A computational framework for observing and understanding the interaction of humans with objects of their environment

Nikolaos Kyriazis
June 2014

University of Crete
Computer Science Department

Thesis submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Doctoral Thesis Committee:
Professor Antonis Argyros, University of Crete (Advisor)
Associate Professor Ioannis Tsamardinos, University of Crete
Dr. Manolis Lourakis, Principal Researcher, ICS-FORTH
Professor Panos Trahanias, University of Crete
Assistant Professor George Papagiannakis, University of Crete
Professor Kostas Daniilidis, University of Pennsylvania
Professor Michael Beetz, Universität Bremen

The work reported in this thesis has been conducted at the Computational Vision and Robotics (CVRL) laboratory of the Institute of Computer Science (ICS) of the Foundation for Research and Technology–Hellas (FORTH) and has been financially supported by a FORTH-ICS scholarship, including funding by the European Commission through projects GRASP (FP7-215821), RoboHow.cog (FP7-288533) and WEARHAP (FP7-ICT-2011-9).

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

A computational framework for observing and understanding the interaction of humans with objects of their environment

Dissertation submitted by
Nikolaos Kyriazis

in partial fulfilment of the requirements for the
Doctor of Philosophy Degree in Computer Science

Author: _____
Nikolaos Kyriazis

Examination committee: _____
Antonis Argyros, Professor, University of Crete

_____
Ioannis Tsamardinos, Associate Professor, University of Crete

_____
Manolis Lourakis, Principal Researcher, ICS-FORTH

_____
Panos Trahanias, Professor, University of Crete

_____
George Papagiannakis, Assistant Professor, University of Crete

_____
Kostas Daniilidis, Professor, University of Pennsylvania

_____
Michael Beetz, Professor, Universität Bremen

Department approval: _____
Panos Trahanias, Professor, Chairman of the Department

Heraklion, June 2014

Acknowledgements

Coincidence has favored me by having situated me among people with great surplus of value and generosity. From these people I have earned a lot and, to them, I owe the greatest part of what I am and what I have achieved.

I am most grateful for the support, guidance and patience of Prof. Antonis Argyros, who has been more than a supervisor for me. Among other traits, his honesty, integrity and thoroughness have had a great impact on me and our work. The fact that Prof. Argyros has always provided me with opportunities to do research on provably interesting topics advocates his vision in the research field and his open mindedness.

I thank Prof. Panos Trahanias for all of his support during my stay at the Computational Vision and Robotics Laboratory (CVRL) which he directs, at the Institute of Computer Science (ICS), Foundation for Research and Technology – Hellas (FORTH). Prof. Trahanias was the first to introduce me to 3D space and robotics during my undergraduate studies and has been most positive and an excellent host since then. Through him, since he has been the Chairman of the Computer Science Department (CSD) during all my studies, I would also like to thank the CSD and the University of Crete (UOC) for providing me with a chance in truly high level education.

I truly appreciate the collaboration with the members of my advisory committee, Dr. Manolis Lourakis, Principal Researcher at ICS-FORTH and Ioannis Tsamardinos, Associate Professor at the University of Crete. I always looked forward to their constructive criticism and have always enjoyed their insight and directness.

I owe great thanks to the rest of the members of my examination committee, who have found enough interest in my work to support the examination process in the most positive way. Thank you Assistant Professor George Papagiannakis, Professor Kostas Daniilidis and Professor Michael Beetz.

My research has been supported, financially and substantially, by the Foundation for Research and Technology – Hellas and the Institute of Computer Science, and the EU integrated projects GRASP, RoboHow.cog and WEARHAP. Big thanks to FORTH-ICS and the numerous collaborators in these projects.

In CVRL-ICS-FORTH I had the chance to collaborate with a series of smart and interesting people, most of which I am also glad to call friends. It is also thanks to friends outside work that I have endured, in spite of difficulties. I sincerely thank each and every one of them. From all the above I want to especially mention Iason Oikonomidis, with whom I have been collaborating closely for many years and have been a friend for even more time.

My driving force has always originated from my family. I work towards opportunities, like this one, so that I may thank them publicly and in a celebratory fashion.

Last but definitely not least, I thank Alexandra, my Alexandra. Understandably, stressful periods can make a person hard to live with. A few lines in an acknowledgements page are far from enough for expressing gratitude, especially given that, indeed, she has done so much more for me.

I thank you all deeply and sincerely.

Abstract

We focus on the problem of vision-based scene understanding, *i.e*. "lifting" a scene which is observed by visual means across time, to a symbolic representation that can be processed by a computational system. We are interested in dynamic indoor scenes, in which humans purposefully interact with their environment. We observe that existing approaches have been performing scene understanding mainly through coarse modelling of the observed processes, as more detailed modelling is very demanding in terms of computational resources and exhibits difficulties with respect to the required integration of computer vision methods.

We suggest that currently, it is indeed feasible to incorporate detailed scene modelling, which can be easily integrated with computer vision techniques and can efficiently cope with the associated computational requirements. With respect to scene understanding, we are in position to model and simulate the process of image acquisition through 3D rendering (appearance), and the dynamics of the observed processes through physics simulation (behavior). Thus, we identify 3D rendering and physics simulation as two significant processes towards scene understanding. We propose the combination of the simulation power of these tools with powerful optimization methods, in order to yield powerful inference tools towards scene understanding.

More specifically, we consider the process of scene understanding as an optimization problem. We design parametric models that describe what can take place in a dynamic scene and how this can be observed by visual means. We define these parameters to constitute the domain of the optimization problem. Optimization is decoupled from modelling and is performed in a hypothesize-and-test framework which is implemented based on black box optimization techniques. The outcome of the optimization is the instance of the parametric models which best "explain" the observations. Ultimately, in the context of this work, the tested hypotheses are in agreement with laws of physics as they originate from physics simulators. For every hypothesis, its compatibility with actual observations of the scene is evaluated through 3D rendering. Thus, our proposal focuses on three points: (a) forward modelling of the scene, (b) incorporation of physics simulation and (c) exploitation of black-box optimization methods.

We have developed a computational framework which, based on the above, performs aspects of 3D scene understanding. We present this framework and its application to the problems of 3D tracking and motion estimation. We emphasize the necessity for the incorporation of physics. More specifically, we show that by acknowledging that visual observations regard physical phenomena governed by laws of physics, we can even apply inference on initially "hidden" parameters. More specifically, we can estimate parameters that prior to incorporating physics were not directly observable, and which can be recovered only by attributing observations to side-effects of physical processes.

The proposed computational framework has been employed to solve problems that vary from tracking a single object to tracking two hands while interacting with many objects, in 3D and from different visual modalities and camera arrangements. Through a series of experiments we show how important it is to incorporate computer graphics and physics processes in 3D scene understanding. These processes were successfully used as black box simulation tools and their inherent complexity has not hindered the integration with computer vision processes, thanks to the design choice of employing black-box optimization. We were also able to show that the proposed framework exhibits a favorable scalability profile when applied to domains of increasing complexity. Through careful design, the invocation of otherwise expensive simulations can be performed so efficiently

that interactive processing frame rates are achieved. All the above advocate a modular computational solution to 3D scene understanding problems with a clear potential for improvement or generalization: substituting parts with better or more general modules automatically improves the entire framework.

Περίληψη

Εστιάζουμε στο πρόβλημα της κατανόησης μιας δυναμικής σκηνής με βάση οπτική πληροφορία, δηλαδή στον μετασχηματισμό μιας τέτοιας σκηνής σε μια συμβολική αναπαράσταση, την οποία να μπορεί να επεξεργαστεί ένα υπολογιστικό σύστημα. Ενδιαφερόμαστε για σκηνές εσωτερικού χώρου, στις οποίες ένας άνθρωπος αλληλεπιδρά σκόπιμα με το περιβάλλον.

Παρατηρούμε ότι οι έως τώρα σχετικές προσεγγίσεις πραγματοποιούσαν κατανόηση σκηνής μέσω κυρίως χονδρικής μοντελοποίησης της παρατηρούμενης διαδικασίας, καθώς λεπτομερέστερες μοντελοποιήσεις είναι πολύ απαιτητικές σε υπολογιστικούς πόρους και δυσκολεύουν την απαιτούμενη ενοποίηση με υπολογιστικές μεθόδους όρασης.

Υποστηρίζουμε πως αυτήν τη στιγμή είναι όντως δυνατό να εκμεταλλευτούμε λεπτομερείς μοντελοποιήσεις, που να ενοποιούνται εύκολα με υπολογιστικές τεχνικές όρασης και να ανταπεξέρχονται στις σχετικές υπολογιστικές απαιτήσεις. Σε ότι αφορά την κατανόηση σκηνής, είμαστε σε θέση να μοντελοποιήσουμε και να προσομοιώσουμε τόσο την διαδικασία ανάκτησης εικόνων μέσω 3D rendering (παρουσιαστικό), όσο και την δυναμική των παρατηρούμενων διεργασιών μέσω προσομοίωσης φυσικής (συμπεριφορά). Έτσι, προσδιορίζουμε το 3D rendering και την προσομοίωση φυσικής σαν δύο σημαντικές διεργασίες για την κατανόηση σκηνής και προτείνουμε τον συνδυασμό της προσομοιωτικής δυνατότητας σχετικών υπολογιστικών μεθόδων με ισχυρές μεθόδους βελτιστοποίησης προς την ανάδειξη αποτελεσματικών εργαλείων συμπερασμού.

Ειδικότερα, θεωρούμε την διαδικασία κατανόησης μιας δυναμικής σκηνής ως ένα πρόβλημα βελτιστοποίησης. Σχεδιάζουμε παραμετρικά μοντέλα που περιγράφουν το τι είναι δυνατόν να διαδραματιστεί σε μια σκηνή και πως αυτό μπορεί να παρατηρηθεί από τα διαθέσιμα οπτικά μέσα. Ορίζουμε σαν πεδίο ορισμού του προβλήματος βελτιστοποίησης τις παραμέτρους καθαυτές. Η βελτιστοποίηση πραγματοποιείται σε ξεχωριστή διαδικασία από αυτή της μοντελοποίησης, με υπόθεση-και-δοκιμή, μέσω μεθόδων βελτιστοποίησης black-box. Το αποτέλεσμα της βελτιστοποίησης είναι εκείνη η παραμετροποίηση των μοντέλων που «εξηγούν» με βέλτιστο τρόπο τις παρατηρήσεις. Οι υποθέσεις που δοκιμάζονται είναι σε συμφωνία με κανόνες φυσικής αφού πηγάζουν από προσομοιωτές φυσικής. Για κάθε υπόθεση αποτιμάται η συμβατότητά της με πραγματικές παρατηρήσεις μέσω 3D rendering. Έτσι, η πρότασή μας εστιάζει σε τρία σημεία: (α) μοντελοποίηση σκηνής, (β) ενσωμάτωση προσομοίωσης φυσικής και (γ) εκμετάλλευση των μεθόδων βελτιστοποίησης black-box.

Έχουμε αναπτύξει ένα υπολογιστικό πλαίσιο που βασίζεται στα παραπάνω για να επιτυγχάνει επίλυση επιμέρους προβλημάτων κατανόησης μιας τρισδιάστατης σκηνής. Παρουσιάζουμε αυτό το πλαίσιο και τις εφαρμογές του σε προβλήματα τρισδιάστατης παρακολούθησης και εκτίμησης κίνησης σε σκηνές εσωτερικού χώρου. Δίνουμε έμφαση στην αναγκαιότητα για ένταξη φυσικής. Πιο ειδικά, δείχνουμε ότι με το να αναγνωρίζουμε ότι οι οπτικές παρατηρήσεις αφορούν φυσικά φαινόμενα που εξηγούνται από κανόνες φυσικής, μπορούμε να εφαρμόσουμε συμπερασμό ακόμα και σε αρχικά «κρυφές» παραμέτρους. Επομένως, μπορούμε να εφαρμόσουμε λογισμό σε παραμέτρους που πριν την ένταξη φυσικής δεν ήταν άμεσα παρατηρήσιμες και τις οποίες μπορούμε να ανακτήσουμε μόνο με τη θεώρηση φυσικών φαινομένων και των συνεπειών τους.

Το προτεινόμενο υπολογιστικό πλαίσιο έχει χρησιμοποιηθεί για τη λύση προβλημάτων που ποικίλουν από την παρακολούθηση ενός αντικειμένου έως την παρακολούθηση δύο χεριών καθώς αυτά αλληλεπιδρούν με πολλά αντικείμενα, στις τρεις διαστάσεις και με βάση παρατηρήσεις που προέρχονται από διάφορα οπτικά μέσα. Μέσα από μια σειρά πειραμάτων δείχνουμε τη σημασία της ενσωμάτωσης γραφικών υπολογιστών και προσομοί-

ωσης φυσικής στην τρισδιάστατη κατανόηση σκηνής. Οι ανωτέρω διαδικασίες χρησιμοποιήθηκαν επιτυχώς σαν προσομοιωτές black-box, χωρίς η εγγενής πολυπλοκότητά τους να εμποδίσει την ενοποίηση με υπολογιστικές μεθόδους όρασης, χάρη στη σχεδιαστική επιλογή της εμπλοκής μεθόδων βελτιστοποίησης black-box. Δείχνουμε επίσης ότι το προτεινόμενο πλαίσιο επιδεικνύει καλά χαρακτηριστικά ως προς την κλιμακώσιμη αντιμετώπιση προβλημάτων μεγάλης πολυπλοκότητας. Μέσω προσεκτικής σχεδίασης, η επίκληση έως τώρα υπολογιστικά ακριβών προσομοιώσεων μπορεί να επιτευχθεί τόσο αποδοτικά ώστε να επιτυγχάνεται επεξεργασία σε γρήγορους ρυθμούς. Τα παραπάνω συνηγορούν υπέρ μιας αρθρωτής υπολογιστικής λύσης σε προβλήματα τρισδιάστατης παρακολούθησης σκηνής, με ξεκάθαρη δυνατότητα για βελτίωση ή γενίκευση: αντικαθιστώντας μέρη με καλύτερες ή γενικότερες υλοποιήσεις βελτιώνεται αυτόματα το σύνολο.

Contents

List of Figures **13**

1 Introduction **23**
1.1 Motivation . . . 23
1.2 Problem specification . . . 25
1.3 Value of solving the problem . . . 26
1.4 Problem difficulty . . . 27
1.5 Our approach . . . 28

2 Contribution **29**
2.1 Methodological contributions . . . 29
2.1.1 Decoupling modelling and optimization . . . 30
2.1.2 Appearance model . . . 32
2.1.3 Behaviour model . . . 33
2.1.4 A joint appearance-behaviour model . . . 33
2.2 Technical contributions . . . 34
2.2.1 Computational framework . . . 34
2.2.2 Acceleration . . . 34
2.2.3 Software availability . . . 34

3 Foundation **37**
3.1 Basics . . . 37
3.1.1 Entities . . . 37
3.1.2 Interaction . . . 38
3.1.3 Forward models . . . 38
3.2 Parametric model based 3D understanding . . . 38
3.3 Optimization . . . 39
3.3.1 Black box optimization . . . 39
3.3.2 Tracking . . . 41
3.3.3 Model complexity . . . 42
3.4 Models . . . 42
3.4.1 Appearance/Rendering . . . 42
3.4.2 Behaviour/Physics . . . 44
3.4.3 Conjoint modelling . . . 46

4 Computational framework **47**
4.1 Preliminaries . . . 47
4.1.1 Transforms . . . 49
4.1.2 Geometry . . . 50

4.1.3 Entities . . . 51
4.2 The loop . . . 52
4.3 CPU Single Core components . . . 53
4.3.1 Problem specification . . . 53
4.3.2 Parallel Search . . . 54
4.4 CPU Multi Core components . . . 55
4.4.1 Decoding . . . 55
4.4.2 Physics Simulation . . . 58
4.4.3 Prior evaluation . . . 59
4.5 CPU Asynchronous components . . . 60
4.5.1 Observation . . . 60
4.5.2 Feature mapping . . . 60
4.6 GPU components . . . 63
4.6.1 Tiled rendering . . . 64
4.6.2 Feature mapping . . . 68
4.6.3 Appearance differentiation . . . 71
4.6.4 Reduction . . . 73

5 Tracking the parts 75
5.1 Single hand 3D tracking . . . 75
5.1.1 The problem . . . 76
5.1.2 The solution . . . 76
5.1.3 Experiments . . . 79
5.1.4 Literature review and contribution . . . 80
5.2 Physically plausible motion estimation of objects . . . 84
5.2.1 The problem . . . 85
5.2.2 The solution . . . 85
5.2.3 Experiments . . . 88
5.2.4 Literature review and contribution . . . 91

6 Tracking the whole 93
6.1 Tracking two interacting entities . . . 93
6.1.1 The problem . . . 94
6.1.2 The solution . . . 94
6.1.3 Experiments . . . 97
6.1.4 Literature review and contribution . . . 97
6.2 Tracking multiple interacting entities . . . 100
6.2.1 The problem . . . 101
6.2.2 The solution . . . 101
6.2.3 Experiments . . . 106
6.2.4 Literature review and contribution . . . 113
6.3 Towards more compact models of interaction . . . 115
6.3.1 The problem . . . 115
6.3.2 The solution . . . 116
6.3.3 Experiments . . . 120
6.3.4 Literature review and contribution . . . 123

7 Higher level inference **127**
7.1 Understanding intention 127
7.2 Extracting manipulation alphabets 129

8 Discussion **131**
8.1 Identification of literature gap 131
8.2 Significance of the problem 131
8.3 Contribution 131
8.4 Limitations 132
8.4.1 Known models 132
8.4.2 Track loss 133
8.4.3 Initialization 133
8.4.4 The single actor hypothesis 133
8.4.5 Complicating variance 134
8.5 Epilogue 135

9 Impact **137**

References **146**

List of Figures

1.1 An image captured from Crysis 3©, a popular computer game. This game includes realistic graphics, as it can be seen in this gameplay footage, but also realistic behaviour, powered by physics simulation. Mores specifically, the environment is destructible and parts move while respecting the laws of the physics engine which powers the game. The whole experience is delivered at high frame rates. 24

1.2 A synthetic scene. What is even more striking than the realistic rendering is the realistic behaviour of the scene. A thin structure, like the fork, interacts with a complex compound of noodles, twisting them, picking them and then dropping them. The outcome can barely be identified as being synthetic. The entire video can be viewed at `http://www.youtube.com/watch?v=oyjE5L4-1lQ` 25

1.3 Cameras in our lives. Cameras are everywhere, as they have been established as a minimally intrusive way of monitoring and understanding. They appear in (a) surveillance, (b) quality assurance, (c) medicine and education or even for leveraging (d) every-day entertainment. 26

4.1 The information flow diagram of the proposed computational framework. All components are detailed in chapter 4. 48

4.2 The viewing frustum, defined in the context of Computer Graphics. It essentially defines a clipping volume, outside which geometry is not visible. 50

4.3 This closed cube is made of 8 vertices and 12 triangles. Each vertex is shared by 4 triangles. Each triangle is defined as a triplet of vertex indices. For example, the two front facing triangles are $[1, 4, 5]$ and $[1, 5, 2]$, which corresponds to the triangle list representation of 3D triangular meshes. . . 51

4.4 Entity 3D models. Entities can be rigid (a), or articulated, *i.e*. compounds of interconnected rigid objects. The hand (d) is an articulated entity that is composed of appropriately transformed spheres (b) and cylinders (c). . 52

4.5 The decoding of a 3D synthetic hand. A hand model comprises several spheres and cylinders, that are appropriately transformed so as to resemble a hand structure. The corresponding decoding is a mapping of 3D mesh references (sphere and cylinder) to 4×4 3D homogeneous matrices (transforms). The hand depicted is composed of 15 cylinders and 22 spheres. Each 4×4 matrix is associated with an id to indicate which hypothesis it regards. In this figure only one hypothesis is decoded, so all indices are set to 0. In the general case, a decoding regards multiple hypotheses and it then acts as a batch. 56

4.6 An example of combination decoding. Decoding an original 54-D hypothesis that regards two hands can be broken down to decoding two 27-D hand hypotheses separately. The partial results can be trivially combined into a merged decoding. 57

4.7 Extraction of 3D information from camera setups. Multiple RGB images from different views (a) can be used to extract implicit or explicit 3D information. On the other hand, specialized and more compact sensors, like the Kinect, directly provide both color (b) and partial 3D structure (c), for one view at a time. 61

4.8 Adaptive Gaussian mixture models for background subtraction [118]. A background is learnt (a) and then pixels of new frames (b) are classified upon whether they belong to the background or the foreground (c). 62

4.9 Skin color detection [4]. A simple skin color model is learnt in training and then used on images to detect skin coloured regions. 62

4.10 3D box classification. A 3D box is defined (a) and only pixels that correspond to 3D points that are contained in this box are regarded as foreground (b). 63

4.11 Edge detection. Boundaries of objects often concur with intensity discontinuities, or (b) edges. Distance transforms (c) are used to accelerate edge distance queries on images. 63

4.12 Tiled rendering. The same geometry is multiply rendered, under different transforms or camera positions and on the same image but in different tiles. Geometry is contained in each tile (as in (a)), *i.e*. it does not *leak* into neighbouring tiles (as in (b)). This rendering is efficiently performed through *geometry instancing* and *multi-viewport clipping*. The separating white lines are not originally included, but are explicitly superimposed to convey the implicit boundaries of multiple tiles. 64

4.13 Deferred shading of 2 meshes into 4 tiles, with 2 instances in each. Two 3D triangular meshes ((a) and (b)), that come with per vertex normals and texture coordinates, are rendered into maps of primitives. In (c), the three distinct colors of the issue id map indicate that two meshes where used in the tiled rendering and that the entire image was not covered by geometry. In (d), the five distinct colors indicate that four instances of each of the two meshes was rendered (plus one color for pixels with no geometry). The 3D position that corresponds to every pixel is color coded in the position map of (e). Accordingly, the orientation of each pixel is stored in the normal map (f). Finally, the per vertex texture coordinates are rasterized into a single texture map (g). 66

4.14 Composition of tiled rendering from maps of primitives (see fig. 4.13). A map of unique colors is created from the combination of issue and instance ids. Lighting is computed based on normals. Using a texture file and the texture coordinates the final texture for every pixel is computed. Finally, intermediate results are multiplied to yield the final rendering. All processes are pixel-wise and can be performed very efficiently on a GPU, using the CUDA framework. The independency in pixel computations relieves of the consideration of tiles. 67

4.15 Deferred shading results for a given pose of a synthetic 3D model of a human hand. The model image (a) has actually been generated through composition of maps (b), (c) and (e). 69
4.16 By keeping pixels with non-zero issue id we identify the foreground silhouettes in renderings. 69
4.17 A discontinuity intensity map (b) can be built from a normal map (a). The former is then easy to threshold in order to identify edges (c). 70
4.18 By keeping the z component of the position map (a) one directly gets a depth map (b). 70
4.19 Silhouettes extracted from observations (a) are replicated in order to perform per pixel comparison with 4 hand configuration hypotheses (b). The logical conjunction (white) and disjunction (white and gray) are depicted in (c). 71
4.20 Through rendering, edges that are comparable to observed edges (a) can be produced (b). The distance transform is computed for the observed edges (a) and the result is tiled so as to match the dimension of the hypotheses rendering (c). By pixel-wise multiplication of (b) with (c) one can get the nearest edge distance for each of the edge pixels for the hypotheses (d). . 72
4.21 Depths measured from sensors (a) and generated for hypotheses (b) are differentiated for every pixel (c). 72

5.1 *Multi-camera 3D hand tracking*: Single hand tracking, in 3D, by exploiting the implicit 3D information provided by a multi-camera system [61]. All views are regarded disjointly, as the back-projection error amounts to a sum of multiple back-projection errors, one for each view. The optimal solution is superimposed, in blue edges, on the original input. 76
5.2 *Visual-hull-based 3D hand tracking*: Single hand tracking, in 3D, by explicitly exploiting the 3D information provided by a multi-camera system [70]. In a direct extension of [61], all views are regarded jointly, as they are used to estimate a 3D volume that corresponds to the visual hull of hand silhouettes. Hypotheses are also mapped to the feature space of volumes, where direct comparison with observations becomes available. . 77
5.3 *RGBD 3D hand tracking:* Tracking a single hand from RGBD input (Kinect) [63]. Every RGB frame is associated with explicit 3D information (calibration and depth). The tracking solutions are superimposed on the original inputs. 77
5.4 The instantiation of the outline in fig. 4.1 for the problem of tracking a single hand in 3D. Edge colors denote information flow which regards distinct implementations, over different scenarios, as presented in works *multi-camera 3D hand tracking* [61], *RGBD 3D hand tracking* [63] and *visual-hull-based 3D hand tracking* [70]. This figure can be read by selecting an edge color and following the corresponding flow (see legend). . 78
5.5 Exemplar results of *multi-camera 3D hand tracking* [61]. Each column corresponds to a different view/camera. 80
5.6 Exemplar results of *visual-hull-based 3D hand tracking* [70]. Each pair of images illustrates the same pose from different views. 81

5.7 Exemplar results of *visual-hull-based 3D hand tracking* [70]. The superiority of [70] over [61], in the case of short baseline stereo is demonstrated. Left image pairs correspond to [70] and right image pairs to [61]. Each pair corresponds to a stereo view. 82
5.8 A ball is thrown towards a table with a high back-spin. By incorporating physics-based simulation, we infer the ball's 3D trajectory (a), and its linear and angular velocities *from a single camera* (cam. 7). The proposed method identifies that a back-spin is the cause of the reduction of the outgoing angle of the bounce. The green ellipses in (b) are projections of an equator of the ball and the arrows represent the direction of the estimated angular speed. 84
5.9 Information flow diagram of [52]. This is an instantiation of the framework depicted in fig. 4.1. 86
5.10 The two phases of the bouncing ball (a) flight and (b) bounce. Red arrows represent impulses and blue arrows represent velocities. Angular velocities are perpendicular to the image plane. Black arrows represent the air flow with respect to the flying ball. 87
5.11 The mean values and standard deviations for the errors on the experiments with (a) synthetic and (b) real observations. 89
5.12 Examples of actual and estimated 3D trajectories. The illustrated trajectories have high left, low left and high right curvature, respectively. The respective average trajectory point estimation errors were $1.38cm$, $0.75cm$ and $0.83cm$. 89
5.13 Estimation of the 3D trajectory of a ball from single camera 2D observations (camera 3) and the assumption of a given physical world. 90
5.14 Single view estimation of the 3D trajectory of a ball from partial 2D observations (camera 6) and the assumption of a given physical world. . . . 91

6.1 Top row, and bottom left: Three views of a hand grasping an object. Skin regions appear in red and edges in black. The hand is partially occluded by the object in all views. The incomplete skin and edge maps of the hand facilitate the generation of a hypothesis for a hand manipulating a compact sphere. At the same time, given this hypothesis, the 3D pose of the hand can be estimated more accurately. Bottom right: the output of the proposed approach superimposed in one of the frames. 94
6.2 Left: A view of two interacting hands. Right: The configuration of the two hands as estimated by the proposed method, superimposed in the left frame. 95
6.3 The instantiation of the generic framework in fig. 4.1 for the purposes of hand-object tracking [65] and two-hand tracking [73]. 96
6.4 Sample frames from the results obtained by [65] in real-world experiments. The results are of such fidelity so that they succeed in visually conveying the tight physical relation between the hand and the object. . . 98
6.5 Snapshots from [67], tracking two hands which interact with each other. Notice that tracking is successful even for extreme cases, like the one in the bottom-left image. It is also noteworthy that these results have been produced without accounting for collisions between the two hands. Only adjacent finger interpenetration is penalized, for each hand independently. 99

6.6 3D tracking of multiple entities. We compare three variants for tackling the tracking problem, to highlight the requirement for a new tracking scheme, *i.e. collaborative tracking*. 100

6.7 Instantiation of the generic framework of fig. 4.1 for the case of joint tracking (JT). In this abstracted information flow diagram the domain of the problem is depicted relatively enlarged to convey that the same principle of 3D tracking may be applied for arbitrarily big problems. However, in practice, it takes moderately sized domains to render this approach ineffective. 102

6.8 The instantiation of the framework depicted in fig. 4.1 for the case of a Set of Independent Trackers (SIT). In this diagram the case of 3 independent trackers is illustrated. The big problem is decomposed into smaller independent ones, in a lossy fashion. The trackers have no means of communication, which introduces problems, pertaining mainly to ambiguities, due to the lack of a mutual exclusiveness in data association, in scenes that are dominated by interaction among the tracked entities. 103

6.9 The instantiation of the framework depicted in fig. 4.1 for the case of an Ensemble of Collaborative Trackers (ECT). In this diagram the case of 3 collaborative trackers is illustrated. The big problem is decomposed into smaller ones. In contrast to SIT and fig. 6.8, the collaborative trackers communicate by sharing their current estimation of the world state to the rest of the trackers. Thus, the decomposition preserves the nice trait of being scalable, but is also brought closer to the spirit of joint tracking. Experimental results actually show that this approach is not only practical in cases where JT is not, but also performs better, overall, compared to JT, in terms of throughput, accuracy or both. 105

6.10 Frames of the two-hands tracking synthetic dataset used here and in [69] for quantitative analysis. 107

6.11 Quantitative results for the two hands tracking regarding (a) accuracy, (b) tracking throughput and (c) average tracking performance across time. . . 108

6.12 Frames from the rotating bottles synthetic dataset. 109

6.13 Quantitative results for the rotating bottles dataset. (a)-(c) tracking error and (d) tracking throughput. 110

6.14 Tracking two interacting hands. The results of (a) SIT , (b) JT and (c) ECT, are compared for 50 particles and 50 generations per hand. SIT has no change of correctly interpreting a scene that is dominated by interaction. Results for JT and ECT are similar, with the results of ECT being slightly more accurate but also faster than JT. 111

6.15 Representative results obtained from the proposed ECT approach on the toy disassembly dataset. It is redundant to show the corresponding results for the other variants, as they are far from being even remotely plausible for this high-dimensional problem. 112

6.16 The exploitation of the *single actor hypothesis* through *physics modelling*, allows physically plausible, heuristic-free 3D tracking of hand-object interactions. (a) RGBD observation of a hand interacting with objects. (b), (c) By searching for hand motion only, we are able to track the 3D state of the entire scene. The state can be *overt* (partially visible hand and objects (b)) or even *covert* (totally occluded objects like the ball-inside-the-cup (c)). 116

6.17 An adjustment of the information flow presented in fig. 5.4 for the case of tracking a hand and multiple entities in a reduced search space. Notice the additions of modules, highlighted in red. The search space regards only the hand, but the entire scene is tracked, by applying the consequences of the hand motion to the simulated state of the rest of the scene. Each new hypothesis is combined with the current simulation state. The simulation state is updated once the optimal hand motion has been recovered. 117

6.18 The physical entities that are considered in our framework. The hand model (a) comprises 22 ellipses and 15 cylinders, appropriately positioned, rotated and scaled. The collision spheres (green) inside the hand model give it physical substance. We consider a variety of object specifications such as a ball and and box (analytical expressions) a cup (designed and then printed) and a bottle (3D scanned). 118

6.19 (a), (b) RGBD input. (d) Masked depth image I_d^o. For a hand motion hypothesis h (c), a synthetic depth map I_d^r is rendered (e). The difference between I_d^o and I_d^r (f) yields the fitness of h. The best scoring h, computed by PSO, is the tracking solution (g) for the current frame. 120

6.20 Quantitative results. Distinct curves correspond to different particle counts. Logarithmic scale was used in the vertical axes for better resolution over small value differences. 121

6.21 Tracking results, superimposed over the respective RGB input. Leftmost column: the sequence that was used for quantitative assessment. Rest columns: qualitative evaluation. Detailed presentation of the entire sequences that highlight the efficacy of the proposed method can be found in the supplementary material. 122

7.1 (a) Human demonstrates a grasp with an intention to *hand-over an apple*. (b) Robot imitates the power grasp configuration used by the human, and fails to hand-over because there is not enough free space for regrasp. (c) Robot estimates that human intends to hand-over the apple. It also learns the task requires leaving enough free space on the object. It applies a precision grasp to achieve the task. This figure has been borrowed from [92]. 128

7.2 The vision system of [92]. A database of known objects is pre-established. The objects in a scene that correspond to entries in this database are automatically extracted from the first observation frame. This information is enough to bootstrap hand-object(s) tracking. The channels of information of each frame that are used in each phase are outlined with green borders. For more details please refer to [92], the original source of this figure. . . 128

7.3 Task of *Pouring* water from mug subdivided into action primitives. Each image depicts the output of hand-object tracking algorithm. For more details please refer to [77], *i.e*. the origin of this figure. 129
7.4 Objects, which belong to different classes, according to the type of their eveyday use, have been considered for the problem of extracting manipulation alphabets. Several videos showing these objects being involved in everyday tasks have been recorded and processed. 130

Chapter 1

Introduction

Computer vision is concerned with the understanding of the physical world through the analysis of its image(s). Such an understanding may be defined at various levels of abstraction. Whatever the level of abstraction may be, this understanding is always associated with a context, *i.e*. an assumption of a generative process that produces the observations. It is convenient to think about such contexts as sets of rules that transform some initial conditions into images.

In this thesis we are concerned with understanding dynamic 3D scenes in which humans interact with their environment. To achieve the desired understanding, we identify an appropriate set of rules that govern such scenes and turn them into inference mechanisms. We do this systematically through a framework which combines effectiveness and efficiency. We show that this framework, which constitutes the main contribution of our work, is capable of tackling a series of computer vision problems related to the understanding of non-trivial interaction of humans with their environment.

1.1 Motivation

Understanding images can be achieved by identifying how they were generated. Asking how may receive an answer whose detail can vary, according to the required level of understanding. For example, trying to understand whether an image is taken from an indoors or outdoors scene is quite abstract, and amounts to discerning whether the generation of the image follows the abstract rules governing the setup of of indoors or outdoors scenes (*e.g*. furniture usually denotes indoors and vegetation usually denotes outdoors). In computer vision this question is usually answered by comparing pixel intensity statistics for indoors or outdoors scenes [98]. Other methods go even deeper by trying to identify specific structure that clarifies the answer [20, 80]. For example, besides answering *indoors or outdoors* they also provide a more detailed explanation, like *it is a room because walls that form a box have been detected*.

The most important problems in the process of image understanding are (a) identifying the domain of rules that are to be inferred but also to (b) define the means of discovering the instantiations of these rules from observations. Trying to tackle both fronts jointly can be a cumbersome task. Fortunately, we observe that these problems can be addressed disjointly.

On the front of identifying the domain of rules, a general remark might be that we are in position to account for the process of image generation, because we can generate synthetic images or sequences of images that are hard to distinguish from their real coun-

Figure 1.1: An image captured from Crysis 3©, a popular computer game. This game includes realistic graphics, as it can be seen in this gameplay footage, but also realistic behaviour, powered by physics simulation. Mores specifically, the environment is destructible and parts move while respecting the laws of the physics engine which powers the game. The whole experience is delivered at high frame rates.

terparts. Otherwise stated, we seem to know how images are formed because we seem to be good at performing the same process. The most striking results can be drawn from the contemporary Computer Gaming industry, which produces games with high degree of realism in both appearance (graphics) and behaviour (physics) (see fig. 1.1). At the same time this degree of realism is served at high frame rates. Another striking example comes from digital art production, where a commodity 3D modelling software combines powerful rendering techniques and commonly available physics simulation packages to help deliver synthetic scenes of unprecedented realism (see fig. 1.2). So, evidently there are tools, methods and pieces of software that can help generate realistic images, or, for the purpose of computer vision methods, hypothetical observations. We hereby propose that such processes should constitute the domain of rules.

These processes are highly complex and highly non analytical, so their incorporation in computer vision methods seems problematic at a first glance, with respect to extracting their instantiations from observations. Nevertheless, they implement a simple interface. That is, given scene specifications regarding objects, materials, lights and camera placement, rendering pipelines can provide realistic images as an output. Physics engines require similar specifications, like shapes, material properties and initial conditions, in order to advance a scene, in time, according to physics rules.

Thanks to advances in black box optimization, *i.e*. optimization of functions that are black boxes, these simple interfaces can be used in inference, thus allowing for tackling the second front. Since black box optimizers do not need to know how a function is computed, but only require that this function maps some parameters to some values, we can incorporate these complex modules in the required optimization phases of computer vision tasks. Simply stated, in order to understand an image or video, we could try to find

Figure 1.2: A synthetic scene. What is even more striking than the realistic rendering is the realistic behaviour of the scene. A thin structure, like the fork, interacts with a complex compound of noodles, twisting them, picking them and then dropping them. The outcome can barely be identified as being synthetic. The entire video can be viewed at `http://www.youtube.com/watch?v=oyjE5L4-1lQ`

this rendering and physics simulation, by means of black box optimization, which yields a result that is most compatible to the observations. Finding or calibrating such a simulation essentially means explaining the scene.

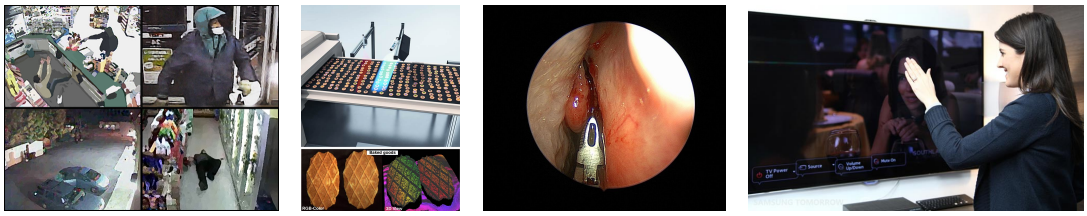
1.2 Problem specification

We consider an indoors scene which comprises a human subject and its environment. This environment might be composed of static or immovable entities, like walls or heavy objects, and dynamic or movable entities, like small objects, tools, furniture, *etc*. The subject can engage into interaction with its environment, with one or multiple objects at a time, including picking, lifting, moving, putting one on top of or inside another, *etc*.

The scene may be observed by one or multiple visual sensors, where each one generates observations in the form of images. Such images may contain color information or depth measurements. Each such sensor is assumed to be calibrated, *i.e*. its position and intrinsic properties are known, and thus each pixel can be associated with 3D space.

The task of understanding such a scene amounts to transforming the observations into a symbolic representation which describes the scene and can be processed by a computer system. The focus of understanding might be an entity or even the entire scene. The symbolic representation may regard arbitrary aspects of state, such as position, orientation, appearance, utility, *etc*.

In brief, scene understanding amounts to telling a computer system what things are, where they are and what they do. Each of these questions may be answered relatively and at different abstraction levels. For example, a cup that is on a table is also in the room containing both, or at a specific 3D point with respect to a camera reference system. A



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

Figure 1.3: Cameras in our lives. Cameras are everywhere, as they have been established as a minimally intrusive way of monitoring and understanding. They appear in (a) surveillance, (b) quality assurance, (c) medicine and education or even for leveraging (d) every-day entertainment.

pencil might act as both a writing accessory or a poking device and thus may write or poke.

Interaction is a dynamic phenomenon which evolves through time. Therefore, the corresponding observations form temporal sequences, which are used as input during scene understanding. In this context, and for the sake of efficiency, we treat the scene understanding problem as a tracking problem. More specifically, temporal continuity is assumed and the aspects of state to be inferred are being tracked, rather than being estimated anew for each new observation. The aspects of state we consider regard the 3D world, and therefore we will be referring to 3D tracking, to describe the process of extracting meaningful interpretations from image sequences.

1.3 Value of solving the problem

Understanding the interaction of humans with their environment can be greatly beneficial in a series of sectors. The list of potential applications is long and covers multiple service areas, such as safety, education, medicine, industry, *etc*.

Visual sensing is an established means of capturing important activity in a fashion that is minimally invasive (see fig. 1.3). To name a few instances, security cameras are installed in safety-critical locations in order to prevent or analyze harmful activities. Specialized visual sensors are installed in production lines to ensure the safety of the production process itself and also assert the quality of the products. Surgical operations are monitored by systems of cameras which extend the sensory capability of surgeons and whose recordings also act as instructional material. Smart appliances employ cameras and other sensors to leverage the interaction quality between them and their users.

In all the aforementioned scenarios, and in others not listed, the captured footage is usually scrutinized by trained personnel in order to draw a series of meaningful and interesting conclusions. The repetitiveness of such tasks along with the great volumes of data hinder the reliability of the results, especially as human performance wears down.

Large data and repetitive tasks are more suitable for computer processing. Lately, with the advances in computer vision technology, some of such tasks have been delegated to computer systems. This transition is slowed down by the complexity of the captured activity. Thus, the simpler of everyday activities have been what usually constitutes the domain in which such technologies function. Simplicity pertains to the ease of successfully processing visual data. Scenes of few entities, with no or little overlap, engaging

into clearly depicted forms of interaction are relatively easy to process.

However, the vast majority of activities, as seen from camera systems, are far more difficult, in the same respect. It takes a simple every-day activity, like cooking, to render most contemporary methods unsuccessful in performing understanding. The amount of involved entities, their complexity that stems from their degrees of freedom, the subtlety of interaction, the frequent overlap and occlusions among entities, the limitations of the visual sensors, *etc*. lead to relatively hard problems. The same still holds for a great series of computer vision problems which are localized indoors, or even in the more restricted scenario, on the top of a table. We argue that successfully tackling such problems and providing the means for extending solutions to other domains too is a valuable contribution to the literature and will also help derive technologies that will improve the quality of every-day living.

1.4 Problem difficulty

The difficulty of the problem stems from (a) the complexity of the entities themselves (b) the cardinality of a scene, composed of multiple entities, (c) the limitations of the sensors and (d) the subtlety of the activities that require understanding.

The interacting entities, the subject and its environment, can pose difficulties in the relevant computer vision tasks due to their complex nature. Entities, like humans, require long description lengths due to their articulated structure and large number of degrees of freedom. Tracking that many degrees of freedom from images is hard, especially when, due to the articulated nature, these entities are only partially sensed, because of severe self-occlusions (*e.g*. hand fingers occluding one another). Variability or uniformity of appearance can cause severe problems, too. For example, although clothing is usually irrelevant to scene understanding tasks, the variability it introduces in the corresponding observations generates a requirement for complex solutions. On the other hand, the uniformity in the appearance of a hand, due to the uniformity of skin color, introduces ambiguity, as *e.g*. one finger can be mistaken for another (see section 5.1 on handling human hands).

As more entities are introduced, the relevant problems become harder. Linearly increasing the number of entities requires a combinatorial increase in tracking resources, as it is also the relationships between entities which are required to be considered, for understanding to be successful. As it is shown in section 6.2, considering multiple entities as if they were independent severely undermines the quality of scene understanding to the point of failure (see our resolution in section 6.2). Moreover, as the number of entities grows, the probability of occlusions increases and even total occlusions become highly probable. This introduces ambiguities that complicate the task of scene understanding (we address this in section 6.3).

