D model and the computed texture map.

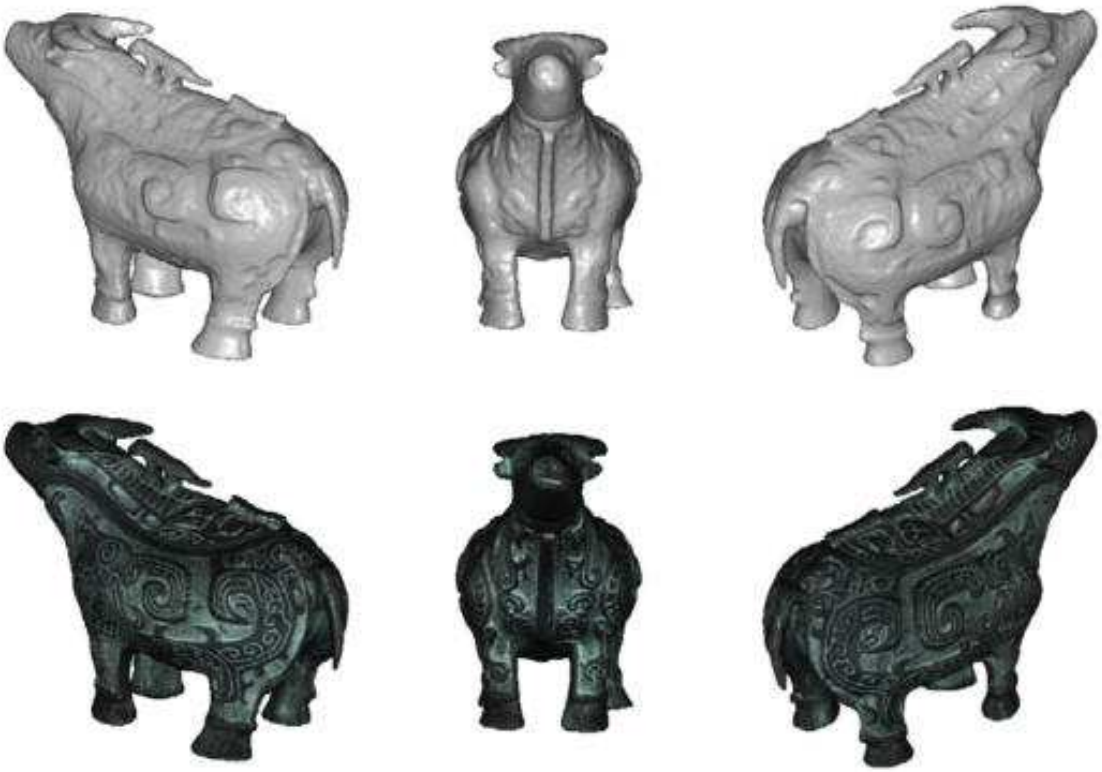


Figure 3.61: *Final Buffle model with 58734 vertices. Top: 3 shaded views. Bottom: the same views with texture mapping.*

3.6 Snapshot of the Best 3D Reconstructions

Figure 3.62: *Some of the reconstructed objects with the proposed technique (1/2).*

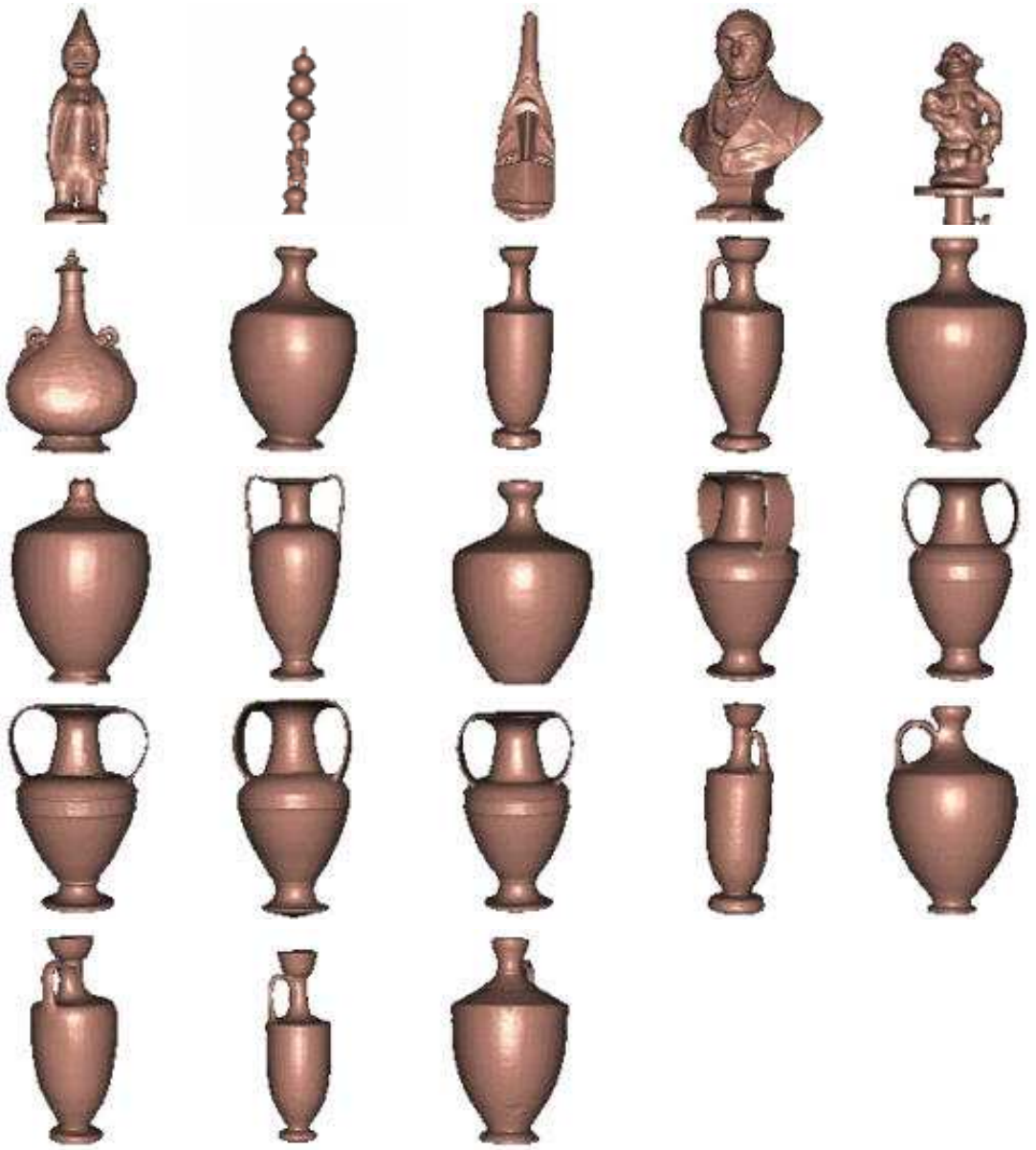


Figure 3.63: *Some of the reconstructed objects with the proposed technique (2/2).*

Conclusions and Perspectives

We have presented a new approach to 3D object modeling from partially calibrated images under circular motion that mixes silhouettes and stereo. We have addressed two main problems in a Computer Vision 3D modeling approach: camera calibration and structure recovery. Camera calibration is accomplished with the definition and implementation of the silhouette coherence criterion, which extends the epipolar tangency criterion when dealing with more than two silhouettes. The 3D modeling algorithm allows us to fuse silhouettes and stereo in a robust way, providing high quality models under real lighting conditions.

The combination of the proposed calibration and reconstruction algorithms provides a complete 3D modeling pipeline that greatly simplifies the modeling process, without the requirement of a calibration pattern. However, this flexibility is obtained at the price of introducing a crucial step into the modeling pipeline: silhouette extraction. Since both the calibration and reconstruction algorithms make an intensive use of silhouettes, their correct extraction is a very important step (and sometimes the bottleneck) of the modeling pipeline. A first general rule to obtain good silhouettes is to use an homogeneous "out of focus" background. Although chromakey techniques can be used, the use of a neutral gray color is preferable to avoid undesirable color shift of the texture. Another possibility is to use a highlighted background to create a Chinese shadow, which requires the acquisition of two sequences: one for the color, and another for the silhouettes.

As a general conclusion, the proposed silhouette-based calibration approach works quite well in practice. Future work should be focused on the definition of a polygon-based similarity measure between the original silhouettes and the visual hull silhouettes. It should improve the convergence properties and the accuracy of the current implementation of the silhouette coherence criterion, which is based on a silhouette contour sampling approach. Il should also enable the criterion to be used in a more general scenario other than circular motion. However, special attention should be paid to the computation time of such a criterion, since it might be too computationally expensive to be used in practice.

Finally, the 3D modeling approach has been extensively tested with more than 100 objects. Short-term improvements should include the visibility handling of the deformable model under the action of the stereo force (the multi-resolution GVF field). In particular, the stereo force should only attract the surfaces that lie between the stereo correlation hits and the cameras that generated them. Furthermore, any optic ray that generated a valid correlation hit should only intersect the final surface in one

point. Also, the deformable model evolution should deal better with convergence and high curvature edges. As a middle-term improvement, the deformable model should be able to recover the correct topology from stereo whenever silhouettes do not recover the right topology. A possible solution could be to detect topology problems and locally update the topology of the mesh. Another possibility would be to launch a local level-set method to allow stereo-driven topology changes. As a long-term work, additional informations could be mixed together with silhouettes and stereo, such as albedo or radiance.

Appendix A

Automatic Silhouette Extraction

A.1 Introduction

If we search for the word *silhouette* in the online Britannica enciclopaedia www.britannica.com, we find the following definition: *an image or design in a single hue and tone, most usually the popular 18th- and 19th-century cut or painted profile portraits done in black on white or the reverse. Silhouette also is any outline or sharp shadow of an object.*

Silhouette-like images can be found among Stone Age cave paintings, ancient Greek vase paintings and Indonesian shadow puppets. But the term and what most people think of as silhouettes originated in the early eighteenth century in Europe. The word was satirically derived from the name of the mid-18th-century French finance minister Étienne de Silhouette, whose hobby was silhouette cutting. The first silhouettes may have been the profiles made of King William and Queen Mary, produced about 1700. The English word for this kind of art piece was "shade". Their popularity was established by 1720, and spread to France and to the United States later in the century. The first silhouettes were painted images, taken from a subject's shadow, and subsequently reduced in size, often with a pantograph. The medium of early silhouettes was lamp black on plaster or glass. Later, silhouettes have been created by cutting a positive shape from black paper from direct observation of a model, at a smaller size, then mounting it on a white ground.

Concerning the scientific application of silhouettes, they are considered as one of the most robust features that can be extracted from an object. Silhouettes are an important source of shape information and are used in many different domains of Computer Vision such as object recognition, object reconstruction or camera calibration.

One of the most difficult steps in Computer Vision is the extraction of information from the original sources: the images. This is also true for all the silhouette-based algorithms: the most difficult step is the silhouette segmentation [1]. Silhouette extraction is obligatory and it is very often short-circuited by imposing heavy constraints to the acquisition problem or by simply doing a manual segmentation. In the context of this work, i.e., the scanning of 3D objects, we need an *almost* automatic way of silhouette segmentation and even if the manual segmentation can be used, it is strongly discouraged: in our usual acquisition 36 high resolution pictures are used.