The visual sensors present limitations that make understanding harder. Natural motion can be too fast to capture from commodity sensors with limited resolutions, low acquisition rates and high shutter times, that lead to low quality images with artifacts (*e.g*. noise, blurring, see section 5.2 on how physics can complement sensing deficiency). Also, interaction between small entities, like the hand and tools or small objects, which are also very common, occurs at a scale which requires a high camera resolution to reveal at a low level (*e.g*. from pixel intensities). However, sensors which are more compact and easy to deploy are also easier to come by, but they provide resolutions that are too low to capture the subtlety of such interactions. The lack of observations may only be remedied by the incorporation of appropriate prior knowledge. The identification of such priors

that are generic enough to maintain applicability across domains is evidently a hard task. Throughout our work, we struggle with such limitations and cope with them by incorporating strong priors, which have been established in fields different than that of Computer Vision (Computer Graphics, Physics).

1.5 Our approach

In the past, the contexts described in the beginning of chapter 1 have only been considered through high level abstractions, since their detailed and explicit modelling was considered to be prohibitive from a computational point of view. Different abstractions from different researchers led to different approaches to similar problems. In order to make these abstractions computationally tractable, researchers have usually made simplifying assumptions which in turn limited the range of applicability of the corresponding methods. However, a series of observations allows us to move away from such cases, towards more detailed, accurate and fast scene understanding.

We observe that, currently, we are able to account for a variety of such contexts in high detail and high fidelity. We are able to model processes that transform initial conditions to images, so well, that we make use of them in order to generate synthetic, yet realistic sequences of images. Computer graphics and Physics provide tools that make it possible to simulate the corresponding real-world mechanisms, up to a point where realistic observations can be (re)produced.

Another observation is that we are able to sample from such elaborate models (simulation) quite quickly, thanks to the advancements in software and hardware engineering. Realistic images of complex geometry with elaborate effects of light (reflections, transparency, depth of field, etc.) can be produced at the millisecond scale. The physical interactions of large scale scenes can be computed thousands of times per second. Exploiting parallel hardware, like concurrent GPUs, for the implementation of such tasks yields a manifold speed-up. Thus, the associated computational demands are currently less of a factor.

However, an issue remains, on how to fuse such tools with computer vision techniques into solutions for computer vision tasks. Given our ability to transform some initial conditions into realistic image sequences through simulation, the question is formed: Can we invert this process? In other words, given visual observations of a real scene and the ability to simulate such realistic scenes, quickly, is it possible to recover some initial conditions that optimally reproduce the actual observations in simulation?

To that end, we follow the recent advances in black box optimization [40] and exploit powerful search heuristics to turn the forward models discussed into inference mechanisms. This thesis is then about using powerful forward models in computer vision processes as inference tools, through black box optimization, in order to tackle a series of computer vision problems that amount to understanding scenes of interaction between humans and their environment.

Chapter 2

Contribution

We are considering the problem of understanding the interaction between a human subject and the surrounding environment. More specifically, we are focusing on scenarios where interaction takes place indoors, on the top of a table, *e.g*. a kitchen or work-place scenario. Besides the apparent specificity, there are indeed several major categories of computer vision problems that fit this description. Such problems are 3D hand tracking, 3D object tracking, segmentation, occlusion handling, *etc*.

We have proposed resolving approaches under such categories, which are described in chapter 5, chapter 6 and chapter 7. In these chapters too, the corresponding problem-specific literature is reviewed.

Our approaches to tackle the considered problems, as shown later, share a common rationale. This is made explicit through the invocation of a common framework (see chapter 4). This framework introduces novelties that go beyond the context of the specific problem domains. These novelties are summarized here, so as to give a central and concise overview of our contributions, at the high level. We make a distinction between methodological and technical contributions. These contributions are related to the corresponding literature.

2.1 Methodological contributions

We are mainly concerned with tracking problems. In such problems, there is a distinction, with respect to the nature of the approach, that distinguishes methods between being discriminative and generative.

Discriminative methods make use of discriminative models, found in the machine learning literature, to capture the dependence of an unobserved variable y, *i.e*. the existence of a *desired statistic*, on observations x, *i.e*. images. In probabilistic terms, discriminative methods model the conditional probability distribution $P(y|x)$, to predict y from x [109]. The *desired statistic* usually regards the appearance characteristics (pixel intensity statistics) of the entity to be tracked. The existence query may receive a categorical answer, *e.g*. yes or no, or a numerical answer, *e.g*. a quantity that is proportional to the likelihood of existence at a given image part. Thus, such methods are usually used to classify regions of images over the existence of one or more entities to be tracked (see [107] and references therein). By repeating this process in sequence, tracking across time is achieved. In the literature, dependence is usually captured through a slow process of training with given samples (*e.g*. [8, 82, 84, 85, 115]). Generalization ability to new data is proportional to the amount of available training samples, the invariance characteristics of the selected

feature space and the generalization bias of the learning method. Classification itself is commonly fast. The primary results of such discriminative methods are usually fused, at post-processing, in a meaningful way, to yield high level results.

Generative methods make use of generative models, to capture the process which generates the expected observations. In probabilistic terms, generative methods model the joint probability distribution $P(x, y)$, over observations x and the generating process itself controlled by parameters y, simultaneously [110]. In contrast to discriminative methods, generative methods fully describe the process of observation generation, and therefore they can be readily used for arbitrary sampling over any of the involved variables. Discriminative methods are constrained in only sampling over the target variables, conditioned on observed quantities. To make the distinction clearer, a generative method can be used to fully simulate the process of observation generation, while a discriminative method may only answer whether some model instantiation is more probable than others, given some observations. While it is common to also employ learning to capture generative models, it is also possible and not uncommon to manually specify the models too. Learning requires the availability of training samples. Manual specification requires a good understanding of the process to be modelled. The employment of generative models usually leads to slow methods, whose tardiness is due to the requirement of performing multiple, usually expensive simulations. However, the results are rewardingly rich.

2.1.1 Decoupling modelling and optimization

We are most fascinated with the potential of generative methods towards providing wealthy explanations over the activities captured in images sequences. While we are interested in tracking, our results, as shown later, do not have to be constrained in just providing locations of objects. Instead, we are enthused by the possibility of also explaining *why* entities behaved like they did (see section 5.2 and section 6.3), while also inferring their 3D status.

In the literature, there are numerous tracking related works which employ generative models in order to perform information-rich tracking of entities, *e.g*. [26, 81, 94, 97] regard generative 3D hand tracking. However, these methods, as well as other generative methods, share a common drawback. Modelling is tightly coupled with optimization. More specifically, in order to transfer solutions to new domains, the required mathematical derivations would have to be performed anew, partly or entirely, if possible at all.

On the other hand, while the existence of rich models (*e.g*. computer graphics, physics simulation) increase the appeal of generative methods, the difficulty or inability to establish analytical descriptions of complex models, with desired optimization traits, such as convexity, continuity, *etc*., and without sacrificing any of the models' expressiveness, makes optimization a hard task. What has been described as a problem here finds its solution in *black box optimization*, as referred to in the optimization literature [40]. In the context of black box optimization, the generative model may have an arbitrary implementation, as long as it can be sampled at arbitrary points (usually trivial). The energy function which relates observations to samples of the model, can also be an arbitrary parametric function, which maps parameters to a single value, representing score, likelihood, similarity, compatibility, *etc*. The implementation of this mapping needs not be known for its optimum to be well approximated. By carefully sampling the function, while treating it as a black box, search heuristics are currently able to perform remarkably in optimization of "hard" objective functions [40].

In loose terms, this means that one may experiment with various, arbitrarily complex

models, in the context of a generative method, without worrying about the technicalities of, perhaps, the most complex step, namely optimization. We are interested in incorporating strong, highly non-analytical models, which regard Computer Graphics and Physics Simulation. These models come in the form of sizable software libraries, that provide their functionality through computational interfaces. By programmatically wiring parametric spaces to the input of these interfaces, and by evaluating samples of these interfaces against actual observations, we establish programmatic functions which fit the pattern of black box optimization. Thus, we employ black box optimization in our methods and move our focus away from optimization and towards modelling. Our results clearly advocate the use of black box optimization methods in computer vision tasks and consequently our design choice to thus focus on modelling.

A note on black box optimization

The objective functions we will be invoking utilize 3rd party libraries, the implementations of which may only be treated as black boxes, since there is no analytical representation of them. So we are restricted in numerical optimization. We hereby advocate our selection of evolutionary optimization techniques (see section 3.3.1) for the optimization task, on the basis of practicality. A major differentiating factor between the selected class of optimization techniques and others is the tightness of coupling optimization with modelling (definition of the objective function). For example, gradient-based optimization techniques are tightly coupled with modelling because in order to perform optimization the gradient of the objective function, which is otherwise irrelevant to modelling, needs also to be derived.

We decompose practicality in several, not entirely independent, axes, *i.e*. convergence quality, convergence speed and flexibility. Quality regards how well an optimizer approximates the optimum. Speed refers to the amount of time required in order to produce the optimization result. Flexibility considers the effort required in order to set-up optimization and the generalization ability towards different optimization domains.

Quality can only be guaranteed in convex problem formulations. Except from some inspiring exceptions, like [86], in the dominant case computer vision problems are either ill-posed (multi-modal) or their formulations include multiple local minima. In our work we too struggle with ill-posed problems with multiple local optima.

Convexity is still useful, even if it holds locally. If the region around a local optimum is convex then quality over achieving this local optimum is also guaranteed, regardless of whether specific traits of the objective function are known. In other words, black box optimization techniques too can be used effectively in this case. For global optimization to be achieved the common approach is to initialize multiple local optimizers in various regions and keep the best result. It is noteworthy that black box optimizers do not require convexity or other traits in order to achieve optimization [40].

Then, the remaining question is on how fast and how easily local optimization can be performed, without sacrificing quality. Here, we can distinguish between two basic cases, judging on whether the gradient of the objective function is known, or not. In the case where the gradient is known a few informed steps suffice to reach the local optimum. However, it is common that the computation of the gradient is more expensive than the computation of the original objective function, as multiple variables and chain rules introduce additional overhead. This is especially true for highly non-linear objective functions, like the ones we consider. If the gradient of the objective function is not known it can be numerically approximated, by sampling the objective function itself in small

intervals around the point of interest. The number of samples is usually large, which automatically renders this approach slower. In contrast, black box optimizers do not require the knowledge or estimation of the gradient, and rely solely on parsimonious sampling of the original objective function. Ultimately, both extremes (gradient-aware, black-box) require sampling, with the gradient-aware ones exchanging some (not all) sampling of the objective function for less sampling of the gradient. The computational balance is not clearly in favor of one or the other, but needs to be decided in a per-case basis. In our case there are concrete examples which show black-box optimization (*e.g*. [63]) to be significantly faster than gradient-aware approaches (*e.g*. [28]), in similar problems, where 3D rendering is incorporated as a part of the objective function computations.

With gradient estimation being disqualified due to large overheads, what remains is to compare flexibility between gradient-aware and black-box optimizers. This is by construction in favor of black-box optimizers, as in this case one needs only define an objective function, without worrying about differentiability, convexity, continuity, *etc*. Moreover, black-box optimization has also been shown to be able to solve hard multi-dimensional problems with additional complications, such as noise and multiple strong local-minima [40]. Black-box optimization methods are in general more fitting to our task, as they operate on a limited set of required assumptions. In our context, the use of black box optimization has allowed for the total disregard of the optimization problem and gave the opportunity for complete and uncompromised focus on modelling. Thus, transferring solutions from one domain to the next or experimenting with new models is no longer constrained by the practical considerations of optimization.

2.1.2 Appearance model

Computer graphics, as a process of generating images from descriptions, can be viewed as our implemented understanding over how images are formed in real life. In fact, as discussed in section 1.1, our understanding is at such a high level that we are able to generate synthetic images which can be indistinguishable from real ones.

In the tracking literature there are examples of generative approaches that use computer graphics as a forward model. Characteristic examples, which push the envelope with persistence over detail, are the work of De La Gorce *et al*. [28] on monocular 3D hand tracking and the approach by Wu *et al*. [114] on full body multi-view performance capture. In both cases, as well as in the relevant literature, tracking is formulated as an optimization problem, by coining an error function which relates problem variables to differences between 3D renderings and actual input. In both cases, optimization is performed numerically, by employing Newton method variants. This requires the analytical derivation of the objective function's Jacobian [111], which, in turn, requires the rendering function to be expressed analytically, too. The requirement for analytical representations is evidently problematic. For example, in [28], significant effort is put in performing the required differentiations of the objective function.

In our line of work, we too consider graphics as a forward model but by building upon common computer graphics practices, along with the advantages that black box optimization introduces, we establish a generic and flexible forward model, which regards the appearance of entities, as captured through visual sensors. We show how the same model is used across distinct problems without ever altering any optimization related aspect of our approaches.

2.1.3 Behaviour model

Physics simulation, as a process of computing Newtonian dynamics, can be viewed as our implemented understanding over how physical entities interact, under the Newtonian laws of motion. This understanding is at such a high level that we are able to generate physical interaction between complex entities which can "trick" the human eye into believing they are real (*e.g*. computer games, movies).

Physics or dynamics[1] constitute a strong prior over motion and interaction. This is clearly appreciated, as shown in the literature, from researchers that have considered physics at various abstraction levels. The prevalent case study of employing physics in vision is the problem of 3D human tracking [14–16, 79, 104] [17]. There are approaches that reflect physics implicitly or metaphorically [12, 13, 19, 30, 74, 88]. Also, approaches can be found which are closer to the spirit our our work, by being generative [56] or by even using physics simulation as a mental model of motion [10, 33]. We show in section 5.2 how we go beyond the state of the art with respect to incorporating strong physics priors in a rigorous, yet flexible fashion.

We believe that what has prevented the use of detailed physics models in computer vision methods is the analytical complexity in the respective optimization tasks. While equations of motion can be well represented in symbolic form, the discrete notion of shape collision is hard to symbolically represent, but also, the implications of detected collisions incommode analytical derivations even more, by introducing discontinuities. On the other hand, there are several, relatively successful efforts to model physical simulation, in software, for the purposes of computer games and movies. These efforts tackle two fronts at the same time. One front is the realism of the simulation. The other front is the confrontation of several technical problems that arise during physics simulation, like limitations in arithmetic precision, performance, scaling, *etc*.

In our line of work we consider physics directly and in detail, but without struggling with the internals. We harness the fruitful effort of physics simulation software developers and introduce the full blown physics modelling in computer vision processes. Our works described in section 5.2 and section 6.3 are the first to fully exploit the direct and unrestricted incorporation of commodity physics simulation software for improving the understanding of the interaction between entities. This is, once again, enabled through the employment of black box optimization.

2.1.4 A joint appearance-behaviour model

Appearance and behaviour are not independent, at least in the context of computer vision methods. This becomes evident through simple examples. Consider the shell game (see section 6.3), in which a bead is trapped inside one of three shells and is moved around, while all shells are shuffled. While the bead remains totally occluded under the moved shell there is no means of sensing it, visually. However, our understanding over how objects interact clearly suggest that the bead is to travel inside the shell. This is very informative and complements lack of appearance, in the context of estimating the bead's location.

The opposite direction is also interesting. Given a basic understanding of dynamics and the ability to fully and accurately observe entities while interacting, based solely on their appearance, physical aspects can be inferred. By tracking a ball in flight and fitting

[1] The terms are used interchangeably in this text.

the free fall motion on the trajectories one can predict its future motion. If these predictions are violated due to a rapid change, *e.g*. caused by bounding on an obstacle, then this already provides information over the structure of the scene. This has been demonstrated by Fossati *et al*. in [35].

Considering appearance and behaviour jointly and connecting the joint model to actual observations, essentially means that a set of observations can be explained by either models or by both, simultaneously. The fusion of these models is hard at the symbolic or analytical level, in the context of optimization. However, it is straightforward enough in other contexts. For example, this is evident in contemporary computer games, in which appearance and physics simulation are of high detail and are also performed fast, while coupled. In out approaches, to turn such simulation power into a generative model, *i.e*. to connect the simulation parameters to observations, we employ black box optimization. We are the first to introduce such an elaborate generative model and experiment with it in contexts where appearance, behaviour or both are required to explain interaction (see section 6.3).

2.2 Technical contributions

The strongest argument against generative methods regards the high requirements in resources and the consequent slow performance. We try to tackle this point exactly by exploiting contemporary parallel commodity architectures, in order to accelerate processing. Therefore, our contributions also include a technical part, which we describe here. These contributions are explained in chapter 4.

2.2.1 Computational framework

The most important technical contribution is the establishment of the proposed methodology in a single computational framework, in the form of a Software Development Kit (SDK). This SDK is governed by an architecture, *i.e*. an arrangement of interchangeable modules. The same architecture is used in order to tackle all the described distinct problems. What changes across distinct problems is the selection of the modules. Implemented like this, our generic approach introduces the flexibility to easily transfer solutions from one domain to another.

2.2.2 Acceleration

A basic focus in the design of the SDK is speed. We were interested in using this SDK to quickly perform many simulations (graphics, physics), quickly, because this is the requirement when using black box optimization methods and targeting on interactive execution rates. We are the first to introduce several acceleration techniques that have either been borrowed from other fields or have been coined exactly to address our specific requirements. We show that the exploitation of contemporary commodity technologies, from within our framework, allows the resolution of difficult problems at remarkable speeds.

2.2.3 Software availability

Making our solutions available to the community also serves the research process in whole, as researchers may need to incorporate, test, compare *etc*. our methods. In this direction,

we have made our hand tracking approaches available in the form of a live demonstration, which an end-user can download and experiment with. We have also created a library version of the same method that facilitates incorporation and direct comparison with other methods. The entire software has been built with focus on facilitating model-based 3D tracking efforts.

In a smaller scale, the software presented here, has been used by several members of the Computational Vision and Robotics Laboratory at the Institute of Computer Science of the Foundation for Research and Technology - Hellas. Notably, this employment of our computational framework has facilitated the development of computer vision applications outside the hereby presented context, too. The same holds for foreign collaborators of the same laboratory in the course of European integrated projects.

Chapter 3

Foundation

In this thesis we are focusing on scene understanding that regards interaction between humans and their environment. Following a generative approach, we first establish a model of what such scenes contain and how they evolve. After doing so, *i.e*. after identifying the rules that govern such scenes and the corresponding observations, we turn this model into an inference mechanism by means of black box optimization.

Section 3.1 describes some basic terms. These terms are used throughout the text but their meaning is easy to identify, without going through the details of section 3.1. In section 3.2 the basic scheme of our approach is compactly presented. section 3.3 shows why and how the optimization problem described in section 3.2 is converted to and tackled as a tracking problem. Section 3.4 regards the forward models employed in our work.

3.1 Basics

Throughout the text the basic terms of entities, interaction and forward models are discussed. Here, we detail the meanings of these terms.

3.1.1 Entities

By *entities* we refer to the constituents of a scene. Such constituents might be a human, a robot, objects, furniture, *etc*. Understanding an interactive scene amounts to inferring the state of the entities it comprises. As we are interested in 3D scene understanding, we focus on the 3D state of the entities, *i.e*. the state that characterizes the entities in 3D space (but also across time).

In this context the aspects of 3D state that concern us are essentially the position and orientation of entities, that are jointly referred to as 3D pose. A lot of information can be extracted by identifying the pose of several entities in a scene, as well as the history of their poses. Therefore, recovering 3D poses accurately is essential in our work.

Although the simpler of entities can be adequately described by their 3D pose, there are more complex entities that require more information to be fully described. In this context we are also considering articulated entities that, put simply, can be regarded as structured groups of simpler rigid entities. For example, the human hand is an articulated structure, that can be regarded as a group of simpler structures, like the fingers, that in turn can be viewed as groups of phalanges, which do not require further decomposition with respect to rigidity.

3.1.2 Interaction

We employ the term *interaction* to refer to the relationships that connect the states of multiple entities. For example, we acknowledge that entities, when conditioned on a system of observing cameras, affect observations of each other by means of occlusions. We are explicitly interested in such relationships and believe that acknowledging them would improve the understanding over the parts, too. The connection between entities, which is regarded as a function of their 3D states, implies that they need to be considered jointly. We argue that, the disjoint consideration of the state of interacting entities yields such a loss in information that would most probably lead to inference failure. As an example, consider an opaque cup containing a ball. The estimation of the state of the hidden ball would fail unless it is connected to the state of the visible cup. In general, all entities in a scene have the potential to interact with any other. Therefore, in our approach, the states of all entities of a scene are considered jointly.

3.1.3 Forward models

The term *forward model* is borrowed from Control Theory, where it refers to a mechanism that provided some control input predicts the motion outcome of a motorized system [113]. In the computer vision literature, as well as in this thesis, we invoke the term to refer to models which, through emulation of actual processes, transform some initial conditions into measurable outcomes. For example, in this text we are discussing Computer Graphics and Physics, as forward models, to refer to mechanisms that are able to produce images from scene descriptions and physical state from initial dynamics, respectively. The transformation process is referred to as *simulation*. The product of simulation, along with the initial conditions, is referred to as *model instantiation*.

3.2 Parametric model based 3D understanding

The main goal in this context is to infer the 3D state of multiple interacting entities, from visual input, across frames. Our approach is model based, *i.e.* the expected variability in appearance and interaction of the entities is captured in predefined forward models. We consider parametric models, thus, the discussed variability is expressed in collections of variables, or parameters. The parameters of these models are connected to observations through an objective function that acts as a compatibility measure between hypothesized model instantiations and actual observations. Inference then amounts to identifying the most compatible model instantiation, *i.e.* the set of parameters that when applied to the forward model best reproduces the observations.

In notation, let a scene comprise N entities whose entire and combined state is given by the parameter vector $x \in X$. Let also $\mathbf{M}$ be a forward model that maps such states into a feature space F:

$$f = \mathbf{M}(x), f \in F, x \in X. \tag{3.1}$$

Given that there exists a procedure $\mathbf{P}$ that maps actual observations $o \in O$ in the same feature space F, and a prior term $\mathbf{L}$ that reflects how likely hypotheses originally are, regardless of observations, we can formulate the following error function $\mathbf{E}$:

$$\mathbf{E}(x, o) = \|\mathbf{M}(x) - \mathbf{P}(o)\| + \lambda \mathbf{L}(x), \tag{3.2}$$

which quantifies the discrepancy between actual observations o and a hypothesized scene state x. Then the problem of estimating the state of the scene for the current frame reduces to finding the minimizer s of the parameters x of $\mathbf{E}$ for given observations o:

$$s \triangleq \arg\min_{x} \mathbf{E}\left(x, o\right). \tag{3.3}$$

3.3 Optimization

Despite the compact and straightforward formulation of the inference problem, there are some issues, which regard the optimization ability and require attention.

3.3.1 Black box optimization

We put emphasis on flexibility and generalizability, *i.e*. we are interested in investigating different instantiations of eq. (3.3) (by employing different versions $\mathbf{M}$, $\mathbf{P}$ and $\mathbf{L}$) across different computer vision problems. Thus, we identify that optimization should be independent to our selection of objective function. Otherwise stated, optimization should be agnostic to the specifics of the objective function, or, the objective function should be treated as a *black box*. Unless this holds, the definition of every new model would also require non-trivial modifications to the optimization process too. Coupling modelling and observation also introduces the requirement of a compromise (*e.g*. simplification of modelling so that a certain optimization process becomes feasible) which hinders flexibility.

Fortunately, there exist methods that can successfully optimize objective functions while being agnostic to their specific traits, like continuity or differentiability and even implementation. These methods are called black box optimizers [40]. Black box optimization methods have the simple interface that provided a function, treated as a black box, they *guess* the optimum of this function. This *guessing* usually relies on careful sampling of the supplied function, driven by heuristics. By introducing a black box optimization method into inference, modelling (definition of forward models) is decoupled from optimization. This remarkably leverages the flexibility in establishing new and generic model-based computer vision methods. We have employed two black-box optimizers, namely Particle Swarm Optimization (PSO) [44] and Differential Evolution (DE) [95].

Particle Swarm Optimization

There exist several black box optimization methods [40], however, we have selected the Particle Swarm Optimization (PSO) heuristic [44, 45], due to several favourable traits: (a) PSO is a compact algorithm and is trivial to implement, (b) it adheres to a simple and intuitive principle that regards social behaviour, (c) it has only a few parameters that are relatively easy to decide (d) it is a parallel algorithm, which favours computational performance and (e) it is comparably powerful in optimization [40].

PSO is an evolutionary algorithm that receives an objective function $F(\cdot)$ and a search space S and outputs the optimum of $F(\cdot)$ in S, while treating it as a black box. Being evolutionary, it is parameterized with respect to a *population* of particles it needs to maintain for optimization. These parameters amount to the particle count N and the generation count G. Additionally, three parameters, namely w (constriction factor [21]), c_1 (cognitive component) and c_2 (social component), adjust the behaviour of the particle population.

PSO maintains a state that consists of a global optimum position G_k and per particle i local optimum $P_{k,i}$, current position $x_{k,i}$ and current velocity $v_{k,i}$, where k is the ordinal of the generation. Initially, particles are sampled uniformly in S. At each generation, the velocity of each particle is updated according to:

$$v_{k+1,i} = w(v_{k,i} + c_1 r_1(P_k - x_{k,i}) + c_2 r_2(G_k - x_{k,i})), \tag{3.4}$$

and the current position of each particle is updated according to:

$$x_{k+1,i} = x_{k,i} + v_{k+1,i}. \tag{3.5}$$

$P_{i,k}$ is set to

$$P_{k+1,i} = \begin{cases} x_{k+1,i}, & F\left(x_{k+1,i}\right) < F\left(P_{k+1,i}\right) \\ P_{k,i}, & \text{otherwise} \end{cases}. \tag{3.6}$$

G_k is set to the best scoring particle's $P_{i,k}$:

$$G_{k+1,i} = P_{k+1,l}, \text{ with } l = \arg\min_m F\left(P_{k+1,m}\right). \tag{3.7}$$

Variables r_1, r_2 represent uniformly distributed random numbers in the range $[0, 1]$. Behavioural parameters are fixed to $c_1 = 2.8$, $c_2 = 1.3$ and the constriction factor is computed as $w = 2/\left|2 - \psi - \sqrt{\psi^2 - 4\psi}\right|$ [21]. We have experimentally confirmed that for the provided w any combination that satisfies $c_1 + c_2 = 4.1$ results in very similar optimization performance. Therefore, the only remaining parameters that need to be set for PSO are the particle count N and generation count G.

For each generation the evaluation of $F(\cdot)$ is performed on the status of each and every particle, independently. This can be exploited on parallel platforms, towards achieving computational efficiency, by delegating distinct particle evaluations to different computational cores. This introduces significant benefits in computational demands and execution times.

Differential Evolution (DE)

Differential Evolution (DE) [24, 95] is another evolutionary optimization method, posterior to PSO. It depends on only a few parameters that have an intuitive explanation and exhibits remarkable performance in difficult problems of large dimensionality. DE effectively handles real-valued multidimensional, potentially non-linear and non-differentiable objective functions. It performs optimization by evolving a set of N_H hypotheses H_g and dimensionality D. Being evolutionary, DE is defined via its mutation, crossover and selection mechanisms that are applied at every generation g.

During mutation, every hypothesis $h_{g,i} \in H_g$ becomes a linear combination of three randomly selected, pairwise different hypotheses of the previous generation $g - 1$.

$$h_{g,i} = h_{g-1,r_1} + F\left(h_{g-1,r_2} - h_{g-1,r_3}\right), r_j \sim U\left(0, |H_g|\right) \wedge r_k \neq r_l \forall k, l. \tag{3.8}$$

Mutation is controlled by the differentiation factor $F \in [0, 2]$. Each mutated hypothesis $h_{g,i}$ is then combined with $h_{g-1,i}$ in order to produce a replacement candidate $\hat{h}_{g,i}$ in the crossover phase.

$$\hat{h}_{g,i}(j) = \begin{cases} h_{g,i}(j), & r_j \leq CR \vee j = idx_{g,i} \\ h_{g-1,i}(j), & \text{otherwise} \end{cases}, \quad j = 1, \ldots, D, r_j \sim U(0,1), idx_{g,i} \sim U(1, D) \tag{3.9}$$

In eq. (3.9), $h_{g,i}(j)$ denotes the j-th component of the i-th hypothesis in the g-th generation. The crossover constant CR controls the combination of individual parameters of $h_{g,i}$ and $h_{g-1,i}$. A random parameter index $idx_{g,i}$ is preselected in order to ensure that at least one parameter of the mutated vector will survive the crossover.

Finally, in the selection phase, the replacement candidate actually replaces the original one in the next generation, if it scores better in the objective function.

The original algorithm is parallel, in the sense that two consecutive generations are two distinct sets. We consider a serial variant, where the two generations are mixed. This means that a mutation may be based on already mutated vectors in the same generation. We have experimentally observed this mixing to add quicker reflexes to the algorithm, leading to faster convergence. We also consider a dithering parameter δ that modulates F at each generation. Dithering improves convergence and helps in avoiding local optima [18]. The DE variant of our choice appears with the coding DE/rand/1/bin in [95]. The input of DE is a real-valued objective function f, the number of generations N_G, hypotheses per generation N_H and constants F, CR, δ. The output of DE is the real-valued parameter vector that optimizes f.

Using PSO and DE in different problems

Evolutionary black-box optimization, especially in the form of the selected optimizers, is not panacea. Interestingly, we have found PSO and DE to be more suitable for one problem or the other (*e.g.* see section 5.2). Suitability regards traits of optimization, like convergence speed, convergence quality, *etc*.

3.3.2 Tracking

In this context we are dealing with complex objective functions over high dimensional spaces. Despite the optimization efficacy of PSO, DE and other black optimization methods, global optimization in these circumstances can be extremely hard and would require immense amounts of computational resources to become effective (aspects of scalability are discussed in section 6.2).

Instead, we exploit temporal continuity, between adjacent input frames, while computing eq. (3.3), and perform optimization locally, in the vicinity of solutions for the previous frames. Thus, search is already initialized close to the true optimum, and therefore much less resources are required for searching for it.

A notion of solution history h emerges, that is incorporated in the respective equations. eq. (3.2) becomes:

$$\mathbf{E}(x, o, h) = \|\mathbf{M}(x, h) - \mathbf{P}(o)\| + \lambda \mathbf{L}(x, h) \tag{3.10}$$

and eq. (3.3) becomes:

$$s \triangleq \arg\min_{x} \mathbf{E}(x, o, h). \tag{3.11}$$

By initializing search near last known solutions and by modulating computations so as to impose a preference over smooth estimation transitions between frames we can effectively substitute global optimization for local optimization and become, at the same time, effective and fast in the state estimation task.

3.3.3 Model complexity

Black box optimization is not panacea. Even if search is initialized close to the true optimum, through tracking, effectively exploring the search space can easily become problematic as dimensionality increases (see section 6.2). Therefore, care needs to be taken when designing forward models and the corresponding objective functions. So, it is always feasible to define a forward model and turn it into an inference mechanism, but this forward model cannot just be arbitrary and still preserve the ability to effectively perform inference upon it.

It is a fact that interaction is inherently complex, since multiple entities are regarded jointly and simultaneously. However, complexity may also be highly structured. Consequently, the involved entities might not need to be considered independently from one another, as this might be redundant. When entities are considered jointly it is likely that the required description length is less that what it would take to describe them disjointly.

For example, consider the human hand. We've shown that its dexterity can indeed be expressed by 26 degrees of freedom [61, 63]. However, and even though we have been able to tackle the problem despite the high dimensionality, we have also shown that even more structure can be identified, when conditioning hand motion on given tasks [31]. This dramatically decreases the dimensionality of the hand motion, as, given some task, it becomes constrained. In fact, parts of the hand need to collaborate in specific coordination which reduces the true degrees of freedom. For example, to describe a hand closing motion it is not required to describe all fingers disjointly, but jointly instead, as they are all performing a *similar* motion.

Consider another example that regards multiple distinct entities. In a scenario where a hand manipulates an object it is not necessary to track the state of both the hand and the object. As it will be shown (see section 6.3), the observation that the object cannot move on its own allows for the formulation of a tracking problem that regards only the degrees of freedom of the hand, as it is the hand alone that can change the state of the entire scene. Then tracking the object too comes as a natural consequence of tracking the hand.

Still, while we acknowledge that all measures should be taken towards defining the most compact models possible and stick to this direction, we also do tackle the problem of scalability (large tracking problems), outside modelling and in a way that proves to be both relatively fast and effective (see section 6.2).

3.4 Models

One of the most important processes in our approach is modelling, *i.e*. identifying the rules that govern the expected observations. In this thesis we have been invoking Computer Graphics and Physics, as forward models. These models are detailed here.

3.4.1 Appearance/Rendering

As already discussed, we are currently in position to model how entities appear on camera. Otherwise stated, we are able to model the acquisition process, *i.e*. the process of acquiring images from visual sensors. This process can be modelled using 3D rendering.

Description of input and output

3D rendering is a process that transforms 3D scene specifications into 2D images. In detail, a scene is specified by declaring object properties, like 3D shape, material and position, light properties, like type of light and position, and finally camera properties, like intrinsic (lens properties) and extrinsic (placement properties) configuration. A fully specified scene can then be passed through a rasterization pipeline (this is one of multiple ways of performing rendering), to generate the final 2D image.

Description of the process

Conventional rendering is currently performed through a programmable pipeline, *i.e.* a series of stages where some of them can be programmed, while others remain fixed. First, the programmer supplies the 3D primitive information, like the 3D surfaces (triangular meshes) that describe the entities, as well as location information (position, orientation). Then a programmable *vertex shading* stage is responsible for transforming the input geometry accordingly, based on the input location information and the camera configuration. The final outcome of vertex shading is a series of 3D triangles that have undergone projection so as to be mapped onto the camera projection plane. Then, a fixed rasterization stage is responsible for traversing the 3D triangles in a way to produce the corresponding pixels in the image plane. At the same time, through z-buffering, it is made sure that parts of triangles that are hidden by other geometry are not output to the final image. Once all visible pixels are computed they are passed through the *pixel shading* phase, where the programmer can decide the final outcome for each pixel.

As far as computer vision processes are concerned, the focus is usually on object and camera properties. Lighting conditions are commonly simplified away. The material of objects in the vast majority of the respective literature is considered to be Lambertian, which means that the apparent brightness of such a material to an observer is the same regardless of the observer's viewing angle. It is also very common that camera properties are specified too, through camera calibration. Therefore, the only free parameters are essentially the objects' shapes and poses. There are exceptions where the texture, *i.e.* a part of objects' material, is also considered [28].

Mathematical formulation of the model

The interface of the appearance forward model is to transform 3D scene descriptions, that comprise object shape, material and location specifications as well as camera intrinsic and extrinsic configurations, into 2D images. This forward model can be used sequentially to generate sequences of images, or videos. With an appearance model one can hypothesize scene configurations and predict how they would look like through a specified virtual camera.

In notation, we assume a function $\mathbf{R}$ that is appropriately set up with the shapes and materials of entities. After being set up it supports the mapping of 3D location of entity references x, along with camera configuration specifications c, to the space I of 2D images:

$$r = \mathbf{R}\left(x, c\right), x \in X, r \in I \tag{3.12}$$

3.4.2 Behaviour/Physics

A great part of how entities behave at the macroscopic level can be explained through Newtonian Dynamics of rigid bodies. In a physical world, no two entities can share the same space. Instead, when they come too close they collide. The outcome of the collision depends on the properties of the objects, the initial conditions of the collision and also external forces, like gravity or wind. Collisions result in change of velocities (accelerations). Non-rigid entities, like articulated bodies, can be directly approximated by ensembles of rigid parts which are assembled through motion constraints. Non-rigid entities, like clothing and elastic materials, referred to as soft-bodies, are inherently difficult to accurately represent and perform simulation upon[1]. We have not considered soft-bodies in the works presented here. However, this is not due to our inability to handle them, since, because of decoupling modelling from optimization (see section 3.3.1), we only require a physics simulator that can handle soft body simulation, like Bullet [23].

The behavioural aspect that is not included in Newtonian Dynamics has to do with the intent behind the actions of entities, which can voluntarily introduce energy into closed systems, *e.g*. a human, a robot or another type of machine. Modelling intent remains outside the scope of this work.

Description of input and output

Physics simulation is the process of computing the evolution of a dynamic scene across time. For example, a falling object's trajectory can be computed across discrete instants in time by integrating acceleration due to gravity. When more entities are introduced, the complexity of physics simulation increases, as collisions between each and every pair of entities need to be checked.

There exist various commonly available physics simulators or physics simulation libraries, whose focus is on powering computer games and introducing realism. A shuttle point then emerges: these physics simulators do not reflect real conditions, instead they generate realistic behaviour, *i.e*. behaviour that tricks the observer into thinking it's real. Still, the fact that everyday object behaviour can be somehow synthetically reproduced is the most significant trait of these simulators.

Description of the process

The vast majority of commodity physics simulators (ODE [90], Bullet [23], Newton Game Dynamics [42], PhysX [59], *etc*.) perform discrete dynamics simulation. More specifically, time is discretized and is only considered at samples. Physics simulation is then responsible for advancing a physical state from one time instant to the next. The physical state comprises per entity specifications that regards their 3D collision shapes, materials and energies. Entities are essentially represented as 3D shapes with inertia tensors, masses, friction and restitution coefficients. Collision shapes can usually be represented analytically, like spheres, cylinders, boxes, convex polytopes, *etc*., and collision checking between analytically represented shapes is also analytical and usually fast. More complex shapes are usually decomposed in compounds of analytical shapes. Inertia tensors and masses reflect a body's resistance towards induced accelerations. Friction coefficients

[1] There exist physics simulation libraries that can handle soft body simulations. Bullet [23], the library we have used in this work is one of those that possess this capability.

express the amount of energy that is transferred from body collisions to tangential accelerations. Restitution factors modulate the amount of energy that is lost during collisions. All of the above hardly reflect realistic conditions and only bare relative significance, since they are simulator- or application-specific.

During two adjacent time samples, or upon a time sample, two or more entities might be colliding or moving towards collision. Physics simulators are then responsible for identifying the exact collision time and location and apply closed system theorems, like preservation of energy and preservation of momentum, to compute how the entities will behave after the collision. Collision is judged based on the specified entity collision models.

Advancing the physical state from a starting point in time to another point later in time, iteratively across adjacent time samples, generates *physically plausible* discrete motion trajectories for the entities involved. To achieve this, physics simulators usually work in the following manner. For a transition from one time sample to the next, potential collisions are checked. Because of the combinatorial nature of collision checking (all versus all) and for the sake of efficiency, a pruning phase, usually called broadphase collision checking, is employed, where rough exclusion of entity pairs which are certain not to collide occurs (their velocities are not enough to cover their current distance so as to collide in the next time sample). During broadphase collision checking, shapes are abstracted to simpler forms, like bounding spheres, oriented bounding boxes (OBB) or axis aligned bounding boxes (AABB) that are highly efficient in performing pessimistic collision checking. The rest of pairs that can yield true collisions are checked in detail according to their true shape specifications. For each collision the appropriate forces are applied to the corresponding entities, that adhere to the closed system assumption. After forces have been computed more forces are added that come from the environment (gravity, wind, magnetic fields, *etc*.). Once all forces are computed, they are accumulated and integrated across time to yield the next physical state of the entities.

Mathematical formulation of the model

The interface of a physics model is a mapping from an initial physical state to the next state, with respect to the two involved time instants (start, next). The free parameters are then the two time instants themselves and the specification of the initial physical properties of the entities. The outcome of physics simulation is the new physical state.

In notation, if S_t is the initial physical state, then the next physical state S_{t+1} is computed via the simulation process $\mathbf{S}$, as a function of the current scene state S_t:

$$S_{t+1} = \mathbf{S}\left(S_t\right), \tag{3.13}$$

where S_k is the full description of the physical state at time k for N entities:

$$\mathrm{S}_k = \left\{\left\{s_i, m_i, I_i, F_i, R_i, \vec{p}_i, \vec{q}_i, \vec{v}_i, \vec{\alpha}_i\right\} | i = 1 \ldots N\right\} \tag{3.14}$$

In eq. (3.14), s_i is the collision shape, m_i is the mass, I_i is the inertia tensor, F_i is the friction coefficient, R_i is the restitution coefficient, $\vec{p}_i$ is the position, $\vec{q}_i$ is the orientation, $\vec{v}_i$ is the linear velocity and $\vec{\alpha}_i$ is the angular velocity of body i, at time step k. For each individual object, the applied forces and torques are the accumulated result of the total simulated interaction. All vectors are in 3D. The required time step Δt is inferred from t and $t + 1$.

3.4.3 Conjoint modelling

How objects look (graphics) and how objects behave (physics) can be modelled individually, but it would be too oversimplifying to consider that they are independent. On the contrary, they are tightly coupled and can thus be complementing to each other.

As a first example consider the case of 3D tracking of articulated objects. The human hand and the human body manifest structural complexities that are difficult to capture, even from multiple cameras. A forward model of how such structures appear from different views can be used to track them. However, there exist ambiguous configurations due to occlusions, from objects or from integral parts themselves, or even poor image acquisition performance. A prior should be used to disambiguate such cases and also restrict the search space in the domain of *plausible* solutions. By considering some physically sound principles like (a) the impossible sharing of space from multiple entities, (b) the physical constraints that govern articulated movement, (c) the preservation of energy in a closed system, *etc*.,we can form the required prior that would enable a computational system to infer the *invisible*, by invoking undeniable facts, *i.e*. the forward modelling of the dynamics (see section 5.1 and [65]).

Another example would be the testbed scenario according to which we require the estimation of the state of a uniformly coloured bouncing ball through its observation by a single or by multiple calibrated cameras. By employing standard computer vision techniques, accounting for the position of the ball at each time step is not trivial. The possibly inadequate acquisition frame rate may lead to aliasing and the possibly large shutter time of the cameras may lead to motion blur. On top of the above mentioned difficulties, for some aspects of the state of the ball (*i.e*., its orientation and/or angular velocity) there is no apparent evidence, whatsoever. The problem becomes even more challenging when, due to occlusions, the available set of visual observations becomes even more limited or when we are interested in solving the above problems based on single-camera observations. Instead of trying to model the appearance of the bouncing ball, as it will be discussed later, we can model the dynamics of the ball and use visual cues as constraints upon the dynamics, in order to perform robust motion estimation.

Mathematical formulation of the model

The combined forward model has an interface that is a straightforward combination of the interfaces of the parts. Given that the physical state already contains location information, the output of a simulation is readily susceptible to 3D rendering. Therefore, one may compute the appearance of a scene after it has evolved, according to eq. (3.12) and eq. (3.13):

$$r = \mathbf{R}\left(\mathbf{S}\left(S_t\right), c\right). \tag{3.15}$$

Once again, the simplicity in describing and using the mentioned interfaces stems from the flexibility introduced through decoupling the modelling process from optimization (section 3.3.1). Forward models can be combined through their interfaces, like in this case, by simply invoking them, as any function in a programming language would be called.