Figure A.1: *Left: greek vase with silhouette forms. Middle: illustration from the first edition of Lavater's Physiognomy, 1790, showing how where created the first silhouette paintings. Right: John Meirs'* 18^{th} *century silhouette art work.*

This makes the automatic silhouette segmentation something highly desirable. In the case of really complicated objects, we also have the possibility of imposing more acquisition constraints in order to simplify the segmentation step. The extreme case being the use of a Chinese shadow technique, i.e., the use of an illuminated background to produce a natural silhouette. This technique has two drawbacks: i) it cannot be easily applied to very big objects, ii) it requires more photographic work at the acquisition time since two pictures per viewpoint with different lighting setup need to be taken: one for the texture and one for the silhouette. Hence, we are more interested in automatic silhouette segmentation of a single image sequence based on color and/or texture information.

In the following sections we describe 3 different algorithms that we have tested for automatic silhouette extraction based on the color similarity: we suppose that the background has a *constant*[2]color.

A.2 JSEG Algorithm

The JSEG algorithm has been developed by [Deng and Manjunath, 2001] as a robust algorithm for automatic segmentation of color-texture regions in images and video. The main approach of the algorithm is to separate into two different steps the use of color information and spatial information. Color information is first used by strongly quantizing the color space of the original image into a few principal color classes which give a class partition of the image [Deng et al., 1999]. Then the image is replaced by the class-map and a spatial segmentation is performed over the class-map. The

[1]Although there exist very performing chromakey techniques used in TV or movie special effects, they are in general not acceptable for museum objects since they produce strong color artifacts on the object surface due to a diffuse or specular reflection of the background color.

[2]The best results in terms of "color transfer" between the object and the background are obtained with neutral background colors as in the example of Fig. A.1 left.

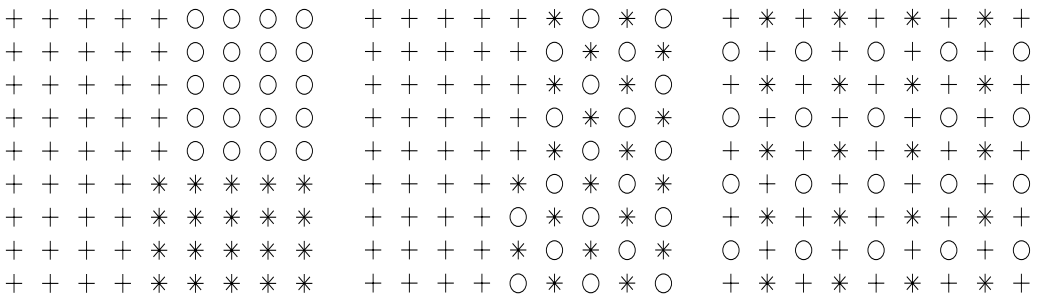


Figure A.2: *An example of different class-maps and their corresponding J values. Images coming from [Deng and Manjunath, 2001].*

spatial segmentation uses a measure of segmentation quality similar to the clustering techniques used in the k-means algorithm (see [Ball and Hall, 1965] or [Diday, 1970]), i.e., a measure that takes into account the ratio between the inter-class distance and the intra-class distance. They propose the quality measure J as follows. Let Z be the set of all N pixel positions $z = (x, y)$ in the class-map. Suppose Z is classified into C classes of N_i data members each, $Z_i,\ i = 1, \ldots, C$. We can define both the class-map mean m and the intra-class mean m_i as,

$$m = \frac{1}{N}\sum_{z\in Z} z, \quad m_i = \frac{1}{N_i}\sum_{z\in Z_i} z.$$

Let S_T be the variance of the entire class-map, S_W the intra-class variance and S_B the inter-class variance:

$$\begin{aligned} S_T &= \sum_{z\in Z} ||z - m||^2, \\ S_W &= \sum_{i=1}^{C} S_i = \sum_{i=1}^{C}\sum_{z\in Z_i} ||z - m_i||^2, \\ S_B &= S_T - S_W. \end{aligned}$$

The measure J is defined as

$$J = S_B/S_W = (S_T - S_W)/S_W.$$

The measure J depends only on the color quantization of the image, since it is the color quantization that determines the class-map distribution. We see in Fig. A.2 three examples of class-maps with three different J values. The J values only depend on the spatial distribution of the $+$,o and $*$ labels.

To take into account a particular segmentation of the class-map, they propose to recalculate J over each segmented region instead of the entire class-map and define the average $\bar{J}$ as

$$\bar{J} = \frac{1}{N}\sum_{k} M_k J_k,$$

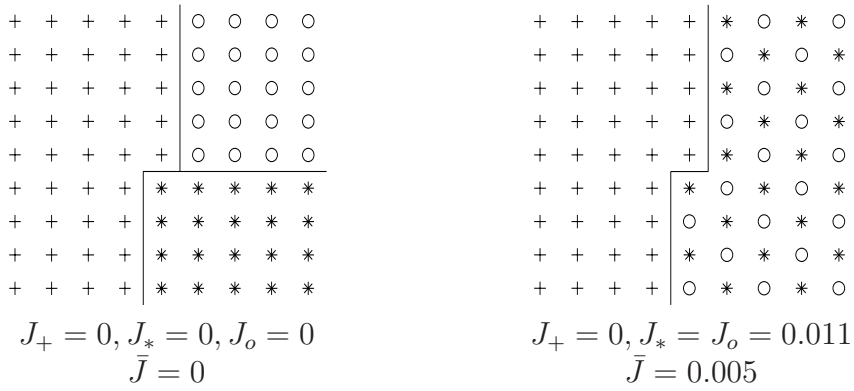


Figure A.3: *Segmented class-maps and their corresponding $\bar{J}$ values. Images coming from [Deng and Manjunath, 2001].*

where J_k is J computed over region k and M_k is the number of points in region k (see Fig. A.3). $\bar{J}$ can be seen as a segmentation criterion where the *best* segmentation is the one that minimizes it. At this point the algorithm suffers from the same problems as the clustering algorithms [Ball and Hall, 1967] or the Mumford and Shah [Mumford and Shah, 1989] formulation: we do not know the number of regions to segment. This implies that this additional information has to be introduced manually. To avoid this problem, the authors propose a different approach to construct the partition minimizing the $\bar{J}$ energy. They consider the local measure of J as a boundary indicator inside the image. For this they introduce the notion of J-image as a gray-scale image whose pixel values are the J values of a local window centered on that pixel. The result is a sort of gradient map where boundaries are represented by high J values and uniform regions are represented by low J values. Based on the J-images they propose an algorithm of region growing where the initial seeds are the centers of low J value regions. To accelerate computations, a multi-scale version of the J-image is used where the class-map is downsampled by half at each new scale level. At the end on the region growing step, an additional region merging step is performed in order to reduce the over segmentation produced by the region growing.

Next we present some practical cases of segmentation using the jseg software available at the jseg home page `vision.ece.ucsb.edu/segmentation/jseg`. In Fig.A.4 we present an original image and two different quantizations of the original image with 8 and 5 colors respectively. In both cases the background is quantized with one color. This means that, except for the dark gray bottom of the left foot that is considered as the background, the image is already well segmented by just choosing the background color class. Following the jseg algorithm, the next step is to compute the J-images of the quantized image using different window sizes, which correspond to the different levels of resolution of the algorithm (see Fig.A.5). The J-images are then used as a height map to detect different regions by region growing. We can see the region growing results corresponding to different levels of resolution in Figure A.6. In Figure A.7 we show the results after region merging. As we can appreciate in the algorithm sequence, the hardest part of the segmentation is achieved by the color quantization

Figure A.4: *From left to right, the original image, quantized image with 8 colors, and quantized image with 5 colors of the Porcinet object.*

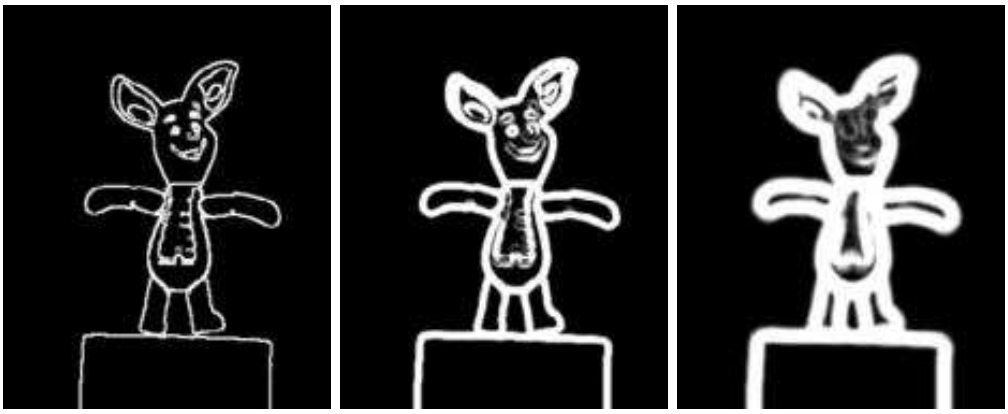


Figure A.5: *From left to right three J-images computed over the 5-class quantized image with a window size of 9, 17 and 33 pixels respectively.*

algorithm. Then, the fact of using multi-resolution $J - images$ certainly allows an overall good segmentation but it also greatly degrades the quantization result produc-

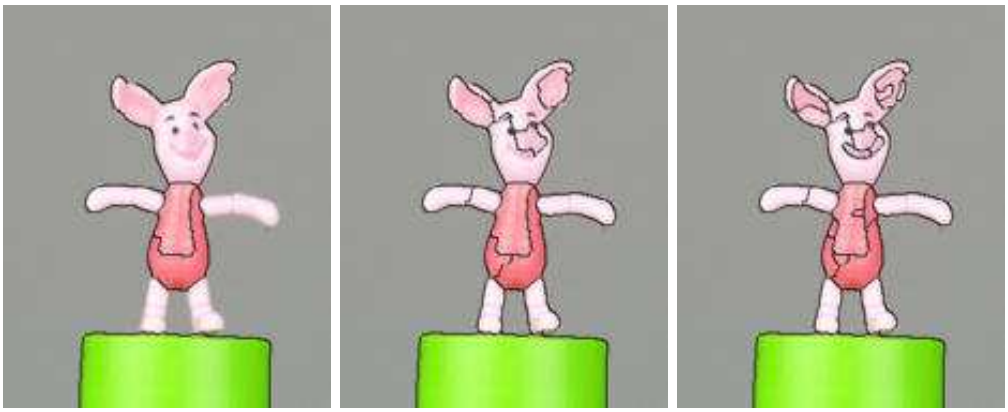


Figure A.6: *Segmentation before region merging using 1, 2 and 3 scale resolutions.*

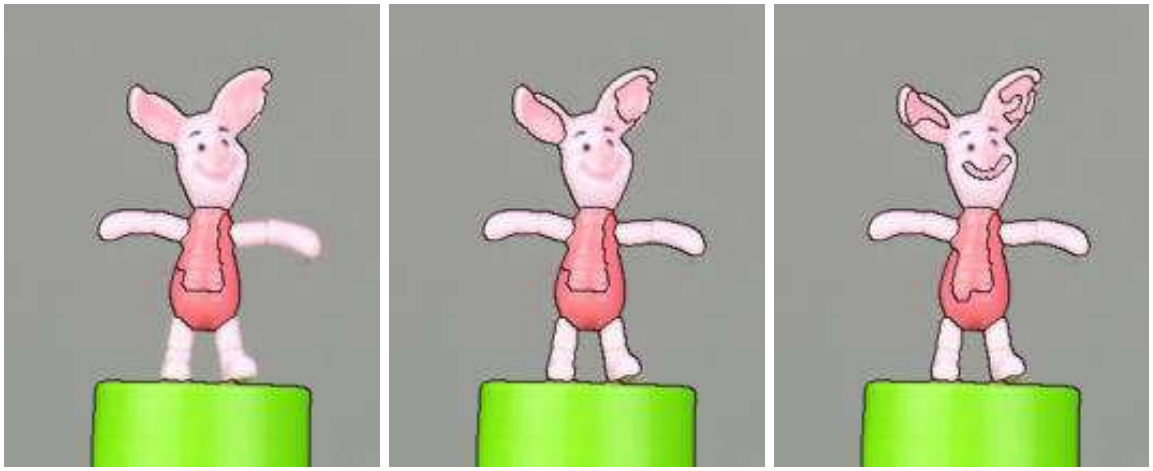


Figure A.7: *Segmentation after region merging for 1, 2 and 3 scale resolutions.*

Figure A.8: *Two additional examples of image segmentation for the Anyi and Coignard statues. We can appreciate both the strengths and the flaws of the algorithm: good color segmentation but poor segmentation precision.*

ing smoothed regions with bad precision. In Figure A.8 we can see two segmentation results obtained on two different statues.

A.3 Level Set Algorithm using the Mumford and Shah Model

A theoretic framework to the problem of image segmentation in computer vision was first developed by [Mumford and Shah, 1989]. Based on this model, there exists abundant literature concerning the existence and uniqueness of a solution to this image segmentation problem. Let $\Omega \in \mathbb{R}^2$ be open and bounded. Let C be a closed subset in Ω made up of a finite set of smooth curves. C divides Ω into a set of connected components Ω_i so that $\Omega = \cup_i \Omega_i \cup C$. A gray-level image u_0 is defined as a function $u_0 : \Omega \rightarrow \mathbb{R}$. The segmentation problem as described by [Mumford and Shah, 1989] is posed as follows: given an observed image u_0, find the *optimal* decomposition C and piecewise smooth approximation u of u_0, so that u varies smoothly within each region Ω_i and discontinuously across the boundaries of Ω_i. To solve this problem, [Mumford and Shah, 1989] propose solving the following energy minimization problem:

$$\inf_{u,C} \left\{ \int_{\Omega} (u - u_0)^2 dxdy + \mu \int_{\cup_i \Omega_i} ||\nabla u||^2 dxdy + \nu |C| \right\}, \tag{A.1}$$

where $\mu > 0$ is the weight controlling the smoothness inside the regions Ω_i and $\nu > 0$ is the weight controlling the length of the curves that define the different regions Ω_i. Among all possible implementations to solve the above minimization problem, one which is particularly interesting is the level set implementation [Osher and Sethian, 1988]. Level sets are a particular category of deformable models. Their most important characteristic is they actually increase the dimension of the problem by one. This provides one of its most powerful properties: the possibility of changing the topology. This is achieved by the use of an implicit representation of the deformable model where the domain is a fixed rectangular grid. In the case of 2D deformable contours, the 2D contour $\vec{\gamma}(s)$ is seen as the zero level set of a scalar function $\phi : \Omega \rightarrow \mathbb{R}$ (see Fig. A.9) so that:

$$\begin{cases} \phi(\mathbf{x}) > 0 & in \quad \omega, \\ \phi(\mathbf{x}) < 0 & in \quad \Omega \backslash \omega, \\ \phi(\mathbf{x}) = 0 & on \quad \partial\omega. \end{cases} \tag{A.2}$$

For a given contour $\vec{\gamma}(s)$, a typical level set function ϕ is given by the signed distance function to the curve.

The level-set evolution equation is a special case of a classic deformable model where the displacement of the model is always done along the normal to the surface. In fact, any classic evolution equation of the form:

$$\frac{\partial \vec{\gamma}(s,t)}{\partial t} = F(\vec{\gamma}(s,t))\vec{n}(s,t), \tag{A.3}$$

F being the amplitude of the force applied to the deformable model, has its corresponding level set equation of the form:

$$\frac{\partial \phi(\mathbf{x},t)}{\partial t} = F(\mathbf{x})||\vec{\nabla}\phi(\mathbf{x},t)||, \tag{A.4}$$

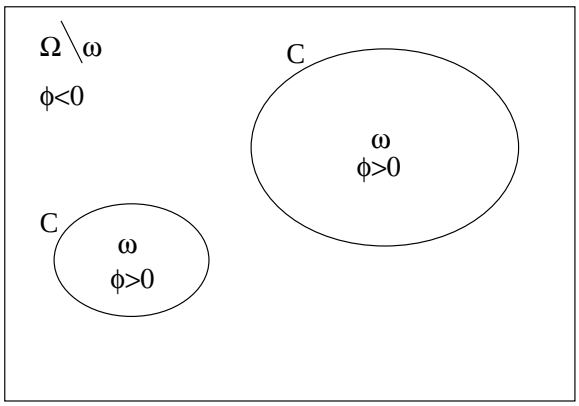


Figure A.9: *Two curves given by the zero level of the function ϕ. The sign of the function ϕ partitions the domain into two regions $\{(x,y) : \phi(x,y) < 0\}$ and $\{(x,y) : \phi(x,y) > 0\}$.*

so that solving equation A.4 is equivalent to solving A.3. The derivation of this equivalence is obtained if we take into account the two constraints that link $\phi(\mathbf{x},t)$ and $\vec{\gamma}(s,t)$:

- The curve $\vec{\gamma}(s,t)$ must be a level set (iso-level contour) of $\phi(\mathbf{x},t)$, meaning that the value of $\phi(\mathbf{x},t)$ must be constant along $\vec{\gamma}$. This condition can be posed as:

$$\left.\frac{\partial\phi(\mathbf{x},t)}{\partial s}\right|_{\mathbf{x}=\vec{\gamma}(s,t)} = \frac{\partial\phi(\mathbf{x},t)}{\partial\mathbf{x}}\cdot\frac{\partial\vec{\gamma}(s,t)}{\partial s} = \vec{\nabla}\phi(\mathbf{x},t)\cdot\frac{\partial\vec{\gamma}(s,t)}{\partial s} = 0, \tag{A.5}$$

  which implies $\vec{\nabla}\phi(\mathbf{x},t)\cdot\vec{n}(s,t) = 0$.

- The level set value corresponding to $\vec{\gamma}(s,t)$ cannot change during the evolution of $\phi(\mathbf{x},t)$, it must stay fixed:

$$\left.\frac{\partial\phi(\mathbf{x},t)}{\partial t}\right|_{\mathbf{x}=\vec{\gamma}(s,t)} = \frac{\partial\phi(\mathbf{x},t)}{\partial\mathbf{x}}\cdot\frac{\partial\vec{\gamma}(s,t)}{\partial t} + \frac{\partial\phi(\mathbf{x},t)}{\partial t} = 0. \tag{A.6}$$

From equation A.5 and the convention of the level set sign (see Fig. A.9), i.e., the region bounded by $\vec{\gamma}(s,t)$ corresponds to $\phi > 0$, the normal $\vec{n}(s,t)$ to the zero level set function $\vec{\gamma}(s,t)$ can be defined as:

$$\vec{n}(s,t) = -\left.\frac{\vec{\nabla}\phi(\mathbf{x},t)}{||\vec{\nabla}\phi(\mathbf{x},t)||}\right|_{\mathbf{x}=\vec{\gamma}(s,t)}. \tag{A.7}$$

From equation A.6 and equation A.7, we get:

$$\frac{\partial\phi(\mathbf{x},t)}{\partial t} = -\vec{\nabla}\phi(\mathbf{x},t)\cdot\frac{\partial\vec{\gamma}(s,t)}{\partial t} = ||\vec{\nabla}\phi(\mathbf{x},t)||\vec{n}(s,t)\cdot\frac{\partial\vec{\gamma}(s,t)}{\partial t},$$

which together with equation A.3 gives:

$$\frac{\partial\phi(\mathbf{x},t)}{\partial t} = ||\vec{\nabla}\phi(\mathbf{x},t)||\vec{n}(s,t)\cdot\vec{n}(s,t)F(\mathbf{x}) = ||\vec{\nabla}\phi(\mathbf{x},t)||F(\mathbf{x}).$$

This equivalence between classic deformable models and level sets requires $F(\mathbf{x})$ to be defined not only on $\vec{\gamma}(s)$ but on all the domain Ω. Sometimes this *extension* to the entire domain can naturally be done while the rest of the time the extension has to be *synthesized*. In the present case of image segmentation we will see that we do not need to synthesize $F(\mathbf{x})$ since we can find a formulation were $F(\mathbf{x})$ is naturally defined over the entire domain.

The composition of $F(\mathbf{x})$ for a level set evolution equation is the same as for a classic deformable model: a data term F_d and a smoothing term F_s:

$$\frac{\partial \phi}{\partial t} = ||\vec{\nabla}\phi||(F_s + F_d).$$

The data term is given by the particular problem that we try to solve. The smoothing term is specific to the level set form and is usually composed by a curvature term:

$$F_s = \nu \text{div}\left(\frac{\vec{\nabla}\phi}{||\vec{\nabla}\phi||}\right),$$

where ν is a weight that controls the degree of smoothing. In the case of a 2D level set the curvature is the following:

$$\text{div}\left(\frac{\vec{\nabla}\phi}{||\vec{\nabla}\phi||}\right) = \text{div}\left(\frac{[\phi_x \ \phi_y]}{(\phi_x^2 + \phi_y^2)^{1/2}}\right) = \frac{\phi_{xx}\phi_y^2 - 2\phi_x\phi_y\phi_{xy} + \phi_{yy}\phi_x^2}{(\phi_x^2 + \phi_y^2)^{3/2}}.$$

For the two-class binarisation problem, i.e., one background and one foreground colors, [Chan and Vese, 2001] propose the following data speed F_d:

$$F_d = (u_0 - c_0)^2 - (u_0 - c_1)^2, \tag{A.8}$$

where c_0 and c_1 are the background and foreground mean colors, respectively, computed as:

$$c_0 = \frac{\int_{\phi<0} u_0 dxdy}{\int_{\phi<0} dxdy}, \ c_1 = \frac{\int_{\phi>0} u_0 dxdy}{\int_{\phi>0} dxdy}.$$

They show that this speed corresponds to the minimization of the following energy (not proven here):

$$E(c_0, c_1, \phi) = \int_{\phi<0} (u_0 - c_0)^2 dxdy + \int_{\phi>0} (u_0 - c_1)^2 dxdy, \tag{A.9}$$

i.e., it searches to partition the image into two constant colors, minimizing the mean square error to decide if a pixel belongs to one class or to another. As noted by [Chan and Vese, 2001], this model has several advantages compared to other active contour models: detection of interior contours, robust initialization, and detection of cognitive contours (without gradient). Equation A.9 can be seen as a specialization of the region-based energy defined by [Cohen et al., 1993, Cohen, 1997]. The main difference is that, in [Cohen et al., 1993], the image is considered to have only one

homogeneous region, while the rest of the image is *not* homogeneous. Equation A.9 is even more specialized and considers that the image can be partitioned into two different homogeneous regions.