Chapter 4

Computational framework

The computational framework presented here essentially regards a tracking loop, as described in chapter 3. It comprises several components, that are substitutable and whose different combinations can lead to distinct methods that tackle different problems. A template information flow of this framework is depicted in fig. 4.1.

We consider 3D tracking problems, *i.e*. problems where the temporal continuity assumption holds and the 3D state estimation of interacting entities is required. Each time sample is associated with observations made by visual sensors. These observations are mapped onto a feature space, in order to easily filter out much of the complicating variability. In parallel, for the given observations a set of *explanations* are hypothesized, that are in the vicinity of the optimal explanation for the previous time sample. These hypotheses can bare physical coherence. Comparability between observations and hypotheses is established by means of 3D rendering and feature mapping on synthetic data. The outcome of the comparison, *i.e*. the compatibility scores, guide the search process into finding the most compatible explanation.

This briefly described information flow is implemented as a software pipeline, that takes advantage of commodity hardware, in order to deliver interactive tracking rates. Some parts of the incorporated algorithms run on Central Processing Unit (CPU) threads and others run on Graphics Processing Unit (GPU) threads. The details are provided in the following sections. Each process, along with its inputs and outputs, corresponds to a section.

The description begins with section 4.1, where the fundamental elements of our framework are explained. The basic information flow of our framework is briefly sketched in section 4.2). The components of the framework run in different computational modes, and their descriptions are grouped according to these modes. Components may be executed in the CPU (see section 4.3), using multiple threads (see section 4.4), involving asynchronous execution (see section 4.5), and can even take advantage of GPUs (see section 4.6).

4.1 Preliminaries

Throughout the following sections references will be made to transforms, geometry and entities. The corresponding notions and representations are detailed here.

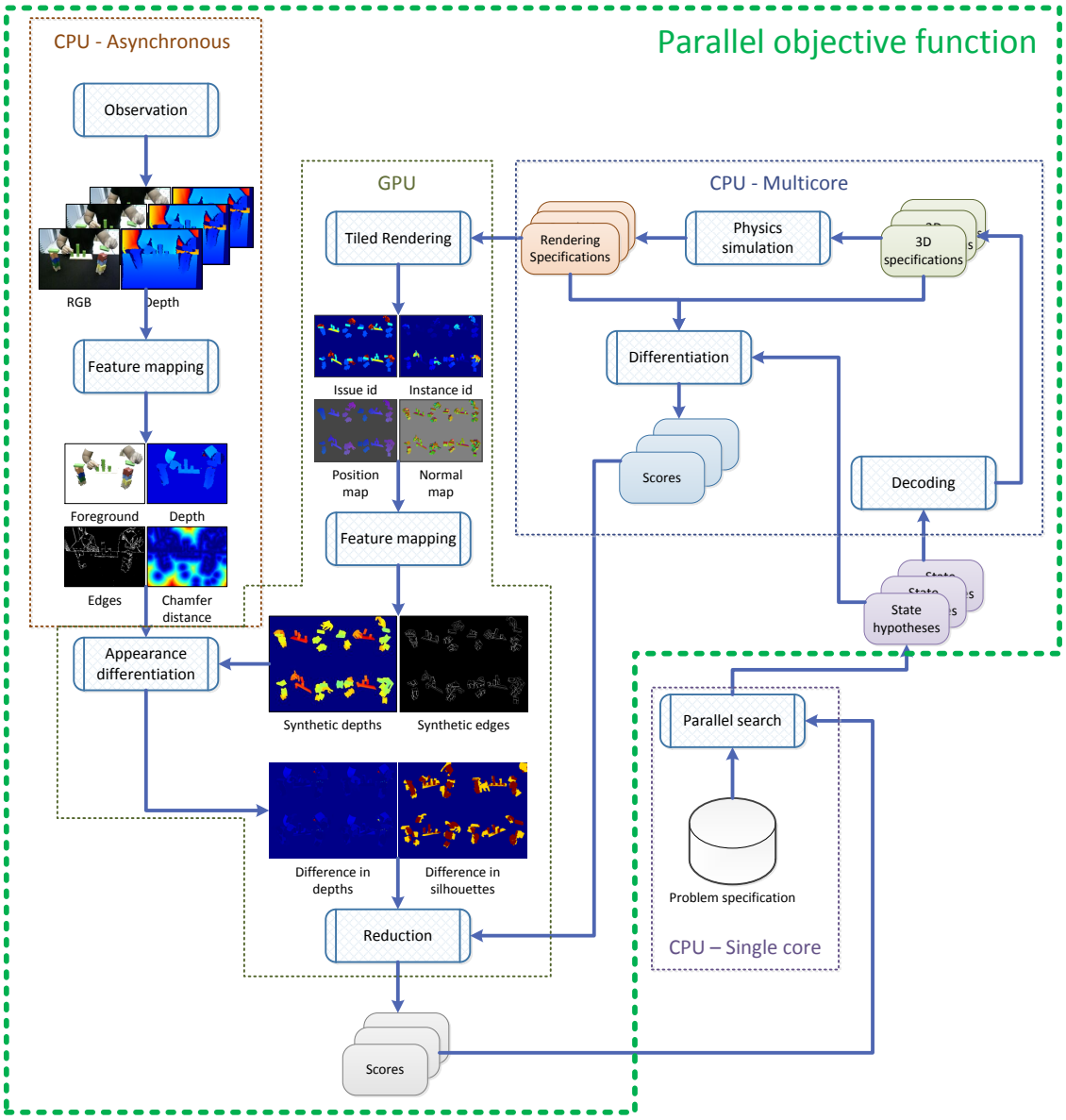


Figure 4.1: The information flow diagram of the proposed computational framework. All components are detailed in chapter 4.

4.1.1 Transforms

Entities are associated with poses. We represent poses, along with other properties, such as scale or skew, by means of 4×4 3D homogeneous transform matrices, as it is commonly done in Computer Graphics. In fact, we model all 3D transforms after such matrices, *i.e.* world, view and projection transforms.

To support 4×4 matrices, as well as operations on them, but also to be able to do so in both CPU and GPU programming we make use of the OpenGL Mathematics library (GLM) [1]. This library supports matrix multiplication, inversion and composition from position representations, like 3D vectors (`glm::vec4`), and orientation representations, like quaternia (`glm::quat`), Euler angles and angle-axis representations.

Rendering pipelines distinguish between world, view and projection transforms. We also employ this convention. In notation, the transform that some geometry undergoes so that it is effectively rasterized is given by the following equation:

$$M = \text{Projection} \times \text{View} \times \text{World} \tag{4.1}$$

World transform

A transform that regards an entity's pose (position and orientation) and scale. This transform is used to map the entity from object space to world space, *i.e.* place in the world's reference system. Given a 3D position vector $\vec{p}$, a quaternion $\vec{q}$ and a 3D scale vector $\vec{s}$, the world transform is computed as:

$$World = T\left(\vec{p}\right) \times R\left(\vec{q}\right) \times S\left(\vec{s}\right), \tag{4.2}$$

where T generates a translation transform (`glm::translate`), R generates a rotation transform (`glm::mat4_cast`) and S generates a scale transform (`glm::scale`).

View transform

Stated briefly, the view transform is the inverse of the world transform of the camera. More specifically, if a camera is known to be at the pose described by matrix W_c then the view transform is:

$$View = W_c^{-1} \tag{4.3}$$

In this context, since the camera already corresponds to an actual entity, *i.e.* the visual sensor, we use the sensor's extrinsic calibration information directly to define $View$. The extrinsic calibration is performed using the standard method of OpenCV [11].

Projection transform

The projection transform regards the intrinsic configuration of a camera. Once more, since an actual visual sensor is regarded, this information can be provided by means of extrinsic calibration (OpenCV). This calibration yields the focal length and camera center of a perspective projection model.

In our case, where Computer Graphics is involved, it is necessary to also define some clipping planes. In Computer Graphics the projection transform needs to map a 3D scene to a normalized volume, the viewing frustum (see fig. 4.2), that comprises six clipping planes (top, down, left, right, near and far). As a convention, graphics pipelines consider

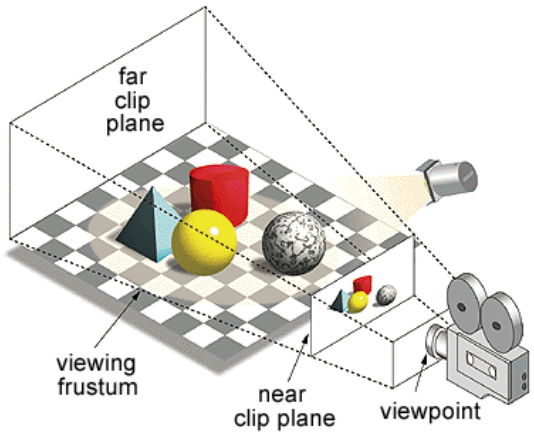


Figure 4.2: The viewing frustum, defined in the context of Computer Graphics. It essentially defines a clipping volume, outside which geometry is not visible.

that, after projection has been performed, only what is included in the range $[-1,1] \times [-1,1] \times [0,1]$[1], for coordinates x, y and z respectively, is actually rasterized. So, an additional task of the projection transform is to also appropriately map the view space to these ranges.

In our case, we consider the following projection matrix, for a given calibration:

$$\text{Projection} = \begin{bmatrix} 2f_x/W & 0 & 2c_x/W - 1 & 0 \\ 0 & -2f_y/H & 1 - 2c_y/H & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{z_f}{z_f - z_n} & -\frac{z_n z_f}{U\left(z_f - z_n\right)} \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{4.4}$$

where f_x and f_y represent the focal length and pixel size in image axes x and y respectively, c_x and c_y constitute the estimated camera center, W and H are the width and height of the projected image, z_n and z_f are the depths of the near and far clipping planes.

4.1.2 Geometry

Each entity is associated with some 3D geometry that represents the volume this entity occupies in space. In that respect and within this context, entities can be simple rigid objects, that can be represented by a single 3D triangular mesh, or articulated structures, *i.e.* compounds of simpler structures.

3D triangular meshes

A 3D triangular mesh is a set of triangles, defined by 3D vertices, that together define a continuous 3D surface. Because we are also interested in physics simulation, these triangular meshes are also watertight or closed, *i.e.* they define a finite volume (or, there is no border edge in the mesh's graph, *i.e.* holes or borders in the surface).

The vertices that are shared among adjacent triangles are unique and are multiply referred to. Otherwise stated, 3D triangular meshes are hereby represented by a vertex buffer

[1] These are the ranges that the Direct3D 9© rendering pipeline considers. For OpenGL® the ranges are $[-1,1] \times [-1,1] \times [-1,1]$.

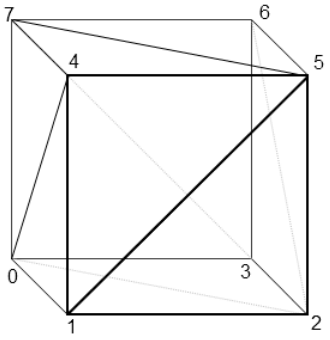


Figure 4.3: This closed cube is made of 8 vertices and 12 triangles. Each vertex is shared by 4 triangles. Each triangle is defined as a triplet of vertex indices. For example, the two front facing triangles are $[1, 4, 5]$ and $[1, 5, 2]$, which corresponds to the triangle list representation of 3D triangular meshes.

and an index buffer. The vertex buffer holds unique vertices and the index buffer holds triplets of vertex references, that define triangles. The shared vertices appear after the same index in the triangles they help form (see fig. 4.3). This triangle list representation is memory efficient for 3D meshes of high degree of vertex reuse. More efficient representations are triangle strips and triangle fans. The order of vertex referencing also denotes the triangle's orientation. The clockwise order is used to define a front facing triangle.

Rigid and articulated entities

Rigid entities require a piece of 3D geometry and a 3D transform to be fully defined. Articulated entities are slightly more complex. They are defined by ensembles of rigid entities, with each one being located, in object space, through a local transform, *i.e*. a transform expressed in the reference system of the articulated entity. Begin articulated means that the parts which comprise these entities are connected through joints.

Mesh references

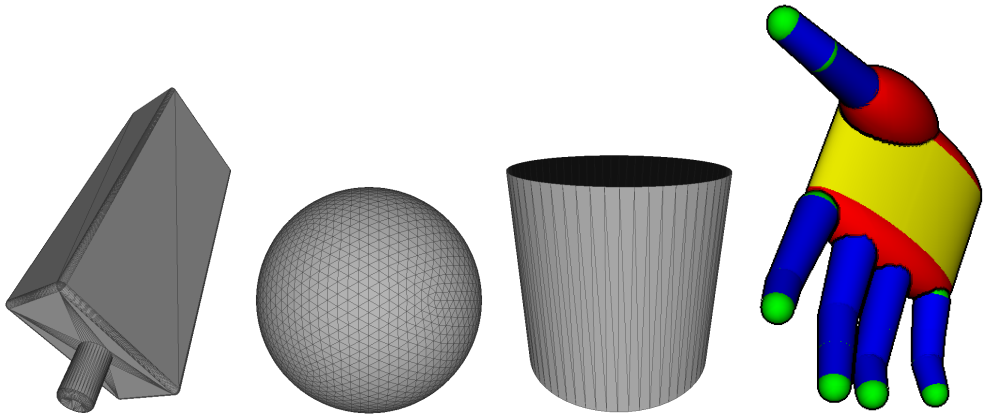
3D triangular meshes are heavyweight objects, *i.e*. they require much memory, which makes them expensive to move around. Instead, meshes are allocated once at one place and then only referred to through lightweight references. In our computational model, meshes are loaded once into the appropriate memory region and then are only referred to through unique references.

4.1.3 Entities

In the tracking problems dealt with within the context of this thesis there are mainly two types of entities: rigid objects and articulated hands.

Rigid entities

Rigid entities are represented by single 3D triangular meshes and have 6 DoFs, that regard pose, *i.e*. position (3D coordinates) and orientation (euler angles). We employ quaternia, as a representation of orientation, because, despite their redundancy, they present overall



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

Figure 4.4: Entity 3D models. Entities can be rigid (a), or articulated, *i.e*. compounds of interconnected rigid objects. The hand (d) is an articulated entity that is composed of appropriately transformed spheres (b) and cylinders (c).

better traits than Euler angles [112]. Therefore, rigid transforms are essentially represented by 7 values and lead to 7 parameters.

Hands

Hands are articulated entities whose pose and articulation is represented by 27 parameters. 7 parameters are used to localize the hand's palm, as if it were a rigid object. Each of the 5 fingers is then configured relatively to the pose of the palm. Each finger is connected at its base to the palm with a 2D hinge joint, that yields 2 DoFs. Then, the 3 phalanges of each finger are interconnected with hinge joints, yielding 2 additional DoFs for each finger. Thus, in total, the fingers require $5 \times (2 + 2) = 20$ parameters.

The hand model is realized by putting together appropriately transformed spheres and cylinders. By means of non-uniform scaling and skewing, the 3D triangular meshes that represent a unit sphere (unit radius) and a unit cylinder (unit radius and unit height) are converted to ellipsoids and truncated cones (see fig. 4.4(d)).

4.2 The loop

As already mentioned, our framework implements eq. (3.11) (see fig. 4.1). The main task is the optimization of a parallel objective function, *i.e*. an objective function that can be efficiently sampled at multiple points over parallel architectures. The optimizer is also parallel, *i.e*. it works by generating and evolving a set of independent hypotheses, or samples. The objective function quantifies the discrepancy between hypothesized state and actual observations. The domain of this objective function, as well as optimizer-specific details, are provided as specifications to the parallel optimizer. Then the optimizer engages in a loop during which hypotheses are generated, evaluated against observations and improved, in order to approximate the optimum state. This process is repeated until a predetermined computational budget is spent or a convergence criterion is met.

The discrepancy between given hypotheses and observations is performed by mapping hypotheses and observations in the same feature space, differentiating the resulting repre-

sentations and then computing a compatibility measure based on the differences. Before hypotheses can be mapped to the selected feature space they are simulated in physics and rendering.

The execution of the various computational components is delegated to the most appropriate computational resource of a commodity computer, in order to achieve optimal optimization throughput. The modules themselves are substitutable. Through substitution difference instantiations of the same architecture are established which tackle different computer vision problems.

The processes depicted in the information flow diagram of fig. 4.1 are visually grouped into clusters, according to the type of computational resource they utilize. Each such group corresponds to a section in this text. In these sections, each part of every group is described in a sub-section.

4.3 CPU Single Core components

When focusing on execution speed, not all computations require parallelization in order to execute quickly. CPU manufacturers have always been focusing on fast serial execution of programs. Simple programs, which contain few iterations and no I/O, should execute on a single core of a CPU for maximum efficiency. Such is the logic of the *Parallel Search* process. The domain of search is defined during problem specification.

4.3.1 Problem specification

Solving the tracking problem amounts to performing an optimization (see eq. (3.11)). Our proposal in this thesis is the incorporation of black box optimization techniques for solving such tasks. The black box optimizer's interface is a simple one. What is required as input is a specification of the search space and the objective function itself.

The *search space* X is the effective area in which the optimizer heuristic should search to find the *optimal* solution. We are considering *parametric* problems, therefore, the search space is also parametric. The number of parameters is also the *dimensionality* of the problem. In this context we are considering search spaces that are defined by *box bounds*, *i.e.* a minimum and maximum value for each and every parameter. This specification yields a volume that in N dimensions can be represented as a N-D box. Therefore, we are considering points x that are inside some bounds b_{min}, b_{max}:

$$S = \{x \in X \,|x_i \geq b_{\text{min},i} \wedge x_i \leq b_{\text{max},i}\}, X \triangleq \mathbb{R}^D, i \in [1, \ldots, D] \tag{4.5}$$

An objective function f in this context is a mapping from points of this search space to scalar values:

$$f : x \to y, x \in S, y \in \mathbb{R}. \tag{4.6}$$

No other property or trait (*e.g.*, continuity, differentiability) is required from the objective function for it to be susceptible to black box optimization. This introduces flexibility as it facilitates the quick setup of arbitrary problems. This also implies that the implementation of the function can also be arbitrary. In our context we are invoking highly non-analytical processes, like 3D rendering and physics simulation, which are used as black-boxes themselves. Therefore, black box optimization is appropriate.

4.3.2 Parallel Search

The strongest argument against model based methods is the high computational costs which lead to long execution times. One of our goals is to deal with this issue directly. To that end, we focus on exploiting parallel techniques in computer vision methods so that we can take advantage of parallel hardware architectures and thus achieve good computational performance.

A central point in our model based approach is the search phase, *i.e*. the phase during which hypotheses are generated and evaluated. The evaluation of these hypotheses is the most time consuming task, as this may rely on elaborate and computationally intensive simulations.

A parallel search heuristic is one that evolves multiple hypotheses which can be evaluated independently. We discuss the difference between serial and parallel in the case of PSO. During optimization and for each generation, PSO alternates between the phases of evaluation, mutation and crossover. In the parallel variant of PSO there is a barrier that separates the evaluation phase from the rest of the three. More specifically, at the beginning of each generation, all particles are evaluated, independently, and then according to these results, there are mutated and crossed over. In the serial variant this barrier is missing, so evaluation is mixed with mutation and crossover. More specifically, every particle's state change is a function of the intermediate results computed so far in the same generation. Therefore, evaluations are not independent. Mixing could yield PSO, and other evolutionary algorithms, *better reflexes* [18], which means that optimization might be achieved earlier, *i.e*. it might take less generations. On the other hand, in parallel platforms, evaluating a non-mixed generation in parallel might be so fast that would still allow for spending some more generations, when compared to execution times of the serial variant. PSO is a search heuristic whose parallel variant has led to faster execution times for our problems, while maintaining satisfactory accuracy levels.

We employ the parallel variant of PSO. The mutation and crossover phases of PSO are very simple and compact, and are executed efficiently on a single CPU core. Regardless of specifics, every parallel search heuristic is responsible for generating hypotheses to be evaluated and to ameliorate them, across time but also as early as possible, until a termination criterion has been met. PSO can be substituted for another black box optimization technique. So, any progress in the area of black-box optimization is directly integrable to the proposed framework.

The other search heuristic we have used is Differential Evolution (DE). This is a serial black box optimization heuristic, which has been experimentally showed to outperform PSO in difficult optimization problems [40]. We have mainly used PSO for local optimization during tracking, when computationally heavy objective functions were involved.

Parallel objective function

The use of a parallel search heuristic calls for the definition of a parallel objective, *i.e*. an objective function that is aware of parallel architectures so that it is efficiently computed. We define a parallel objective function to be a straightforward generalization of the serial objective function of eq. (4.6):

$$f_p : x \to y, x \in S^N, y \in \mathbb{R}^N. \tag{4.7}$$

More specifically, instead of mapping a single N-D point to a single scalar, the generalization maps a collection of such points to a scalar vector. This simple interface allows

for serial objective functions to be promoted to parallel objective functions but also for parallel objective functions to be demoted to serial objective functions. Each different computer vision problem may connect observations to hypotheses differently, which in turn requires the formulation of new objective functions.

In the diagram of fig. 4.1 the architecture of a parallel objective function is depicted. Except for the processes of *parallel search* and *problem specification* (lower right group), all modules are interconnected so as to implement the evaluation of hypotheses against observations, in parallel. This architecture acts as a template and can be instantiated. More specifically, different objective functions can be generated by employing different variants for the parts (*e.g.*, feature mapping, physics simulation, rendering, *etc.*).

Serial Objective Function $\rightarrow$ Parallel Objective Function Given a serial objective function f_s, a parallel objective wrapper function f_w can be defined that appropriately delegates computations to f_s:

$$f_w\left(f_s, x_0, \ldots, x_N\right) = \left\{f_s^{C_0}\left(x_0\right), \ldots, f_s^{C_N}\left(x_N\right)\right\}, \tag{4.8}$$

where indices C_i indicate the computational core at which the partial evaluation will take place. In a serial system, it holds that $C_i = 0, \forall i$ and all evaluations are serialized. In a parallel system with many cores it may hold that $C_i = i, \forall i$, so that each distinct evaluation is performed on a separate core. In GPU implementations where granularity can be even more detailed, the evaluation of a single invocation alone may regard multiple cores at a time, as it will be discussed later (see section 4.6).

Parallel Objective Function $\rightarrow$ Serial Objective Function A parallel objective function may be also required to be optimized by a serial optimizer. Demotion is then trivial. Given a parallel objective function f_p, a serial objective wrapper function f_w may be defined:

$$f_w\left(f_p, x\right) = f_p(\{x\})_0. \tag{4.9}$$

More specifically, this wrapper is a simple function which delegates the evaluation of a trivial collection of a single point to f_s and maps its result to the first scalar of the trivial 1-element result vector.

4.4 CPU Multi Core components

There are tasks which can benefit from multi-core architectures. These tasks usually contain lengthy iterations of heavy computations and no I/O. In the presented diagram, these processes are *Decoding*, *Physics Simulation* and *Prior Evaluation*.

4.4.1 Decoding

A hypothesis, *i.e.* a parameter vector, might not, in itself and immediately, provide 3D specifications, as required for it to undergo physical simulation and 3D rendering. To bridge this gap we introduce the notion of decoding. The main task of a *Decoder* is to map arbitrary hypotheses to 3D state, or, to provide 3D geometry specifications associated with 3D world transforms. Evidently, decoding is tightly coupled with the tracking problem itself, as it provides the means to make hypotheses comparable with observations.

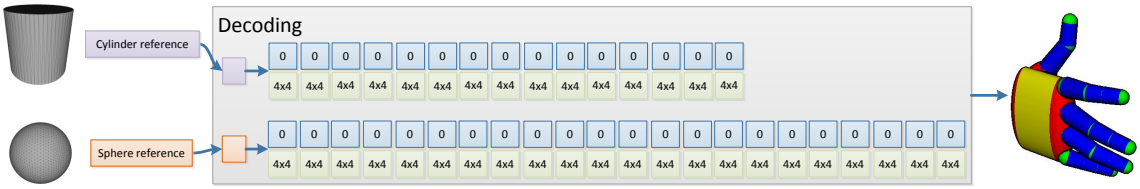


Figure 4.5: The decoding of a 3D synthetic hand. A hand model comprises several spheres and cylinders, that are appropriately transformed so as to resemble a hand structure. The corresponding decoding is a mapping of 3D mesh references (sphere and cylinder) to 4×4 3D homogeneous matrices (transforms). The hand depicted is composed of 15 cylinders and 22 spheres. Each 4×4 matrix is associated with an id to indicate which hypothesis it regards. In this figure only one hypothesis is decoded, so all indices are set to 0. In the general case, a decoding regards multiple hypotheses and it then acts as a batch.

In order to exploit parallelism, a parallel interface is also established for the decoder. A decoder maps a series of hypotheses to instantiations of known geometry, *i.e.* to pose specifications that are tied with geometry references. More specifically, a set of hypotheses are mapped to a decoding, *i.e.* a collection of transforms, where each one is associated with a reference to a piece of geometry and an index that identities which hypothesis this very transform regards (see fig. 4.5). Indeed, hypotheses may involve articulated entities, whose parts require multiple transforms to localize. Therefore, multiple transforms may regard the same hypothesis. Transforms that refer to the same pieces of geometry are grouped together, regardless of which hypothesis they regard. This allows for efficient batching of geometry rendering, as it will be discussed later (see section 4.6.1). Overall, the aforementioned constitutes a piece of information that, along with camera configurations, fully specifies a 3D rendering.

Decoders are inherently parallel. While decoding multiple hypotheses, they may delegate the decoding of distinct hypotheses to distinct cores. We are using CPU cores for that matter. In the tracking problems tackled, mainly three decoders were employed, namely a hand decoder, a rigid decoder and a combination decoder. These are detailed here.

Rigid decoder A rigid decoder is responsible for mapping 7-D points, representing rigid transformations in the form $[p_x, p_y, p_z, q_w, q_x, q_y, q_z]$, to transformation matrices that regard only translation and rotation. Translation is defined by a 3D position vector, extracted from values p_x, p_y, p_z, and a quaternion extracted from values q_w, q_x, q_y, q_z. From the translation vector a translation matrix is generated and from the quaternion a rotation matrix is computed, that are then multiplied together to yield the final world transform of the decoded entity. Each such decoder is associated with a mesh reference, which is associated with the transforms during decoding.

Hand decoder A hand decoder is responsible for mapping 27-D points to 3D state that resembles a hand structure (see fig. 4.5). These 27 dimensions correspond to the 27 parameters of the modelled hand (see section 4.1.3). Decoding the hand amounts to computing its forward kinematics, *i.e.* traversing its kinematic structure and emitting global transformation matrices for each part that are the result of accumulation of local transforms. The same decoder can be used for decoding a left or a right hand. More details over decoding a hand hypothesis can be found in [61, 63, 69, 70].

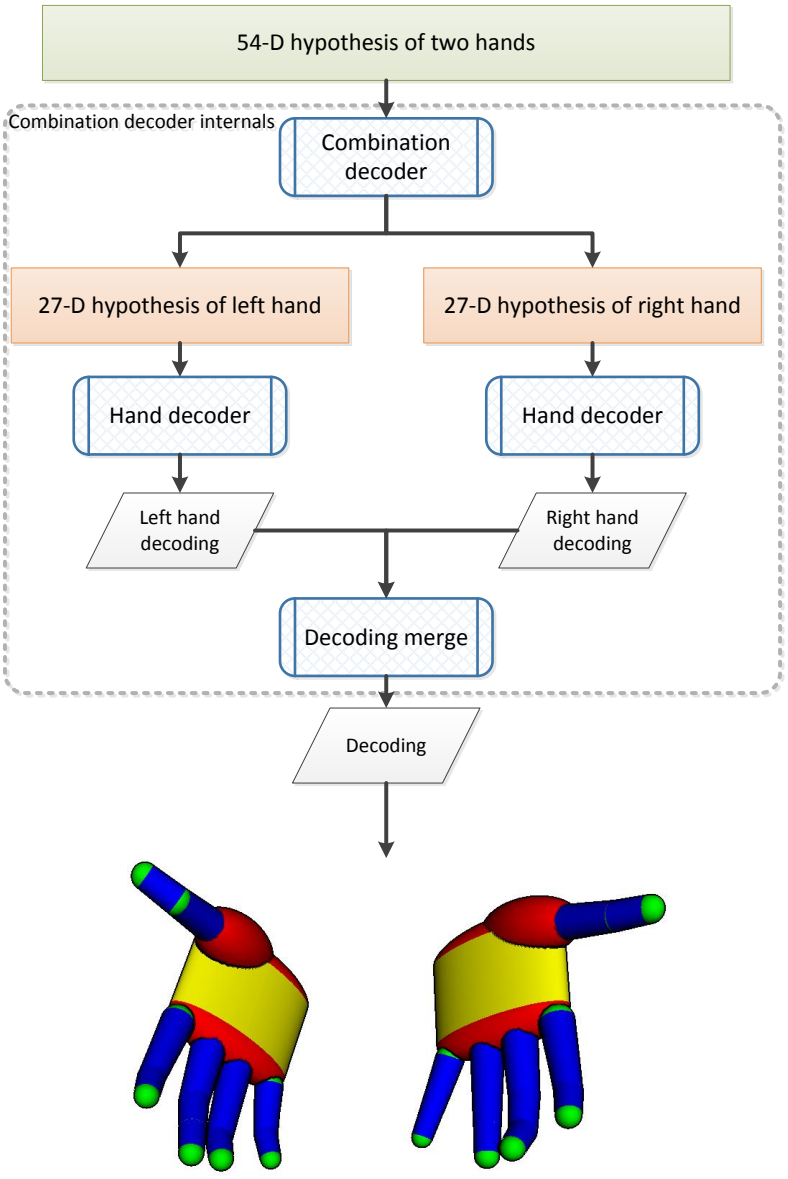


Figure 4.6: An example of combination decoding. Decoding an original 54-D hypothesis that regards two hands can be broken down to decoding two 27-D hand hypotheses separately. The partial results can be trivially combined into a merged decoding.

Combination decoder Decoders are composable modules. More specifically, they can be combined in order to decode more complex hypotheses. The considered type of combination bares concatenation semantics. More specifically, more complex hypotheses are considered to be concatenation of simpler hypotheses. For example, a hypothesis that regards two hands is just a juxtaposition of two hypotheses, one for the left hand and one for the right hand. Thus, a parameter vector of 54 dimensions can be broken down to two 27-D vectors. Each vector can be input to a distinct decoder and then all results can be combined into one big decoding (see fig. 4.6).

4.4.2 Physics Simulation

Physics simulation, as described in section 3.4.2, is the process of evolving a physical state, as described in eq. (3.14). Decodings are provided as input to physics simulation processes, but they need to be augmented so that they fully specify eq. (3.14). More specifically, decodings only provide static configurations of entities, like position and shape, and do not include dynamic aspects of state, like velocities, energies and materials. The physics simulation is then responsible for augmenting these decodings accordingly and performing the necessary book-keeping to map decodings to their augmented counterparts. The augmented state needs only be known by the physics simulation process. So, in system terms, the physics simulation process transforms decodings into decodings, by augmenting the input, reasons upon it to yield new augmented state, and strips it of the augmentations to deliver it to the next processes in the pipeline as a pure decoding that contains only static information.

In this context we are using physics simulation processes as black boxes. More specifically, we employed 3$^{\text{rd}}$ party components, *i.e*. the Newton Game Dynamics[2] and Bullet Game Physics Simulation[3] libraries to perform physics simulation. They both provide a similar interface, where an initial physical state is properly setup and given as input, along with a time step, and the output is the advanced physical state, changed due to external forces or internal interactions (*e.g*. collisions).

In physics simulator notation, the entire physical state constitutes a *world*. More specifically, a world is fully specified by providing the shape, material, poses and velocities of all entities involved. Each hypothesis in our pipeline corresponds to a distinct world. These worlds are independent and can therefore be simulated independently too. Each world is simulated on a different CPU thread, and multiple threads can be processed in parallel by a multi-core CPU. The performance gain is then proportional to the amount of cores available.

However, performing multiple simulations in parallel has not been a goal for commodity physics simulators. Therefore, design has not been influenced towards providing efficiency in this direction. In fact, to the best of our knowledge, we are the first to explore the parallelization potential of this process in computer vision methods. It can be expected that as this requirement becomes established, parallel physics simulation, *i.e*. performing multiple simulations in parallel[4], will receive enough attention so that even more efficient implementations will become available.

[2] `http://newtondynamics.com/forum/newton.php`

[3] `http://bulletphysics.org/wordpress/`

[4] As opposed to parallelizing and accelerating the internals of a simulation of a single world, which has received significant attention.

4.4.3 Prior evaluation

An objective function should connect observations to hypotheses so that the compatibility of the latter can be computable. We are mainly using appearance as a means of evaluating this compatibility, *i.e*. we look for hypotheses that *look like* the observations. This takes care of the 1st term in eq. (3.10).

The remainder, and focus of this section, is the evaluation of the hypotheses against priors (the 2nd term ofeq. (3.10)). We consider priors to be functions that map hypotheses to scores, directly, without considering observations at all. The only product of tracking that is utilized during prior computation is the history of estimations, *i.e*. the tracked trajectory of 3D states itself. In eq. (3.10) this history is denoted by h.

Arbitrary feature spaces and priors can be incorporated in different substitutions of the prior evaluation process. The output is a score for each and every input hypothesis. This output can be computed across multiple cores, as computations are independent across hypotheses.

Adjacent finger interpenetration

The hypotheses themselves can be mapped to different feature spaces. To begin with, there is the original search space, a trivial feature space. Already in this space hypotheses may hold enough information to evaluate priors upon. For example, in [61,63,67], finger interpenetration was disallowed during tracking by inducing a penalty that was a direct function of the finger angles encoded in some of the 27 parameters of a hypothesis h:

$$L_0\left(h\right) = \sum_{p \in Q} -\min\left(\phi\left(p, h\right), 0\right), \tag{4.10}$$

where Q amounts to the pair of adjacent fingers and ϕ is the abduction-adduction difference, in radians, of adjacent fingers, excluding the thumb.

Collision detection

Another trivial feature space in this context is the decodings themselves, *i.e*. the collection of 4×4 transforms, along with the geometry references. This has been used in order to perform 3D collision checking on multi-entity hypotheses. More specifically, since each transform in a decoding is associated with a mesh, a collection of mesh instantiations can be generated, one for each corresponding transform. Then, it can be checked whether there exist any two such instantiations that occupy the same space and penalize for it.

In physics simulators, penetration can be quantified into a term called *penetration depth*. Penetration depth between two intersecting 3D shapes is defined to be the length of the minimal translation that either shape needs to undergo so that there is no intersection [37]. Physics simulators have built-in modules to compute penetration depths between analytical shapes. This includes spheres, cylinders, capsules, boxes and convex shapes. More elaborate shapes need to be decomposed into simpler ones, before penetration depth computations can be applied to them. The most dominant example is the decomposition of concave 3D meshes (the most generic case of shape) to compounds of convex parts that take up exactly the same volume. We perform this decomposition by employing the algorithm in [37] as implemented in the CGAL library [2].

We assume a function **PD** that can compute the penetration depth between two shapes. Then, in a collection of multiple shapes w, total penetration depth (**TPD**) can be defined

as the sum of all pairwise penetration depths:

$$\mathbf{TPD}\,(w) = \sum_{x,y \in w \wedge x \neq y} \mathbf{PD}\,(x, y). \tag{4.11}$$

Then, if **D** is a decoding function, a prior term that penalizes penetration can be defined:

$$L_1\,(h) = \mathbf{TPD}\,(\mathbf{D}\,(h))\,. \tag{4.12}$$

4.5 CPU Asynchronous components

Asynchronous are the processes that are executed out of synchronization. This implies that there is an execution thread, where synchronous processes are serially executed, and at least another thread, where asynchronous executions triggered from the main thread are performed.

We make use of asynchronous processes in order to efficiently *fetch* the next observations for our pipeline. Getting the next observations includes performing I/O to get images from cameras, files or network and also applying some image filters on them to perform the required feature mapping. In order to hide the latency these tasks introduce, we perform them asynchronously, *i.e*. these results are pre-computed or pre-fetched, from a dedicated thread, for every next tracking frame, while the current tracking frame is being computed.

4.5.1 Observation

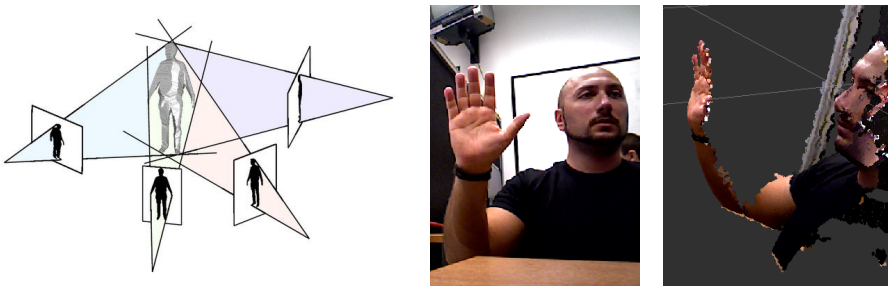
We consider intrinsically and extrinsically calibrated visual sensors that can deliver sequences of synchronized images. A set of images acquired from a set of such sensors at the same moment in time is called a *multiframe*. If $M_i = \{I_1, I_2, \ldots\}$ is a multiframe of a sequence $S = \{M_1, M_2, \ldots\}$ then I_j denotes the image from the j-th camera/view at the i-th time step.

These images usually contain intensity values (RGB) but could also contain other types of information. For example, a commodity nowadays is the use of RGBD sensors, *i.e*. sensors that provide two images, an RGB image and a depth image. The depth image contains depth measurements that are associated with the corresponding pixels in the RGB image.

For the purposes of this thesis, and since 3D information is required to be extracted, we have considered the use of multiple synchronized and calibrated cameras, but also the use of compact RGBD sensors. Multiple cameras provide complementary views of a scene that can be fused into 3D information. On the other hand, RGBD sensors provide 3D information directly, through their depth measurements. However, since this information is captured by considering only one view, 3D reconstruction is partial (see fig. 4.7). Nonetheless, through careful modelling, the exploitation of this partial information can still be very useful, as it will be shown in section 5.2.

4.5.2 Feature mapping

Observations, acquired as described earlier, contain everything the corresponding sensors can capture. So besides the entities in interest, they may capture the surroundings, which



(a) (b) (c)

Figure 4.7: Extraction of 3D information from camera setups. Multiple RGB images from different views (a) can be used to extract implicit or explicit 3D information. On the other hand, specialized and more compact sensors, like the Kinect, directly provide both color (b) and partial 3D structure (c), for one view at a time.

can be irrelevant or misleading, given a specific computer vision task. Moreover, the entities themselves may exhibit variability that makes them visually distinct despite them belonging to the same class. For example, clothing may induce complicating differences in the appearance of human bodies when the focus may lie upon reasoning about human bodies, regardless of clothing.

In order to filter out the irrelevant variability, and preserve only the important parts of an image, feature mapping is performed. More specifically, images are processed and transformed from the original space (RGB or depth) to other image spaces, like foreground masks, edge images, *etc*.

This pre-processing of images is also performed asynchronously, on the CPU, as it is usually fast enough to compute before the processing of the previous frameset is performed, but it is also undesirable to stall the main processing thread by waiting for it. The feature spaces and transforms considered in this thesis are detailed below.

Foreground detection

What visual sensors perceive can be coarsely classified between two classes, namely foreground and background. The foreground comprises all pixels which regard entities of interest. The rest of the pixels constitute the background. Thus, it is mainly a pixel classification process. To identify the foreground we have used three approaches.

Adaptive Gaussian mixture models We assume that a scene being captured by visual sensors is initially empty of interesting entities. Then, whatever is captured constitutes the foreground. As entities enter the scene they take up the image space previously allocated to background. We make use of the work of Zivkovic [118] in order to compute the state of every pixel, *i.e*. whether it belongs to the background or the foreground.

This approach operates by modelling the color variance of each pixel after a mixture-of-Gaussians distribution over RGB colors. Such models are learnt from a few images of the background, *e.g*. the very first frames of an image sequence of an empty scene. Then, for the rest of the images, all pixels are classified upon whether they *agree* with the learnt models or not. Agreement leads to classification as background and the opposite to classification as foreground (see fig. 4.8). An efficient GPU-powered implementation of this process has been provided by Tzevanidis [101], which delivers real-time performance.

(a) Background (b) RGB image (c) Foreground mask

Figure 4.8: Adaptive Gaussian mixture models for background subtraction [118]. A background is learnt (a) and then pixels of new frames (b) are classified upon whether they belong to the background or the foreground (c).

(a) RGB image (b) Skin mask

Figure 4.9: Skin color detection [4]. A simple skin color model is learnt in training and then used on images to detect skin coloured regions.

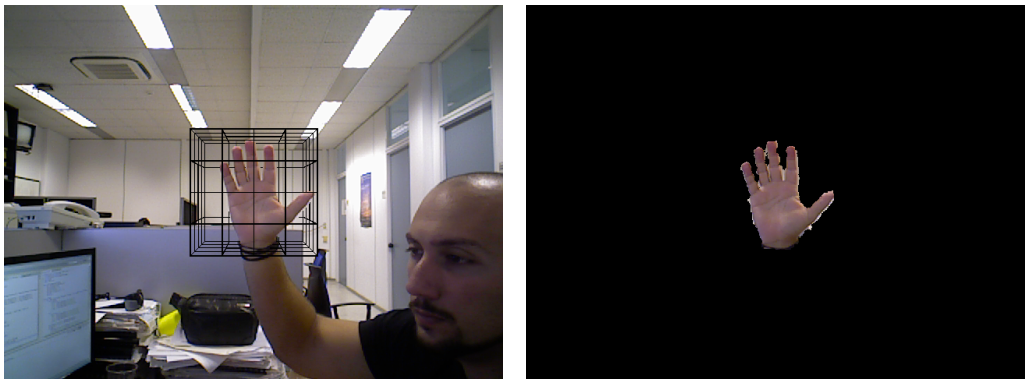
Skin color detection It is often required to identify human parts like the face or hands. What is characteristic of these parts is their color, *i.e.* skin color. Through training, an effective skin color classifier can be defined. We make use of the skin color detection algorithm of Argyros and Lourakis [4] to implement this process. This process requires an offline training phase and does not need any initialization frames in the online execution.

3D box classification The 3D information that RGBD sensors provide can be used to do 3D-based filtering. In the scenarios discussed throughout this thesis it is common that a human subject interacts with objects that lie on a table. This implies a working area that can be approximated by a 3D bounding box. For a pair of RGB and depth images, pixels, associated with depths, can indeed be filtered upon whether they correspond to 3D points that are contained in such a box, or not (see fig. 6.18(c)).

Edge detection

Another important feature regards the boundaries of entities. It is common that separable parts present discontinuities in the perceived intensities. These discontinuities can be identified by means of intensity differentiation. We are using the Canny edge detector, as implemented in the OpenCV library [11] (see fig. 4.11).

An important transform over edge images is the Distance Transform (DT) [34]. It



(a) RGB image (b) mask

Figure 4.10: 3D box classification. A 3D box is defined (a) and only pixels that correspond to 3D points that are contained in this box are regarded as foreground (b).

(a) RGB image (b) Edges (c) Distance transform

Figure 4.11: Edge detection. Boundaries of objects often concur with intensity discontinuities, or (b) edges. Distance transforms (c) are used to accelerate edge distance queries on images.

amounts to an image, that has the same size as the input edge map, where each pixel holds the distance to the closest edgel (see fig. 4.11(c)). It is computed efficiently through dynamic programming and is used to accelerate edge distance queries.

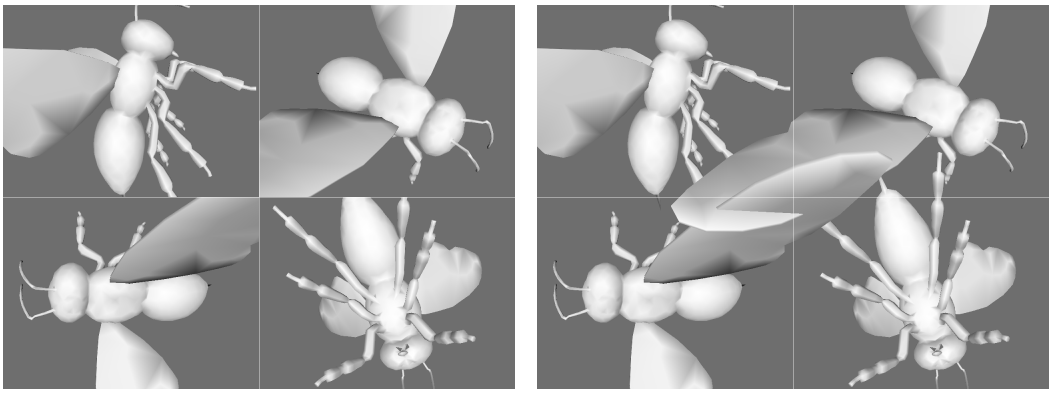
Depth measurements

A trivial feature space for the case of RGBD sensors regards depth maps. Depth maps are images which contain depth measurements, where each one corresponds to a color value in a RGB image, or, otherwise stated, the depth map is registered to a RGB image. Depths, along with camera specifications, facilitate 3D reconstruction (see fig. 4.7(c)).

4.6 GPU components

One heavy task in the presented framework regards the data term of eq. (3.10), *i.e.* 3D rendering of hypotheses and image comparison with observations. More specifically, at every tracking frame, entire generations, *i.e.* hypothesis batches, are rendered into 2D images, mapped to the same feature space as the observations and then compared pixel by pixel. All pixel comparisons are reduced to per-hypothesis scores.

These processes are computationally demanding because they regard great volumes of data, like geometry and pixels. GPU architectures have evolved so as to tackle exactly



(a) Tiled and clipped. (b) Tiled but not clipped.

Figure 4.12: Tiled rendering. The same geometry is multiply rendered, under different transforms or camera positions and on the same image but in different tiles. Geometry is contained in each tile (as in (a)), *i.e*. it does not *leak* into neighbouring tiles (as in (b)). This rendering is efficiently performed through *geometry instancing* and *multi-viewport clipping*. The separating white lines are not originally included, but are explicitly superimposed to convey the implicit boundaries of multiple tiles.

these matters efficiently. Therefore, we exploit GPU programming to accelerate appearance simulation, through 3D rendering, and evaluation of the data term, through Generic Purpose GPU (GPGPU) programming. We perform rendering by carefully customizing existing rasterization pipelines, like Direct3D 9© and OpenGL®. We perform GPGPU computations by adopting the CUDA© paradigm [60], which allows accelerations of parallel computations on NVidia GPUs with coding practices that successfully abstract away the architecture heterogeneity.

4.6.1 Tiled rendering

3D rendering of geometry is an established process. Driven by the demanding Computer Game industry, Computer Graphics have evolved algorithmically and in hardware support so as to constitute a luxurious commodity. More specificallystunning 3D graphics, which involve intense processing, can be produced at high frame rates and at a low cost.

The generic requirement for speed, which is a major factor in our context, is already well aligned with contemporary Computer Graphics. A GPU has many weak processing units, or cores, which can quickly perform, among others, most linear algebra computations involved in Computer Graphics. The number of cores is far less than the number of independent computations (*e.g*. per pixel). However, by issuing large batches of parallel tasks, to be executed on these cores, contemporary GPUs have the ability to appropriately schedule these executions so that no core remains idle at any point in time [60]. For example, when a core is executing a slow memory fetch on one context, another context that requires computations can be swapped and activated, while waiting the response from the memory unit. This interleaving capability yields the generic rule of thumb that *the more the parallel tasks the more the achieved acceleration*, compared to serial executions.

Towards this direction, and since we are considering batches of hypotheses at a time, we go from rendering one hypothesis at a time to rendering multiple hypotheses at a time. This rendering is output in tiled images, where all tiles have the same dimensions and each tile regards a different hypothesis (see fig. 4.12).

This unorthodox use of graphics pipelines generates issues that require special attention, as they are specific to our pipeline and diverge, more or less, from the common requirements in Computer Game Graphics. These issues are, namely, *geometry instancing*, *multi-viewport clipping* and *deferred shading*.

Geometry instancing

In the process of rendering multiple hypotheses there is a repetition of references to the same geometry. For example, in fig. 4.12, the same geometry is rendered four times, under different hypotheses over its rigid world transform. This has also been the requirement of the game industry, where usually scenes to be rendered contain multiple instances of the same geometry (*e.g.*, multiple trees, buildings, people *etc*.). Thanks to that, GPU rendering pipelines currently include devoted computational stages to efficiently perform such tasks.

We exploit this feature in order to quickly render multiple hypotheses that regard shared geometry. Under the Direct3D 9© pipeline we make use of *hardware instancing*. Under OpenGL®, we make use of *shader instancing*. For more details please refer to the 3rd chapter of [78].

The instanced geometry specifications used in this work are decodings, which comprise a mapping from mesh references to arrays of transform matrices, which are accompanied by hypothesis identifiers (see fig. 4.5). The hypothesis identifiers indicate the tile that is to contain each geometry instantiation.

Multi-viewport clipping

The term clipping refers to identifying and enforcing containment regions during rasterization (see fig. 4.12). Put plainly, clipping is used to contain rendering into given ranges, in the image plane or even in 3D. Traditionally, graphics pipelines perform image plane clipping for one viewport at a time. In that context, multi-view clipping could be supported by serially iterating over all desired viewports and rendering the corresponding content. However, this is in contrast to the goals described in section 4.6.1.

It is required that all geometry that regards a batch of hypotheses is rendered through geometry instancing, for the sake of efficiency. Thus, rendering should not be broken down into steps in order to perform clipping. To support multi-viewport clipping we modify both the vertex processing stage and the pixel processing stage, so that rendering is not broken.

At the vertex processing stage, and during instancing, we assign a viewport to each and every vertex, according to the issued specifications. During rasterization, the values that regard the viewport specifications at every vertex are interpolated across all generated pixels. However, for every rasterized triangle, the defining three vertices correspond to the same viewport, by construction, yielding a viewport for every pixel that is the same with the viewport of any of the corresponding triangle's vertices. Although this approach might seem as an overkill, it is necessary but also efficient.

After rasterization, and during pixel processing, each and every pixel carries a viewport specification. Then, it is trivial to check whether the image coordinates of these pixels are contained within the specified viewports. A fail in this check issues the abortion of outputting the pixel to the frame buffer. Thus, out of bounds geometry is never rendered (see fig. 4.12).

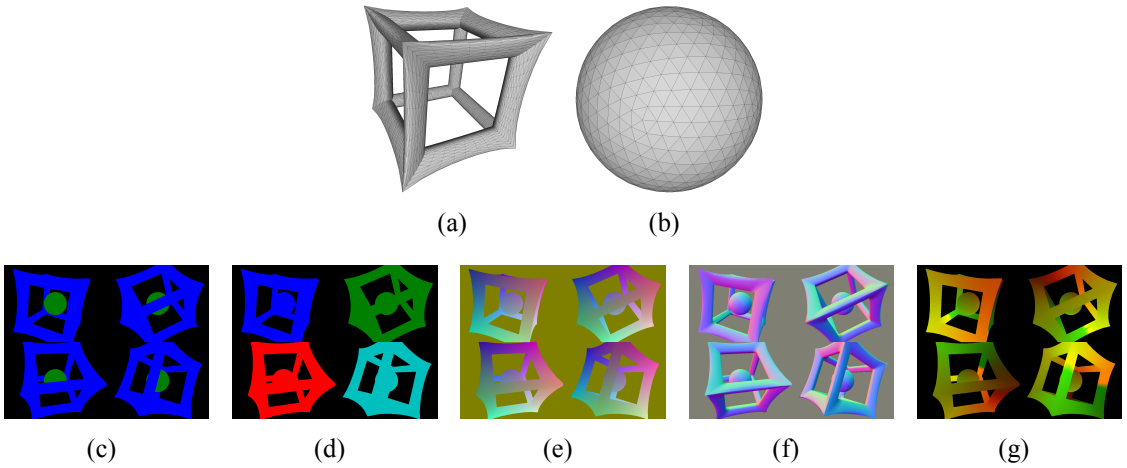


Figure 4.13: Deferred shading of 2 meshes into 4 tiles, with 2 instances in each. Two 3D triangular meshes ((a) and (b)), that come with per vertex normals and texture coordinates, are rendered into maps of primitives. In (c), the three distinct colors of the issue id map indicate that two meshes where used in the tiled rendering and that the entire image was not covered by geometry. In (d), the five distinct colors indicate that four instances of each of the two meshes was rendered (plus one color for pixels with no geometry). The 3D position that corresponds to every pixel is color coded in the position map of (e). Accordingly, the orientation of each pixel is stored in the normal map (f). Finally, the per vertex texture coordinates are rasterized into a single texture map (g).

Deferred shading

This framework is intended for use across different domains. As the domains vary, the notion and type of acquisition varies accordingly. Therefore, instead of providing specific acquisition models we provide the fundamental means of supporting arbitrary acquisition models.

In this direction, instead of producing fully specified renderings, *e.g*. photorealistic, we generate maps of rendered primitives, according to the practice of *deferred shading* [108]. These maps of primitives are results of rasterization and contain various types of per pixel information. From these maps and through composition, multiple types of renderings can be generated (see fig. 4.13).

In the work supported by the presented framework it has sufficed to provide the following maps of primitives: 1) issue id map 2) instance id map 3) 3D position map 4) normal map and 5) texture coordinate map. These maps are generated by appropriately programmed vertex processing and pixel processing units. The maps are efficiently exposed to the CUDA© framework, so that further GPGPU processing can be performed upon them.

Issue id map During rendering multiple 3D triangular meshes might be considered. After rasterization geometry occlusions have been resolved and only visible (by the camera) geometry is output. It is convenient to identify which parts of geometry has actually been rendered. Therefore, in the issue id map, we associate every pixel with a 1-based index to the geometry whose projection generated it. A pixel that does not correspond to projected geometry has an issue id equal to 0. We refer to the elements of this map as issue ids, as batched geometry is *issued* sequentially, and can thus be identified by an index. See

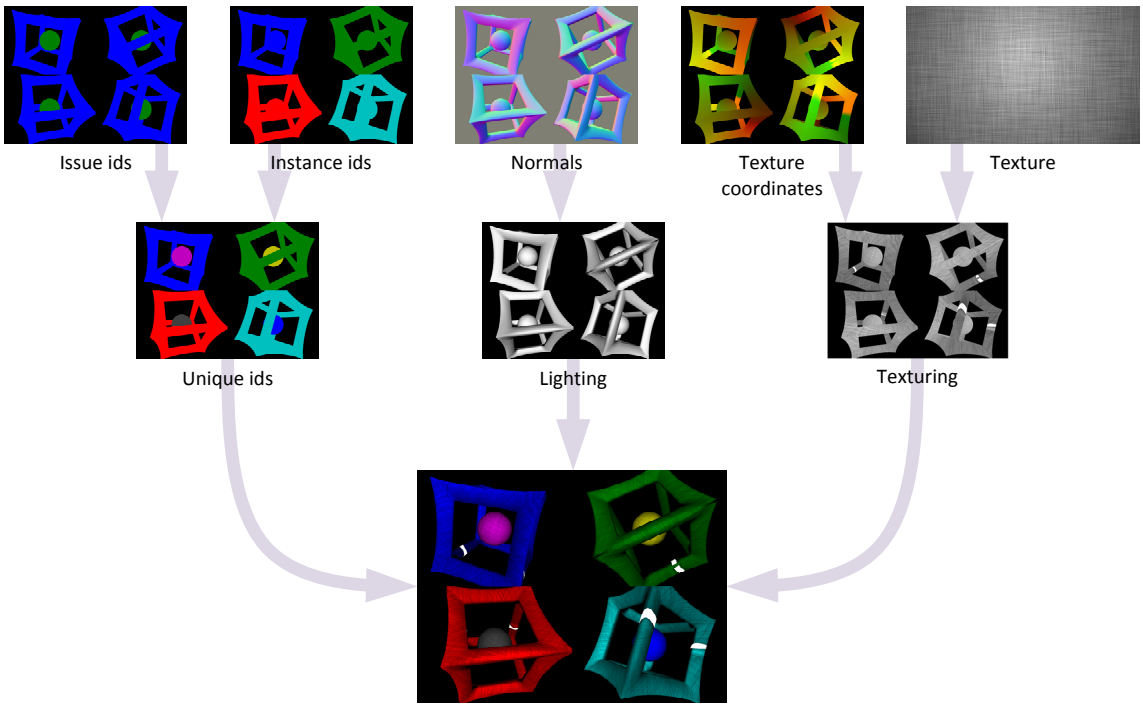


Figure 4.14: Composition of tiled rendering from maps of primitives (see fig. 4.13). A map of unique colors is created from the combination of issue and instance ids. Lighting is computed based on normals. Using a texture file and the texture coordinates the final texture for every pixel is computed. Finally, intermediate results are multiplied to yield the final rendering. All processes are pixel-wise and can be performed very efficiently on a GPU, using the CUDA framework. The independency in pixel computations relieves of the consideration of tiles.

fig. 4.13(c).

Instance id map Since geometry renders are issued in batches it is important to identify which instance of a given batch is responsible for a given pixel. Therefore, in the instance id map we assign pixels to 1-based indices that correspond to instances. See fig. 4.13(d).

3D position map Every rendered pixel is due to the projection of a 3D point in the surface of geometries to the image plane. For every pixel we efficiently store this information, that is represented as a 3D point with respect to the camera reference frame. A pixel with the $(0, 0, 0)$ 3D point is not associated with any projection of any geometry. See fig. 4.13(e).

Normal map Provided that a triangular mesh has per vertex orientation information, stored as 3D normal vectors, this information can be propagated to the final rasters. Each pixel is then associated with a 3D normal, *i.e*. a 3D normalized vector, which signifies the orientation of the surface belonging to the geometry responsible for the rendering of this pixel. This normal is defined with respect to the coordinate system of the camera. See fig. 4.13(f).

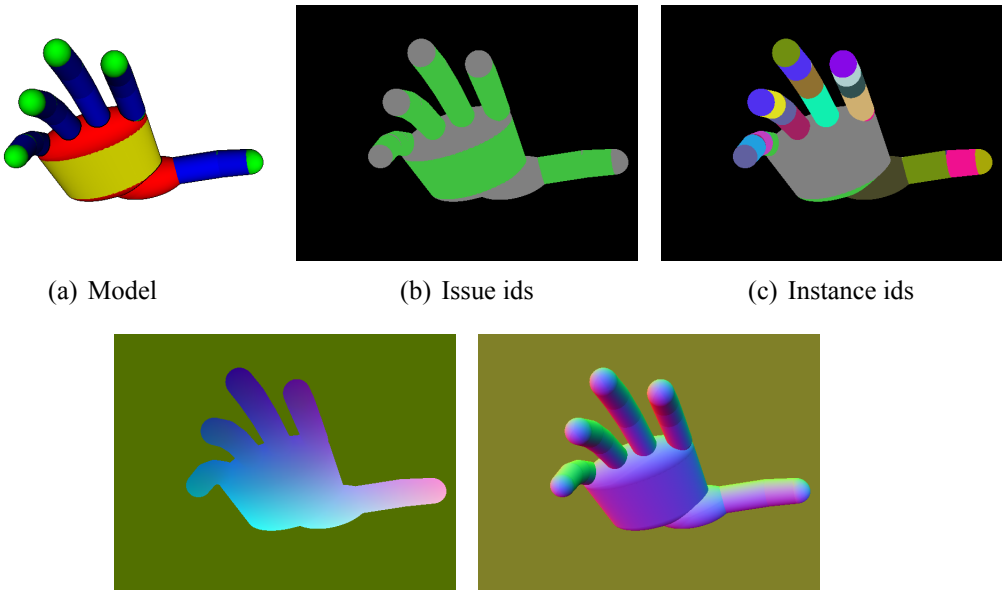
Texture coordinate map It is common that 3D triangular meshes themselves cannot be dense and detailed enough to convey realism. Therefore, they are usually *dressed* with images, whose intensity values are interpolated across the surface of the geometries, according to some normalized per-vertex texture coordinates. These coordinates are 2D normalized vectors that correspond to a position in a 2D texture image. Given that such information is provided it can be rasterized and thus propagated to a corresponding map. See fig. 4.13(g).

CUDA exposition What is most important in the rendering phase is computational efficiency. The aforementioned results can be efficiently provided by our rendering implementations and will reside in the memory of the GPU. Another GPU framework that is complementary to the Graphics pipeline is the CUDA© framework, whose computational platform has traditionally been the GPU. We have implemented, through the API provided by NVidia, an efficient interoperation scheme, where rendered data are efficiently provided to and processed by GPGPU CUDA-based processes.

4.6.2 Feature mapping

Rendering is employed in order to model the acquisition process. For renderings to be comparable with actual observations, the rendering products need to be transformed to the same feature space. This mapping is pixel-wise, *i.e*. computations for every pixel are independent to any other, and, therefore, the respective mappings are efficiently performed on the GPU, using CUDA©.

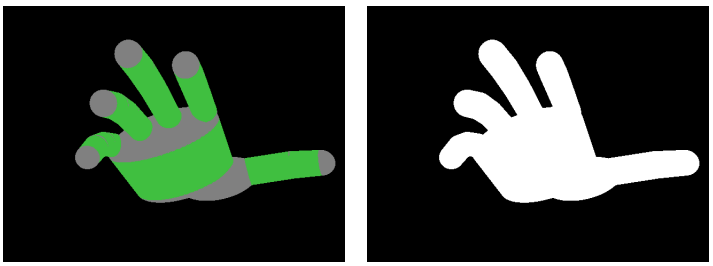
To illustrate this feature mapping procedure we are considering the example of 3D rendering of a synthetic human hand 3D model, at a given configuration (see fig. 4.15).



(a) Model (b) Issue ids (c) Instance ids

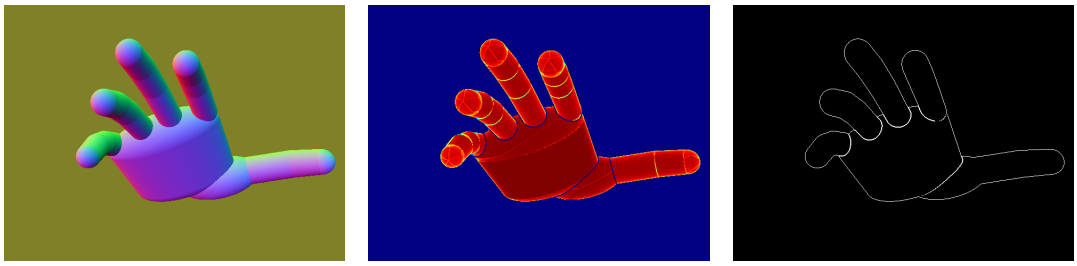
(d) 3D positions (e) 3D Normals

Figure 4.15: Deferred shading results for a given pose of a synthetic 3D model of a human hand. The model image (a) has actually been generated through composition of maps (b), (c) and (e).



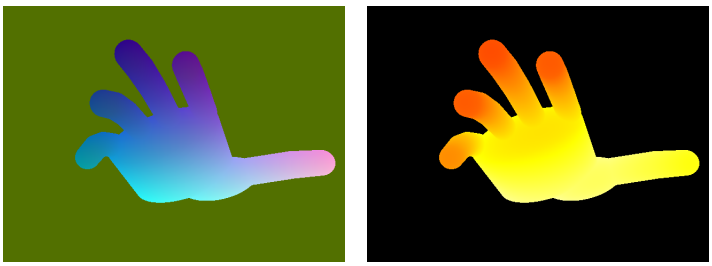
(a) Issue ids (b) Foreground silhouettes

Figure 4.16: By keeping pixels with non-zero issue id we identify the foreground silhouettes in renderings.



(a) 3D Normals (b) Discontinuities (c) Edges

Figure 4.17: A discontinuity intensity map (b) can be built from a normal map (a). The former is then easy to threshold in order to identify edges (c).



(a) 3D positions (b) Depth map

Figure 4.18: By keeping the z component of the position map (a) one directly gets a depth map (b).

Foreground detection

In a rendered 3D scene it is trivial to identify the foreground that through comparison would correspond to the detected foreground in actual images. In rendering, foreground silhouettes amount to pixels occupied by projected geometry. This is trivial to extract from any map of primitives, as they all encode non-rendered pixels. For example, foreground can be defined as the pixels with non-zero issue id, *i.e.* pixels that have been associated with geometry (see fig. 4.16).

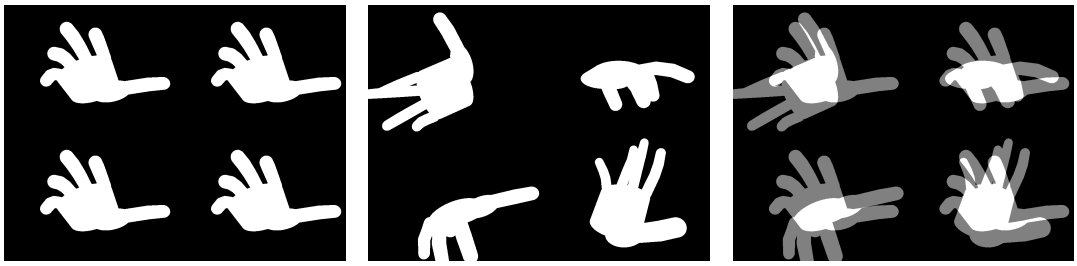
Edge detection

It is also straightforward to extract edges from 3D renderings. We assume that significant edges are due to structure discontinuities, and therefore try to extract edges from the mapped 3D structure.

In detail, we consider the surface orientation of the rendered entities (normal map) and extract edges, in screen space, through local differentiation of normals. This differentiation is performed on every normal, by measuring the mean dot product of the normal in question with each of its neighbourhood. By thresholding this result significant edges can be extracted (see fig. 4.17).

Depth measurements

In the special case where a RGBD sensor is available, structure itself is an important feature. RGBD sensors provide per pixel depth measurements. On the other hand, it is trivial to generate depth measurements for rendered geometry, by just keeping the z



(a) Observation silhouettes (b) Hypotheses silhouettes (c) Silhouette comparison

Figure 4.19: Silhouettes extracted from observations (a) are replicated in order to perform per pixel comparison with 4 hand configuration hypotheses (b). The logical conjunction (white) and disjunction (white and gray) are depicted in (c).

coordinate of the position map (see fig. 4.18).

4.6.3 Appearance differentiation

This process regards computations over the compatibility of observations and hypotheses, that are all mapped on the same set of feature spaces. For illustrative purposes, the differentiation between features that regard a synthetic human hand model will be regarded.

During differentiation, pixels in feature images of observations and hypotheses are compared into meaningful quantities. Processing is then fine-grained and tile-independent, and is thus suitable for straightforward GPGPU processing. This stage can be considered as the mapping phase in a *map-reduce* formalization [29] for GPU computing.

Since pixel-wise operations are discussed, we employ the notation $M\left(i,j\right)$ to refer to the value of M at coordinates i,j.

Silhouette differentiation

Foreground processes are considered as classification procedures, which classify each pixel according to whether it belongs to the foreground or the background. Given that classification is boolean, the results can be represented as logicals. Therefore, the respective per pixel operations are logical operations, *i.e.* conjunction ($\wedge$) and disjunction ($\vee$) (see fig. 4.19).

In notation, if S_o is the map of observed silhouettes and S_r is the map for the hypothesized ones, then two quantities are computed, namely a conjunction map $S_\wedge$ (intersection) and a disjunction map $S_\vee$ (union):

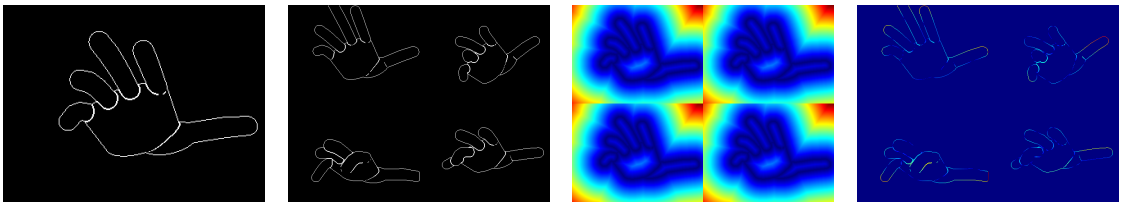
$$S_\wedge\left(i,j\right) = S_o\left(i,j\right) \wedge S_r\left(i,j\right) \tag{4.13}$$

and

$$S_\vee\left(i,j\right) = S_o\left(i,j\right) \vee S_r\left(i,j\right). \tag{4.14}$$

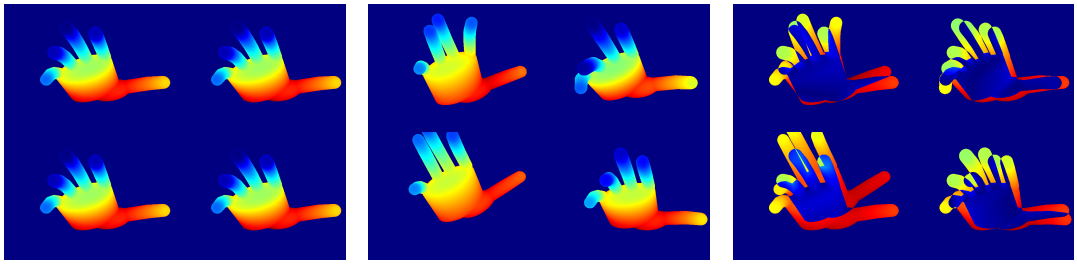
Edge differentiation

Edges have little occupancy over images. Simple edge overlap measurement yields low information gain due to lack of density. To introduce some density in the edge differentiation penalty we perform nearest neighbour queries. More specifically, for each edge



(a) Observation edges (b) Hypotheses edges (c) Distance transform (d) Distance coded edges

Figure 4.20: Through rendering, edges that are comparable to observed edges (a) can be produced (b). The distance transform is computed for the observed edges (a) and the result is tiled so as to match the dimension of the hypotheses rendering (c). By pixel-wise multiplication of (b) with (c) one can get the nearest edge distance for each of the edge pixels for the hypotheses (d).



(a) Observation depths (b) Hypotheses depths (c) Depth differences

Figure 4.21: Depths measured from sensors (a) and generated for hypotheses (b) are differentiated for every pixel (c).

pixel in the hypotheses we assign a penalty that is equal to the minimum distance from all edge pixels in the observations. To do so, instead of considering the edges extracted from observation, we consider the distance transform of the edge map. Then, per-pixel error is computed by multiplying edge occupancy in the hypothesis edge map with nearest edge distance, in the corresponding position of the distance transform (see fig. 4.20).

In notation, if E_o is the map of edges extracted from observations, then E_d is the distance transform of E_o, *i.e.* a map that in each pixel stores distance to the nearest edge pixel in E_o:

$$E_d\left(i,j\right) = \min_{x,y \wedge E_o(i+y,j+x)>0} \sqrt{x^2+y^2}. \tag{4.15}$$

Then, the hypothesized edges in E_r are modulated with E_d to yield the final map E_s:

$$E_s\left(i,j\right) = E_d\left(i,j\right) \cdot E_r\left(i,j\right). \tag{4.16}$$

Depth differentiation

When available, depth information is vital, since it complements the information in the remaining dimension, which is lost during projection. To quantify discrepancy in this feature space we simply compute per-pixel absolute differences (see fig. 4.21).

In notation, given an observed depth map D_o and a hypothesized depth map O_r, the

depth difference map O_s is defined as:

$$O_s(i,j) = \min\left(T_d, |O_o(i,j) - O_r(i,j)|\right). \tag{4.17}$$

Depth differences are clamped within the range $[0, T_d]$, so that overly large depth differences, due to noise or absence of measurements, do not dominate.

4.6.4 Reduction

The fine-grained parallelism in the pixel-wise operations in feature mapping is tile-agnostic and is directly and efficiently exploited by assigning each pixel pair to a distinct GPU core, through CUDA© programming.

Reducing the values for each and every tile to a per tile score is an elaborate task. Fortunately, the per tile reduction task can be mapped to a parallel segmented scan problem [87] for which a highly efficient implementation is provided, in the CUDPP library, using CUDA©. To reduce a tiled map using parallel prefix scan we first apply the process of *deblocking*, *i.e.* linearizing the per-tile storage, which is a parallel pixel-wise permutation.

We are using the inclusive prefix sum variant of parallel segmented scan, which is equivalent to the computation of cumulative sum, and we denote the computed totals with the $\sum$ operator, to identify per tile summations. Thus, it is only necessary to describe the per-tile reduction, as the same process is efficiently repeated for all tiles, in parallel.

Silhouette term

To produce a silhouette compatibility quantification, between observed and hypothesized silhouettes, we employ the *harmonic mean* $\bar{S}$ of the conjunction and disjunction silhouette areas $\sum S_\wedge$ and $\sum S_\vee$:

$$\bar{S} = 2\frac{\sum S_\wedge \cdot \sum S_\vee}{\sum S_\wedge + \sum S_\vee}. \tag{4.18}$$

This score reflects the simple idea that the compatibility between two silhouettes is proportionate to the amount at which their intersection matches their union. Intersection is important as it directly regards overlap. The union being similar to the intersection guarantees that silhouettes that just *contain* each other (same intersection) are not better than silhouettes that *match* each other (different union).

$\bar{S}$ is a normalized positive score of compatibility, in the range $[0, 1]$. It is turned into a normalized penalty D_s, in the same range, by:

$$D_s = 1 - \bar{S}. \tag{4.19}$$

Edge term

Edge discrepancy amounts to the sum of all distances from hypothesized edges to observed edges. This information is encoded in E_s whose sum yields total edge discrepancy:

$$D_e = \lambda_e \frac{\sum E_s}{\|\dim(E_s)\|}, \tag{4.20}$$

where, $\dim(\cdot)$ yields the dimension vector $(width, height)$ of its argument, λ_e is a rough estimate of the edge occurrence rate in an image and $\|\cdot\|$ is the Euclidean norm.

Depth term

The discrepancy between an observed depth map and a hypothesized one is plainly the sum of all pixel wise depth differences:

$$D_d = \frac{\sum O_s}{\text{prod}\left(\dim\left(O_s\right)\right) T_d}, \tag{4.21}$$

where, prod yields the product of all elements of its input, thus, prod $\left(\dim\left(O_s\right)\right)$ yields the area of O_s. Since the values are clamped we normalize the sum of differences according to the number of pixels multiplied by the maximum allowed depth distance.

Chapter 5

Tracking the parts

The scenes we are interested in involve several entities. Before we go deeper into tracking the interaction of these entities we go about modelling and tracking them, in isolation, one entity at a time. This process establishes an important foundation upon which inference, over the interaction of the entities too, can be performed and explained.

In this chapter we discuss how the presented computational framework (see chapter 4) can be instantiated in order to help track or estimate the motion of isolated entities, in 3D. The entities discussed here are the human hand and manipulatable objects. In section 5.1 we discuss the case of tracking a single hand, as observed from different sensing modalities and under a slight variation of contexts. Then, we discuss the incorporation of physics in 3D state inference and study the specific, yet significantly informative, case study of a bouncing ball (see section 5.2). Seeing these two seemingly distinct problems, in a common context provides the insight towards generalizing state estimation to multiple entities too.

5.1 Single hand 3D tracking

In a scene where a subject interacts with objects it is fundamentally important to understand its actions, as they hold information about the subject itself and the manipulated objects as well. As the dominant kind of manipulation is performed through hands, their tracking, in 3D, constitutes a significant part of scene understanding.

Human hands are difficult to track due to several cascading challenges. The kinematics of the human hand make it a complex structure that is hard to accurately model and represent. Its articulated configuration is defined over a large search space, which incommodes effective optimization, towards performing pose estimation. To make things worse, the uniformity in appearance and sub-structure re-occurrence (*e.g.* finger, phalanges) introduces ambiguities in observations, that hinder accurate, disambiguated estimation. The above are aggravated by the low resolutions at which human hands are captured, as they are part of broader scene observations and are viewed by sensors from significant distance.

What has been described here constitutes the main focus of the PhD of Oikonomidis [72]. In the joint work with Oikonomidis [31, 63, 63, 70], presented here, our contribution comprises the computational platform for the implementation of markerless model-based 3D hand tracking. Although our concern goes well beyond tracking a single hand, in 3D, still, this very specific case is very important in our context and is also well expressed in our architecture.

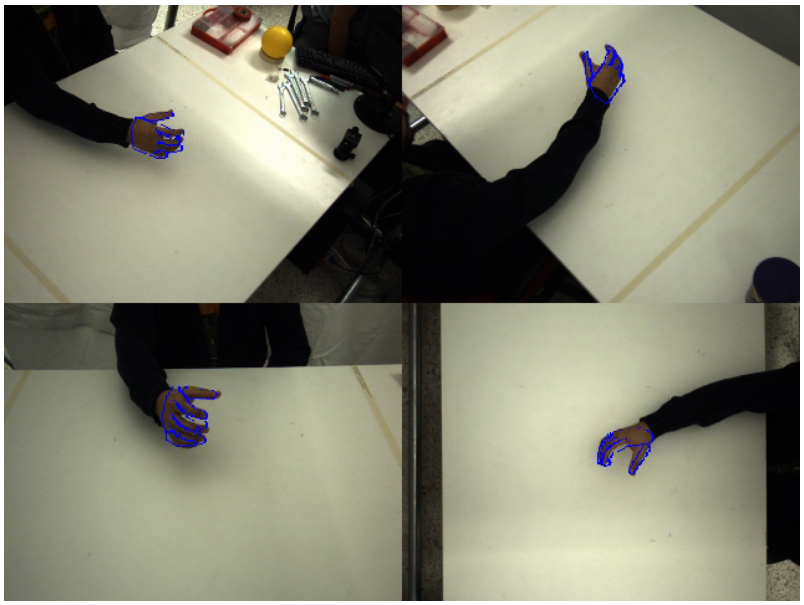


Figure 5.1: *Multi-camera 3D hand tracking*: Single hand tracking, in 3D, by exploiting the implicit 3D information provided by a multi-camera system [61]. All views are regarded disjointly, as the back-projection error amounts to a sum of multiple back-projection errors, one for each view. The optimal solution is superimposed, in blue edges, on the original input.

5.1.1 The problem

We are interested in tracking the pose and articulation of a human hand, in 3D, from sensors which generate 3D observations, implicitly or explicitly. We consider the two distinct cases of a multi-camera system (see fig. 5.1) and a RGBD sensor (see fig. 5.3), as two different sensor setups. Over these setups three variants of the 3D hand tracking problem are formulated, that correspond to works [52,61,70] (see fig. 5.1, fig. 5.2 and fig. 5.3), and which will be referred to as *multi-camera 3D hand tracking*, *visual-hull-based 3D hand tracking* and *RGBD 3D hand tracking*, respectively. Tracking the 3D pose and articulation of the observed hand, for every frame in a sequence, is initialized with an estimate of its state for the previous frame.

5.1.2 The solution

In this section the works in [61,63,70] will be presented. The rationale, which is common among all works, boils down to the principle that if it is possible to come up with a parametric forward model of how a hand looks through several sensor setups, then, this model can be turned into an inference mechanism, through black-box optimization, and thus the 3D hand tracking problem can be tackled. What differentiates the presented works is the selection over a sensor setup and a target feature space. As shown in fig. 5.4, all works share a great portion of processes and data.

Multi-camera systems and RGBD cameras can be used to implicitly or explicitly generate 3D information over observed scenes (see fig. 4.7). Given a 3D model of the human hand, its appearance, as seen through these sensors, can be predicted, by means of 3D rendering. Through a hypothesize-and-test approach, a hand configuration is sought, such that when compared to input images optimally matches hand observations.

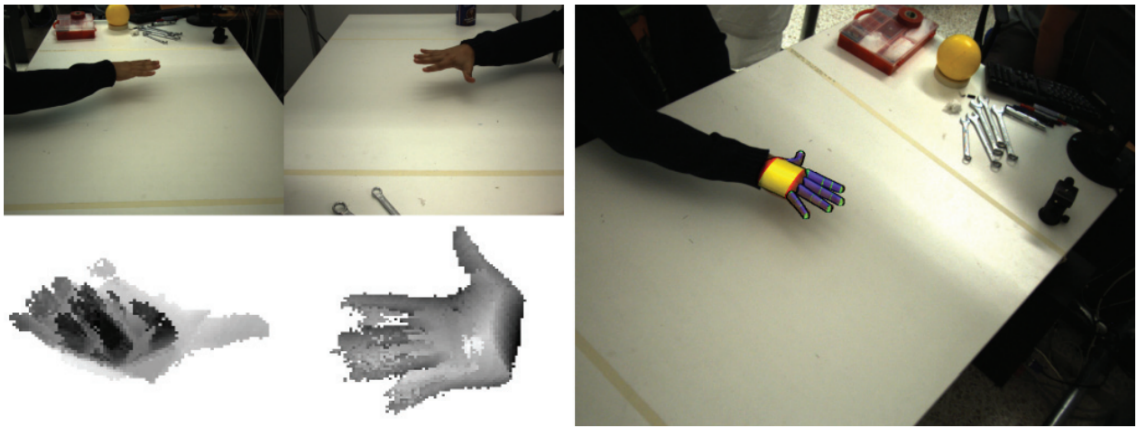


Figure 5.2: *Visual-hull-based 3D hand tracking*: Single hand tracking, in 3D, by explicitly exploiting the 3D information provided by a multi-camera system [70]. In a direct extension of [61], all views are regarded jointly, as they are used to estimate a 3D volume that corresponds to the visual hull of hand silhouettes. Hypotheses are also mapped to the feature space of volumes, where direct comparison with observations becomes available.

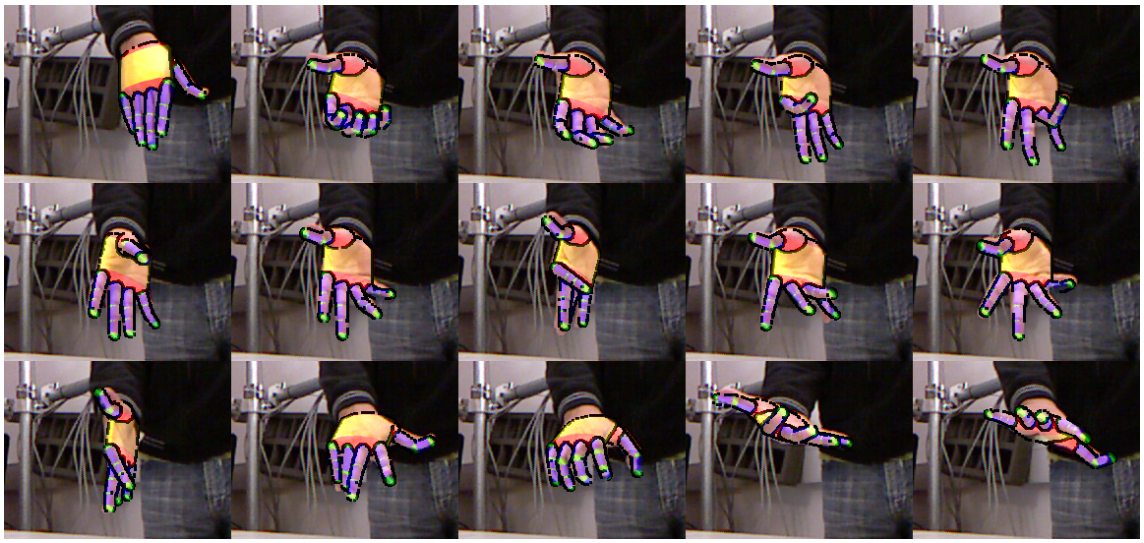


Figure 5.3: *RGBD 3D hand tracking:* Tracking a single hand from RGBD input (Kinect) [63]. Every RGB frame is associated with explicit 3D information (calibration and depth). The tracking solutions are superimposed on the original inputs.