In [Vese and Chan, 2002] they extend the original idea to n-phase segmentation, n being the number of desired regions of the partitioned image. The main difference with other multi phase level set segmentation methods is the way they use the level sets to code the different regions. All the previous methods such as [Zhao et al., 1996], [Samson et al., 2000] or [Paragios and Deriche, 2000] use one level set per phase, which poses some problems of overlap and vacuum since, to define an image partition, the intersection of all the regions must be empty and the union must be the entire domain. These problems are solved by imposing additional constraints in order to guarantee the partition properties. [Vese and Chan, 2002] proceed in a different way and use all the possible regions created by the combination of all the available level sets. If we use m level sets, we have up to 2^m regions, where each region is *coded* with a 2-base m-tuple $(b_0, \cdots, b_{m-1})$, with $b_{j=0,\cdots,m-1} \in \{0, 1\}$. Each bit b_j represents a level set and its value (0 or 1) corresponds to the region defined by $\phi_j < 0$ or $\phi_j > 0$. A given region $\omega_{i=0,\cdots,2^m-1}$ is then defined as the intersection of all the corresponding level set regions:

$$\omega_i = \bigcap_{j=0}^{m-1} \begin{cases} \phi_j < 0 & \text{if} \quad b_j = 0 \\ \phi_j > 0 & \text{if} \quad b_j = 1 \end{cases}, \quad i = \sum_{j=0}^{m-1} b_j \cdot 2^j.$$

Coding the regions in this way allows [Vese and Chan, 2002] to use only $m = \log_2(n)$ level sets to partition an image into n different regions. The following step is to construct a valid set of speeds $\{F_d^j, j = 0, \cdots, m-1\}$ that minimize the desired energy. For a n-phase segmentation problem, the energy to minimize is simply the extension of equation (A.9):

$$E(c_0, \cdots, c_{n-1}, \phi_0, \cdots, \phi_{m-1}) = \sum_{i=0}^{n-1} \int \chi_i (u_0 - c_i)^2 dxdy, \tag{A.10}$$

where χ_i is the characteristic function of the region ω_i. To find the speed set F_d^j, [Vese and Chan, 2002] propose to simply minimize equation (A.10), which gives:

$$F_d^j = \sum_{i=0}^{n-1} \chi_i \left((u_0 - c_{A(i,j)})^2 - (u_0 - c_{B(i,j)})^2 \right), \tag{A.11}$$

where the indices $A(i, j)$ and $B(i, j)$ are computed as:

$$\begin{aligned} A(i,j) &= \sum_{k=0, k\neq j}^{m-1} b_k \cdot 2^k + 0 \cdot 2^j, \\ B(i,j) &= \sum_{k=0, k\neq j}^{m-1} b_k \cdot 2^k + 1 \cdot 2^j, \end{aligned}$$

with $(b_0, \cdots, b_{m-1})$ being the base-2 decomposition of the index $i = \sum_{k=0}^{m-1} b_k \cdot 2^k$.

For a given level set ϕ_j, equation (A.11) simply compares the color of a given pixel with the colors of the two regions obtained changing the sign of ϕ_j. This is logic since, for a given level set ϕ_j at a particular pixel position, the only choice that we have is whether to preserve the actual sign of the level set or to change the speed sign in order to change the level set sign. This is the same behavior as equation (A.8). The problem with the speed is that the choice of its sign is only based on a subset (actually only two) of all the region colors. This means that, for a given pixel, the choice of the *best* region is only made using a few colors and not all the region colors. On the one hand, as shown by [Vese and Chan, 2002], this suffices for a large number of images to converge to a correct segmentation. On the other hand, the initialization takes a very important place since, depending on the initial image partitioning, the segmentation will converge to a local minimum due to the impossibility of selecting the globally *best* color. We propose a way to improve this convergence problem by using all the region colors in the decision, which can be achieved using the following speed set:

$$F_d^j = \min_{i \in \mathcal{I}_0(j)} \left\{ (u_0 - c_i)^2 \right\} - \min_{i \in \mathcal{I}_1(j)} \left\{ (u_0 - c_i)^2 \right\}, \tag{A.12}$$

with

$$\begin{aligned} \mathcal{I}_0(j) &= \left\{ i = \sum_{k=0}^{m-1} b_k \cdot 2^k,\ b_j = 0 \right\}, \\ \mathcal{I}_1(j) &= \left\{ i = \sum_{k=0}^{m-1} b_k \cdot 2^k,\ b_j = 1 \right\}. \end{aligned}$$

Here we compare the color of a given pixel with the color of *all* the regions and, based on this comparison, we decide to keep or to change the sign of the speed function. This new speed set dramatically improves the convergence properties of the multi-phase segmentation algorithm.

In order to better see the shape of equations (A.11) and (A.12), we can develop them for the special case of $m = 2, n = 2^m = 4$, i.e., a 4-phase segmentation with two level sets, where the indices i, $A(i,j)$ and $B(i,j)$ of equation A.11 are represented in base-2 for more clarity, i.e., $\chi_{b_1 b_0}$, $c_{b_1 b_0}$. Equation (A.11) becomes:

$$\begin{aligned} F_d^0 &= \chi_{00}\left((u_0 - c_{00})^2 - (u_0 - c_{01})^2\right) + \chi_{01}\left((u_0 - c_{00})^2 - (u_0 - c_{01})^2\right) + \\ &\quad \chi_{10}\left((u_0 - c_{10})^2 - (u_0 - c_{11})^2\right) + \chi_{11}\left((u_0 - c_{10})^2 - (u_0 - c_{11})^2\right), \\ F_d^1 &= \chi_{00}\left((u_0 - c_{00})^2 - (u_0 - c_{10})^2\right) + \chi_{01}\left((u_0 - c_{01})^2 - (u_0 - c_{11})^2\right) + \\ &\quad \chi_{10}\left((u_0 - c_{00})^2 - (u_0 - c_{10})^2\right) + \chi_{11}\left((u_0 - c_{01})^2 - (u_0 - c_{11})^2\right), \end{aligned} \tag{A.13}$$

where χ_i is the characteristic function of the region ω_i:

$$\begin{aligned}
\chi_{00}(\mathbf{x}) &= \begin{cases} 1 \text{ if } \phi_0(\mathbf{x}) < 0 \text{ and } \phi_1(\mathbf{x}) < 0 \\ 0 \text{ else} \end{cases} \\
\chi_{01}(\mathbf{x}) &= \begin{cases} 1 \text{ if } \phi_0(\mathbf{x}) > 0 \text{ and } \phi_1(\mathbf{x}) < 0 \\ 0 \text{ else} \end{cases} \\
\chi_{10}(\mathbf{x}) &= \begin{cases} 1 \text{ if } \phi_0(\mathbf{x}) < 0 \text{ and } \phi_1(\mathbf{x}) > 0 \\ 0 \text{ else} \end{cases} \\
\chi_{11}(\mathbf{x}) &= \begin{cases} 1 \text{ if } \phi_0(\mathbf{x}) > 0 \text{ and } \phi_1(\mathbf{x}) > 0 \\ 0 \text{ else} \end{cases}
\end{aligned} ,$$

and c_i is the mean color of the original image u_0 computed inside the region ω_i:

$$c_i = \frac{\int_{\omega_i} u_0 dxdy}{\int_{\omega_i} dxdy} = \frac{\int_\Omega \chi_i u_0 dxdy}{\int_\Omega \chi_i dxdy}.$$

Equation (A.12) gives:

$$\begin{aligned}
F_d^0 &= \min\left\{(u_0 - c_{00})^2,\ (u_0 - c_{10})^2\right\} - \min\left\{(u_0 - c_{01})^2,\ (u_0 - c_{11})^2\right\}, \\
F_d^1 &= \min\left\{(u_0 - c_{00})^2,\ (u_0 - c_{01})^2\right\} - \min\left\{(u_0 - c_{10})^2,\ (u_0 - c_{11})^2\right\}.
\end{aligned} \tag{A.14}$$

In Figures A.10 and A.11 we can see the level set evolution for the equation (A.13). In Fig.A.10 we show the evolution of the zero level sets of the two level sets functions ϕ_0 and ϕ_1 together with the original image. In Fig.A.11 we show the same evolution with the regions colored with their corresponding mean colors c_i. As we can appreciate, the level sets converge to the global minimum, giving a very good segmentation except for the right foot, where the bottom of the foot is integrated into the background due to the shadows. The same evolution but with a different initialization is shown in Figures A.12 and A.13. As a difference with the previous case, equation (A.13) gets locked into a local minimum and is unable to converge properly. The problem is that some pixels need to change both signs of the two level sets but they cannot since passing through one of the intermediate states (only one level set changes at a time) temporary increases the overall energy: in Fig.A.13 a green pixel classed as gray (class c_{00}) needs to be temporary classed as a pink or red pixel (classes c_{01} and c_{10}) in order to be finally classed as green (class c_{11}). The same applies to red pixels that are classed as pink and vice versa.

This problem does not happen with the proposed speed set (A.14) since, for the same initialization, the color comparison will always give us the best classification independently of the current one (see Fig.A.14). This implies that the overall energy may temporary increase when one level set changes its sign quicker than the other, which in this case means that some pink pixels will be classed as gray or green until the slowest level set changes its sign (see Fig.A.15).

Although convergence can be greatly improved by reinitializing the level sets to the distance function of the zero level sets, the results shown have been obtained without reinitialization, which increases the overall number of iterations.

Figure A.10: *4-phase segmentation using [Vese and Chan, 2002] speed. The level sets are able to converge to the global minimum.* ϕ_0 *and* ϕ_1 *zero level set are drawn in black and white color, respectively.*