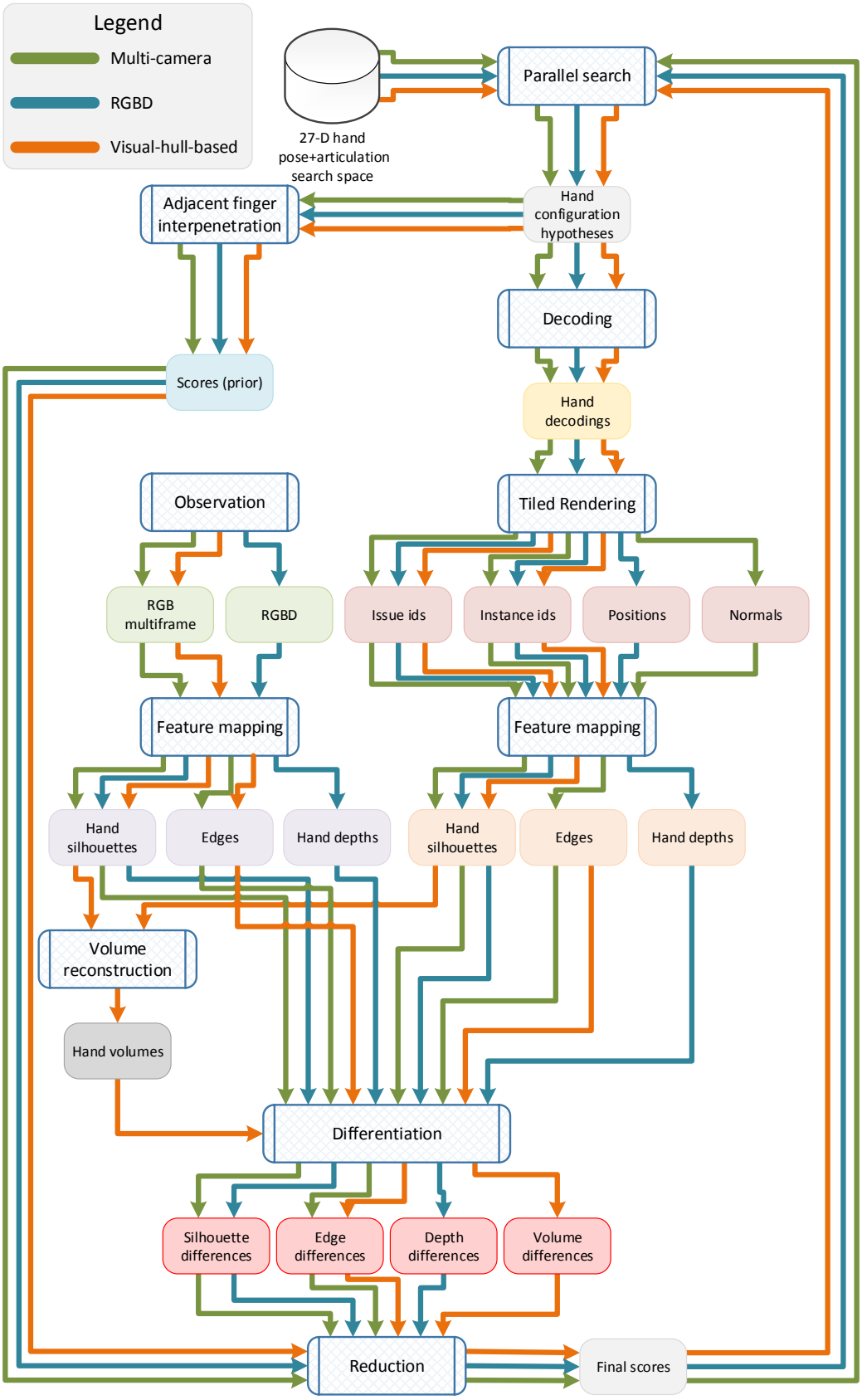


Figure 5.4: The instantiation of the outline in fig. 4.1 for the problem of tracking a single hand in 3D. Edge colors denote information flow which regards distinct implementations, over different scenarios, as presented in works *multi-camera 3D hand tracking* [61], *RGBD 3D hand tracking* [63] and *visual-hull-based 3D hand tracking* [70]. This figure can be read by selecting an edge color and following the corresponding flow (see legend).

3D hand tracking is formulated as an optimization problem, in accordance to section 3.3.2 and eq. (3.11). A parametric model of 27 parameters, which regards a hand's global position and relative articulation is fit into sequences of observations. Temporal continuity is exploited to constraint the search space for every new tracking frame so that optimization becomes tractable.

We describe the solution to the problem in terms of the computational framework presented in chapter 4. For the two sensing modalities three instantiations of the pipeline presented in fig. 5.4 will be detailed. Once more, the instantiations of the computational framework have as goal to implement eq. (3.11).

The observations that come from the calibrated multi-camera setup are sequences of 2D multi-frames. These multi-frames are mapped to the feature spaces of foreground (skin color detection) and edge detection, using the methods presented in section 4.5.2.

The observations that come from a RGBD sensor (*e.g.* Kinect) correspond to a RGB image and a depth map (see fig. 4.7). The RGB part is transformed into silhouettes by means of skin color detection. The depth map is masked with the silhouette generated from the RGB image, in order to filter out depths that do not correspond to non-skin parts, hopefully maintaining only measurements that correspond to the human hand.

For all cases, the hypothesis space is defined over the 3D pose and articulation of the human hand, and is thus of 27 dimensions. Points in this search space are transformed in renderable hypotheses by using the hand decoder (see section 4.4.1).

For the multi-camera and visual-hull-based cases, rendered hypotheses are mapped into the feature spaces of silhouettes and edges by exploiting the issue id (see section 4.6.2) and normal maps (see section 4.6.2). For the visual-hull-based case, silhouettes can be further transformed to the feature space of volumes, by means of visual-hull computations [100]. For the RGBD setup, silhouettes are also generated from the issue id map, and depth measurements are simulated by exploiting the position map (see section 4.6.2).

In multi-camera and RGBD 3D hand tracking, silhouettes are differentiated (see section 4.6.3) and turned into scores by computing the harmonic mean of the intersection area and union area, between observed and hypothesized silhouettes (see section 4.6.4). The same process, but for volumes rather than 2D areas, is employed for the visual-hull-based case, accordingly. In both cases, the generated scores are complemented with the edge discrepancy quantification (see section 4.6.3 and section 4.6.4). In the case of RGBD input, silhouette scores are accompanied by depth discrepancy computations (see section 4.6.3 and section 4.6.4). In all cases, the finger interpenetration penalty (see section 4.4.3) was employed.

In all variants, the optimization part of tracking was performed using PSO. In eq. (3.11), history of tracked solutions h was used to search for the next frame in the vicinity of the solution for the previous frame.

5.1.3 Experiments

For the multi-camera system the computational budget for each tracking frame amounted to 16 particles and 25 generations, which yielded a tracking throughput of 1.44fps. The visual-hull-based method employed 64 particles and 30 generations, leading to a tracking throughput that is $85\times$ slower (budget increase and additional processing steps) than the multi-camera approach. For the RGBD case, 64 particles and 25 generations were used, which yielded a tracking throughput of 25fps. Despite the $4\times$ increase in the number of objective function invocations, the RGBD method is more than $17\times$ faster, because

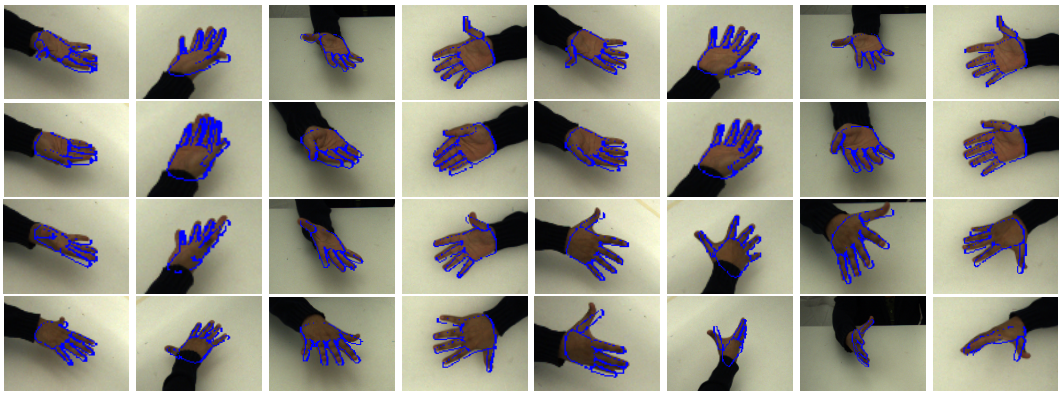


Figure 5.5: Exemplar results of *multi-camera 3D hand tracking* [61]. Each column corresponds to a different view/camera.

(a) each objective invocation for the RGBD case is much cheaper, due to the fewer amount of observation data, and (b) since each generation is evaluated in parallel, linear increases in budget lead to sub-linear increases in processing time. For more details please refer to [61, 63].

For a series of quantified analyses over the tracking methods the reader is referred to the original papers [61, 63, 70]. Here, qualitative results are presented so that the potential of the presented methods, in tracking 3D hand motion, is substantiated. In fig. 5.5, various results are shown for multi-camera 3D hand tracking [61], regarding different hand motions and different views. The tracking efficacy of the visual-hull-based method [70] is illustrated in fig. 5.6. A comparison that specifically targets the short baseline stereo case is depicted in fig. 5.7. Finally, results of RGBD 3D hand tracking over real sequences can be viewed at fig. 5.3. The entire video of these results can be viewed at `http://youtu.be/Fxa43qcm1C4`.

5.1.4 Literature review and contribution

For an extensive review of single hand 3D tracking the reader is referred to the thesis of Oikonomidis [72], or, even more specifically, to the joint work with Oikonomidis on single hand 3D tracking [31, 63, 63, 70]. In this context, we are interested on identifying how our hypothesis-and-test computational framework helped in advancing the state of the art, in markerless 3D hand tracking methodologically. This identification is performed via direct comparison with the state-of-the-art. We compare our approach against (a) methods which model the relation between the variables of the problem (*e.g.* variables which describe a 3D hand configuration) and the observations directly [25, 96] and (b) methods which learn this relation [83, 106]. Our approach falls within (a). In between stand methods which perform partly modelling and partly learning, like [9]. There are several aspects on which our approach is clearly differentiated. We do not discuss aspects which regard the quality of 3D hand tracking, as they are elaborated, favorably to our approach, in [31, 63, 63, 70, 72]. Instead, we discuss methodological traits, such as (i) ease-of-use, (ii) generalizability and (iii) speed.

In what regards the ease-of-use, 3D hand tracking is implemented, in our framework, as simply as defining an articulated 3D model (borrowed from [93]), in the form of a decoder (see section 4.4.1). Optimization is decoupled and delegated to PSO. Comparison with observations is implemented through simple image comparison (observations ver-

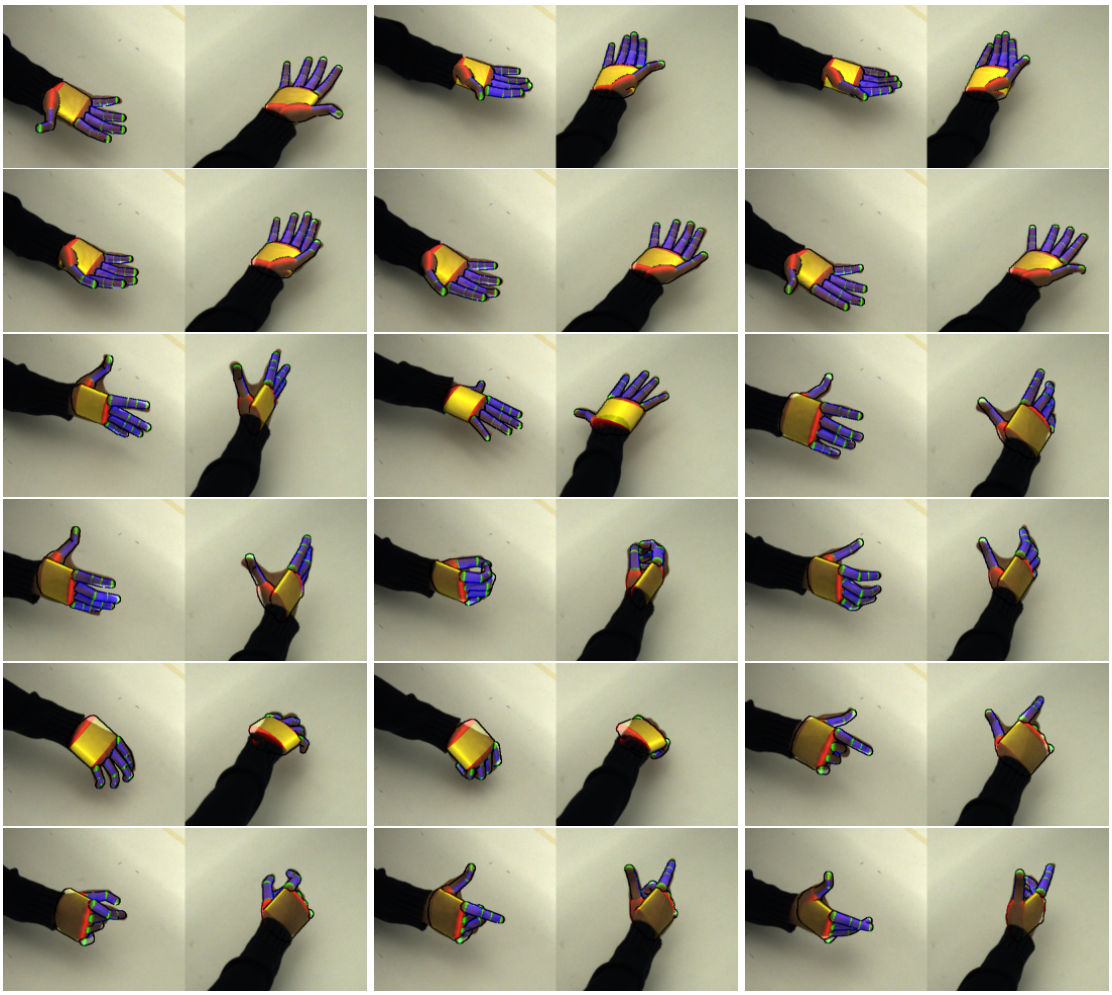


Figure 5.6: Exemplar results of *visual-hull-based 3D hand tracking* [70]. Each pair of images illustrates the same pose from different views.

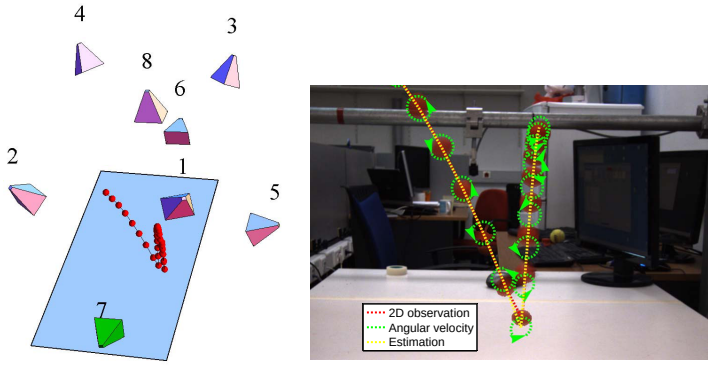
Figure 5.7: Exemplar results of *visual-hull-based 3D hand tracking* [70]. The superiority of [70] over [61], in the case of short baseline stereo is demonstrated. Left image pairs correspond to [70] and right image pairs to [61]. Each pair corresponds to a stereo view.

sus hypotheses). Although the ease of definition of a 3D hand model also holds for the methods in (a), the strong coupling of optimization with modelling introduces pragmatical hurdles. In detail, Sudderth *et al*. [96] take special care for finger occlusions and interpenetrations, in their filtering approach, by carefully scheduling special messages in their message passing schema. Handling of occlusions comes naturally and effortlessly in our framework, being directly incorporated in 3D rendering. Interpenetration avoidance is implemented as simply as adding a penetration penalty as a prior in eq. (3.10). De la Gorce *et al*. [96], followed a direct optimization approach, in which the success of non-linear optimization relied on the careful derivation of the objective function's derivative, which also constitutes the major contribution of this work. In our approach derivatives are not required at all. The approaches in (b) are, by construction, harder to set-up and evaluate, as they involve intense machine learning tasks (establishment of adequately large training sets, definition of features, learning from data, model selection, *etc*.).

As far as generalizability is concerned, a simple example can be used to highlight how our approach compares favorably to the state-of-the-art. Consider the generalization of tracking two interacting hands (see section 6.1), instead of one. As we show, in our framework, this is as simple to test as adding another hand in the decoding phase (see fig. 4.6). Experimentation has shown that doing simply and only this results in very effective tracking of two strongly interacting hands (see section 6.1). Adding another hand in [96] or [27] would also require the non-trivial revision of the strongly coupled optimization step. The introduced overhead is even greater for the training-based methods of (b), for which the entire process needs revision. This is becomes evident by comparing 3D tracking of a hand [84] against tracking a hand while in interaction with an object [83]. Another thing to note is that while the methods in (a) operate on the full continuous domain of the hand tracking problem, the methods in (b) are restricted to work in a discrete domain implicitly defined by the training sets.

With respect to speed, our fastest approach to tracking a single hand from a RGBD sensor [63] yields a performance of $25fps$. The model based approaches in (a) are reportedly far slower, while operating on monocular input. Of course, RGBD is far richer than monocular input, but, in methodological terms, our processing of more information is far faster than processing inputs of smaller sizes. Our RGBD 3D hand tracking approach is also faster than the methods in (b). However, as expected, there do exist training-based methods which operate far faster [46, 89]. Still, the former expectation that model-based approaches are prohibitively slow has been turned around in favor of our model-based, top-down approach, given the additional benefits described above.

Our works in [61, 63] were indeed the first ones to propose practical approaches to 3D hand tracking that surpassed in efficiency and accuracy the previous status. A standard was set for the quantified analysis in 3D hand tracking, using synthetic sequences, and the qualitative results demonstrated tracking success in scenarios of great complexity. It was also the first time that Particle Swarm Optimization (PSO) was used to tackle the 3D hand tracking problem, which introduced both accuracy and computational efficiency. PSO allowed for the decoupling of modeling from optimization, which facilitated the use of arbitrary models in 3D tracking. The parallel nature of the PSO variant employed allowed for efficient GPU-powered implementations of 3D hand trackers. The work in [70] introduced a better formulation of the 3D hand tracking problem, than [61], for the multi-camera case, by exchanging the feature space over pixels to a feature space over volumes. Despite the increased demands in computational resources, [70] presents a tracking performance that is similar to [61] for multiple cameras, but far better for the case of short



(a) (b)

Figure 5.8: A ball is thrown towards a table with a high back-spin. By incorporating physics-based simulation, we infer the ball's 3D trajectory (a), and its linear and angular velocities *from a single camera* (cam. 7). The proposed method identifies that a back-spin is the cause of the reduction of the outgoing angle of the bounce. The green ellipses in (b) are projections of an equator of the ball and the arrows represent the direction of the estimated angular speed.

baseline stereo, all under a concise and powerful formulation.

5.2 Physically plausible motion estimation of objects

In this work, we are interested in deriving a *physically plausible explanation* of a dynamic scene. Thus, the respective rules governing the generative process are the *laws of physics*.

We argue that by exploiting this type of context as a prior, we can derive very useful information for a dynamic scene, that is difficult or even impossible to derive by other means. Consider for example the scenario according to which we are interested in estimating the state of a uniformly colored bouncing ball through its observation by a single or by multiple calibrated cameras (see fig. 5.8). By employing standard computer vision techniques, accounting for the position of the ball at each time step is not trivial. The possibly inadequate acquisition frame rate may lead to aliasing and the possibly large shutter time may lead to motion blur. On top of the above mentioned difficulties, for some aspects of the state of the ball (i.e., its orientation and/or angular velocity) there is no direct evidence, whatsoever. The problem becomes even more challenging when we are interested in solving the above problems based on single-camera observations and/or when, due to occlusions, the available set of visual observations becomes even more limited.

We show that through the direct incorporation of explicit physics we are able to tackle these challenges. We demonstrate how hidden variables like the position of the ball when it is occluded, its orientation and angular velocities, can be estimated. We highlight that physics provide a strong prior, which permits the successful treatment of these challenges, even for the case of single camera 2D observations that may be incomplete due to occlusions. The incorporation of physics is performed in a clean, top-down fashion that could be generalized and scaled towards solving larger problems in different contexts.

5.2.1 The problem

Let a colored (red) ball be thrown on a table so that it bounces for several times and then rests. We consider the problem of estimating the motion of the ball. The 2D image position of the ball can be roughly recovered for every time step and for every camera that views it, for the case of moderate velocities. Accurate recovery is even more problematic for the case of larger velocities and especially around bounces, due to blurring and aliasing (see fig. 5.8). These problems hinder a bottom-up resolution of the problem, but, they do not prevent our top-down approach from being effective.

5.2.2 The solution

We treat the motion estimation problem as an optimization problem, which we relate to our computational framework in chapter 4. We consider the *physical explanation* e of the bouncing of the observed ball. We assume that certain scene properties (mass, inertia, collision properties) and initial conditions (position and velocities of the throw), together with the laws of physics, generate a 3D trajectory which optimally projects back to all cameras and matches the observations o. We define an objective function S that quantifies the discrepancy between the actual observations and the camera back-projection of a simulated parameterized ball throwing experiment. The latter can be sub-sampled to match the acquisition rate of the actual camera set. This also accounts for the aliasing effects of the acquisition process.

Since whatever is observed must be *physically plausible*, the *physical explanation* e is the minimizer parameter vector x of this objective function. The domain of x is defined in section 5.2.2. In notation:

$$e = \arg\min_{x} S(o, x) \quad \text{where} \quad S(o, x) = \text{BackProjectionError}\left(o, \text{Simulation}(x)\right) . \tag{5.1}$$

The original observations comprise an image array, whose one dimension corresponds to the number of cameras and whose remaining dimension corresponds to samples in time. Due to motion blur the ball does not appear clearly, therefore, image-based per-frame estimation of its state is problematic. We therefore consider all observations jointly. For each frame in the array, we perform simple color detection and connected components computations to identify red silhouettes. Each of the silhouettes is tested upon how well it approximates an ellipse, by computing the Mahalanobis distance of the silhouette's contour to the silhouette itself. The most elliptic silhouette is kept. The center of mass of this silhouette constitutes a 2D point in each frame, which we consider as the feature space.

The search space is defined over the parameters of a ball throwing simulation, that include material description and initial conditions. Then, a decoder transforms these parameters in the initial state of a physics simulation. Physics simulation is then performed to evolve the initial conditions through time and thus generate hypothetical 3D trajectories of the motion of the ball. These trajectories are projected onto virtual views, that correspond to the real cameras, and are thus converted into 2D trajectories, *i.e.* they are transformed into the same feature space with the observations.

No rendering is required in this process. Observations and hypotheses, transformed in the 2D trajectory feature space, are trivially differentiated to yield a score for each hypothesis. To optimize eq. (5.1) we use DE (see section 3.3.1). The information flow, as an instantiation of the framework of fig. 4.1, is illustrated in fig. 5.9.

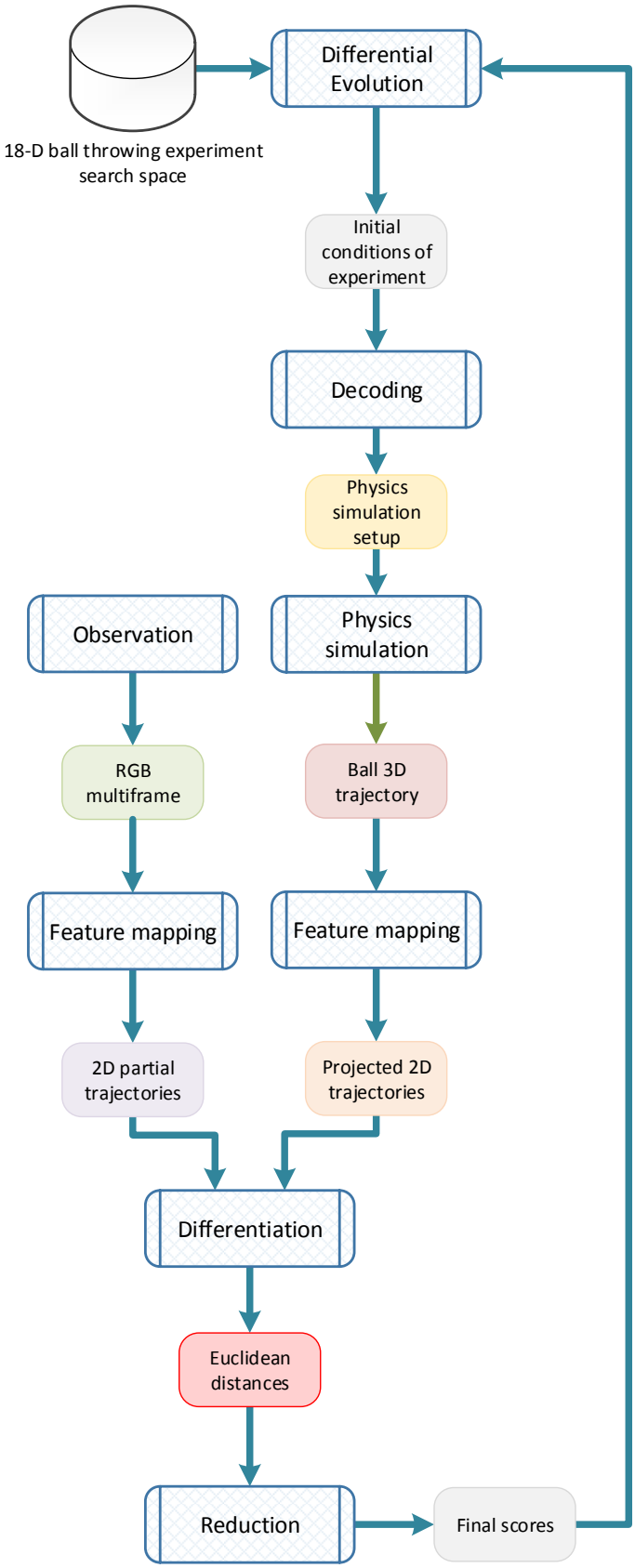
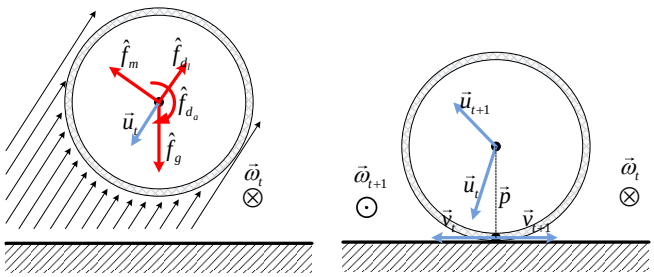


Figure 5.9: Information flow diagram of [52]. This is an instantiation of the framework depicted in fig. 4.1.



(a) (b)

Figure 5.10: The two phases of the bouncing ball (a) flight and (b) bounce. Red arrows represent impulses and blue arrows represent velocities. Angular velocities are perpendicular to the image plane. Black arrows represent the air flow with respect to the flying ball.

Physics of the bouncing ball

In order to account for the dynamics of the trajectory of a bouncing ball we explicitly reason upon an idealized, yet sufficient, physics model. We identify two alternating phases, namely ball flying and ball bouncing. The two phases are detailed in fig. 5.10. Since we consider average effects over generally small time intervals, we discuss impulses rather than forces. For a time interval dt and a function of force $\vec{f}$ over that interval, the respective impulse is defined as $\hat{f} = \int \vec{f} dt$.

During its flight (see fig. 5.10(a)), the ball undergoes velocity changes that are inflicted by the gravitational attraction and the resistance of the air. Gravity constantly exerts a downwards impulse of $\hat{f}_g = m \cdot \vec{g}$, where m is the mass of the ball and $\vec{g}$ is the gravitational acceleration. Given enough air resistance, at each time step t, the linear velocity $\vec{u}_t$ and angular velocity $\vec{\omega}_t$ are decreased in magnitude due to friction (linear damping d_l and angular damping d_a). Also, an impulse $\hat{f}_m = K \cdot (\vec{u}_t \times \vec{\omega}_t)$ that is perpendicular to the linear velocity and the axis of angular velocity, makes the ball travel in a curved trajectory [5]. For every part of the flight the standard equations of motion hold and suffice in order to predict the state of the ball.

At every bounce (see fig. 5.10(b)), a portion of the ball's energy is lost according to an elasticity factor $\beta \in [0, 1]$. An amount of dynamic friction redistributes energy between its linear and angular motion in the horizontal, according to a friction factor $\alpha \in [-1, 1]$. The friction model adopted here is an extension of [6] to the 3D case and identifies friction as the reason that scales the total linear velocity $\vec{v}_{t+1}$ of the contact point. This modeling accounts for a great variety of frictions. For example, both the glass ball (no friction) and the super ball (extreme friction) can be modeled for $\alpha = 1$ and $\alpha = -1$, respectively. The vertical axis of rotation has no contribution to the horizontal contact. Moreover, the impulse which changes the horizontal linear momentum is also the negative impulse that changes the horizontal angular momentum. All the aforementioned constraints define a system of equations:

$$\begin{array}{lcl} S_y \vec{u}_{t+1} & = & -\beta S_y \vec{u}_t \\ S_y \vec{\omega}_{t+1} & = & S_y \vec{\omega}_t \\ S_{xz} \vec{v}_{t+1} & = & \alpha S_{xz} \vec{v}_t \\ m \cdot \vec{p} \times S_{xz} (\vec{v}_{t+1} - \vec{v}_t) & = & -I \cdot S_{xz} (\vec{\omega}_{t+1} - \vec{\omega}_t) \end{array} \quad \text{with} \quad \begin{array}{c} S_y = Diag([0,1,0]) \\ S_{xz} = Diag([1,0,1]) \\ \vec{v}_k = \vec{u}_k + \vec{p} \times \vec{\omega}_k \end{array} \tag{5.2}$$

These equations linearly relate the pre-bounce velocities $\vec{u}_t, \vec{\omega}_t$ to post-bounce velocities $\vec{u}_{t+1}, \vec{\omega}_{t+1}$. Solving this system for time $t+1$ yields the post-bouncing state of the ball.

Physics-based simulation

Ubiquitous physics simulators are already able to account for the most part of the presented physics modeling. They also ease the incorporation of more detailed models via modular architectures that are carefully designed for that purpose. In our scenario we augment such a simulator by incorporating the effects of $\hat{f}_g$ and $\hat{f}_m$ and by adjusting the collision module so that it also accounts for the exchange of horizontal energy. The vertical is already in agreement with our equations. The parameter vector of a simulation is $(m, I, \beta, \alpha, d_l, d_a, K, \vec{s}_0, \vec{u}_0, \vec{\omega}_0, T)$. The state of the ball $\left(\vec{l}_t, \vec{q}_t, \vec{u}_t, \vec{\omega}_t\right)$ is the result of the invocation of the simulator for a given parameter vector, where t is a time step of the whole duration T and $\vec{l}_t$, $\vec{q}_t$, $\vec{u}_t$, $\vec{\omega}_t$ represent position, orientation, linear and angular velocity (all in 3D space).

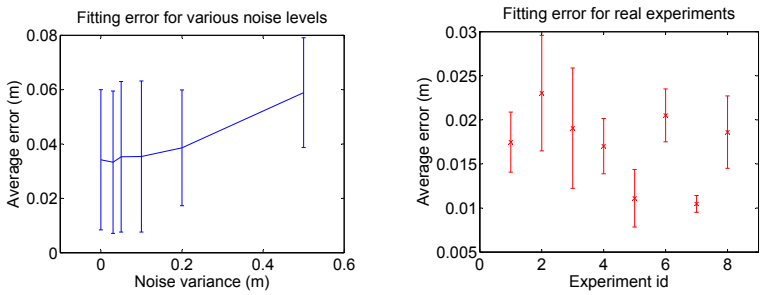
5.2.3 Experiments

A series of experiments were conducted to assess our method's ability to account for 3D and 2D observations of a bouncing ball. From an implementation point of view, we used the DE implementation of the SwarmOps[1] library, the Newton Game Dynamics[2] simulator and the MATLAB[3] platform for the rest of the logic. A commented video summary of the performed experiments can be found at `http://youtu.be/Lr5wq5It4io`.

Results on synthesized image sequences

A first series of experiments were carried out to assess the capability of the proposed method to come up with physically plausible explanations of various simulated ball throws, performed in different world contexts and initial conditions. We distinguished the parameters representing scene properties $(m, I, \beta, \alpha, d_l, d_a, K)$ and those representing initial conditions $(\vec{s}_0, \vec{u}_0, \vec{\omega}_0)$. We generated 3 random scene property parameter vectors and 3 random initial condition vectors. We then considered all possible combinations resulting in a total of 9 experiments. The experiment parameterizations generated 9 ball 3D trajectories for a time duration of $T = 4s$, each. Each 3D trajectory was considered in conjunction with 6 levels of Gaussian noise at each of the 3 spatial dimensions, separately and with variances $0m$, $0.03m$,$0.05m$,$0.1m$, $0.2m$ and $0.5m$, respectively. For each set of parameters, 20 repetitions were executed. This protocol led to a total of $3 \times 3 \times 6 \times 20 = 1080$ experiments accounting for various world properties, initial conditions and amounts of noise. For each experiment, the physics-based simulator produced a ground truth 3D trajectory of the ball and the proposed method was employed to provide a physical explanation of it. We evaluated the optimization accuracy by measuring, for each experiment, the average of the Euclidean distances between corresponding points of the simulated and the recovered 3D ball trajectories. The results presented in fig. 5.11(a) show that the proposed method is able to perform well even under severe Gaussian noise. This is because by conception, the method allows for physically plausible solutions, only. Thus, observations

[1] http://www.hvass-labs.org/projects/swarmops/c/
[2] http://www.newtongamedynamics.com
[3] http://www.mathworks.com/products/matlab/



(a) (b)

Figure 5.11: The mean values and standard deviations for the errors on the experiments with (a) synthetic and (b) real observations.

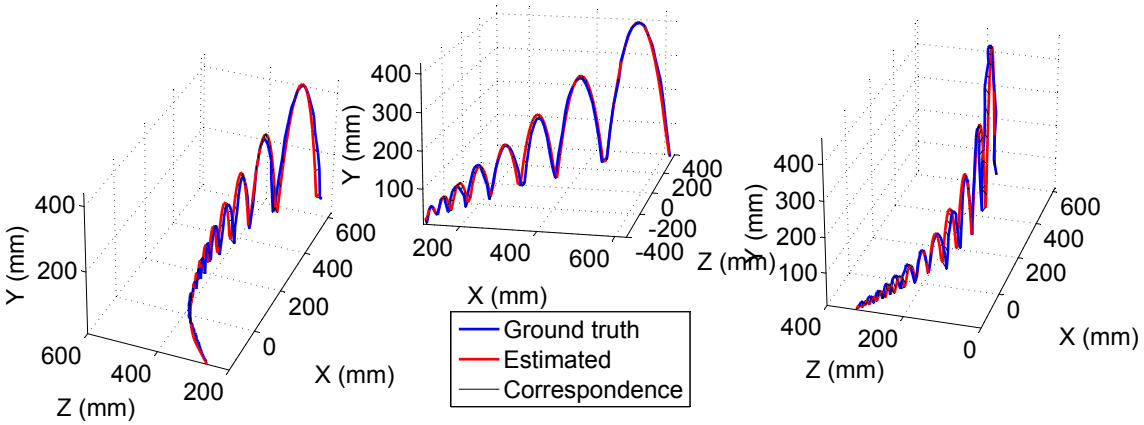


Figure 5.12: Examples of actual and estimated 3D trajectories. The illustrated trajectories have high left, low left and high right curvature, respectively. The respective average trajectory point estimation errors were $1.38cm$, $0.75cm$ and $0.83cm$.

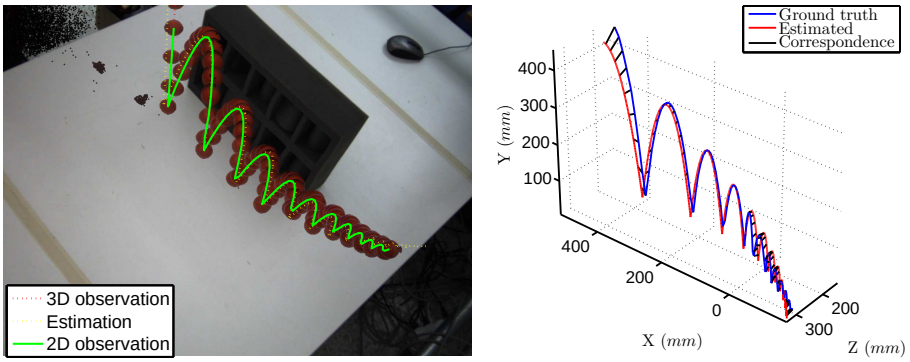
that are heavily contaminated by this type of noise cannot distract the estimation towards physically implausible solutions.

Multiview estimation of 3D trajectories

Experiments analogous to those of section 5.2.3 were also performed in the real world, i.e., using multicamera observations of an actual bouncing ball. For these experiments, we employed the setup that is illustrated in fig. 5.8(a). It consists of a $2 \times 1m^2$ hard table, a red table tennis ball of radius $2cm$, and 8 synchronized and calibrated Flea2 PointGrey cameras. All cameras provided images at a resolution of 1280×960 and at an acquisition rate of $30fps$. Processing was performed on a workstation that has an Intel Core i7 950 CPU @ $3.07Ghz$ and $6GB$ of RAM. All computations were performed on a single thread of a single core of the CPU.

As a first step, the ball was detected in every frame for all sequences (all cameras inclusive). We applied color thresholding to isolate red areas in every frame. We then filtered each extracted connected blob based on its shape, to ensure high confidence detection and excluded partially occluded and/or significantly blurred detections. 3D ball positions were then estimated through multiview 3D reconstruction of the ball centroids.

We conducted several ball throwing experiments in our physical setup. We selected 8 of them according to our empirical criterion of diversity. Each one was input to our method



(a) (b)

Figure 5.13: Estimation of the 3D trajectory of a ball from single camera 2D observations (camera 3) and the assumption of a given physical world.

20 times. The optimization process of section 5.2.3 was employed, where synthetic data were replaced by real data. The obtained results are presented in fig. 5.11(b). Some examples of actual trajectories and the respective estimates are shown in fig. 5.12. As it can be verified, the proposed method faithfully reproduces the actual 3D observations.

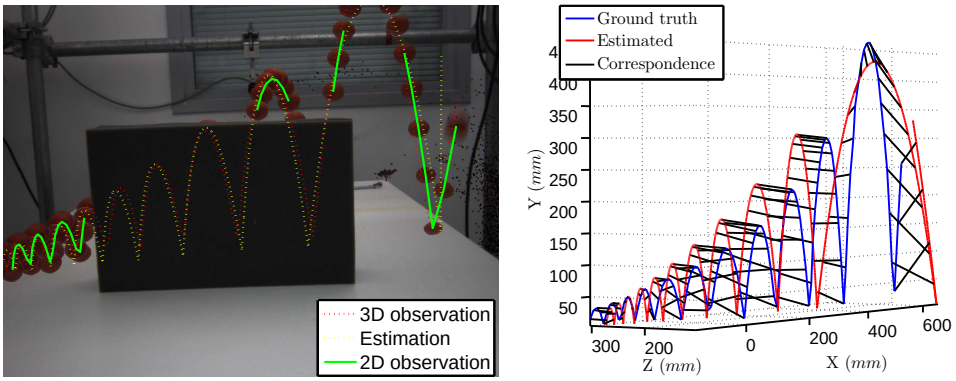
Single view estimation of 3D trajectories

We are interested in estimating the ball 3D trajectory by a single camera. Without any physics-based prior information and for the case of a ball of known size, single view ball 3D localization depends on the ability to accurately estimate the ball's projected shape and size. In practice, this is problematic due to acquisition and processing artifacts, which lead to errors in depth estimation that are difficult to treat in a bottom-up fashion. However, by modeling the physics of the process, we are able to infer depth from a more reliable source, the 2D trajectory of the ball on the image plane of a single camera. To demonstrate this, we optimized S for 2D observations of a single camera and the non trivial cases of non-planar (due to spin) trajectories. During optimization, the simulator generated 3D data from which 2D reference trajectories were produced by means of projection. The back-projection error, i.e., the average Euclidean distance between back-projected and observed 2D positions of the ball was guiding the optimization process. An exemplar 3D estimation is illustrated in fig. 5.13. It can be verified that the estimation from a single camera is almost indistinguishable from the ground truth.

Interestingly, no post-processing is required to enforce the plausibility of the solution because implausible hypotheses are not considered at all. Even more importantly, even though 3D estimation from 2D trajectories relies on the knowledge of the respective ball heights, we do not account for this knowledge explicitly. 3D reconstruction comes effortlessly, as a byproduct of physics-based simulation. Another interesting observation is that the points at which bounces are observable suffer from aliasing. However, since we also sub-sample the simulator's behavior at real acquisition rate, we also account for this type of aliasing.

Seeing through walls with a single camera

We also tested our method's effectiveness under considerable lack of constraints, i.e., in the case of partial observations due to occlusions. We recorded ball throws that were



(a) (b)

Figure 5.14: Single view estimation of the 3D trajectory of a ball from partial 2D observations (camera 6) and the assumption of a given physical world.

largely invisible to camera 6 (see fig. 5.8(a)) due to a purposefully placed large obstacle (see fig. 5.14(a)). Based on this partial evidence, the proposed method estimated the 3D trajectory of the ball from the (single) view of camera 6. The ball was still visible to some of the rest of the cameras. This information was only used to estimate a kind of ground truth for the 3D trajectory of the ball. Figure 5.14 shows the actual observations, the ball trajectory as this was estimated by camera 6 and the ground truth as this was measured by the rest of the cameras. The estimation of the 3D trajectory (see fig. 5.14(b)) is not that accurate due to the lack of enough constraints. Still, it is quite satisfactory given the fact that it has been obtained through single camera observation and in the presence of occlusions.

Inferring angular velocity

The ball's angular velocity cannot be estimated by any direct vision method in any of the considered experiments. However, evidence regarding this parameter is encapsulated in the overall dynamic behavior of the ball. By seeking for a physically plausible explanation of the observed scene, the proposed approach reveals, as a byproduct, information regarding the latent variable of angular velocity.

We performed a series of ball throws with high back-spin, so that the ball resists its original tendency to move forward (see fig. 5.8). We then optimized S for the resulting 2D observations of camera 7. An exemplar bounce is shown in fig. 5.8(b). The proposed method inferred the 3D state of the ball accurately (once more, ground truth and estimated ball positions are indistinguishable). Moreover, as demonstrated in fig. 5.8(b), we were able to compute a qualitative measure of the ball's angular velocity.

5.2.4 Literature review and contribution

We show that through the direct incorporation of explicit physics we are able to tackle the challenges in section 5.2. We demonstrate how hidden variables like the position of the ball when it is occluded, its orientation and angular velocities, can be estimated. We highlight that physics provide a strong prior, which permits the successful treatment of these challenges, even for the case of single camera 2D observations that may be incomplete due to occlusions. The incorporation of physics is performed in a clean, top-down fashion

that could be generalized and scaled towards solving larger problems in different contexts.

In the past, several researchers have stressed the benefits stemming from the consideration of physics as integral part of computer vision processes. Although beyond the scope of this work, we mention approaches [22, 41, 58, 117] that exploit the physical nature of light to process images and estimate or predict otherwise unaccountable information.

The prevalent case study of employing physics in vision is the problem of 3D human tracking. The dynamics of the human body can be exploited towards the formation of strong priors. Popović and Witkin [79] rectified 3D motion capture data to make them compliant to physical constraints. Vondrak *et al*. [104] fused motion planning, contact dynamics and a ground assumption to track humans from multiple cameras. Brubaker *et al*. [14–16] employed realistic metaphors of the lower body dynamics to estimate and predict walking. Going further, they incorporated a friction model for a ground that affords human motion upon it [17].

There are also approaches that reflect physics implicitly or metaphorically. Brand *et al*. [12, 13] exploited the physical notion of causality in order to perform qualitative reasoning in computer vision problems. Delamarre [30] assigned a physical behaviour to a contour model that drove the optimization process of recovering it. Chen *et al*. [19] were able to track a basketball, while in the air and despite occlusions, by assuming the parabolic trajectories of free flight. Papadourakis and Argyros [74] identified the physical notion of object permanence as the ambiguity resolver for the case of multiple objects tracking. Sethi and Roy-Chowdhury [88] gave physical substance to image features and used methods, usually employed in physics, in order to model activities in image sequences.

This work is most closely related to the works in [10, 33, 56]. Metaxas and Terzopoulos [56] defined a continuous Kalman filter that was able to track a deformable object. This was achieved by a detailed motion model that was tightly coupled to the optimization process. Although interesting, the extensibility of their approach is hindered by this tight coupling. Bhat *et al*. [10] performed 3D tracking of an object by searching over parameterized experiments that optimally project back to an image sequence. However, the shape of the object and the restriction that it is tracked while in flight does not expose the full potential of employing physics. Finally, Duff and Wyatt [33] used physical simulation and search heuristics to track a fast moving ball, despite occlusions. They reasoned upon the ball's 2D position but they did not consider the 3D case, or the hidden variables of ball orientation and angular velocity.

Despite the significant amount of existing work, no existing study demonstrates the full potential of binding vision to physics-based simulation. We try to fill this gap by proposing a method that is generic, top-down, simulation based and incorporates realistic simulation of physics. As a result, and to the best of our knowledge, the proposed method is the first to consider physical properties that can be estimated through physics-based simulation, even in the case of single camera observations and severe occlusions.

Chapter 6

Tracking the whole

The study in the previous chapter over tracking the parts forms the basis upon which tracking the whole is established. Tracking the whole means tracking all parts of a scene, as well as their interaction. The complexity of the interaction grows quadratically with linear increases in entity counts. The interactions themselves (collisions, occlusions, *etc*.), as well as their implications (problem size, scalability) require special attention.

Our first step towards tracking big scenes regards a hand plus another entity (a manipulated object [65] or another hand [67], see section 6.1). We then try to find an affordable generalization to scenes that contain far more entities, while they all interact [49] (see section 6.2). Finally, we identify a special yet common case of interaction which allows to model complex scenes (a hand manipulating many objects while they all interact) compactly and efficiently (see section 6.3). Interestingly, all approaches presented here are instantiations of the computational framework presented in chapter 4.

6.1 Tracking two interacting entities

In section 5.1, the ability to track a single hand and a single object, in isolation, was established. Moreover, the incorporation of physics as a complement in the lack of visual evidence was discussed (see section 5.2). However, it is rarely the case that hands and objects appear disjointly in the scenes upon which understanding is required to be performed. The most common, and also most interesting case, is the interaction between hands and other entities. In this section we discuss the problem of tracking a hand while in strong interaction with another entity.

It was discussed in section 5.1 that tracking the human hand is not an easy task, even from rich 3D input. Most issues originate from the high complexity of the hand's structure, which introduce ambiguity in explaining observations. As soon as the hand interacts with other objects, more difficulties and challenges arise. An object being handled by a hand is occluding the hand and the hand also occludes it. Thus, the information that regards entities in isolation is overall reduced. In the case of the manipulated entity being another hand, even more issues arise due to the introduced ambiguity. When viewing a scene of two hands, in tight formation, it is hard to tell, even for humans, which finger belongs to which hand. In both cases, the size of the search space is increased, which makes optimization harder.

However, the interacting entities need not be considered disjointly. On the contrary, our proposal is their joint consideration. What translates as reduction of input during the disjoint consideration of the interacting entities actually constitutes additional information

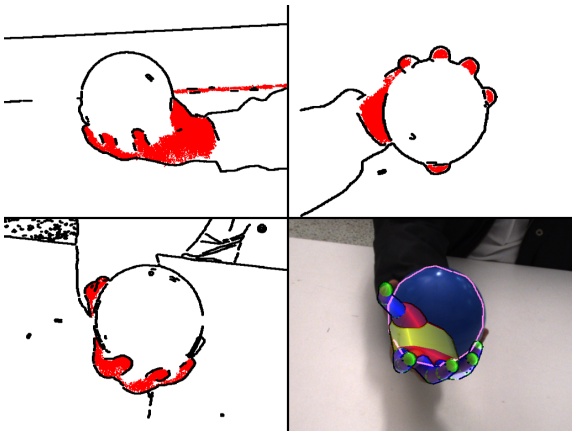


Figure 6.1: Top row, and bottom left: Three views of a hand grasping an object. Skin regions appear in red and edges in black. The hand is partially occluded by the object in all views. The incomplete skin and edge maps of the hand facilitate the generation of a hypothesis for a hand manipulating a compact sphere. At the same time, given this hypothesis, the 3D pose of the hand can be estimated more accurately. Bottom right: the output of the proposed approach superimposed in one of the frames.

when considering them jointly. By modelling the process by which observations on parts can be expected to be reduced due to occlusions we turn occlusions into a feature to be exploited.

Tracking a single hand while in interaction with another entity falls within the main focus of the PhD of Oikonomidis [72]. In the joint work with Oikonomidis [65, 67], presented here, our contribution comprises the computational platform for the implementation of markerless 3D tracking of a hand while interacting with another hand or an object. Although our concern goes well beyond tracking two interacting entities, in 3D, still, this specific case is very important in our context and is also well expressed in our architecture.

6.1.1 The problem

We consider the problem of tracking an entity like the human hand while manipulating another entity. The second entity may be an ordinary passive object [65], or, even another hand [67]. It is then required that, from image sequences, and given an estimation of the initial poses, the configuration of the the two entities is tracked, across time. We consider two types of image sequences: (a) sequences of 2D multi-frames coming from a calibrated multi-camera system and (b) sequences of pairs of RGB images and depth maps, that come from a RGBD sensor (*e.g.* Kinect [57], X-tion [7]).

6.1.2 The solution

Both works in [65, 67] are extensions of the single hand tracking work presented in section 5.1. Therefore, the description of these works will be performed incrementally with respect to section 5.1.

As in section 5.1, we formulate an optimization problem after eq. (3.11), we efficiently compute the formed objective function by employing the computational framework of chapter 4 and we optimize by employing PSO.

As [65] corresponds to [61], and [67] corresponds to [63], in terms of input, the same

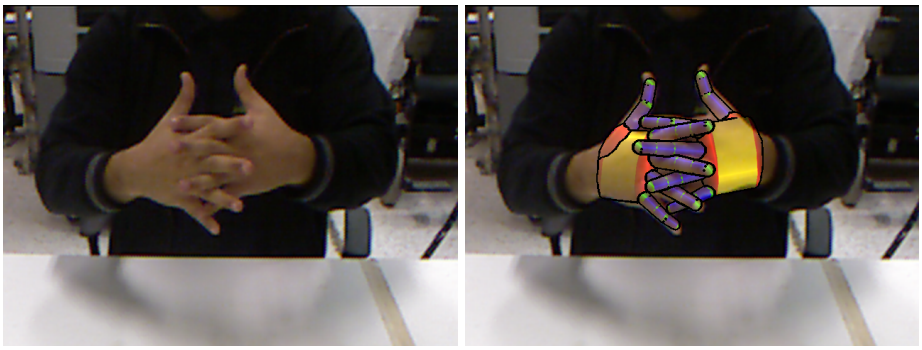


Figure 6.2: Left: A view of two interacting hands. Right: The configuration of the two hands as estimated by the proposed method, superimposed in the left frame.

feature spaces, as described in section 5.1, were used. This, in turn, means that we are also considering the same objective functions, with some exceptions that will be discussed here.

With respect to state decoding, in [65], tracking regarded a hand and an object whose shape could be parameterized (parallepipeds, ellipsoids and cylinders). According to shape, the parameterization of the object varied from 9 to 10 parameters (position, orientation and intrinsic shape parameters). The parameterization of the hand remains the same as in [61]. Similarity transforms (translation, rotation and anisotropic scaling) can be used to transform discretized representations of a unit cube into an arbitrary parallepiped, a unit sphere into an arbitrary ellipsoid and a unit cylinder into an arbitrary elliptic cylinder. Thus, 3 triangular meshes and a decoder that generates such transforms from 9 to 10 parameters suffice to model the objects. Then, the entire search space (hand + object) can be decoded to 3D representations by combining a hand decoder (see section 4.4.1) and the decoder implicitly defined here. For [67], the same procedure if followed by having the second decoder be a mirrored version of the hand decoder, to model a left hand.

With respect to feature mapping, and as far as [65] is concerned, the hand state is explicitly associated with observations, as there are silhouettes that constrain hypotheses over the state of the hand. However, as there is not process of extracting silhouettes for arbitrary objects, there is no *explicit* evidence of the state of the object (apart from spurious characteristic edges). However, there is still *implicit* evidence, encoded into the skin color silhouettes. For example, in fig. 6.1 it is evident for a human observer that the reason the hand silhouette is occluded is because of a grasped sphere. One can generate a corresponding hypothesis, over the state of the hand and the object, by rendering a hand and a grasped object and then keeping pixels only for the issue and instance ids that correspond to hand geometry. Thus, occlusions are explicitly modeled and can be used in a hypothesize and test fashion to reason about the occluding object too. By penalizing hypotheses that correspond to intersecting geometries (see eq. (4.11)) for the hand and object we are constrained in solutions that are not physically implausible.

With respect to [67], the objective function is identical to the one in [52], the only difference being the decoder (an object decoder was swapped for another hand decoder) and the fact that the finger interpenetration penalty needs to be computed twice (once for each hand). Interestingly, adding a term that penalized inter-hand penetrations was not required, as ambiguities in observations could already be resolved.

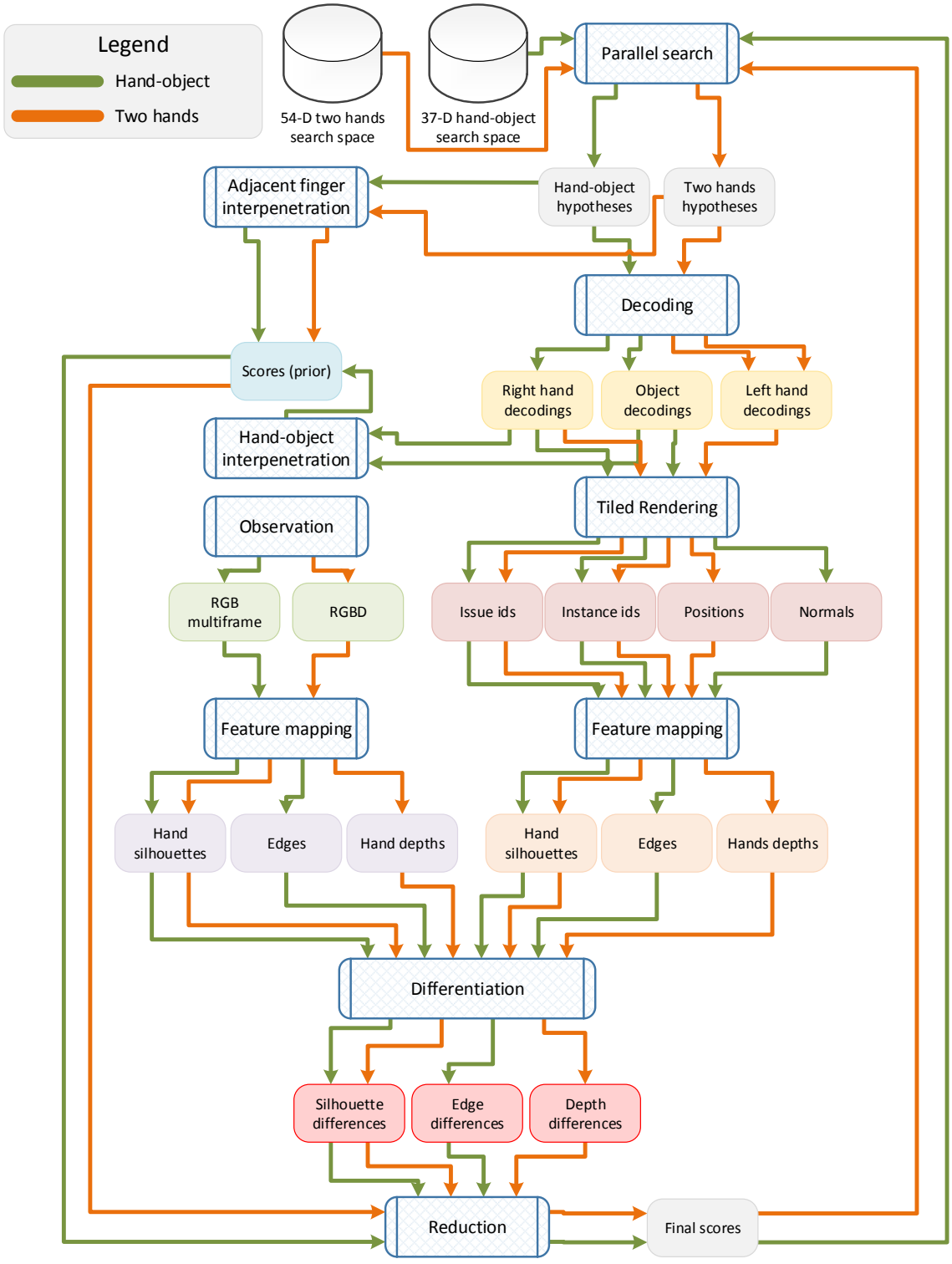


Figure 6.3: The instantiation of the generic framework in fig. 4.1 for the purposes of hand-object tracking [65] and two-hand tracking [73].

6.1.3 Experiments

A series of synthetic experiments were carried out to assess various aspects of the performance of the two methods. In brief, by experimenting across various PSO budgets, an effective tradeoff between accuracy and speed was established. Hand-object tracking in [65] was performed by a PSO configuration of 64 particles and 40 generations, which for a 4-camera system yields a tracking throughput of 2fps. Tracking two hands in [73] was performed at 4fps, by employing 64 particles and 40 generations.

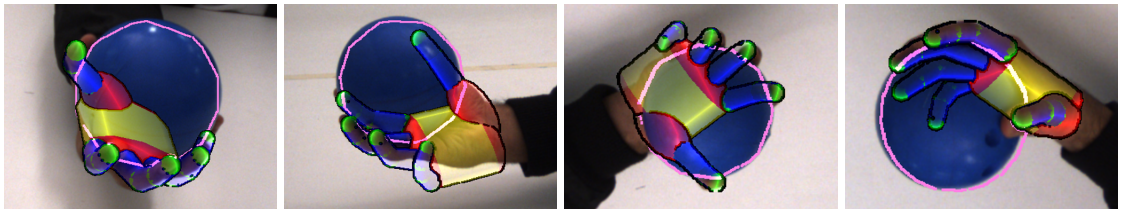
With respect to synthetic experiments, it was shown that indeed, explicitly modelling the occluder is more beneficial than adding another camera to the multi-camera approach [65]. Moreover, the correlation between the hand and the occluder, through reduced observations, is so strong that might as well be used to classify the occluder, into belonging in one of the three classes (parallepiped, ellipsoid, elliptic cylinder). Comparing against the absence of occluder modelling yields the significant contribution of accounting for interaction to accuracy.

With respect to real world experiments, some challenging scenarios were tested. The interaction of a hand with three types of geometry was successfully captured, as shown in fig. 6.4. The physical constraints, that disallow hand-object penetrations, evidently contribute to the extraction of naturally-looking hand-object configurations. A video demonstration of hand-object tracking can be found at `http://youtu.be/N3ffgj1bBGw`. For the case of two hands, a challenging sequence with severe occlusions and finger tangling was successfully tracked (see fig. 6.5). A video showing 3D tracking of two strongly interacting hands can be viewed at `http://youtu.be/e3G9soCdIbc`.

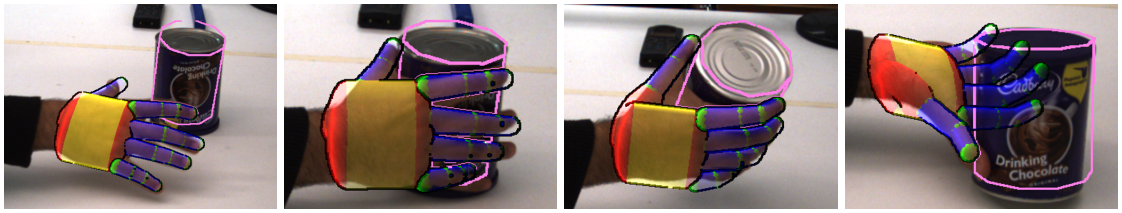
6.1.4 Literature review and contribution

For extensive literature reviews regarding the matters of tracking a hand while interacting with an object or tracking two interacting hands, as well as for comparisons with the state-of-the-art, the reader is referred to [72] and the joint work with Oikonomidis [65, 67]. In this context we focus on the methodological contributions of our model-based, top-down framework in the context of these two problems. It is important to note that while tracking a hand while interacting with an object has been tackled in the past [39, 83], ours was the first model-based approach. Introducing models in 3D tracking yields the benefits discussed in section 5.1.4. For the case of tracking two interacting hands there is actually no prior work at all.

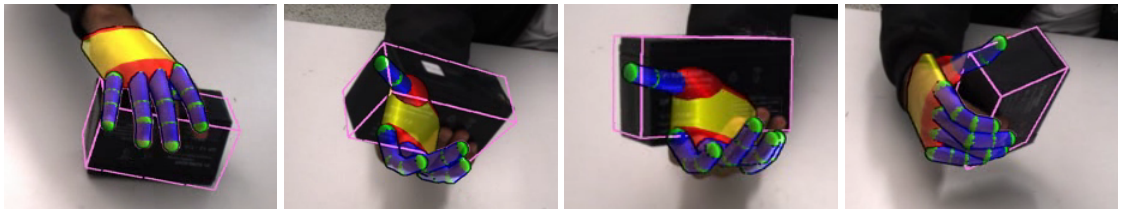
Going from tracking one hand or one object to tracking ensembles brings the discussion over generalizability. Interestingly, tracking two hands is a simple extension for our framework to handle, on the basis of tracking a single hand (see also section 5.1.4). In fact, adding an additional complex entity, like the hand, makes little or no difference compared to adding a simpler rigid object. This is to be contrasted with the existing hand-object tracking methods [39, 83]. As far as [39] is concerned, special reasoning is required in order to maintain hand tracking robustness under the assumption of a distractor object. In our case, the object is not a distractor to be robust against, but rather yet another source of useful information. With respect to [83], we remind of the discussion in section 5.1.4, concerning the generalization problem in training-based methods. In our approach, tracking a single hand, a hand and an object or two hands is equally simple to establish, despite the varying problem dimensionalities ($27D$, $36D$ and $54D$ respectively) and complexities introduced by also considering interactions between entities. Once again, the advantages derive directly from our design choice over decoupling optimization from modelling and



(a) Sphere sequence, frame #103, 4 views



(b) Cylinder sequence, 4 frames, view #2



(c) Box sequence, 4 frames, view #1

Figure 6.4: Sample frames from the results obtained by [65] in real-world experiments. The results are of such fidelity so that they succeed in visually conveying the tight physical relation between the hand and the object.

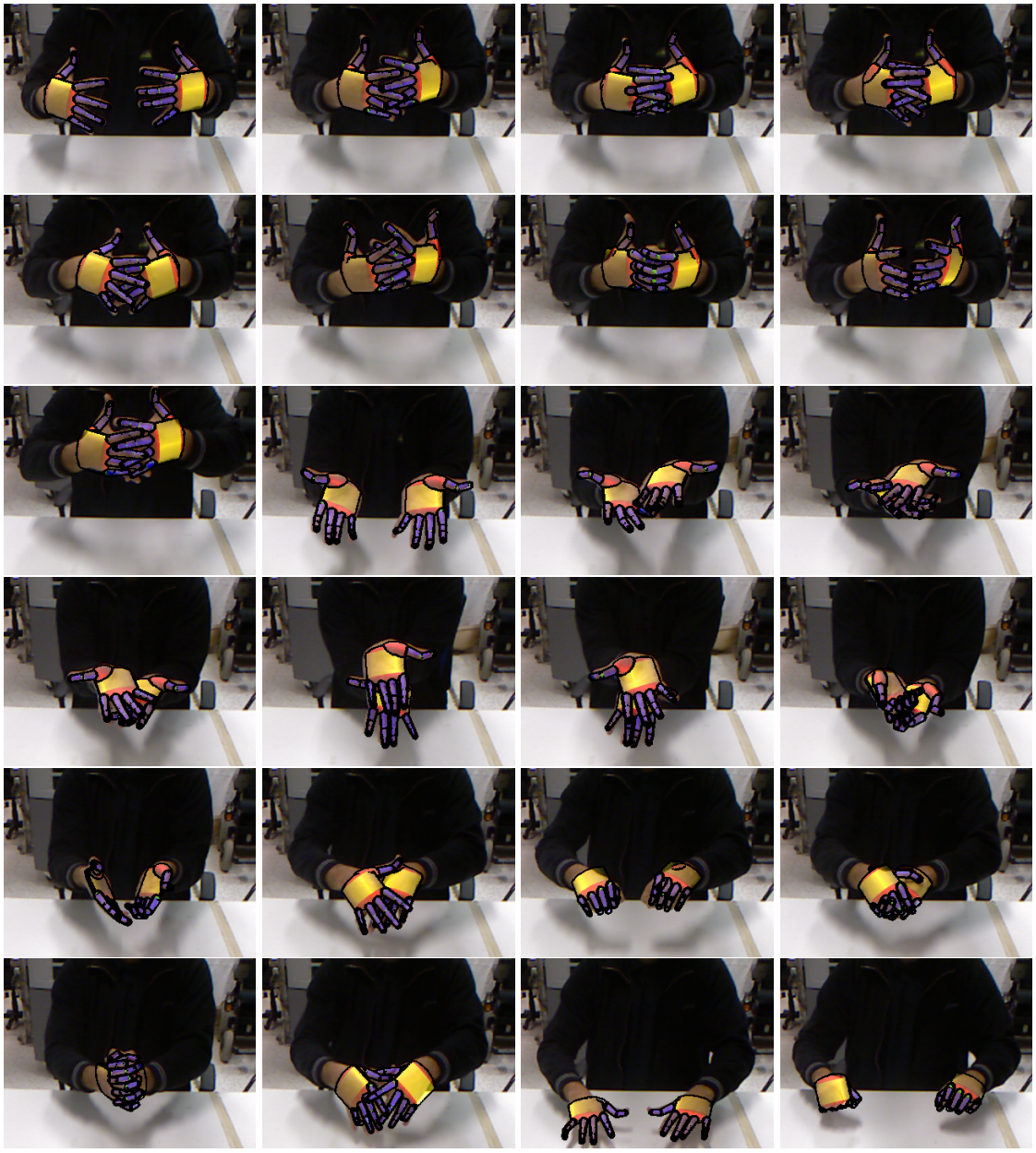


Figure 6.5: Snapshots from [67], tracking two hands which interact with each other. Notice that tracking is successful even for extreme cases, like the one in the bottom-left image. It is also noteworthy that these results have been produced without accounting for collisions between the two hands. Only adjacent finger interpenetration is penalized, for each hand independently.

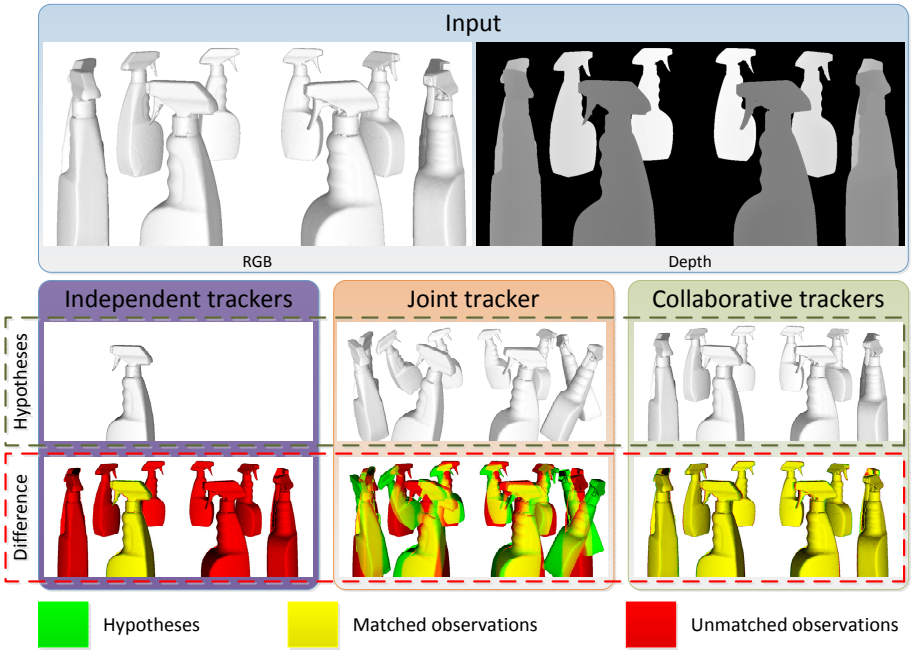


Figure 6.6: 3D tracking of multiple entities. We compare three variants for tackling the tracking problem, to highlight the requirement for a new tracking scheme, *i.e. collaborative tracking*.

the fact that we employ rich models defined over continuous variables.

6.2 Tracking multiple interacting entities

We consider the problem of tracking multiple interacting entities in 3D, using RGBD input. A fundamental difficulty in this kind of multi-entity tracking problems results from the occlusions that occur when the physical, 3D space is projected onto the 2D image of the camera observing the scene. If no such occlusions exist, the whole problem can be easily decomposed. Assuming the availability of a tracker for each and every entity, we may consider a Set of Independent Trackers (SIT) in a divide-and-conquer approach. This straightforward decomposition has an overall accuracy that depends on the accuracy of the individual trackers, alone. From a computational point of view, it scales well (linearly) with the number of objects to be tracked.

However, in the most interesting and most often encountered case, entities do occlude each other in the field of view of the camera. This is particularly true in the object manipulation case we are interested in, where the interaction of hands with objects results in strong hand-object and object-object occlusions. Occlusions have an almost catastrophic effect to the SIT approach. This is because, due to occlusions, each individual tracker is fed either with missing or with ambiguous observations of the object it tracks. To handle this problem, state of the art approaches [65, 67] suggest a Joint Tracker (JT) that performs joint optimization over the parameters that encode the full 3D state of all entities to be tracked. Thus, occlusions are not treated as a distractor but rather as a source of information that has an active role in tracking the state of the scene.

The JT approach has been shown to accurately track scenes of up to 54 DOFs, involving two strongly interacting hands [67]. However, as the complexity of the scene objects

increases[1], optimization becomes much harder, and, as demonstrated here, the resources required to achieve a constant level of tracking accuracy increase geometrically with the number of the objects to be tracked.

6.2.1 The problem

We are interested in tracking the full state of a scene consisting of multiple moving and interacting entities. For the purposes of this work we consider rigid and/or articulated entities, which interact in front of a static RGBD camera. An example scenario is that of observing hands interacting with several objects in assembly/disassembly tasks. In such a context, the full state of the scene, at a given time, consists of the 3D position and orientation (3D pose) of all rigid entities as well as all the degrees of freedom (DOFs) of the articulated entities.

6.2.2 The solution

In this variant, and with respect to eq. (3.10), the specialization of the generic framework (see chapter 4) presented in section 5.1 (see fig. 5.4), which regards RBGD input, has been adopted and extended. The single 3D hand tracking approach of section 5.1 is hereby used as a building block, where multiple 3D trackers which are associated with distinct entities are combined into a single 3D tracker, which regards the entire set of entities, simultaneously. Such complex scenes can be decomposed to their constituents, where each one is not more complex than an articulated hand. In fact, most entities are far simpler (*e.g.* manipulated objects). Regardless of the complexity of an entity, given a parameterization of its appearance and the ability to decode some parameters into renderable state, the same principles, as in 3D hand tracking, apply.

Tracking with a Joint Tracker (JT)

Unless all entities are considered jointly, the corresponding forward model cannot capture their interaction and, therefore, cannot predict possible occlusions. The state of the art approaches [65, 69] tackle the problem of eq. (3.2) directly. The configuration space of all objects combined is considered as the joint hypothesis space. This allows for the objective function of eq. (3.2) to regard all entities simultaneously and to account for their interactions.

Two problems are associated with this approach. First, computing interactions and features for a large number of objects at every invocation of the objective function is computationally expensive. Second, as the search space grows, it becomes increasingly harder to solve the optimization problem. In fact, we demonstrate experimentally that JT does not scale well with the number of objects to be tracked and that JT is incapable of handling the complexity of the scenes we are interested in. A description of the instantiation of the generic framework presented in chapter 4 for the case of JT is given in fig. 6.7.

Set of Independent Trackers (SIT)

An oversimplification of the problem would be to consider all objects disjointly. In detail, multiple independent problems are solved, where each one addresses the tracking of

[1] In this context, the complexity of a scene is quantified as the number of parameters/DOFs that represent its state.

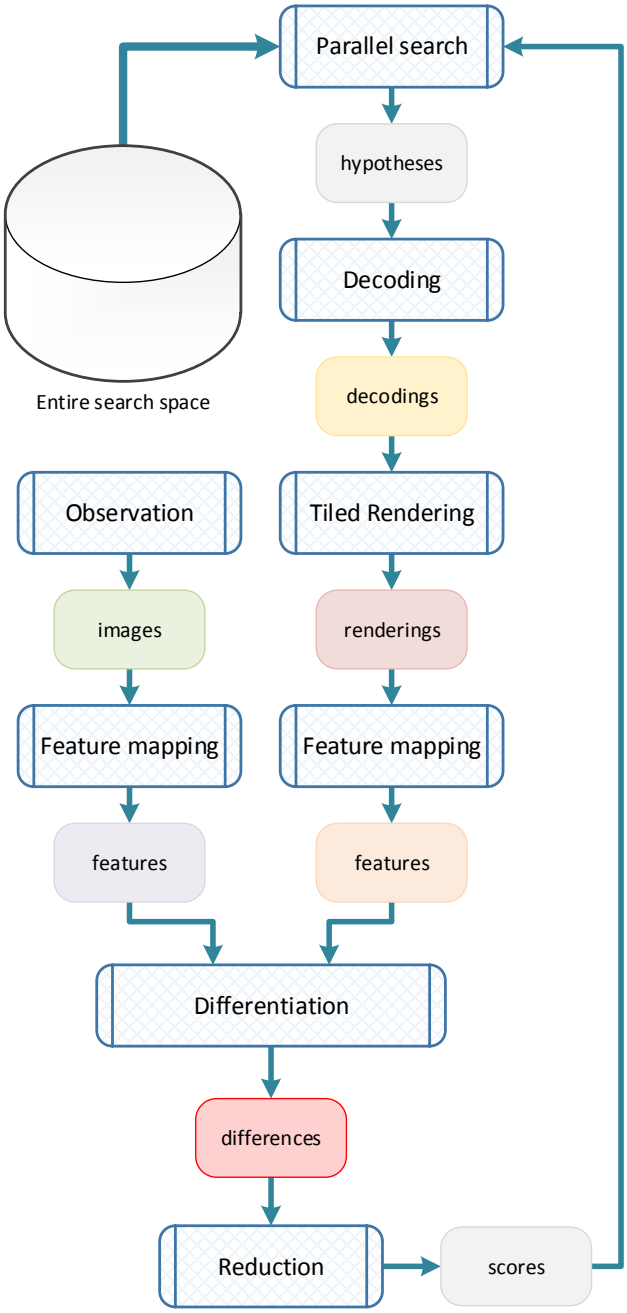


Figure 6.7: Instantiation of the generic framework of fig. 4.1 for the case of joint tracking (JT). In this abstracted information flow diagram the domain of the problem is depicted relatively enlarged to convey that the same principle of 3D tracking may be applied for arbitrarily big problems. However, in practice, it takes moderately sized domains to render this approach ineffective.

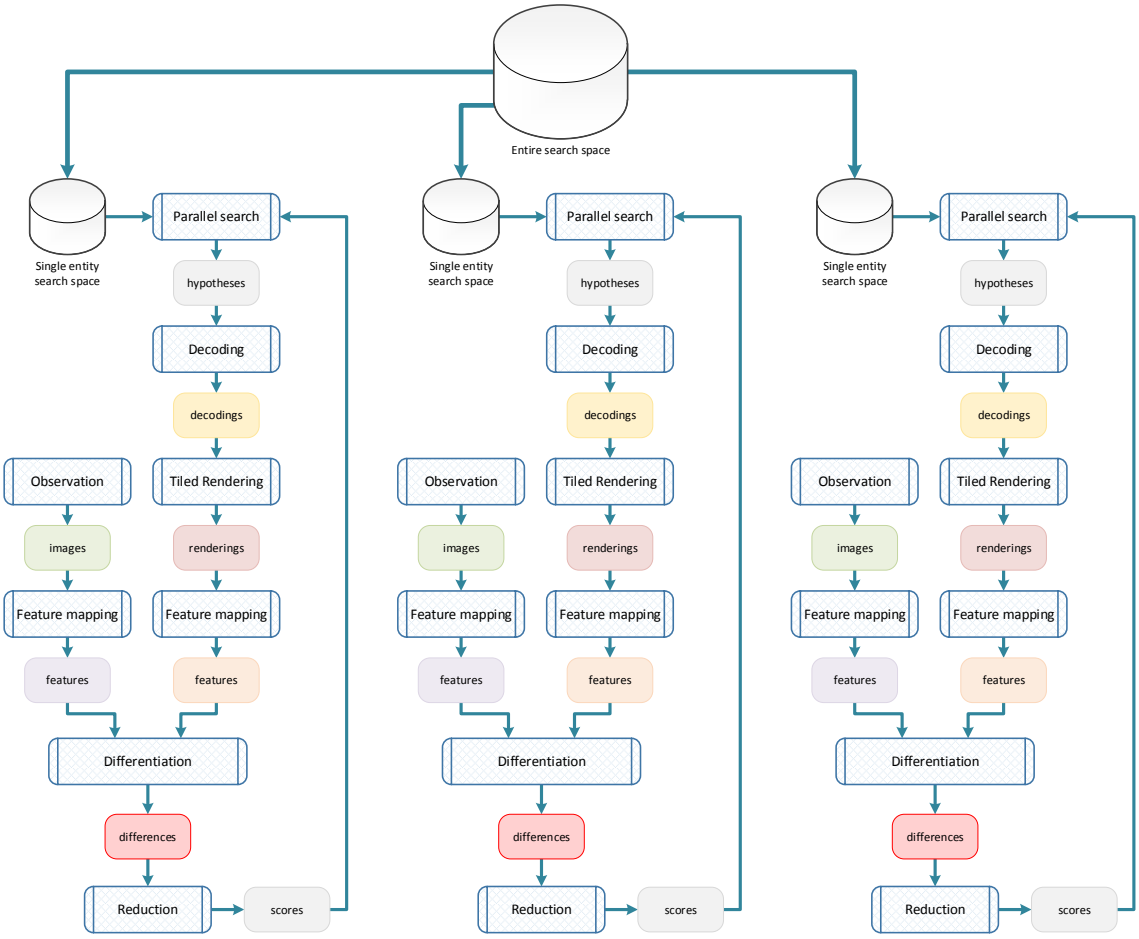


Figure 6.8: The instantiation of the framework depicted in fig. 4.1 for the case of a Set of Independent Trackers (SIT). In this diagram the case of 3 independent trackers is illustrated. The big problem is decomposed into smaller independent ones, in a lossy fashion. The trackers have no means of communication, which introduces problems, pertaining mainly to ambiguities, due to the lack of a mutual exclusiveness in data association, in scenes that are dominated by interaction among the tracked entities.

an individual object in isolation. Thus, the hypothesis space for each tracker is the configuration space of a single object. This yields a method that is significantly faster than JT due to the lossy decomposition of the big problem into multiple smaller ones. In case that there are no occlusions among the objects to be tracked, the quality of the solution (*i.e*. tracking accuracy) is determined by the accuracy of the solution to each individual problem. However, as interaction between objects and the resulting occlusions increases, the method is expected to deliver tracking results of progressively lower accuracy. This is because occlusions contaminate the observations of each individual object with missing or ambiguous evidence. Formally, the computed objective function, due to the lossy decomposition, no longer corresponds to eq. (3.2), and therefore the respective minimizer does not respect joint constraints, such as the required mutual exclusiveness in the allocation of observations. This has been demonstrated experimentally in [69, 72] where the JT approach managed to successfully track two hands in a dataset featuring two hands in strong interaction. Then an independent single-hand tracker (SIT approach) was employed to the same dataset. SIT performed accurately when there was no occlusion between hands. However, as soon as interaction started, it failed to recover the correct state. A depiction of the instantiation of the generic framework presented in chapter 4 for the case of SIT is given in fig. 6.8.

Ensemble of Collaborative Trackers (ECT)

Our proposal is a middle ground between SIT and JT. As in SIT, we employ multiple trackers, each of which is responsible for tracking the state of a single object. However, the trackers are not independent. On the contrary, each of them considers the state for the rest of the objects to be static for the current frame, according to the most updated view of the other trackers. Reversely, each tracker broadcasts the estimated state for the associated object as soon as this becomes available, and this information is regarded as static by the rest of the trackers during their very next optimization step. Formally, this still allows for the full consideration of all possible interactions since the invocation of eq. (3.2) regards the combined state. At the same time (a) optimization is clearly separated for each object, yielding a clear scalability profile and (b) less computations are required for the interactions and features, because the most part regards static information that needs only be computed once per frame.

The rationale behind this choice is that by breaking down the problem into multiple smaller ones optimization should be guaranteed, as in SIT, but at the same time, interactions will also be considered. In contrast to SIT, ECT has no ambiguity issues. The appropriate parts of observations that correspond to other trackers are already allocated to them, through the consideration of the broadcast results, leaving only the part of observations that correspond to each tracker, plus some noise that is introduced by the one-frame lag of these broadcast results. It is shown experimentally that the level of this noise is well inside the trackers' tolerance. An illustration of the instantiation of the generic framework presented in chapter 4 for the case of ECT is given in fig. 6.9.

Decomposition of computations In each frame, each tracker of the ensemble needs to estimate the state of the object it is associated with. To do so, it renders and evaluates object pose hypotheses (dynamic state) in a context that is formed by what has been estimated for the rest of the objects, from the rest of the trackers, in the previous frame (static state). Essentially, dynamic state corresponds to the optimization task performed by each

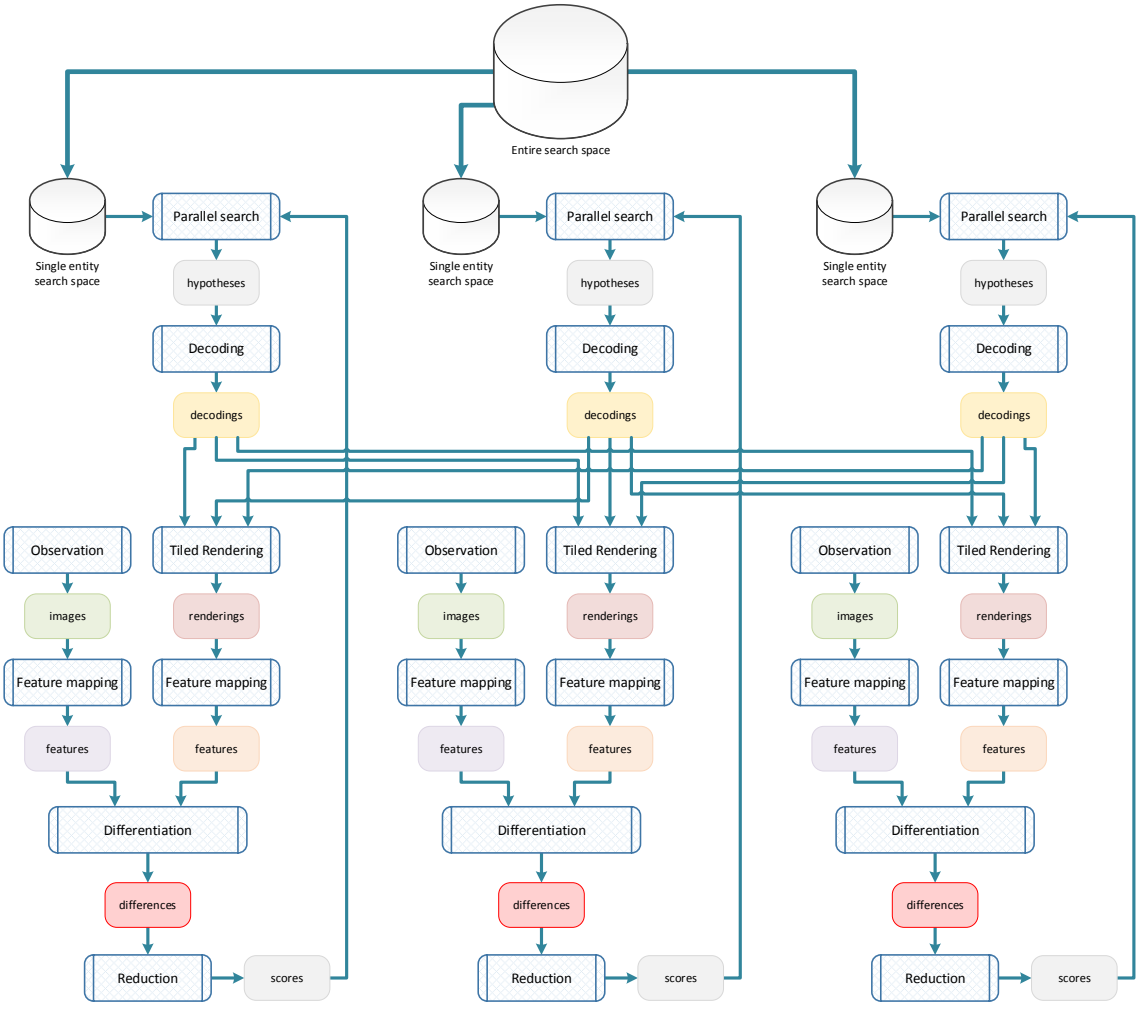


Figure 6.9: The instantiation of the framework depicted in fig. 4.1 for the case of an Ensemble of Collaborative Trackers (ECT). In this diagram the case of 3 collaborative trackers is illustrated. The big problem is decomposed into smaller ones. In contrast to SIT and fig. 6.8, the collaborative trackers communicate by sharing their current estimation of the world state to the rest of the trackers. Thus, the decomposition preserves the nice trait of being scalable, but is also brought closer to the spirit of joint tracking. Experimental results actually show that this approach is not only practical in cases where JT is not, but also performs better, overall, compared to JT, in terms of throughput, accuracy or both.

tracker and static state corresponds to a frozen state of the scene as it has been estimated so far. This distinction is quite important because it leads to considerable computational performance gains.