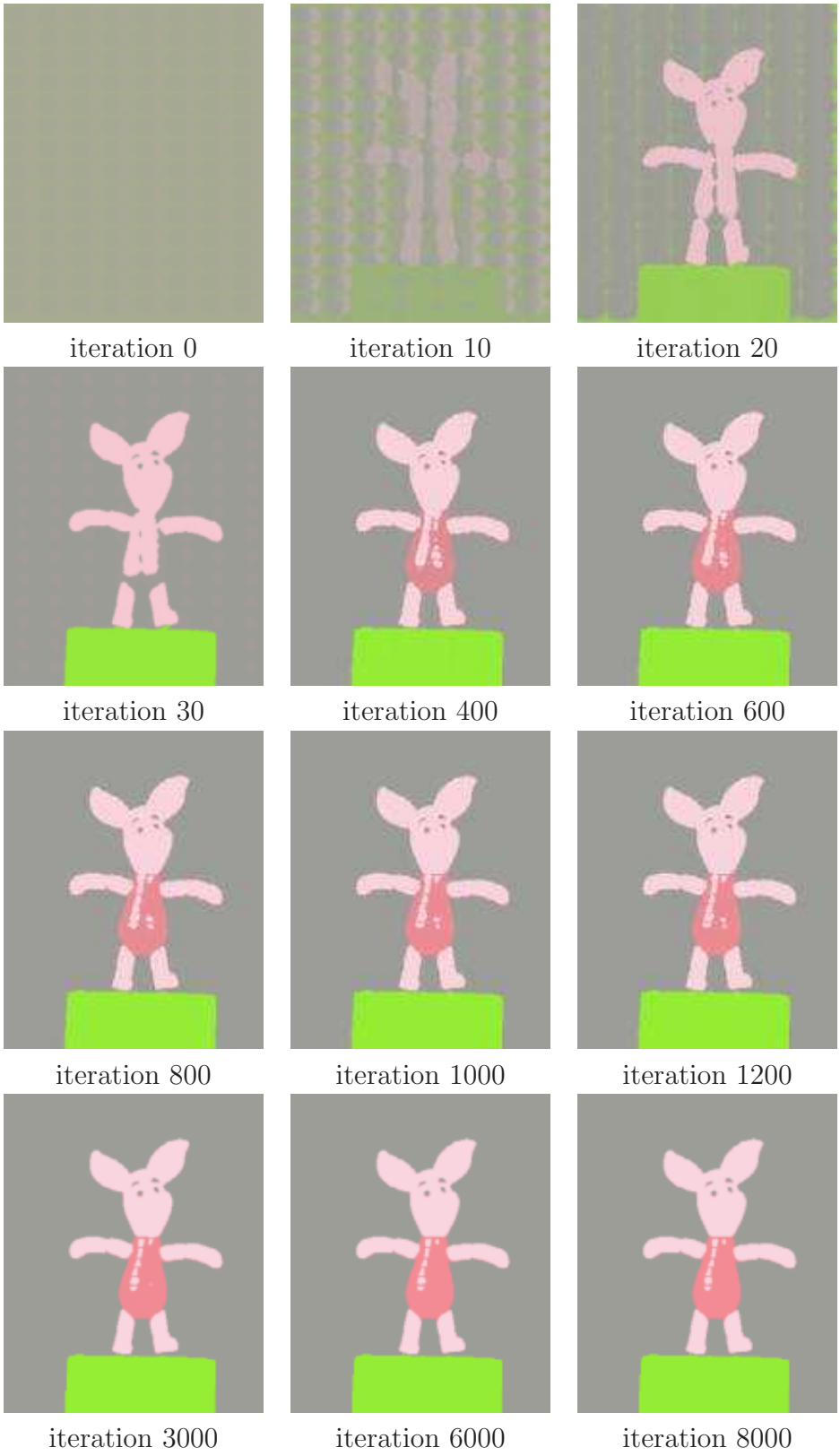


Figure A.11: *4-phase segmentation using [Vese and Chan, 2002] speed. The level sets are able to converge to the global minimum. The four regions are drawn with their corresponding mean color.*

Figure A.12: *4-phase segmentation using [Vese and Chan, 2002] speed. The level sets get locked into a local minimum due to the initialization.* ϕ_0 *and* ϕ_1 *zero level set are drawn in black and white color, respectively.*

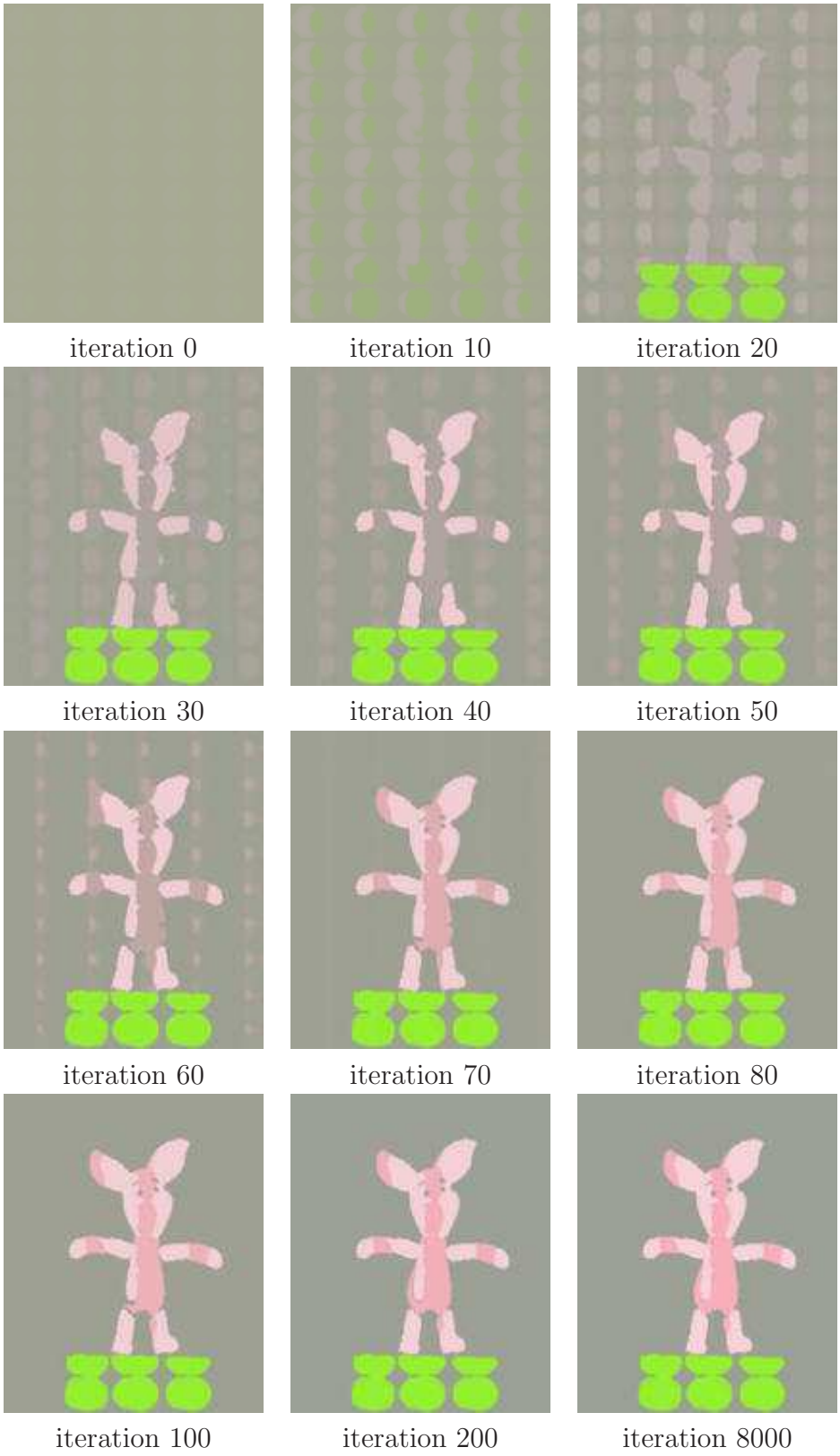


Figure A.13: *4-phase segmentation using [Vese and Chan, 2002] speed. The level sets get locked into a local minimum due to the initialization. The four regions are drawn with their corresponding mean color.*

Figure A.14: *4-phase segmentation using the proposed speed. The level sets are able to converge to the global minimum.* ϕ_0 *and* ϕ_1 *zero level set are drawn in black and white color, respectively.*

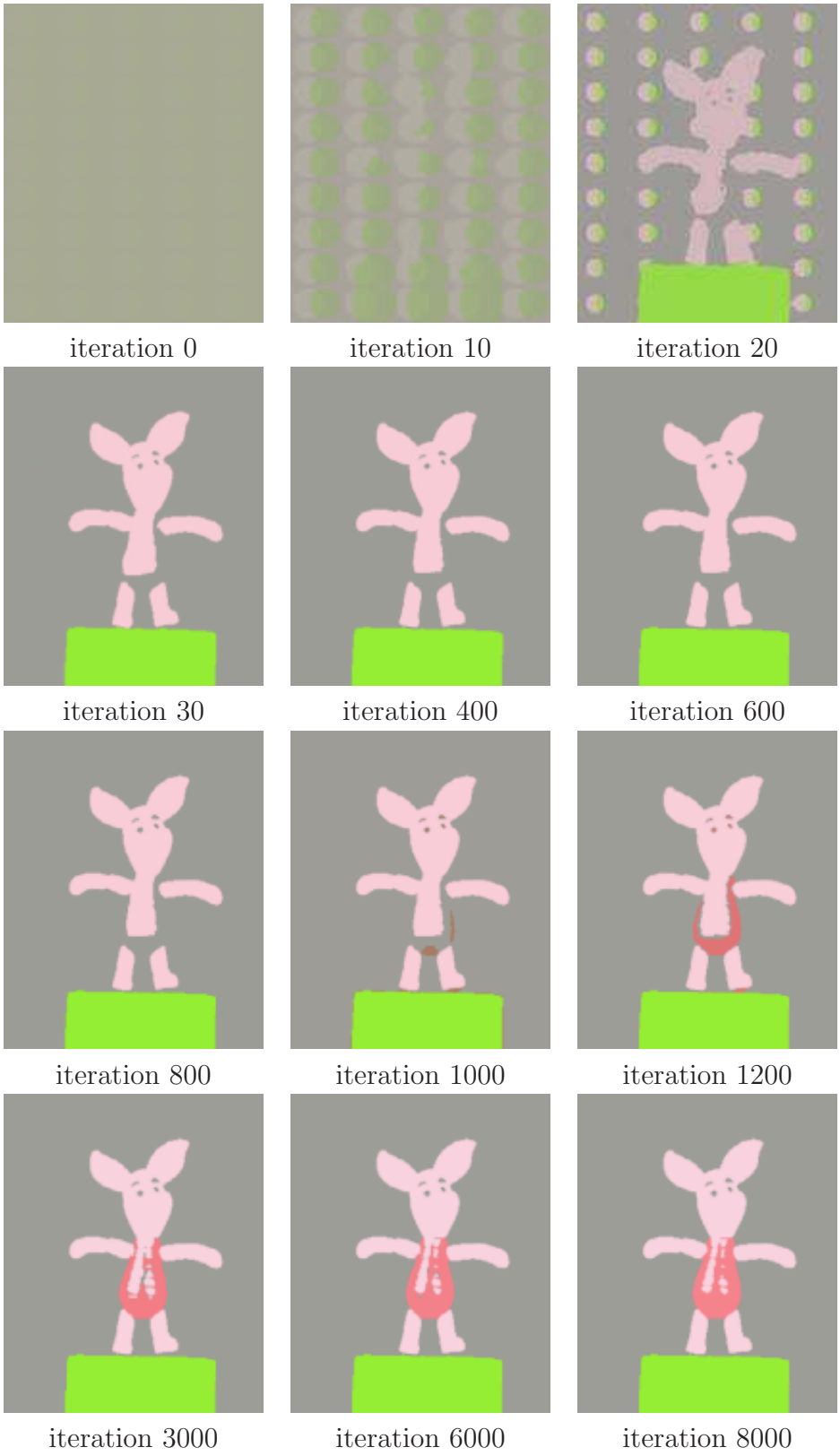


Figure A.15: *4-phase segmentation using the proposed speed. The level sets are able to converge to the global minimum. The four regions are drawn with their corresponding mean color.*

A.4 Color Histogram Algorithm

This simple method makes the hypothesis that the colors appearing on the background are different from those of the object. Both Commercial and free software such as Photoshop or Gimp dispose of a basic tool (called magic wand) that performs a simplified version of this algorithm. The basic implementation selects a background color (a point in 3D color space) and chooses the color cube with a given edge size (the color tolerance) centered on the selected background color as the histogram of the background. The color space does not need to be RGB and can be a more sophisticated one such as Lab. In our implementation, we have developed a more sophisticated approach by constructing the color histogram using a learning region of the image.