For each tracking frame the terms in eq. (3.2) that are related to static state are precomputed. Then, during optimization for the same frame, the precomputed terms are fused with computations for the dynamic state, in order to properly assemble an invocation to the designed objective function. Given that dynamic state accounts typically for a small fraction of the scene (*i.e.*, one out of many objects), the computational gains are high.

Object rendering regards both static and dynamic state. At the beginning of each tracking frame, the static state of every collaborative tracker (*i.e.* the most up to date view of the scene according to the rest of the trackers) is rendered once into a depth map and stored. For each hypothesis generated during optimization (dynamic state), the corresponding depth map is fused with the stored one through z-buffering. The final depth map is identical to what the rendering of the entire state would have yielded. Nevertheless, it is much cheaper computationally during optimization.

Collision checking also regards both static and dynamic state. The total penetration depth TPD for a collection of shapes is defined as the sum of all pairwise penetration depths PD:

$$\begin{aligned} \text{TPD}\left(h\right) = \sum_{x,y \in h} \text{PD}\left(x,y\right) = \\ \sum_{x,y \in h_s} \text{PD}\left(x,y\right) + \sum_{x,y \in h_d} \text{PD}\left(x,y\right) + \sum_{\substack{x \in h_s \\ y \in h_d}} \text{PD}\left(x,y\right), \end{aligned} \tag{6.1}$$

where h is a 3D configuration hypothesis that regards multiple entities, h_s is the part of h that regards static state and h_d is the part of h that regards dynamic (per collaborative tracker) state, such that $h = h_s \cup h_d$ and $h_s \cap h_d = \emptyset$. The computations that regard h_s alone need only be computed once at the beginning of each tracking frame. During optimization this precomputed term is simply added to the penetration depth computations for the rest of the pairs.

It should be noted that for both ECT and JT the objective function is invoked over the entire state of the scene. However, the same invocation to the same objective function is computationally cheaper for ECT, because for JT there is no static/fixed state whose processing could be re-used.

6.2.3 Experiments

All experiments were executed on a machine with a quad-core Intel i7 920 CPU, 6 GBs RAM and a 1581GFlops Nvidia GTX 580 GPU with 1.5 GBs RAM. For the image acquisition process we employed a Kinect sensor and the OpenNI framework. Acquisition was performed at a $30fps$ rate. For collision checking complex/concave objects were decomposed into convex parts. Convex decomposition was performed using the method in [37] as implemented in the CGAL library [2]. Collision checking was performed using the Bullet physics simulator [23]. During optimization, λ was set to 1 when $\mathbf{L}$ amounted to penalization of adjacent finger interpenetration (measured in radians), and 0.001 when $\mathbf{L}$ amounted to the total penetration depth penalty (measured in millimeters). Videos with results from all experiments are available at `http://youtu.be/SCOtBdhDMKg`..

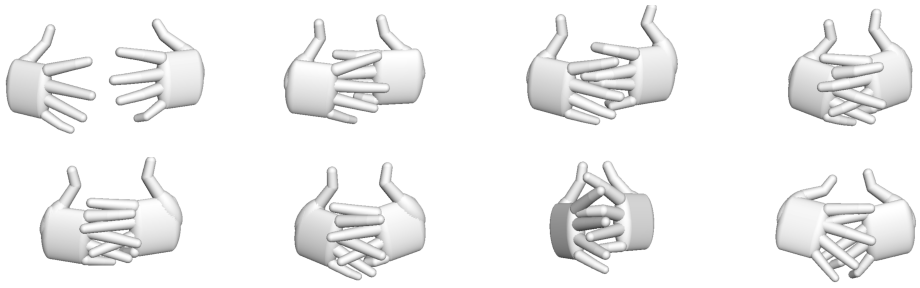


Figure 6.10: Frames of the two-hands tracking synthetic dataset used here and in [69] for quantitative analysis.

Quantitative analysis

The tracking performance of JT, SIT and ECT was compared in a series of experiments where ground truth was available. The establishment of ground truth in real-world acquisition is known to be hard, especially for the case of hand tracking [63, 65, 69]. Therefore, we employed the common practice of generating synthetic datasets, where ground truth establishment was guaranteed. The fact that in that case observations were ideal is irrelevant, as the focus of this experiment lies in the quantitative comparative evaluation of the three tracking methodologies. Their efficacy in real-world scenarios pertaining to real noise, is demonstrated in section 6.2.3. Efficacy was computed for various configurations of PSO. This configuration amounted to specifying the number of particles and generations. The product $budget = particles \times generations$ yielded the number of objective function invocations during the corresponding tracking frames.

Each of the employed datasets involved objects of the same type. Thus, the budget allocated to each tracker of SIT and ECT was identical. For experiments to be fair, at any given point of comparison the three tracking methods were provided the same budget. If N entities were involved, and for a selection of p particles and g generations, each tracker of SIT and ECT was allocated a budget of $p \times g$ objective function invocations. Thus, for these two variants each tracking frame amounted to $N \times p \times g$ objective function invocations. JT was allocated the same budget, by maintaining the generation count at g and by setting the particles count to $N \times p$.

Quantitative experiments were conducted across a 2D grid of budgets, that was generated by varying particle and generation counts. At every point of that grid several tracking experiments were conducted to balance effects stemming from the stochastic nature of PSO.

For each experiment we computed an accuracy measure E, that amounted to the average 3D registration error, in millimeters, between the true configurations of the entities and the tracked configurations, for all frames and entities. Registration error of rigid entities amounted to the mean value of the registration error of each point of the corresponding 3D model. For the articulated entities the mean registration error across all joints was considered, as in [69].

Tracking two hands The problem of tracking two interacting hands in 3D amounts to solving a 54-D (27 dimensions per hand) problem, for every tracking frame. This problem has already been tackled in [69] which in this context can be viewed as the JT method. This is compared against SIT and ECT on the synthetic dataset that was used in the quan-

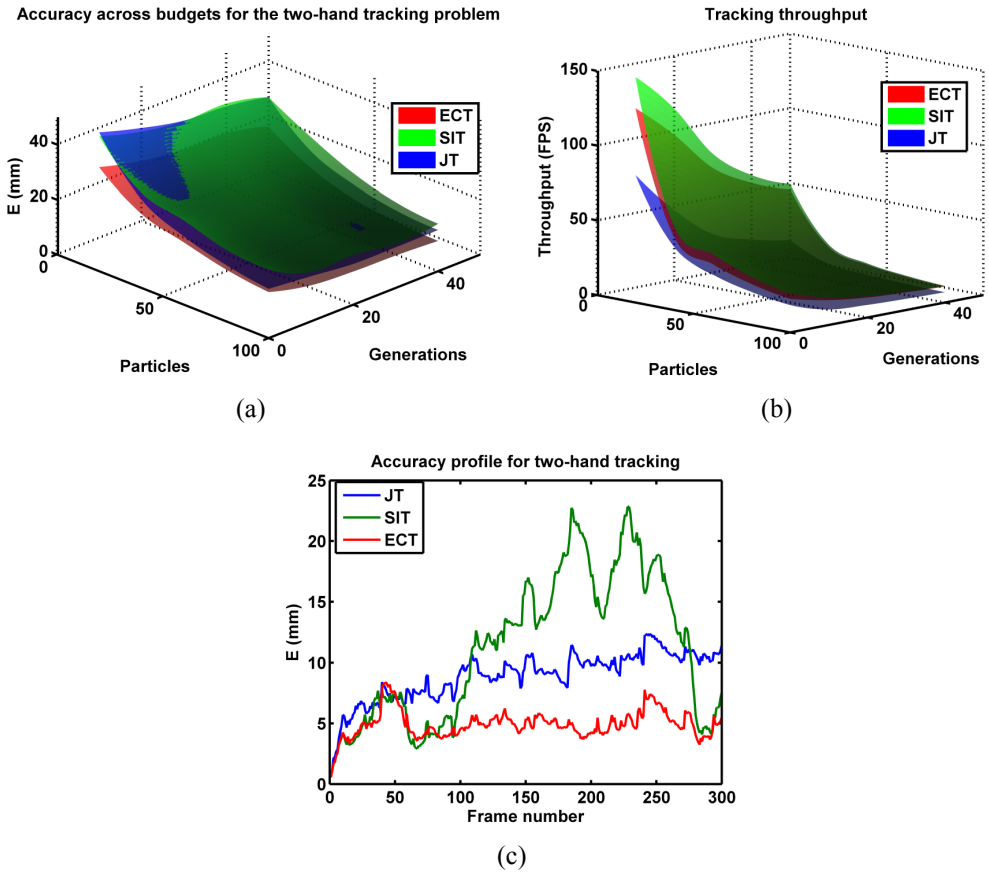


Figure 6.11: Quantitative results for the two hands tracking regarding (a) accuracy, (b) tracking throughput and (c) average tracking performance across time.

titative analysis of [69] (see fig. 6.10). This dataset consisted of 300 frames showing two hands engaged in increasingly strong interaction. Thus, it was more probable that tracking accuracy deteriorated at later frames, as inaccuracies accumulate and drift increased. During optimization, we used the finger interpenetration penalty as a prior.

A budget grid was considered, where particle counts varied in the range $(0, 100]$ and generation counts varied in the range $(0, 50]$. The corresponding results are shown in fig. 6.11. It can be verified that ECT always outperformed both SIT and JT. Another striking result is that, on average, SIT was not much worse than the rest of the methods. The details in fig. 6.11(c) yield an explanation. The accuracy for the JT method gradually degraded as the interaction complexity increased, due to accumulated drift. SIT and ECT had very similar behaviours, being more accurate on average than the JT method, until the two hands interacted (frame 100). After this point, the accuracy for the SIT deteriorated quickly, since it did not account for the intense interaction. The ECT method was unaffected by the increase in interaction complexity. We attribute the superiority of ECT against JT to (a) the explicit isomerization of optimization budget and (b) the fact that a difficult problem was broken down to two easier ones.

Multi-object tracking The problem of tracking two interacting hands included the following challenges: (a) the problem dimensionality was high and (b) the articulated nature of the entities yielded a difficult problem to solve, due to interaction intensity. While (a) remains a constant pursuit, since we are targeting large problems, more insight can be

(a) radius of $120mm$ (b) radius of $250mm$

Figure 6.12: Frames from the rotating bottles synthetic dataset.

provided by modulating (b). We therefore preserved the dimensionality of the problem and instead of considering two complex entities we regarded a larger number of simpler entities.

We animated 8 instances of a spraying bottle model as if they stood equidistantly on a rotating turntable (see fig. 6.12). By modulating the radius of the turntable, occlusions became lighter (larger radii) or heavier (smaller radii), *i.e.* occlusions occurred over smaller or bigger areas and included less or more objects simultaneously. The motion of the bottles was rigid, therefore for each bottle, during tracking, the pose was sought, which amounted to 7 parameters (3D position and quaternion-based orientation). Thus, the total problem dimensionality was $8 \times 7 = 56$. As objects were by construction penetrating each other we employed neither the total penetration depth penalty nor any other.

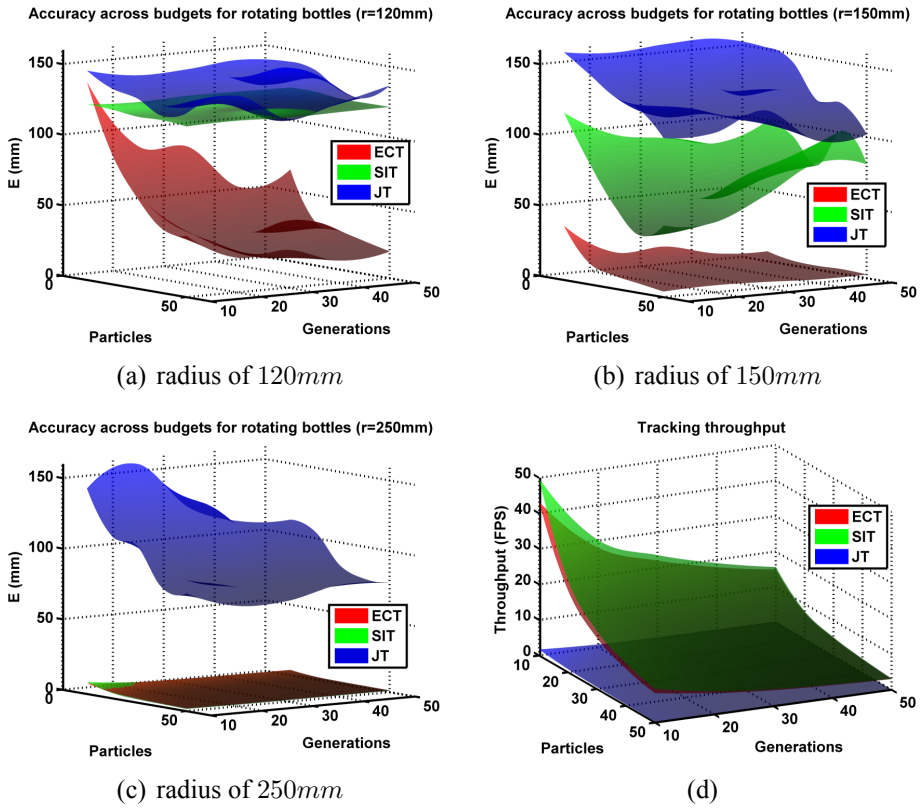
For this problem we considered a budget with particles varying in the range $(0, 60]$ and generations varying in the range $(0, 50]$. The results are shown in fig. 6.13. As it can be verified, in all cases ECT yielded the most accurate performance. What is striking is that JT was always the least accurate. The accuracy of SIT varied between the accuracies of the two other methods, and degraded, as expected, as occlusions became more severe, *i.e.* in lower radii (see fig. 6.13(a)). For the case of the largest radius the performance of SIT and ECT was identical (see fig. 6.13(c)), which demonstrates a single tracker's adequate tolerance against contaminated observations. The order in tracking throughput was preserved, however, the differences were exaggerated due to the increase of the number of objects, rendering JT $25\times$ to $47\times$ slower, than ECT (see fig. 6.13(d)). The presented figures are greatly skewed in favor of ECT as the number of objects increases.

One thing to note is how much slower JT becomes as the number of objects increases. This is due to the ability of ECT to reuse computations over static state (see section 6.2.2). Considering large parts of static state dramatically decreases the amount of object pairs which require dynamic consideration, *i.e.* dynamic redefinition, rendering and comparison. The quadratic amount of object pairs (object to object), which require consideration in JT, becomes linear for ECT (object to scene).

Qualitative analysis

To complete the comparison of the three tracking methods we present tracking results that regard real sequences.

Tracking two hands The dataset that was used in the qualitative assessment of [69] was also used here, to compare the proposed ECT tracking methodology to that of JT and SIT. As in [69], we used skin color detection to identify the foreground. We used the total penetration depth penalty as a prior. The results are shown in fig. 6.14. It is evident that SIT could not cope with the complexity of the problem (see fig. 6.14(a)). The performances



(a) radius of $120mm$ (b) radius of $150mm$

(c) radius of $250mm$ (d)

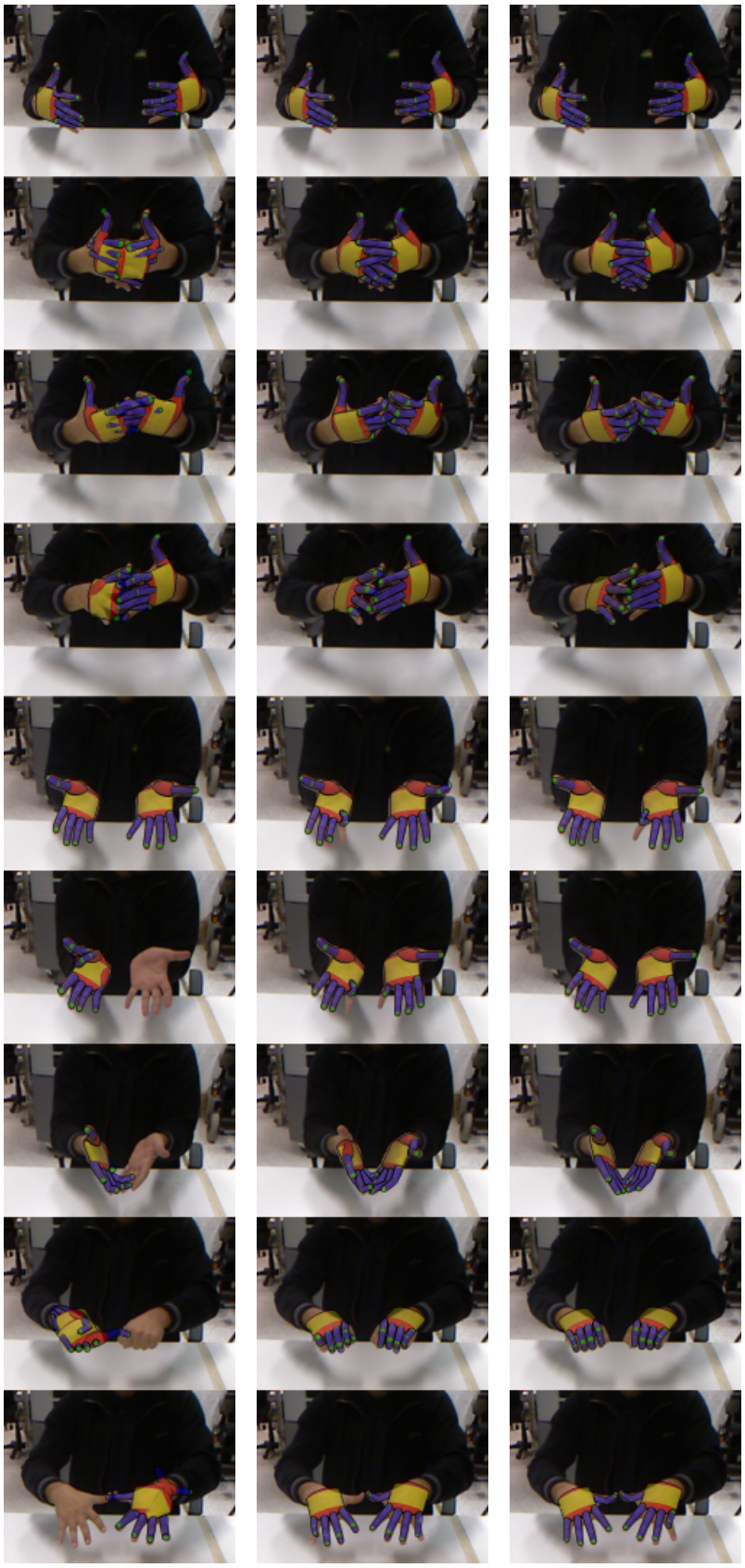
Figure 6.13: Quantitative results for the rotating bottles dataset. (a)-(c) tracking error and (d) tracking throughput.

of ECT (see fig. 6.14(c)) and JT (see fig. 6.14(b)) were comparable on average, with ECT yielding better results, overall. Besides the slightly better results, ECT was also up to $3\times$ faster than JT for larger budgets.

Disassembly tracking We recorded a sequence of two hands disassembling a toy made of 15 parts, *i.e.* 1 base, 6 columns and 8 cubes (see fig. 6.15). Each hand corresponded to 27 parameters and each rigid part to 7, so the entire search space had 159 dimensions. Because it was an assembly toy, parts fit in tightly, creating multiple and severe occlusions. Also, due to the similarity of the parts, one part could be mistaken for another.

We employed SIT, JT and ECT to track this scene. The dataset consisted of more than 2900 frames. We identified the foreground by keeping only points non belonging to the surface of the table. We complemented with skin color detection to enhance poor foreground detection on the hands. We allocated a budget of 30 particles and 30 generations to each rigid part. For each of the hands we considered a budget of 50 particles and 50 generations.

SIT failed to track this complex scene, right from the start, as it did not account for interactions, which were dominant. JT failed too, as the budget was inadequate for exploring viable solutions in such a large space, despite the fact that most of the entities remained stationary for prolonged periods of time. In fact, during tracking, JT was unable to depart from the initialization hypothesis, which was maintained even across frames where data were clearly not supporting it. The results obtained from ECT are shown in fig. 6.15. As it can be verified, ECT was successful in tracking the scene. Some tracking issues did occur, at moments where the poses of the hands generated ambiguous observations in the



(a) (b) (c)

Figure 6.14: Tracking two interacting hands. The results of (a) SIT , (b) JT and (c) ECT, are compared for 50 particles and 50 generations per hand. SIT has no change of correctly interpreting a scene that is dominated by interaction. Results for JT and ECT are similar, with the results of ECT being slightly more accurate but also faster than JT.

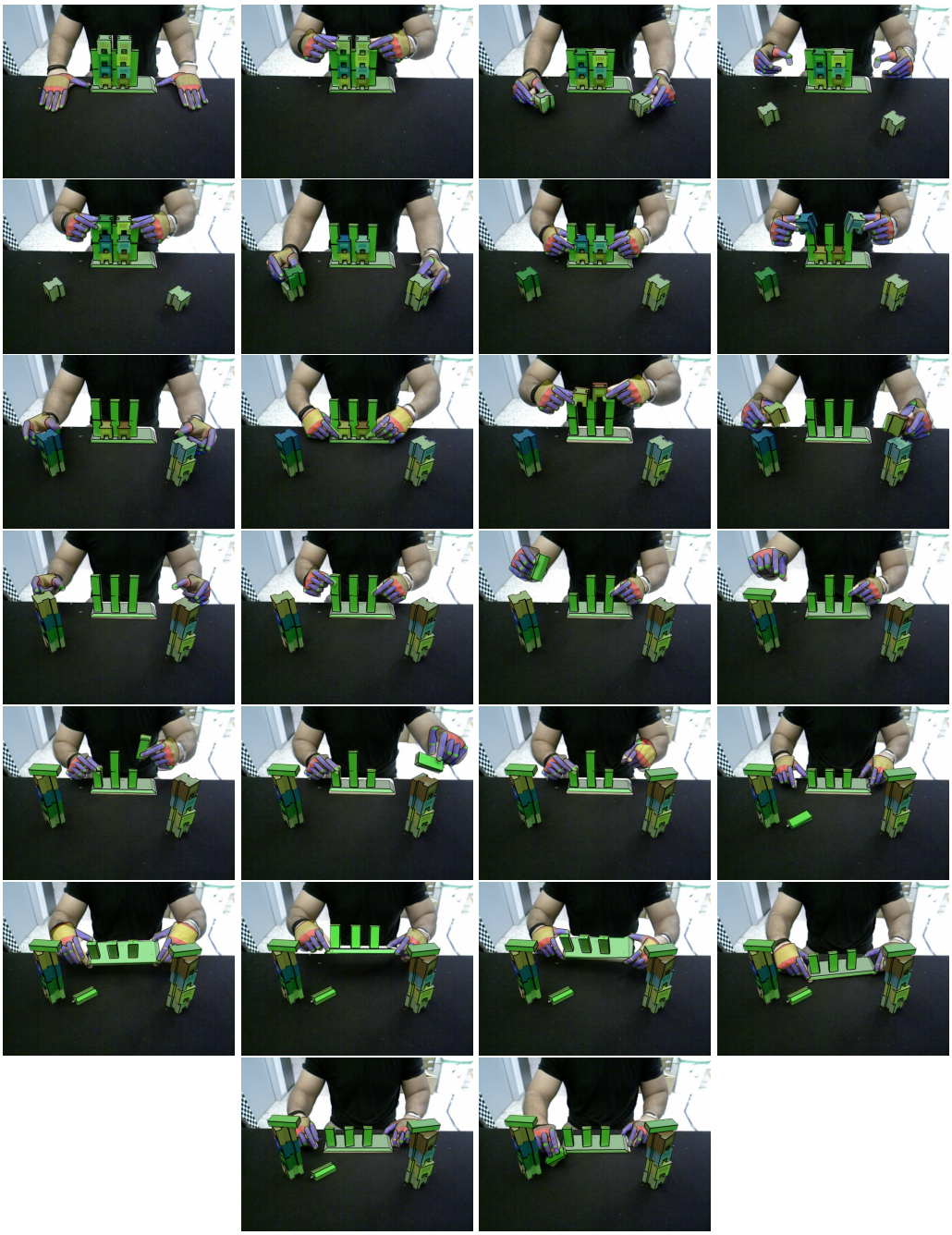


Figure 6.15: Representative results obtained from the proposed ECT approach on the toy disassembly dataset. It is redundant to show the corresponding results for the other variants, as they are far from being even remotely plausible for this high-dimensional problem.

depth map. This, however, is an inherent problem to the approach of [63,69] and does not relate to collaborative tracking (issues occurred when hands were fully visible). As far as execution times are concerned, SIT required $0.95s$ per frame ($1.05fps$), ECT required $2.06s$ per frame ($0.48fps$, $2\times$ slower than SIT) and JT required $106.7s$ per frame (less than $0.01fps$, $50\times$ slower than ECT). ECT is slightly more expensive in computational terms and for the exact same budget than SIT because of implications of considering the entire scene state jointly at each individual tracker. ECT is far more inexpensive than JT, for the same budget, because of the careful decomposition and reuse of computations (see section 6.2.2).

6.2.4 Literature review and contribution

We propose a hypothesize-and-test method that tracks, accurately and robustly, the 3D state of complex scenes, in which strong interactions result in significant occlusions. Model-based tracking constitutes the basis of the proposed approach, which stands between the extremes of SIT and JT, trying to combine the best traits from both worlds. As in SIT, we employ a number of trackers, one for each object to be tracked. However, these trackers are not independent but they rather form an Ensemble of Collaborative Trackers (ECT). Being collaborative, the individual trackers solve the problem in a synergistic manner, bringing ECT closer to the spirit of the joint optimization performed in JT. Collaboration between trackers has the form of exchange of intermediate information. More specifically, each tracker is delegated with the task of tracking a single object, while regarding the rest of the objects as being statically defined. At certain stages of the processing, each tracker broadcasts its intermediate tracking results to all other trackers which update their current knowledge of the state of the other tracked objects. This way, ECT achieves two things simultaneously. Firstly, as in SIT, the full, joint optimization problem is decomposed in a number of smaller problems of lower dimensionality. Secondly, as in JT, occlusions among interacting objects are effectively taken into account.

Several experiments with a variety of scenes and complexities have been carried out to assess the proposed ECT approach, quantitatively and qualitatively. In all cases, the obtained results were compared to those of the SIT and JT approaches. In a representative experiment (see fig. 6.15), we considered two hands as they disassembled a toy consisting of 15 rigid objects. Tracking the articulated motion of the hands and the 3D pose of all objects corresponded to a problem of 159 dimensions. It is shown that while both SIT and JT failed (for different reasons) to track this scene, the proposed ECT method provided an accurate solution at a speed that was only $2\times$ slower than SIT and $50\times$ faster than JT. Similarly, for the rest of the experiments and for every optimization budget, ECT outperformed both SIT and JT in tracking accuracy, while it was only slightly more expensive than SIT and far cheaper than JT in computational time.

Multi-object tracking and occlusion handling are two strongly connected problems. The respective literature lists several approaches that tackle both problems simultaneously. The overwhelming majority of these works track 2D regions of interest (ROIs) and, as such, their full review is out of the scope of this work. Generally speaking, the dominant approach is to treat occlusions as a distractor, *i.e.* perform special occlusion detection and rectification while or after tracking, as in [3]. Similar in spirit is the approach proposed in [74]. That work also reviews the state of the art and the relevant literature in robust occlusion handling while tracking multiple objects in 2D.

Such methods are, by construction, limited in estimating 2D information, with some

exceptions that only go as far as considering depth qualitatively (*e.g*. depth layering). In the current work we focus on methods for the 3D tracking of multiple interacting objects. *i.e*. we focus on quantitative reasoning in all spatial dimensions. In a class of such methods, bottom-up evidence, provided by strong discriminative tools, is fused into coherent interpretations through higher-level generative modelling. Hamer *et al*. [38] proposed a robust reconstruction method that performs 3D tracking of a hand manipulating an object. This method was based on strong bottom-up evidence that was used to identify occlusion, so that their effect was disregarded during inference. Then, hand pose hypotheses were constructed by generative means. Romero *et al*. [82] exploited their ability to realistically synthesize the outlook of a hand-object interaction scenario to also track a hand manipulating an object in 3D. A non-parametric discriminative model was generated from a large synthetic dataset. This model was used to track hand pose from image sequences through classification. Inference over hand poses close in time were regularized so as to adhere to some smoothness criteria. While both methods set the basis for robust hand-object tracking, their extension to other types of interaction or their extension to tracking more objects is not straightforward.

Other methods go beyond treating occlusions as a distractor by explicitly accounting for them in interaction models. These methods are generative in nature and operate on raw or on slightly preprocessed input. Tracking is treated as an optimization problem that involves an objective function that quantifies the discrepancy between hypothesized interpretations of a scene and its actual observations. In the work presented at section 6.1 (see [65]), an object and the hand manipulating it were jointly tracked in 3D. An interaction model that directly accounted for potential occlusions and the fact that two different objects cannot occupy the same physical space was considered. The same line of reasoning has been successfully applied to the more challenging problem of tracking two strongly interacting hands [69]. In both [65, 69] black box optimization was employed. Ballan *et al*. [9] followed the same general principle for tracking two hands in interaction with an object. However, they incorporated stronger preprocessing of their input that was based on elaborate discriminative modelling and they considered the Levenberg-Marquardt algorithm for performing optimization. Interestingly, in all works it is acknowledged that inference over parts is more successful when all constituents are considered jointly, through their accounted interactions.

Multi-object tracking has also been approached as a Bayesian filtering problem. The Probability Hypothesis Density (PHD) filter is a prevalent approach to the multi-object tracking problem. PHD, and generalizations *e.g*. cardinalized PHD (CPHD) [103], have been used to tackle 2D and 3D trajectory estimation problems for point entities. These frameworks have not been used for problems with nature related to ours. We believe the reason for this, among others, is the non-triviality in handling highly articulated entities. This is exemplified in the filtering-based 3D hand tracking method in [96].

The works in [47,49,86] are most relevant to our proposal in the sense that they provide 3D positions of multiple objects across time. Kim *et al*. [47] performed simultaneous camera and multi-object pose estimation in real time by exploiting SIFT features. However, the bottom up nature of the work allows for limited robustness and extensibility. Salzmann and Urtasun [86] derived a convex formulation over a physical model of interaction of multiple objects that enabled tracking in weak perspective projection scenarios. Despite the beneficial optimization traits, it is not straightforward to extend this approach in contexts that might not still allow for convex optimization. As shown later in section 6.3, we were able to track multiple entities, a hand and several objects of known structure,

from RGBD sequences by employing the physics-powered single actor hypothesis. This hypothesis distinguished entities into being active or passive, and allowed for virtually arbitrary counts of passive objects to be tracked without increasing the search space. However, the same did not hold for active entities, where the same issue as with [65,69] applies. In contrast to the above mentioned state of the art approaches, the proposed Ensemble of Collaborative Trackers performs multi object 3D tracking that scales very well with the number of active objects to be tracked.

To accurately and efficiently perform 3D tracking of multiple entities we extended our previous methods (see section 6.1). The extension is not straightforward, as simply augmenting the tracking problem by adding more parameters to be optimized soon reaches an infeasibility state, where black box optimization becomes impractical, given limited resources.

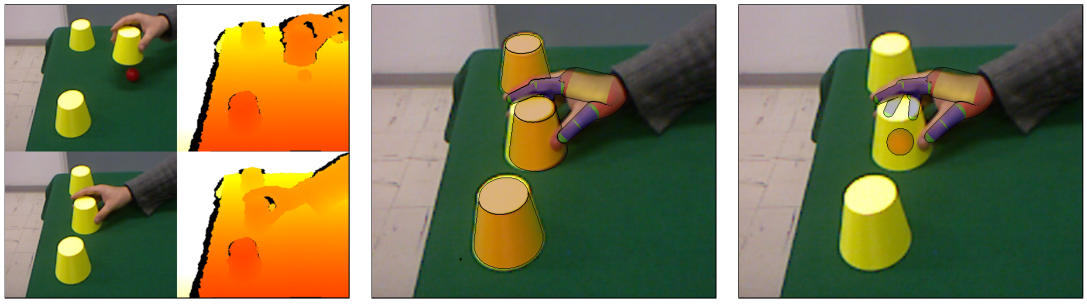
We exploit temporal continuity and carefully reformulate the tracking problem, so instead of a single big problem, several simpler problems are formed. This decomposition intuitively assigns smaller problems to distinct optimization phases. These problems regard distinct entities. However, the decomposition is not a simple disregard of the interactions between the entities (represented by SIT here). We carefully interconnect the smaller and easier problems, so as to best approximate, in practice, the desired traits of our previous joint optimization proposals (see section 6.1), which constituted the state of the art. As a result, we came up with a method that not only outperforms our previous state of the art methods but also solves significantly harder problems, that neither our previous methods, nor any other, could handle, at all.

6.3 Towards more compact models of interaction

In several hand-object(s) interaction scenarios, the change in the objects' state is a direct consequence of the hand's motion. This has a straightforward representation in Newtonian dynamics. We present the first approach that exploits this observation to perform model-based 3D tracking of a table-top scene comprising passive objects and an active hand. Our forward modelling of 3D hand-object(s) interaction regards both the appearance and the physical state of the scene and is parameterized over the hand motion (26 DoFs) between two successive instants in time. We demonstrate that our approach manages to track the 3D pose of all objects and the 3D pose and articulation of the hand by only searching for the parameters of the hand motion. In the proposed framework, covert scene state is inferred by connecting it to the overt state, through the incorporation of physics. Thus, our tracking approach treats a variety of challenging observability issues in a principled manner, without the need to resort to heuristics.

6.3.1 The problem

One of the major goals of computer vision is to extract meaningful interpretations of the world based on the analysis of visual data. This work focuses on a scenario, where the hand of a human actor interacts with a number of objects placed on a table. A fundamental step towards the interpretation of this interaction is the monitoring of the state of the scene, *i.e.*, the 3D position and orientation of the objects and the 3D position, orientation and full articulation of the actor's hand.



(a) (b) (c)

Figure 6.16: The exploitation of the *single actor hypothesis* through *physics modelling*, allows physically plausible, heuristic-free 3D tracking of hand-object interactions. (a) RGBD observation of a hand interacting with objects. (b), (c) By searching for hand motion only, we are able to track the 3D state of the entire scene. The state can be *overt* (partially visible hand and objects (b)) or even *covert* (totally occluded objects like the ball-inside-the-cup (c)).

6.3.2 The solution

Dynamics, as a rich and powerful modelling tool, constitutes an excellent framework where the *single actor hypothesis* is naturally expressed. It is rich because it introduces new types of data, such as mass, energy, friction, restitution, *etc*. Additionally, it is powerful because the predictive power of dynamics is the most elaborate reflection of how entities interact in a truly physical world.

The approach can be summarized as follows. A hand motion is sought that best explains the evolution of a scene between two consecutive time instants t and $t+1$. Hand motion is parameterized as the transition from a reference hand pose h_t (*e.g*. provided by tracking) to a new hand pose h_{t+1} and, thus, is defined by h_{t+1}, alone. As new observations arrive, a new tracking frame is defined, for which the tracking solution is established by a hypothesize-and-test fashion, driven by Particle Swarm Optimization (PSO) [43]. Hypotheses of hand motion are tested in a physics-based simulation environment and the outlook of the induced scene state is rendered into maps that are comparable to the observations. The discrepancy between observations and rendered hypotheses is quantified in an objective function that is minimized through PSO. The sought solution is a physically plausible scene interpretation that is most compatible with the observations. The instantiation of the generic framework (see chapter 4) is depicted as an augmentation of the instantiation for single hand tracking (see section 5.1) in fig. 6.17.

Forward model

We use a forward model that regards the physical state of a scene and its appearance, as observed by a camera. This model is parameterized over a hand motion, *i.e*. two hand poses (a source and a target) in successive time instants. Given a hand motion, forward modelling produces two different outputs. First, through dynamics simulation, it updates the poses and velocities of objects, as these have been altered due to the hypothesized hand motion. Second, the resulting scene state is rendered so that a direct comparison between hypotheses and actual observations is possible.

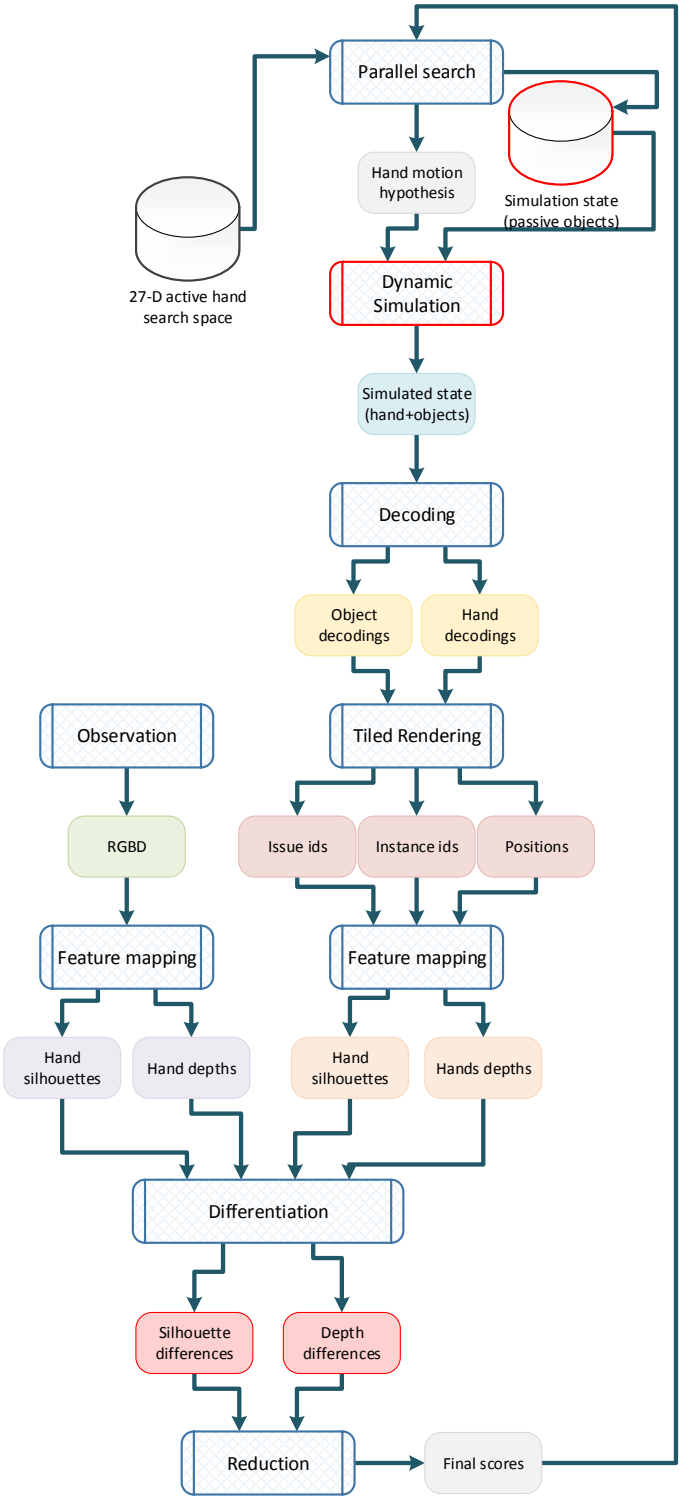
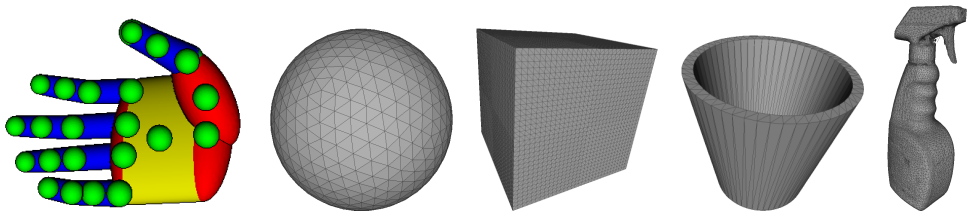


Figure 6.17: An adjustment of the information flow presented in fig. 5.4 for the case of tracking a hand and multiple entities in a reduced search space. Notice the additions of modules, highlighted in red. The search space regards only the hand, but the entire scene is tracked, by applying the consequences of the hand motion to the simulated state of the rest of the scene. Each new hypothesis is combined with the current simulation state. The simulation state is updated once the optimal hand motion has been recovered.



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

Figure 6.18: The physical entities that are considered in our framework. The hand model (a) comprises 22 ellipses and 15 cylinders, appropriately positioned, rotated and scaled. The collision spheres (green) inside the hand model give it physical substance. We consider a variety of object specifications such as a ball and and box (analytical expressions) a cup (designed and then printed) and a bottle (3D scanned).

Dynamics model We are interested in table-top scenes, that consist of a static table, multiple objects and a right hand, all in 3D. All entities are represented in a dynamics simulator (*Bullet* [23]). Entities are essentially represented as 3D shapes with inertia tensors, masses, friction and restitution coefficients. Inertia tensors and masses reflect a body's resistance towards accelerations. Friction coefficients express the amount of energy that is transferred from body collisions to tangential accelerations. Restitution factors modulate the amount of energy that is lost during collisions. All of the above hardly reflect realistic conditions and only bare relative significance, since they are simulator- or application-specific. Still, the selected simulator can generate realistic dynamic behaviour, which is the key in extracting physically plausible scene interpretations.

Bullet supports collision shape representations such as analytical expressions, convex hulls and arbitrary shapes in the form of triangular meshes. We represent a table as an appropriately sized and positioned parallelepiped. Its mass and inertia are infinite so that it is immovable. Shapes can be provided as a pre-production or post-production specification of everyday objects (see fig. 6.18). For the case of shapes like boxes, ellipses, cylinders and their compounds, analytical representations are used as collision models. In all other cases, we resort to triangular shape approximations. All objects are considered to be of equal mass and the inertia tensors are automatically computed by *Bullet,* through shape analysis. The gravitational force is always exerted, at all objects, in a direction that is opposite to the normal vector of the table's top surface, and with a magnitude of $10m/s^2$.