Once we have the histogram, all the implementations proceed in the same way. Based on the histogram, the algorithm simply tags each pixel on the image as belonging or not to the color histogram, which already gives us a binary image. To obtain more reliable results, the biggest foreground component is selected and background components smaller than a minimum size are rejected.

Although this technique is quite robust, it can fail in two ways:

- The first one is not extremely harmful and happens when a region inside the object has a color belonging to the histogram, which produces a hole in the binary image. This type of holes can be filled either manually or in a semi-automatic way. Automatic detection of this kind of holes can work since in general, the contours of the hole are not smooth and have a very characteristic shape.

- The second case is more problematic. It occurs when the object has a color of the histogram *on the silhouette contour*. This makes the silhouette of the object be eroded and, in general, there are few possibilities to automatically correct this kind of problem. The only solution is to try to refine the background color histogram to exclude that particular object color while keeping a good segmentation.

This algorithm is the one that has been used for the extraction of the silhouettes from the color sequence whenever the silhouette sequence was not available in Chapter 3.

Thesis Publications

Journal

- **Using Silhouette Coherence for Camera Motion and Focal Length Recovery under Circular Motion**
  C. Hernández Esteban and F. Schmitt
  Submitted to *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004.

- **Silhouette and Stereo Fusion for 3D Object Modeling**
  C. Hernández Esteban and F. Schmitt
  To be plublished in *Computer Vision and Image Understanding, Special issue on "Model-based and image-based 3D Scene Representation for Interactive Visualization"*, third quarter 2004.

Conference with Proceedings

- **Une approche par modèle déformable pour la reconstruction 3D de haute qualité d'objets photographiés**
  C. Hernández Esteban et F. Schmitt
  RFIA 2004, 14ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle, Toulouse, France, 28 - 30 Janvier 2004, vol. 2, pp. 905-914.

- **Silhouette and Stereo Fusion for 3D Object Modeling**
  C. Hernández Esteban and F. Schmitt
  3DIM 2003, 4th International Conference on 3D Digital Imaging and Modeling, Banff, Alberta, Canada, October 2003, pp. 46-53.

- **A Snake Approach for High Quality Image-based 3D Object Modeling**
  C. Hernández Esteban and F. Schmitt
  2nd IEEE Workshop on Variational, Geometric and Level Set Methods in Computer Vision, Nice, France, October 2003, pp. 241-248.

- **Multi-Stereo 3D Object Reconstruction**
  C. Hernández Esteban and F. Schmitt
  3DPVT - 1st International Symposium on 3D Data Processing Visualization and Transmission, Padova, Italy, June 2002, pp. 159-166.

- **Reconstruction 3D d'objets par multi-stéréo**
  C. Hernández Esteban and F. Schmitt
  AFIG 2001 - 14èmes journées de l'Association Française d'Informatique Graphique, Limoges, November 2001, pp. 27-36.

Technical Reports

- **Using Silhouette Coherence for 3D Image-based Object Modeling under Circular Motion**
  C. Hernández Esteban and F. Schmitt
  Technical report 2003D011, Ecole Nationale Supérieure des Télécommunications, September, 2003.

Bibliography

[Abdel-Aziz and Karara, 1971] Abdel-Aziz, Y. I. and Karara, H. M. (1971). Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. In *Proc. ASP/UI Symp. on CloseRange Photogrammetry*, pages 1–18. 1.1.3, 1.1.3, 1.1.3

[Adalsteinsson and Sethian, 1995] Adalsteinsson, D. and Sethian, J. (1995). A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118:269–277. 2.2.1

[Armstrong et al., 1994] Armstrong, M., Zisserman, A., , and Beardsley, P. (1994). Euclidean structure from uncalibrated images. In *Proceedings of the 5th BMVC*, pages 508–518, York, UK. BMVA Press. 1.1.4

[Ball and Hall, 1965] Ball, G. and Hall, D. (1965). ISODATA, a novel method of data analysis and pattern classification. Technical report, Stanford Research Institute. A.2

[Ball and Hall, 1967] Ball, G. and Hall, D. (1967). A clustering technique for summarizing multivariate data. *Behav. Sci.*, 12:153–155. A.2

[Baumgart, 1974] Baumgart, B. G. (1974). *Geometric Modelling for Computer Vision*. PhD thesis, Standford University. 2.1

[Berger and Mohr, 1990] Berger, M. O. and Mohr, R. (1990). Towards autonomy in active contour models. In *Proceedings of ICPR*, pages 847–851, Atlantic City. 2.3

[Besl and McKay, 1992] Besl, P. J. and McKay, N. D. (1992). A method for registration of 3-d shapes. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14(2):239–256. 1.9.2

[Bottino and Laurentini, 2003] Bottino, A. and Laurentini, A. (2003). Introducing a new problem: Shape-from-silhouette when the relative positions of the viewpoints is unknown. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(11):1484–1493. 1.2

[Broadhurst et al., 2001] Broadhurst, A., Drummond, T., and Cipolla, R. (2001). A probabilistic framework for the Space Carving algorithm. In *Proc. 8th ICCV*, pages 388–393, Vancouver, Canada. IEEE Computer Society Press. 2.4

[Brown, 1976] Brown, D. C. (1976). The bundle adjustment - progress and prospects. *Int. Archives Photogrammetry*, 21(3). 1.1.3

[Caselles et al., 1993] Caselles, V., Catte, F., Coll, T., and Dibos, F. (1993). A geometric model for active contours. *Numerische Mathematik*, 66:1–31. 2.2.1

[Chan and Vese, 2001] Chan, T. and Vese, L. (2001). An active contour model without edges. *IEEE Transactions on Image Processing*, 10(2):266–277. A.3, A.3

[Chen and Williams, 1993] Chen, S. and Williams, L. (1993). View interpolation for image synthesis. In *SIGGRAPH '93*, pages 279–288. 2.1

[Cheung, 2003] Cheung, K. (2003). *Visual Hull Construction, Alignment and Refinement for Human Kinematic Modeling, Motion Tracking and Rendering*. PhD thesis, Carnegie Mellon University. 1.2, 1.5

[Cheung et al., 2003] Cheung, K., Baker, S., and Kanade, T. (2003). Visual hull alignment and refinement across time: A 3d reconstruction algorithm combining shape-from-silhouette with stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 375–382. 1.2

[Cohen et al., 1992] Cohen, I., Cohen, L. D., and Ayache, N. (1992). Using deformable surfaces to segment 3-D images and infer differential structures. *CVGIP*, 56(2):242–263. 2.2.1

[Cohen, 1991] Cohen, L. D. (1991). On active contour models and balloons. *CVGIP: Graphical models and Image Processing*, 53(2):211–218. 2.2.1

[Cohen, 1997] Cohen, L. D. (1997). Avoiding local minima for deformable curves in image analysis. In *Curves and Surfaces with Applications in CAGD*, pages 77–84. A. Le Méhauté, C. Rabut, and L. L. Schumaker (eds.). A.3

[Cohen et al., 1993] Cohen, L. D., Bardinet, E., and Ayache, N. (1993). Reconstruction of digital terrain model with a lake. In *Proceedings SPIE 93 Conference on Geometric Methods in Computer Vision*, San Diego, CA. autre version RR Inria 1824, Décembre 1992. A.3

[Cohen and Cohen, 1993] Cohen, L. D. and Cohen, I. (1993). Finite element methods for active contour models and balloons for 2-D and 3-D images. *PAMI*, 15(11):1131–1147. 2.6

[Courteille et al., 2004] Courteille, F., Crouzil, A., Durou, J., and Gurdjos, P. (2004). Towards shape from shading under realistic photographic conditions. In *Proc. Reconnaissance des formes et l'Intelligence artificielle 04*, volume 2, pages 925–934. France. 2.1

[Cross and Zisserman, 2000] Cross, G. and Zisserman, A. (2000). Surface reconstruction from multiple views using apparent contours and surface texture. In Leonardis,

A., Solina, F., and Bajcsy, R., editors, *NATO Advanced Research Workshop on Confluence of Computer Vision and Computer Graphics, Ljubljana, Slovenia*, pages 25–47. 2.1, 2.4

[Curless and Levoy, 1996] Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *SIGGRAPH '96*, pages 303–312. 2.1

[Das, 1949] Das, G. (1949). A mathematical approach to problems in photogrammetry. *Empire Survey Review*, 10(73). 1.1.3

[Debevec et al., 1996] Debevec, P. E., Taylor, C. J., and Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry and image-based approach. In *SIGGRAPH '96*, pages 11–20. 2.1

[Delingette, 1994] Delingette, H. (1994). *Modélisation, Déformation et Reconnaissance d'objets tridimensionnels a l'aide de maillages simplexes*. PhD thesis, Ecole Centrale de Paris. 2.2.2

[Deng et al., 1999] Deng, Y., Kenney, C., Moore, M., and Manjunath, B. S. (1999). Peer group filtering and perceptual color image quantization. In *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems*, volume 4, pages 21–24. A.2

[Deng and Manjunath, 2001] Deng, Y. and Manjunath, B. S. (2001). Unsupervised segmentation of color-texture regions in images and video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(8):800–810. A.2, A.2, A.3

[Desbrun et al., 1999] Desbrun, M., Meyer, M., Schroder, P., and Barr, A. H. (1999). Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH'99*, pages 317–324. 2.6

[Diday, 1970] Diday, E. (1970). Une nouvelle méthode de classification automatique et de reconnaissance des formes : la méthode des nuées dynamiques. *Revue de Statistique Appliquée*, 19(2). A.2

[Dürer, 1977] Dürer, A. (1977). *Underweysung der Messung*. Albaris Books, New York, original edition 1525 edition. Nuremberg. 1.1

[Faugeras and Keriven, 1998] Faugeras, O. and Keriven, R. (1998). Variational principles, surface evolution, pdes, level set methods, and the stereo problem. *IEEE Transactions on Image Processing*, 7(3):336–344. 2.1, 2.4

[Faugeras and Luong, 2001] Faugeras, O. and Luong, Q. (2001). *The Geometry of Multiple Images*. MIT Press. 1.1, 1.2

[Faugeras and Toscani, 1986] Faugeras, O. and Toscani, G. (1986). The calibration problem for stereo. In *Proceedings IEEE CVPR 86*, pages 15–20. 1.1.3

[Faugeras, 1995] Faugeras, O. D. (1995). Stratification of 3-dimensional vision: Projective, affine, and metric representations. *Journal of the Optical Society of America*, 12(3):465–484. 1.1.4

[Faugeras et al., 1992] Faugeras, O. D., Luong, Q. T., and Maybank, S. J. (1992). Camera self-calibration: Theory and experiments. In Verlag, S., editor, *Proc. 2nd European Conf. on Computer Vision*, LNCS 588, pages 321–334. 1.1.4

[Fitzgibbon et al., 1998] Fitzgibbon, A. W., Cross, G., and Zisserman, A. (1998). Automatic 3D model construction for turn-table sequences. In R., R. K. and Van-Gool., L., editors, *3D Structure from Multiple Images of Large-Scale Environments, LNCS 1506*, pages 155–170. Springer-Verlag. 1.2, 1.6.3

[Fletcher, 1965] Fletcher, R. (1965). Function minimization without evaluating derivatives—a review. *The Computer Journal*, 8(1):33–41. 1.7.1

[Fletcher, 1987] Fletcher, R. (1987). *Practical Methods of Optimization*. John Wiley. 1.7.2

[Fletcher and Reeves, 1964] Fletcher, R. and Reeves, C. (1964). Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154. 1.7.2

[Fornberg, 1988] Fornberg, B. (1988). Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of Computation*, 51:699–706. 2.4.4

[Franco and Boyer, 2003] Franco, J.-S. and Boyer, E. (2003). Exact polyhedral visual hulls. In *Fourteenth British Machine Vision Conference (BMVC)*, pages 329–338. Norwich, UK. 1.10

[Fua and Leclerc, 1995] Fua, P. and Leclerc, Y. (1995). Object-centered surface reconstruction: Combining multi-image stereo and shading. *International Journal of Computer Vision*, 16:35–56. 2.1, 2.4

[Fua and Leclerc, 1996] Fua, P. and Leclerc, Y. G. (1996). Taking advantage of image-based and geometry-based constraints to recover 3–d surfaces. *Computer Vision and Image Understanding*, 64(1):111–127. 2.1

[Furukawa et al., 2004] Furukawa, Y., Sethi, A., Ponce, J., and Kriegman, D. (2004). Structure and motion from images of smooth textureless objects. In *to appear in ECCV 2004*, Prague, Czech Republic. 1.2

[Fusiello et al., 2000] Fusiello, A., Trucco, E., and Verri, A. (2000). A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12(1):16–22. 3.5.1

[Gilmore and Kelley, 1995] Gilmore, P. and Kelley, C. T. (1995). An implicit filtering algorithm for optimization of functions with many local minima. *SIAM J. Opt*, 4:269–285. 1.7

[Griewank, 2000] Griewank, A. (2000). *Evaluating derivatives: principles and techniques of algorithmic differentiation*. Society for Industrial and Applied Mathematics. 1.7

[Han et al., 2003] Han, X., Xu, C., and Prince, J. L. (2003). A topology preserving level set method for geometric deformable models. *IEEE Transactions on PAMI*, 25:755–768. 2.2.1

[Haralick et al., 1991] Haralick, R., Lee, C., Ottenberg, K., and Noelle, M. (1991). Analysis and solutions of the three point perspective pose estimation problem. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 592–598. 1.1.3

[Hartley, 1994] Hartley, R. (1994). Euclidean reconstruction from uncalibrated views. *Lecture Notes in Computer Science*, 825:237–256. 1.1.4, 1.1.4

[Hartley and Zisserman, 2000] Hartley, R. and Zisserman, A. (2000). *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK. 1.1

[Heyden and Aström, 1996] Heyden, A. and Aström, K. (1996). Euclidean reconstruction from constant intrinsic parameters. *Proc. 13th International Conference on Pattern Recognition*, pages 339–343. 1.1.4

[Hooke and Jeeves, 1961] Hooke, R. and Jeeves, T. A. (1961). Direct search solution of numerical and statistical problems. *Journal of the ACM*, 8:212–229. 1.7.1

[Hoppe et al., 1993] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W. (1993). Mesh optimization. In *SIGGRAPH'93*, pages 19–26. ACM Press. 3, 2.7

[Horn and Brooks, 1989] Horn, B. and Brooks, M. (1989). *Shape from Shading*. The MIT Press. 2.1

[Hough et al., 2001] Hough, P. D., Kolda, T. G., and Torczon, V. J. (2001). Asynchronous parallel pattern search for nonlinear optimization. *SIAM J. Scientific Computing*, 23(1):134–156. 1.7, 1.7.1

[Huang and Faugeras, 1989] Huang, T. S. and Faugeras, O. (1989). Some properties of the e-matrix in two view motion estimation. *IEEE Trans. Pattern Analysis and Machine Intell.*, 11(12):1310–1312. 1.1.4

[Ilic and Fua, 2003a] Ilic, S. and Fua, P. (2003a). Generic deformable implicit mesh models for automated reconstruction. In *ICCV workshop on Higher-Level Knowledge in 3D Modelling and Motion Analysis*, Nice. France. 2.1

[Ilic and Fua, 2003b] Ilic, S. and Fua, P. (2003b). Implicit meshes for modeling and reconstruction. In *Conference on Computer Vision and Pattern Recognition*, Madison, WI. 2.1

[Isidoro and Sclaroff, 2003] Isidoro, J. and Sclaroff, S. (2003). Stochastic refinement of the visual hull to satisfy photometric and silhouette consistency constraints. In *Proc. ICCV*, pages 1335 –1342. 2.1, 2.4

[Jiang et al., 2002] Jiang, G., Tsui, H., Quan, L., and Zisserman, A. (2002). Single axis geometry by fitting conics. In *ECCV*, volume 1, pages 537–550. 1.2

[Jin et al., 2000] Jin, H., Yezzi, A., and Soatto, S. (2000). Stereoscopic shading: Integrating shape cues in a variational framework. In *Proc. Intl. Conf. on Computer Vision and Pattern Recognition*, pages 169–176. 2.1

[Kass et al., 1988] Kass, M., Witkin, A., and Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of Computer Vision*, 1:321–332. 1.8.2, 2.2.1

[Kobbelt, 2000] Kobbelt, L. (2000). $\sqrt{3}$-subdivision. In *SIGGRAPH 2000*, pages 103–112. 2.7

[Kolda et al., 2003] Kolda, T. G., Lewis, R. M., , and Torczon, V. (2003). Optimization by direct search: New perspectives on some classical and modern methods. *SIAM J. Scientific Computing*, 45(3):385–482. 1.7.1

[Kruppa, 1914] Kruppa, E. (1914). Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung. *Math, Naturw. Abt. IIa*, 122:1939–1948. 1.1.4

[Lachaud and Montanvert, 1999] Lachaud, J. O. and Montanvert, A. (1999). Deformable meshes with automated topology changes for coarse-to-fine 3d surface extraction. *Medical Image Analysis*, 3(2):187–207. 2.9

[Laurentini, 1994] Laurentini, A. (1994). The visual hull concept for silhouette based image understanding. *IEEE Trans. on PAMI*, 16(2). 1.2, 2.1, 2.3

[Lavest et al., 1998] Lavest, J. M., Viala, M., and Dhome, M. (1998). Do we really need an accurate calibration pattern to achieve a reliable camera calibration? In *Proc. ECCV*, volume 1, pages 158–174. Germany. 1.1.3, 1.8.3, 3.1

[Lazebnik et al., 2001] Lazebnik, S., Boyer, E., and Ponce, J. (2001). On computing exact visual hulls of solids bounded by smooth surfaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 156–161, Kauai, Hawaii. 2.3.1

[Lensch et al., 2001] Lensch, H., Heidrich, W., and Seidel, H. P. (2001). A silhouette-based algorithm for texture registration and stitching. *Journal of Graphical Models*, pages 245–262. 1.2, 2.8

[Levenberg, 1944] Levenberg, K. (1944). A method for the solution of certain problems in least squares. *Quart. Appl. Math.*, 2:164–168. 1.7.2

[Levoy et al., 2000] Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., and Fulk, D. (2000). The digital michelangelo project: 3d scanning of large statues. In *SIGGRAPH 2000*, pages 131–144. 2.1

[Li et al., 2003] Li, M., Magnor, M., and Seidel, H. (2003). Improved hardware-accelerated visual hull rendering. In *VMV: Vision, Modeling, and Visualization*, pages 151–158. 1.2

[Li et al., 2002] Li, M., Schirmacher, H., Magnor, M., and Seidel, H. (2002). Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes. In *Proceedings of IEEE 2002 Workshop on Multimedia and Signal Processing*, pages 9–12. 2.1

[Liedtke et al., 1991] Liedtke, C.-E., Busch, H., and Koch, R. (1991). Shape adaptation for modelling of 3d objects in natural scenes. In *Proc. Conference on Computer Vision and Pattern Recognition*, pages 704–705. USA. 2.1, 2.4, 2.4.2

[Lorensen and Cline, 1987] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of SIGGRAPH '87*, volume 21, pages 163–169. 2

[Luong and Faugeras, 1997] Luong, Q.-T. and Faugeras, O. (1997). Self-calibration of a moving camera from point correspondences and fundamental matrices. *Int. Journal of Computer Vision*, 22(3):261 289. 1.1.4

[Malladi et al., 1995] Malladi, R., J.A.Sethian, and Vemuri, B. (1995). Shape modelling with front propagation: A level set approach. *IEEE Tr. on PAMI*, 17(2):158–175. 2.2.1

[Marquardt, 1963] Marquardt, D. (1963). An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, 11:431–441. 1.7.2

[Martin and Aggarwal, 1983] Martin, W. and Aggarwal, J. K. (1983). Volumetric descriptions of objects from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):150–158. 2.3

[Matsumoto et al., 1999] Matsumoto, Y., Fujimura, K., and Kitamura, T. (1999). Shape-from-silhouette/stereo and its application to 3-d digitizer. In *Proceedings of Discrete Geometry for Computing Imagery*, pages 177–190. 2.1, 2.4, 2.4.3

[Matsumoto et al., 1997] Matsumoto, Y., Terasaki, H., Sugimoto, K., and Arakawa, T. (1997). A portable three-dimensional digitizer. In *Int. Conf. on Recent Advances in 3D Imaging and Modeling*, pages 197–205. Ottowa. 2.1

[Matsushita and Kanedo, 1999] Matsushita, K. and Kanedo, T. (1999). Efficient and handy texture mapping on 3d surfaces. *Computer Graphics Forum*, 18:349–358. 1.2

[Matusik et al., 2000] Matusik, W., Buehler, C., Raskar, R., Gortler, S., and McMillan, L. (2000). Image-based visual hulls. *SIGGRAPH 2000*, pages 369–374. 1.2, 1.6, 1.6, 1.6.1, 1.6.1, 1.6.2, 2.1, 2.3

[Maybank and Faugeras, 1992] Maybank, S. and Faugeras, O. D. (1992). A theory of self-calibration of a moving camera. *Int. Journal of Computer Vision*, 8(2):123–151. 1.1.4

[Mckinnon, 1998] Mckinnon, K. (1998). Convergence of the nelder-mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158. 1.7.1

[McMillan and Bishop, 1995] McMillan, L. and Bishop, G. (1995). Plenoptic modeling: An image-based rendering system. In *SIGGRAPH '95*, pages 39–46. 2.1

[Medioni et al., 2000] Medioni, G., Lee, M.-S., and Tang, C.-K. (2000). *A Computational Framework for Segmentation and Grouping*. Elsevier. 2.4

[Melen, 1994] Melen, T. (1994). *Geometric Modeling and Calibration of Video Cameras for Underwater Navigation*. PhD thesis, Institute for Teknisk Kybernetikk, Norges Tekniske Hogskole, Trondheim. 1.1.3

[Mendonça et al., 2001] Mendonça, P. R. S., Wong, K.-Y. K., and Cipolla, R. (2001). Epipolar geometry from profiles under circular motion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(6):604–616. 1.2

[Morse and Feshbach, 1953] Morse, P. M. and Feshbach, H. (1953). *Methods of Theoretical Physics, Part I*, pages 434–443. Asymptotic Series; Method of Steepest Descent. McGraw-Hill, New York. 1.7.2

[Mumford and Shah, 1989] Mumford, D. and Shah, J. (1989). Optimal approximations by piecewise smooth functions and associated variational-problems. *Communications on Pure and Applied Mathematics*, 42(5):577–685. A.2, A.3

[Nelder and Mead, 1965] Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, pages 308–313. 1.7.1

[Neugebauer and Klein, 1999] Neugebauer, P. J. and Klein, K. (1999). Texturing 3d models of real world objects from multiple unregistered photographic views. *Computer Graphics Forum*, 18:245–256. 1.2

[Niem and Wingbermuhle, 1997] Niem, W. and Wingbermuhle, J. (1997). Automatic reconstruction of 3d objects using a mobile monoscopic camera. In *Int. Conf. on Recent Advances in 3D Imaging and Modeling*, pages 173–181. Ottowa. 2.1, 2.3

[Nobuhara and Matsuyama, 2003] Nobuhara, S. and Matsuyama, T. (2003). Dynamic 3d shape from multi-viewpoint images using deformable mesh models. In *Proc. of 3rd International Symposium on Image and Signal Processing and Analysis*, pages 192–197. 2.1, 2.4, 2.5

[Nocedal and Wright, 1999] Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer-Verlag. 1.7.2

[Osher and Sethian, 1988] Osher, S. and Sethian, J. A. (1988). Fronts propagating with curvature-dependant speed: Algorithms based on hamilton-jacobi formulation. *Journal of Computational Physics*, 79:12–49. A.3

[Paragios and Deriche, 2000] Paragios, N. and Deriche, R. (2000). Coupled geodesic active regions for image segmentation: A level set approach. In Press, I. C. S., editor, *Proceedings of the 6th European Conference on Computer Vision*, volume 2, pages 224–240. A.3

[Park and Subbarao, 2002] Park, S. and Subbarao, M. (2002). Pose estimation of two-pose 3d models using the base tangent plane and stability constraints. In *VMV 2002*, pages 379–386. 1.2

[Plass and Stone, 1983] Plass, M. and Stone, M. (1983). Curve fitting with piecewise parametric cubics. *SIGGRAPH'83*, 17(3):229–239. 1.8.2

[Plänkers and Fua, 2003] Plänkers, R. and Fua, P. (2003). Articulated soft objects for multi-view shape and motion capture. *Transactions on Pattern Analysis and Machine Intelligence*, 25(10):394–401. 2.1

[Polak, 1971] Polak, E. (1971). *Computational Methods in Optimization: A Unified Approach*. Academic Press, New York. 1.7.1

[Pollefeys and Van-Gool, 1997] Pollefeys, M. and Van-Gool, L. (1997). A stratified approach to self-calibration. *Proc. 1997 Conference on Computer Vision and Pattern Recognition*, pages 407–412. 1.1.4

[Pollefeys et al., 1996] Pollefeys, M., Van-Gool, L., and Oosterlinck, A. (1996). The modulus constraint: A new constraint for self-calibration. *Lecture Notes in Computer Science*, 1064:31–42. 1.1.4

[Porrill and Pollard, 1991] Porrill, J. and Pollard, S. B. (1991). Curve matching and stereo calibration. *Image and Vision Computing*, 9(1):45–50. 1.2

[Potmesil, 1987] Potmesil, M. (1987). Generating octree models of 3d objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40:1–29. 2.1, 2.3, 2.3.1

[Powell, 1964] Powell, M. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 17:155–162. 1.7.1, 1.8

[Prados and Faugeras, 2003] Prados, E. and Faugeras, O. (2003). Perspective shape from shading and viscosity solutions. In *IEEE Proceedings of ICCV'03*, volume 2, pages 826–831. 2.1

[Press et al., 1992] Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1992). *Numerical Recipes in C (The Art of Scientific Computing)*. Cambridge University Press. 1.7, 2.3.1

[Quan and Lan, 1998] Quan, L. and Lan, Z. (1998). Linear n $\leq$ 4-point pose determination. In *IEEE Int. Conf. Computer Vision*, pages 778–783. 1.1.3

[Rieger, 1986] Rieger, J. H. (1986). Three dimensional motion from fixed points of a deforming profile curve. *Optics Letters*, 11(3):123–125. 1.2

[Rosenbrock, 1960] Rosenbrock, H. (1960). An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3:175–184. 1.7.1

[Samson et al., 2000] Samson, C., Blanc-Féraud, L., Aubert, G., and Zerubia, J. (2000). A level set model for image classification. *International Journal of Computer Vision*, 40(3):187–197. A.3

[Sarti and Tubaro, 2002] Sarti, A. and Tubaro, S. (2002). Image based multiresolution implicit object modeling. *EURASIP Journal on Applied Signal Processing*, 2002(10):1053–1066. 2.1, 2.4

[Scharstein and Szeliski, 2002] Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1):7–42. 2.4

[Schmitt et al., 1986] Schmitt, F., Barsky, B., and Du, W. (1986). An adaptative subdivision method for surface-fitting from sampled data. In *SIGGRAPH '86*, pages 179–188. 2.1

[Schmitt and Yemez, 1999] Schmitt, F. and Yemez, Y. (1999). 3d color object reconstruction from 2d image sequences. In *IEEE Interational Conference on Image Processing*, volume 3, pages 65–69. 2.8, 2.8.2

[Seitz and Dyer, 2000] Seitz, S. and Dyer, C. (2000). Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 38(3):197–216. 2.4

[Seitz and Dyer, 1997] Seitz, S. M. and Dyer, C. R. (1997). Photorealistic scene reconstruction by voxel coloring. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 1067–1073. 2.1

[Seitz and Kutulakos, 1998] Seitz, S. M. and Kutulakos, K. N. (1998). Plenoptic image editing. In *Proc. 6th Int. Conf. Computer Vision*, pages 17–24. 2.1

[Slabaugh et al., 2001] Slabaugh, G., Culbertson, W. B., Malzbender, T., and Shafer, R. (2001). A survey of methods for volumetric scene reconstruction from photographs. In *International Workshop on Volume Graphics 2001*. 2.1

[Slabaugh et al., 2002] Slabaugh, G., Schafer, R., and Hans, M. (2002). Image-based photo hulls. In *3DPVT '02*, pages 704–862. 2.1

[Soatto et al., 2003] Soatto, S., Yezzi, A. J., and Jin, H. (2003). Tales of shape and radiance in multi-view stereo. In *Proc. ICCV*, pages 974 –981. 2.1

[Soucy et al., 1996] Soucy, M., Godin, G., and Rioux, M. (1996). A texture-mapping approach for the compression of colored 3d triangulations. *The Visual Computer*, 12:503–514. 2.8, 2.28, 7, 7, 7, 2.8.1

[Spendley et al., 1962] Spendley, W., Hext, G. R., and Himsworth, F. R. (1962). Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4:441–461. 1.7.1

[Sturm, 1997a] Sturm, P. (1997a). Critical motion sequences for monocular self-calibration and uncalibrated euclidean reconstruction. In *CVPR - IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1100–1105, Puerto Rico. 1.4

[Sturm, 1997b] Sturm, P. (1997b). *Vision 3D non-calibrée: contributions à la reconstruction projective et études des mouvements critiques pour l'auto-calibrage*. PhD thesis, INP de Grenoble. 1.4

[Sturm, 2000] Sturm, P. (2000). A case against kruppa's equations for camera self-calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1199–1204. 1.1.4

[Sullivan and Ponce, 1998] Sullivan, S. and Ponce, J. (1998). Automatic model construction, pose estimation, and object recognition from photographs using triangular splines. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20(10):1091–1096. 1.2, 2.1

[Terzopoulos et al., 1987] Terzopoulos, D., Witkin, A., and Kass, M. (1987). Symmetry-seeking models for 3D object reconstruction. *International Journal of Computer Vision*, 1(3):211–221. 2.2.1

[Torczon, 1989] Torczon, V. J. (1989). *Multi-directional search: a direct search algorithm for parallel machine*. PhD thesis, Rice University, Houston, Texas. 1.7.1

[Torczon, 1997] Torczon, V. J. (1997). On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25. 1.7.1

[Triggs, 1997] Triggs, B. (1997). Autocalibration and the absolute quadric. In *Proceedings of the Conference on Computer Vision and Pattern Recognition, Puerto Rico, USA*, pages 609–614. IEEE Computer Society Press. 1.1.4

[Triggs, 1999] Triggs, B. (1999). Camera pose and calibration from 4 or 5 known points. In *IEEE Int. Conf. Computer Vision*, pages 278– 284. 1.1.3

[Triggs et al., 2000] Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (2000). Bundle adjustment – A modern synthesis. In Triggs, W., Zisserman, A., and Szeliski, R., editors, *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag. 1.1.3

[Trivedi, 1988] Trivedi, H. P. (1988). Can multiple views make up for lack of camera registration? *Image and Vision Computing*, 6(1):29 32. 1.1.4

[Tsai, 1987] Tsai, R. (1987). A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE J. Robotics and Automation*, 3(4):323–344. 1.1.3

[Vaillant and Faugeras, 1992] Vaillant, R. and Faugeras, O. (1992). Using extremal boundaries for 3d object modelling. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14(2):157–173. 2.1

[Vese and Chan, 2002] Vese, L. and Chan, T. (2002). A multiphase level set framework for image segmentation using the mumford and shah model. *International Journal of Computer Vision*, 50(3):271–293. A.3, A.3, A.3, A.10, A.11, A.12, A.13

[Wong and Cipolla, 2001] Wong, K.-Y. K. and Cipolla, R. (2001). Structure and motion from silhouettes. In *8th IEEE International Conference on Computer Vision*, volume II, pages 217–222, Vancouver, Canada. (document), 1.2, 1.4, 1.8, 1.10

[Wong et al., 2003] Wong, K.-Y. K., Mendonça, P. R. S., and Cipolla, R. (2003). Camera calibration from surfaces of revolution. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(2):147–161. 1.2, 1.8.2

[Wright, 1996] Wright, M. H. (1996). Direct search methods: once scorned, now respectable. *Numerical Analysis 1995. Papers from the Sixteenth Dundee Biennial Conference held at the University of Dundee, Dundee, June 27-30, 1995 (Ed. D. F. Griffiths and G. A. Watson)*, pages 191–208. 1.7.1

[Xu, 2000] Xu, C. (2000). *Deformable Models with Application to Human Cerebral Cortex Reconstruction From Magnetic Resonance Images*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA. 2.6

[Xu and Prince, 1998] Xu, C. and Prince, J. L. (1998). Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, pages 359–369. 1.8.3, 2.4.4

[Yezzi et al., 2002] Yezzi, A., Slabaugh, G., Cipolla, R., and Schafer, R. (2002). A surface evolution approach of probabilistic space carving. In *3DPVT '02*, pages 618–621. 2.1

[Yezzi and Soatto, 2001] Yezzi, A. and Soatto, S. (2001). Stereoscopic segmentation. In *Proc. Intl. Conf. on Computer Vision*, pages 59–66. 2.1

[Zhang and Seitz, 2001] Zhang, L. and Seitz, S. M. (2001). Image-based multiresolution shape recovery by surface deformation. In *Proc. of SPIE: Videometrics and Optical Methods for 3D Shape Measurement*, pages 51–61. 2.1

[Zhao et al., 1996] Zhao, H. K., Chan, T., Merriman, B., and Osher, S. (1996). A variational level set aproach to multiphase motion. *Journal of Computational Physics*, 127:179–195. A.3