The human hand is a special case. It's 3D structure is defined similarly to [63] and thus is represented by 27 parameters, that regard the absolute 3D pose of the palm and the relative articulation of the fingers. In contrast to the rest of the objects, the hand's collision model is abstracted in order to make modelling and inference tractable. The hand is able to change the state of the scene by means of forces that are the result of its accelerated surface contacting the surface of the objects. We approximate the effective surface of the hand by a compound of spheres that are strategically inscribed at various locations inside the 3D volume of the hand's structure (see fig. 6.18(a)). If s_k is the k-th sphere of the collision model and its 3D position is given through the application of the kinematics function $\mathbf{K}_k(h)$ for a hand pose h, then for a hand motion from h_t to h_{t+1}, s_k is given a velocity $\vec{v}_k = \left(\mathbf{K}_k\left(h_{t+1}\right) - \mathbf{K}_k\left(h_t\right)\right)/\Delta t$. Due to their mass, velocity and friction, these spheres can act as point forces on the surface of the scene's objects. The spheres of the hand's collision model are not allowed to rotate, so that all tangential collision energy is

transferred to the colliding object and is not spent on the rotation of the spheres, too. This enables the hand to pick up objects without allowing them to slide through rolling spheres. The collisions among the spheres are ignored so as to better approximate the flexibility of the hand's surface, by accounting for the whole hand collision model as a union rather as a collection of independent entities. By modulating the mass and friction coefficient of the spheres the hand becomes less/more capable of manipulating heavy or slippery objects. No gravitational force is assigned to the spheres as it is assumed to always be eliminated by the torques of the hand joints.

For a hand motion, a given state of the rest of the scene and a time step, simulation of dynamics is responsible for evolving the scene into a new, physically plausible state. The hand spheres bare kinetic energy and transfer that energy, through collision, to other objects. Dynamics simulation is responsible for applying collision checking, force direction estimation and preservation of energy and momentum in order to transform the old scene state to the new one. All states contain information that regards pose (position and orientation, and thus, potential energy) and velocity (and therefore kinetic energy) for every entity being simulated.

In notation, if h_t is the initial hand pose of a hand motion and h_{t+1} is the target hand pose, then the next scene state S_{t+1} is computed via the simulation process **S**, as a function of the current scene state S_t:

$$S_{t+1} = \mathbf{S}\left(h_t, h_{t+1}, S_t\right), \tag{6.2}$$

where S_k is the full description of the physical state at time k for N entities:

$$\mathrm{S}_k = \{\{s_i, m_i, I_i, F_i, R_i, \vec{p}_i, \vec{q}_i, \vec{v}_i, \vec{\alpha}_i\} \,|i = 1 \ldots N\} \tag{6.3}$$

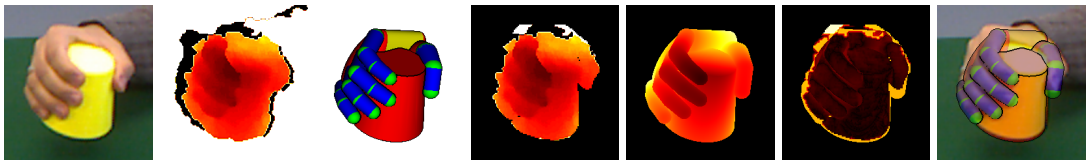
In eq. (6.3), s_i is the collision shape, m_i is the mass, I_i is the inertia tensor, F_i is the friction coefficient, R_i is the restitution coefficient, $\vec{p}_i$ is the position, $\vec{q}_i$ is the orientation, $\vec{v}_i$ is the linear velocity and $\vec{\alpha}_i$ is the angular velocity of body i, at time step k. For each individual object, the applied forces and torques are the accumulated result of the total simulated interaction. All vectors are in 3D. The time step Δt is inferred from t and $t+1$.

Appearance model Every hand motion hypothesis yields a new expectation over the physical state of the scene. This expectation needs to be made comparable to observations so that the corresponding hypothesis can be evaluated. A hypothesis scores well when its simulated expectations over the scene evolution match the new observations well.

The result of a simulation is the 3D state of all accounted entities. Such results can be made comparable to observations by means of rendering. Appearance modeling here is identical to the one in section 6.2. As an exception, the foreground is subtracted from the background by means of background modeling (see section 4.5.2).

Tracking loop

Given a table-top scene containing objects and a human hand, initialization is performed. The table is detected by means of RANSAC plane fitting, and from its normal vector the gravity vector's direction is inferred. 3D models of the objects that can be found in the scene are assumed to be available in a database. The initial registration (estimating 3D position and orientation) of these models to the actual 3D point cloud provided by the RGBD camera is performed using the method described in [75]. The initial configuration of the hand is predefined (see fig. 6.21).



(a) (b) (c) (d) (e) (f) (g)

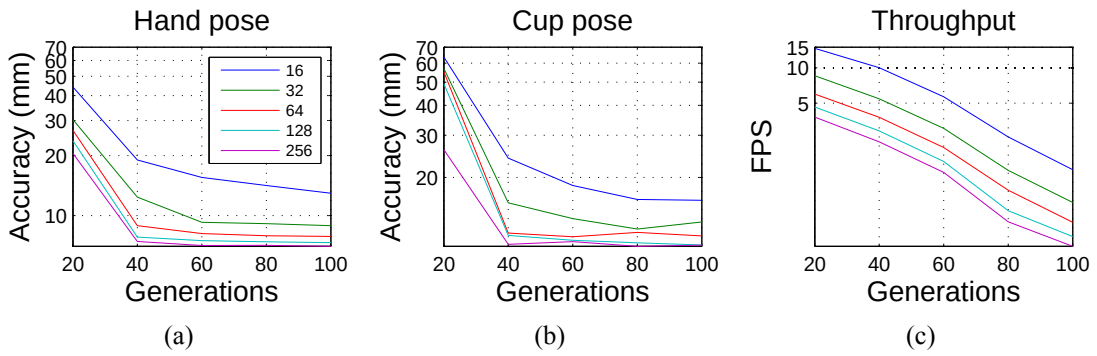
Figure 6.19: (a), (b) RGBD input. (d) Masked depth image I_d^o. For a hand motion hypothesis h (c), a synthetic depth map I_d^r is rendered (e). The difference between I_d^o and I_d^r (f) yields the fitness of h. The best scoring h, computed by PSO, is the tracking solution (g) for the current frame.

A new iteration of tracking is performed as soon as new observations become available at time t. As a first step, foreground segmentation is applied. A hypothesis of a new hand pose h_t amounts to a relative motion with respect to h_{t-1}. All simulations conducted to estimate S_t are evolutions of S_{t-1}. PSO is delegated with the task of minimizing the penalty function $\mathbf{E}$, for the new observations, in order to find the minimizer hand motion. The PSO population of hand poses is initialized in the vicinity of the solution for the previous frame, so that search is more efficient on image sequences that are sampled densely in time. As PSO searches for the fittest hypothesis, multiple scenarios of interaction are simulated, rendered and compared to actual observations. Although the actor's arm/body is not modelled and belongs to the foreground, it does not influence inference. This is because the computations of eq. (3.10) are performed inside a 2D bounding box of the image projection of each and every modelled entity (hand, objects). Being tight, such bounding boxes contain very few observations of the actor other than the hand. The hypothesis that optimally explains the evolution of a scene in terms of appearance is dubbed as the tracking solution for the current frame. The scene state that accompanies the winning hypothesis replaces S_{t-1} for the next tracking frame.

6.3.3 Experiments

For the evaluation of the presented framework we employed a parallel implementation. We used GPU threads for 3D rendering and objective evaluation and CPU threads for dynamics simulation. All experiments were executed on a machine with the following specifications: quad-core Intel i7 920 CPU, 6 GBs RAM and a 1581GFlops Nvidia GTX 580 GPU with 1.5 GBs RAM. For the image acquisition process we employed a Kinect sensor and the OpenNI framework. Acquisition was performed at a $30fps$ rate and therefore the dynamics simulation time interval was set to $\Delta t = 1/30s$. This interval was subsequently subdivided into smaller intervals ($10\times$), so as to remedy the lack of robust continuous collision detection for complex geometries.

Unless otherwise stated, PSO parameterization amounted to 64 particles and 100 generations. Each hypothesis was rendered in a series of surfaces (one per simulated entity) of 50×50 pixels each. The effective area over the observations that rendering and evaluation considered at each step was based on per-entity bounding box computations for the solution of the previous tracking step. The values $\lambda_P = 10$, $\lambda_D = 1$ and $\lambda_F = 0.9$ were used in all experiments. The mass of the hand model collision spheres was set to 1 and the friction factor to 10. This combination of factors yields a powerful and dexterous hand, that still requires at least two opposing fingers in order to pick up objects. T_d was set to $40mm$.



(a) (b) (c)

Figure 6.20: Quantitative results. Distinct curves correspond to different particle counts. Logarithmic scale was used in the vertical axes for better resolution over small value differences.

Quantitative evaluation

For the problem of scene tracking and when a hand is involved, it is very difficult to acquire ground truth information. In an effort to produce data that can be used as a ground truth we conducted the following experiment. A human hand grasped a cup firmly, lifted it and moved it around in various angles. In a sequence of 500 frames, the hand grasped the cup firmly in the last 370 frames (see the 1^{st} column of fig. 6.21). By construction, the pose of the hand was correlated with the pose of the cup. We tracked this scene and thus gained access to the inferred poses of the hand and the cup. We measured the standard deviation of the distance between the estimated position of the hand and that of the object, during grasping, and we found this to be $3.7mm$. Similarly, the standard deviation of the difference in hand/cup poses was equal to $1.3°$. These measurements indicate that tracking was successful in capturing the tight spatial hand/object relationship, despite the object's rotational symmetry, that hinders orientation estimation, and despite the strong hand-hand and hand-cup occlusions.

In a second experiment, we tested the optimizer's ability to effectively solve the tracking problem by using synthetic data for which ground truth was available. Such data (depths and silhouettes) were produced from the tracking result of the previous experiment. Being an output of our framework, these data are compatible by construction. We tested different budgets (*i.e.*, allowed count of objective function evaluations) that were distributed across various particle and generation count combinations. For each budget, a tracking experiment was repeated 100 times and the distance between the resulting track and the ground truth was measured using the accuracy measure proposed in [65]. The corresponding accuracy results are shown in fig. 6.20(a) and fig. 6.20(b), where each point represents the median error across all tracking repetitions. The tracking frame rate for each budget is shown in fig. 6.20(c).

What can be seen from fig. 6.20(a) and fig. 6.20(b) is that generally, as budget grows, a better accuracy is achieved. As expected, we can trade accuracy for speed. In the synthetic experiments even low budgets suffice for adequately accurate tracking of both the hand and the object. There, budgets that yielded as much as $3fps$ could produce adequately accurate tracks. For real-world experiments, exhibiting non-ideal observations and inexact modelling, we resorted to a greater budget that still yielded acceptable performance (64 particles and 100 generations, $0.5fps$).

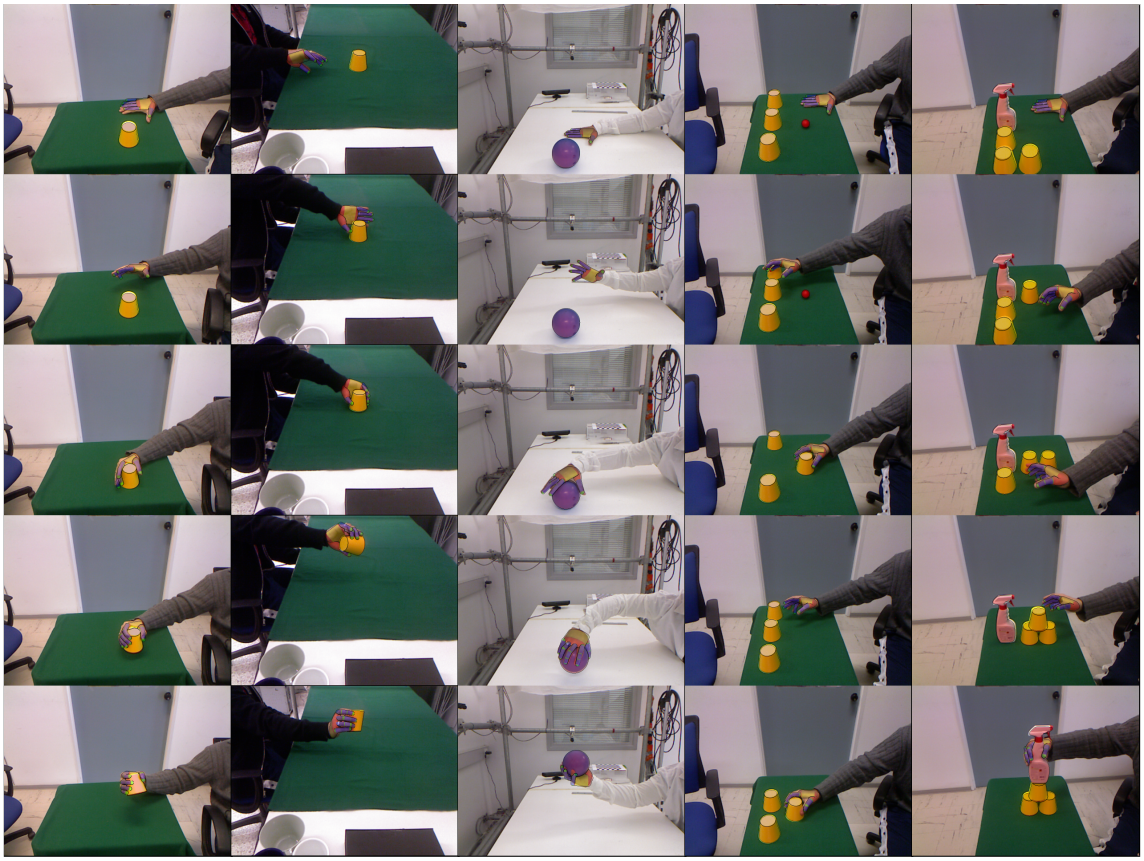


Figure 6.21: Tracking results, superimposed over the respective RGB input. Leftmost column: the sequence that was used for quantitative assessment. Rest columns: qualitative evaluation. Detailed presentation of the entire sequences that highlight the efficacy of the proposed method can be found in the supplementary material.

Qualitative evaluation

We conducted a series of real-life experiments to test whether (a) our modeling of dynamics and appearance is adequate to match real observations (b) tracking complex interactions can be achieved with the proposed optimization framework and, (c) our *single actor hypothesis* is well reflected "in the wild". In all experiments, optimization was performed only on the parameters of the actor's hand. The (correct) motion of all objects was inferred as a consequence of the hand's motion that best explained the observations in total. All videos were recorded at 30fps. They were processed in $1\times$ and $2\times$ speeds, in order to simulate both normal and faster motion, yielding identical tracking performance. The recorded sequences and the respective results can be viewed at `http://youtu.be/0RCsQPXeHRQ` .

The first experiment considered a hand and a cup (2^{nd} column of fig. 6.21). At this footage the hand picked up the cup and put it back on the table in an upside-down orientation. As it can be verified in fig. 6.21, the proposed method successfully provided a physically plausible track.

In the second experiment a hand lifted and manipulated a plastic bowling ball that was barely graspable due to its size (3^{rd} column of fig. 6.21). Given enough friction, our hand modelling was able to explain the lifting and manipulation of the object. Even when almost the entire hand was occluded by the ball we came up with plausible hypotheses. In

both previous situations, consulting physics yielded that an object that was to remain in the air required finger support, which, even if barely observed, was hypothesized in order to achieve overall consistency with the observations.

In the third experiment we considered a more elaborate case of interaction that induced cascaded occlusions. We demonstrated the shell game with three cups and one ball (4^{th} column of fig. 6.21). One cup trapped the ball and was moved around, moving the ball inside it and pushing other cups when in its way. At some point in time, a chained interaction occurred. The hand pushed an empty cup, which in turn pushed the cup containing the ball. As the hand shuffled the cups intense occlusions occurred that did not prevent our framework from maintaining plausible hypotheses about the 3D position and orientation of the fully/partially occluded hand, cups and of the truly invisible ball.

A final experiment regarded object stacking. Concave objects were stacked, one by one, forming a pyramid (5^{th} column of fig. 6.21). This scenario challenged both the dynamics modelling and the optimization module, because stacking of generic geometry is indeed a difficult problem for dynamics simulators to handle stably, which in turn yields an erratic behaviour in the objective function. However, PSO, overcoming the problem, yielded plausible hypotheses and thus the pyramid was tracked well.

6.3.4 Literature review and contribution

A key observation is that the human hand is the *single actor* and *scene state changes can be attributed to the actions of the human hand and their induced consequences*. Given that the physical world and its visual observations are determined by the laws of physics, we focus on how computer vision may benefit from *explicitly* accounting for these laws. Thus, we model the dynamics of a hand interacting with a physical world. Moreover, we make a distinction between active and passive entities, to come up with effective, physically plausible interpretations of scenes, exhibiting complex hand-object(s) interaction.

As in the rest of our work, we monitor the 3D state of the scene by means of tracking (see fig. 6.16), which is defined as an optimization problem. Once again, the objective function is a quantification of the discrepancy between a given hypothesis over the scene state and observations. Here, this function is parameterized over a hand motion between two successive instants in time. A hypothesized hand motion is simulated in a physics-based simulation environment that reflects the latest state of the scene, as it has been tracked up to that point in time. The simulated hand-object(s) interaction yields an expectation over the appearance of such a hypothesis, that regards both the hand and the object(s). A comparison of this expectation to actual observations quantifies the compatibility of the hand motion hypothesis to the data. A highly preferable hypothesis is one that explains (a) where the hand is in the new tracking frame and (b) the consequences of its interaction with the scene, as those are reflected in the observations. The expectation and comparison mechanisms are implemented as a forward model that accounts for the dynamics and the appearance of a scene. This model is turned into an inference mechanism over the physical state of the scene by means of black-box optimization.

We show that under the *single actor hypothesis*, our approach is able to track complex scenes. At the same time, by modelling dynamics, we bring scene understanding to a level where various problems (constrained observability, scene cardinality, *etc*.) are resolved effortlessly, uniformly and without the need to resort to heuristics.

Our work aims at deriving physically plausible interpretations of the interaction of a human with the environment. In this context, we are interested in approaches that study

the interaction of humans with their environment and/or incorporate physics/dynamics to improve vision processes.

Several researchers have exploited dynamics, by introducing interesting abstractions of physical phenomena, in order to tackle scene understanding problems. Brand *et al*. [12] exploited the physical notion of causality to perform qualitative reasoning in computer vision problems. Mann *et al*. [55] methodically generated 2D hypotheses for simple scenes viewed by a single camera. The roles of the scene's constituents were then ranked based on *physical plausibility*. Delamarre [30] assigned physical behavior to a contour model that drove a reconstructive optimization process. Papadourakis and Argyros [74] identified the physical notion of object permanence as the ambiguity resolver for the case of multiple objects tracking. Gupta *et al*. [36] used the notion of physical stability to hypothesize physically plausible 3D scene interpretations.

Several other approaches consider dynamics explicitly, but restrict understanding to either the actor or a single object, only. Human body dynamics has been exploited towards the formation of strong yet compact priors [105]. Urtasun et al. [102] modeled the dynamics of the golf swing motion to track golf swings in 3D from a single camera. Popović and Witkin [79] rectified 3D motion capture data to make it compliant to physical constraints. Vondrak *et al*. [104] fused motion planning and contact dynamics to track humans from multiple cameras and a ground assumption. Brubaker *et al*. [16] employed realistic metaphors of the lower body dynamics to estimate and predict walking. Going further, they incorporated a friction model for a ground that affords human motion upon it [17]. Bhat *et al*. [10] performed 3D tracking of an object by searching over parameterized experiments that optimally projected back to an image sequence. Duff and Wyatt [33] used physical simulation and search heuristics to track a fast moving ball, despite occlusions and for the 2D case. In previous work [52], we performed 3D motion estimation for a bouncing ball, from a single camera and despite severe occlusions by exploiting dynamics modelling. Ye and Liu [116] synthesized physically plausible hand movements, from pour or absent hand observations, that explained the manipulation of objects with known trajectories from a hand whose rough location was also known.

There are also approaches that go beyond abstractions of dynamics while considering ensembles of entities rather than entities in isolation. Metaxas and Terzopoulos [56] defined a continuous Kalman filter that was able to track a deformable object. Although interesting, the proposed approach is of limited extensibility and is susceptible to overfitting. Salzmann and Urtasun [86] approached the problem of 3D tracking by attributing motion of parts to net forces that act upon them at each tracking frame. Because of the lack of explicit structure, this method is also susceptible to overfitting. Scaling to different types of interaction or introducing more structure is not straightforward.

Kjellstrom et al. [48] improved the estimation of the 3D pose of the human body while in interaction with easy to track objects, by constraining the hands. In previous work we tracked the constellation of a hand and an object from multiple cameras [65], and the full articulation of two interacting hands from a RGBD sensor [69], all in 3D, by employing synthetic 3D models. Ballan et al. [9] captured delicate interaction between two hands and an object from multiple high resolution cameras using 3D models of high fidelity. While inspired by the fundamentals of interactions, none of these approaches considered dynamics directly. Moreover, in all cases, the consideration of more objects would require the increase of the problem dimensionality.

In this work, by considering (a) the dynamics of a scene as the core representation in a dynamics simulator, (b) 3D rendering as an appearance forward model and (c) black-

box optimization as the solution to decoupling inference from modelling, we come up with a framework that introduces a novel forward model for dynamic scenes. We use this framework to tackle a series of challenging vision-based tracking scenarios. To the best of our knowledge, none of the existing techniques can cope with the complexity of these scenarios. Still, all of them are treated invariably and are handled effectively within the proposed framework.

As an example, in a table-top scenario, regardless of how many cameras overlook the scene, there are always problems related to *observability* due to occlusions. While a hand transports an object, we do not directly perceive the hand touching it. Still, we know that touching, *i.e*. force exertion, happens, because otherwise the object could not be lifted. Our recently proposed method in [65] demonstrates multicamera-based joint hand-object tracking that is performed based on two criteria: (a) the appearance of the hand-object ensemble matches the observations and (b) the hand does not share the same space as the object. Both criteria are fulfilled by a hand that has proper articulation, is close to the object but does not touch it. But if the hand manages to lift/transport the object, it is clear that the above interpretation is not plausible. Being physics-based, our framework is forced to compute a plausible solution.

Another challenging observability issue is severe occlusions, *e.g*. caused because of *containment*. In the example of the shell game (see fig. 4.1), a ball being covered by a cup can no longer be seen. However, as humans, we do hold expectations over an evolving scene, despite the complexity of interaction and the severe and temporally extended occlusions. This has been successfully identified in [74], where "object permanence", *i.e*. the expectation of an occluded object reappearing close to its occluder, gave rise to a discrete logic that can handle challenging tracking scenarios. Still, the "object permanence" principle will fail if the ball-in-the-cup passes over a hole of the table. Containment and the resulting occlusions go beyond the heuristic of "object permanence". The laws of physics guarantee that a ball that is trapped between a cup and the table, has to travel inside the cup being moved by a hand. Such cases, where *lack of observation* can be remedied by the consideration of physics, are effortlessly handled within our framework.

Another important issue is *scene cardinality*. In the approaches taken by [9,48,65,69], tracking additional rigid entities requires increasing the problem dimensionality which, in turn, makes optimization increasingly harder. Instead, within our framework and as long as the *single actor hypothesis* holds, tracking scenes of different cardinalities does not alter the dimensionality of the problem.

Chapter 7

Higher level inference

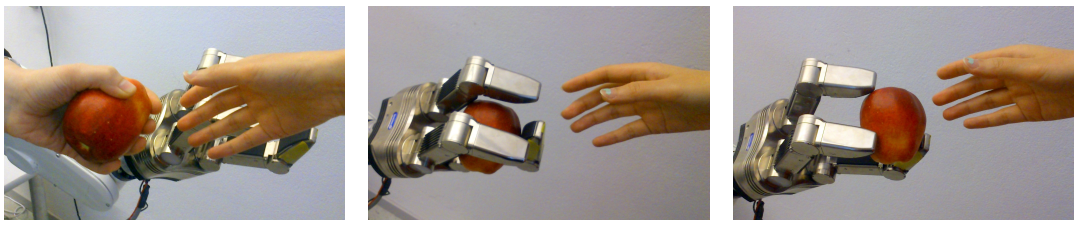
Here, we present two approaches that exploit our 3D hand tracking methods to perform higher level inference, which regards understanding of hand motion, in the context of object manipulation. These approaches have been proposed by Song *et al*. [92] and Patel *et al*. [77].

We employed our 3D hand-object tracking approaches, to process captured human performances and extract accurate 3D configurations for the hand and a manipulated object. Our involvement in these works has constituted an enabling technology. However, our role was limited in providing input to be further processed by higher level modules. Therefore the discussed works are presented here compactly, so as to convey, through examples, what can be achieved upon detailed and accurate 3D tracking.

7.1 Understanding intention

To use a hammer against a nail humans always grasp it by the designed handle, because this optimizes the task. Whereas, in handing a hammer over to another human, the head of the hammer is preferred so as to leave room for the handle to be immediately grasped by the receiver (also a matter of optimization). This reasoning can be generalized over several objects and tasks of known types. By knowing the detailed 3D configuration of the hand with respect to an object and the type of the object itself an inference task can be designed which answers what the manipulation intention is.

This idea has been investigated and implemented by Song *et al*. [92]. There, intention inference, probabilistically implemented by employing a Bayesian formulation, is the significant part of a process which tackles robotic imitation. More specifically, intention is lifted to a representation which is understandable and executable by a robot. This lifting is required as the human hand is so different to robotic embodiments that directly mapping control from one to the other becomes problematic (see fig. 7.1). Our role in this work has been to capture, in 3D and in detail, such manipulations. The joint probability density function (jPDF) of the 3D tracks, object types and manipulation intents, are learned from such captures. This jPDF can then be queried for the probability of manipulation intents given pre-grasp object-relative hand configurations and object types. Intuitively, our direct and accurate 3D scene representation has successfully supported intention inference, by providing data with high discriminative value. More specifically, if different actions can be distinguished then their 3D representations should differ as well. For details the reader is referred to [92]. Our role, as a part of the vision pipeline, is depicted in fig. 7.2.



(a) Demo of Hand-over Task (b) Imitate Power Grasp (c) Imitate Hand-over Task

Figure 7.1: (a) Human demonstrates a grasp with an intention to *hand-over an apple*. (b) Robot imitates the power grasp configuration used by the human, and fails to hand-over because there is not enough free space for regrasp. (c) Robot estimates that human intends to hand-over the apple. It also learns the task requires leaving enough free space on the object. It applies a precision grasp to achieve the task. This figure has been borrowed from [92].

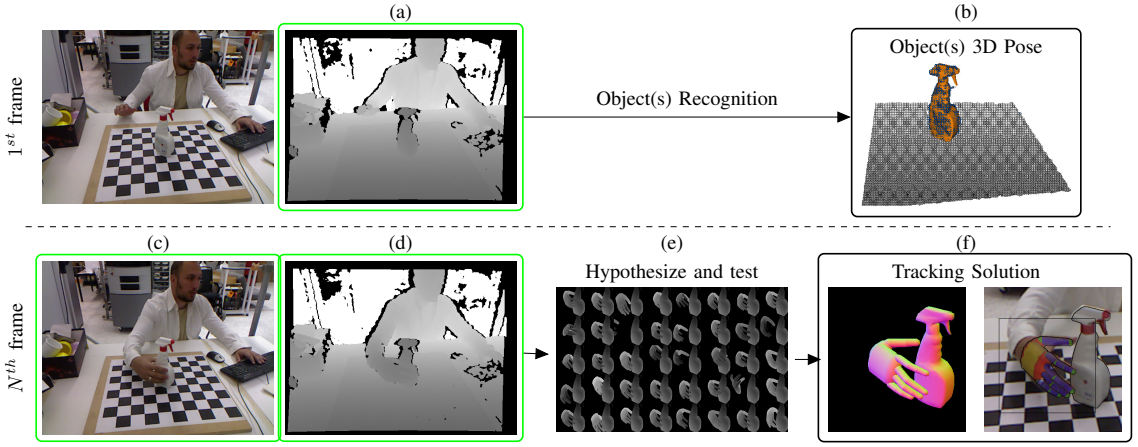


Figure 7.2: The vision system of [92]. A database of known objects is pre-established. The objects in a scene that correspond to entries in this database are automatically extracted from the first observation frame. This information is enough to bootstrap hand-object(s) tracking. The channels of information of each frame that are used in each phase are outlined with green borders. For more details please refer to [92], the original source of this figure.

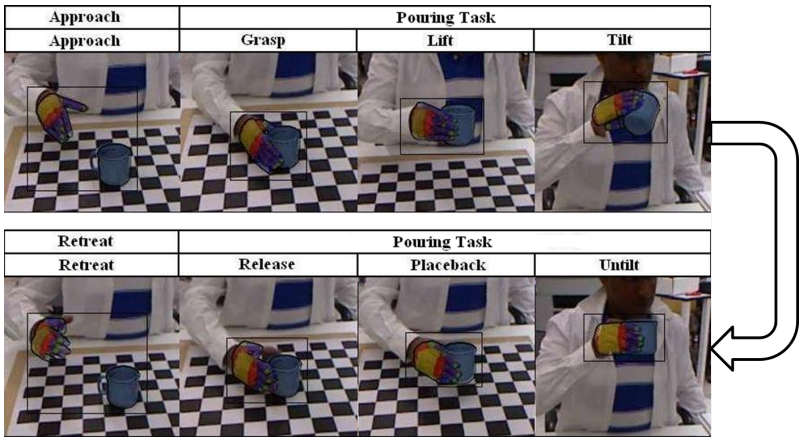


Figure 7.3: Task of *Pouring* water from mug subdivided into action primitives. Each image depicts the output of hand-object tracking algorithm. For more details please refer to [77], *i.e*. the origin of this figure.

7.2 Extracting manipulation alphabets

Everyday tasks come as repetitions over what could be identified as common sub-tasks. For example, pouring the content of a cup can be decomposed into a sequence of parts, some of which can be also met in other actions (see fig. 7.3). In the same figure it can be seen that this sequencing can also be hierarchical. For example, a pouring action, at a high level, requires approaching the object, performing the actual pouring motion and then moving away from the object. In more detail, pouring means grasping, lifting, tilting, untilting, placing back and releasing the cup, in sequence.

This pattern is identified and exploited by Patel *et al*. [77]. In this work observations of humans performing everyday tasks are *parsed*, in a probabilistic fashion, with an *action grammar*. This grammar is established, in the form of a Hierarchical Hidden Markov Model (HHMM), over predefined symbols (tasks, sub-tasks, observations). The objects accounted for can be seen at fig. 7.4. The sequencing of symbols is learned from annotated examples. These examples are manually annotated 3D recordings of a human subject performing everyday tasks. Our role in this approach is to provide accurate 3D tracks of the subject's hand and a single rigid object (several types), which is also involved in some everyday task. We provided such tracks, with high accuracy, by employing our 3D tracking framework (see section 6.1). The learned grammar can be used to parse new sequences, tracked with our framework, in order to lift the detailed 3D tracks we provide into the human-friendly representation of hierarchical symbol sequences.

Figure 7.4: Objects, which belong to different classes, according to the type of their evey-day use, have been considered for the problem of extracting manipulation alphabets. Several videos showing these objects being involved in everyday tasks have been recorded and processed.

Chapter 8

Discussion

The theme of this thesis has been the proposal of computational methods for performing 3D scene understanding. This thesis serves as an answer to a series of reserach-related questions. These questions are discussed in the following.

8.1 Identification of literature gap

The main problem this thesis discusses is 3D scene understanding. This is an old problem, in its generality, which, however, has not yet received a fulfilling answer, even partly. We identify that the reason for this has been the constraints in the tools incorporated, which stem from the difficulty in defining appropriate models in computer vision processes. Appropriateness regards both the expressiveness of the models, *i.e*. how well they encapsulate the expected variability in observations, and the cost of using such models, *i.e*. how easy or even feasible it is to incorporate such models successfully and efficiently. We have identified that there are areas in which there is plenty of room for improvement. At the low level, according to which knowing some type of scene state precedes understanding itself, we have identified that the previous state of the art was limiting in handling complex scenes with intense interaction, let alone doing so accurately and efficiently. The availability of tools which would provide detail and accuracy of high levels would have also allowed for higher level methods to be richer and more robust.

8.2 Significance of the problem

The usefulness of improving the current state of the art in the area of 3D scene understanding is axiomatic. A glimpse of the impact of providing strong methods that would tackle real-life problems has been given in section 1.3. The magnitude of the significance is proportionate to the expected improvement and impact. We have discussed in chapter 2 and shown in chapter 5, chapter 6 and chapter 7 how our contributions have inflicted major improvements in several domains.

8.3 Contribution

The research questions that have guided the process of formulating this thesis have not been answered in full. In other words, we do not claim to have solved the discussed problems completely. However, the improvement on the previous state of the art is significant.

Even more importantly, the foundation which this thesis has established sets the loam for further investigation, with rigor and flexibility, starting at a point already much closer to the final goal (see section 8.4). At the low level we have made breakthroughs with respect to the accuracy and efficiency of 3D tracking, for problems of increasing difficulty, *e.g*. tracking a single hand, two hands, hands and a few objects or even hands with many objects. These results cannot be said to be purely low level, as they also rigorously incorporate high level reasoning, such that involves behaviour and dynamics. Still, purely high-level methods have also been enabled, implemented and presented, thanks to the detail and accuracy of our 3D tracking results.

8.4 Limitations

One of the main pursuits of this thesis has been the attainment of such a degree of flexibility which would allow for the uncompromised incorporation of strong and already established modelling tools, from the fields of Computer Graphics and Physics, in computer vision processes. A significant aftereffect of flexibility has been generalizability, *i.e*. the ability to apply our solutions to greater and/or different domains. Our goal has been achieved up to the point delineated by the hereby reported solutions.

The presented contributions come with limitations. We argue that such limitations can be lifted through specialization. The solutions we have provided are basic and can be elaborated to tackle specific problems, which have been peripheral or disjoint to our central contributions. Next we discuss identified limitations and sketch how future work might be conducted to lift them.

8.4.1 Known models

Throughout this thesis we have been discussing model-based approaches. As suggested by their title, such approaches require the preexistence of models. Models are known beforehand, *i.e*. they are not coined or revised during inference. This is seemingly limiting, in that we can reason only upon models we have good knowledge about, a priori.

However the term known should not be confused with the term static. In the presented work we have been dealing with models, significant parts of which were static, *i.e*. remained the same throughout inference. For example, in 3D hand tracking (see section 5.1) the kinematics remained constant, as well as the 3D primitives which comprised the hand's shape. In reality, the hand is far less rigid, with musculoskeletal actuation and blood flow changing its shape well beyond rigid articulation. Such effects can be incorporated in augmented parametric models. Taking it one step further, the fact that we have been reasoning over rigid or piece-wise rigid entities does not mean our framework cannot handle *e.g*. deformable entities. For example, the state of matter could be expressed as a function of the bond strength between the particles which form a shape. Thus, another parameter which represents exactly that could help model entities that could be solid, liquid or even gas, at one instant or another. Finally, similar to the state of matter, the knowledge of the shape of an entity could also be modelled in a dynamically revisable manner, through careful exposition in a constant number of parameters. The final point is that models might be far more elaborate and still satisfy the simple interface of being parametric, with a known number of parameters. This note is not restricted for modelling 3D shape, but could also be generalized to models of interaction. A far-fetched but clarifying example might be the

parameterized transition from one theory of physics to another one, based on the requirement of different scales (*e.g*. Newtonian dynamics to quantum physics). Clearly, there are a lot of potentially very expressive models to consider, incorporate and test before we identify the requirement for new models, which might fall outside the capabilities of the presented framework.

8.4.2 Track loss

A great issue which regards tracking and that we have not included in our prioritized list of concerns is track loss. The primary reason for losing track is the actual violation of the time continuity assumption. Otherwise stated, tracking might fail when the next true estimate is not *close enough* to the previous estimate. This proximity, which the term close implies, can be modelled in multiple ways. For example, when discussing tracking 3D position there are time continuity models which assume constant position (next estimate is close to the previous one), constant velocity (next estimate is a linear extrapolation), constant acceleration (next estimate is a quadratic extrapolation), *etc*. In our framework, which is powered by evolutionary black-box optimization, there is no restriction in adopting a single motion model, or a transition model in general. Multiple such models can be incorporated rather explicitly. A noninvasive approach might be to initialize multiple competing trackers, with each one adopting a different transition model. This is computationally expensive but it is clear and has a good prospect of being executed efficiently, given the independence of the trackers and the parallelization potential. An invasive approach might be to configure the generation of new hypotheses. Instead of having them all be randomly distributed in the vicinity of the previous estimate, some of them could be spent for testing known and competing motion models, based on the tracking history.

8.4.3 Initialization

A problem, which can be related to track loss (see section 8.4.2), is that of initialization. In all approaches presented (except that of section 5.2), we have been performing local optimization, which was always initialized in the vicinity of a *previous estimate*. However, when the track is lost or when the first frame of a tracking sequence is concerned there is no previous reliable estimate or estimate at all. In this case, local optimization would have to be exchanged for global optimization. We have indeed shown that for a moderately large and complex problem (see the bouncing ball case study in section 5.2) global optimization is indeed feasible. The practical difference in going from local to global is the increase of the optimization budget, which translates in significantly slower execution. Another issue that needs care is the design of the objective function itself. However, we are optimistic that the task is feasible and, also, that, in time, the computational costs will stop imposing an insuperable obstacle.

8.4.4 The single actor hypothesis

In section 6.3 we presented a scalable approach to tracking an actor along with a theoretically arbitrary number of passive objects. This was enabled by the observation that passive objects cannot move on their own and it is only through the actions of the actor that they occasionally do. Our implementation worked well in tracking objects while

they were actuated but, because of design, it cannot handle objects which are given initial energy and are then let go (*e.g*., throw a die on a table). This is because the peculiarities in actual physical interaction cannot be attributed (and thus be reasoned about) to an almighty actor when this actor is no longer in control. One obvious way to tackle this problem is to add more detail in the interaction model, *i.e*. such a model which accounts for the expected peculiarities. However, we think that the defining modification, which would enable reasoning for objects which are not actuated, would be the consideration of a broader observational horizon. Simply stated, the ability to process a few frames ahead would dramatically improve the accuracy in estimating the motion of flying objects. This implies that the optimization problem, which regards the actor alone, would have to be solved for more than one frame at a time (as performed in what was described in section 6.3). So, instead of optimizing over infinitesimally small motion coarser motion models would have to be considered. Our experience in this problem suggests that this is also a feasible task.

8.4.5 Complicating variance

In all the presented work and in the entire literature presented there was always an initial preprocessing step which mapped raw observations to a feature space. This mapping was performed in a way that filters the complicating variance. For example, in 3D hand tracking we expect and model beforehand the 3D shape variance, through 3D rendering, but do not go into detail to model per-pixel coloring, like in [28]. We find the pixel-wise coloring variance to be complicating and filter it out using *e.g*. skin color detection or background subtraction. This reasoning holds for all other presented works, the set of which has led to the definition of the specific feature mapping phases for observations (see section 4.5.2) and for hypotheses (see section 4.6.2). We are certain that the presented feature mapping tools are not complete, and that additional tools would be required in different domains than the ones discussed. However, we do not see this as a problem, but rather as a feature to exploit. More specifically, having clearly identified in a computational framework where complicating variance is filtered it is relatively easy to extend the corresponding phases as required.

An idea we find interesting and has a straightforward application in our framework, in the way sketched so far, is the adoption of an hierarchy of feature mapping tools. The levels of this hierarchy would correspond to different levels of abstraction, at both the observation path and the hypothesis path. Then, comparing features at different levels would then readily answer the question at which abstraction level observations and hypotheses are most compatible. This means that we would thus enable model-based scene understanding which would work in a spectrum of abstractions rather than at a certain level.

Consider the exemplary problem of trying to capture and understand from images the waving motion of the sea. For the waving of a scene we might have abstract simulation models that are able to generate samples of calm or wavy sea conditions. Then, on the other side, we might also have detailed simulation models which are continuous in the effect of several factors (*e.g*., wind state, time in the day, weather conditions, *etc*.) and rich in the samples they provide (*e.g*., prediction of pixel color). This would enable the definition of an hierarchy of features that go all the way from identifying whether the sea is calm or wavy to identifying the conditions which best explain the waviness. The latter seems hard to accomplish, but the hierarchy reasoning allows for both its inexpensive

consideration and the elegant regression to more abstract models when detailed ones fail.

8.5 Epilogue

We have presented a short list of points that could be identified as limitations or future work. These points are not shortcomings of the the computational framework, but rather of our specific instantiations. We have focused on tackling specific problems which regarded interaction, scene scale and, ultimately, flexibility in modelling. We believe our design choices are not restricting and, as discussed earlier, they leave enough room for both specialization and generalization. This specialization or generalization can be performed, as already sketched, through clear modifications of parts of our computational framework. We find most exciting that (a) improving a part automatically improves the whole framework (as exemplified in the works presented) and that (b) the different modules of the framework (*e.g*. rendering, optimization, decoding, *etc*.) themselves correspond to significant areas of research, which means that improvements are indeed to be expected from the respective research communities. Indicatively, the simulation tools we incorporated are not our home-brew products but they have been borrowed from different disciplines. Our design choice over the decoupling of modelling from optimization is what allows modules to maintain simple interfaces, which in turn allows for easy integration of external tools.

Chapter 9

Impact

The presented work has been reported and evaluated by the research community in the form of published papers. Aspects of this work has also been exposed to the research community in the form of downloadable software. Our work has been awarded in two distinct competitive occasions. We provide additional information that is indicative of the overall acceptance of our work.

- Publications
  - Thesis-related publications
    * N. Kyriazis and A. Argyros. Scalable 3d tracking of multiple interacting objects. In *CVPR*, 2014.
    * N. Kyriazis and A. Argyros. Physically plausible 3d scene tracking: The single actor hypothesis. In *CVPR*, 2013.
    * I. Oikonomidis, N. Kyriazis, K. Tzevanidis, and A.A. Argyros. Tracking hand articulations: Relying on 3d visual hulls versus relying on multiple 2d cues. In *Proceedings of the IEEE International Symposium on Ubiquitous Virtual Reality 2013 (ISUVR 2013)*, July 2013. By invitation.
    * P. Douvatzis, I. Oikonomidis, N. Kyriazis, and A. Argyros. Dimensionality reduction for efficient single frame hand pose estimation. In *International Conference on Vision Systems, (ICVS)*, July 2013.
    * D. Song, N. Kyriazis, I. Oikonomidis, C. Papazov, A. Argyros, D. Burschka, and D. Kragic. Predicting human intention in visual observations of hand-object interactions. In *ICRA*, 2013.
    * M. Patel, C. H. Ek, N. Kyriazis, A. Argyros, J. Valls Miro, and D. Kragic. Language for learning complex human-object interactions. In *ICRA*, 2013.
    * I. Oikonomidis, N. Kyriazis, and A. Argyros. Tracking the articulated motion of two strongly interacting hands. In *CVPR*. IEEE, June 2012.
    * I. Oikonomidis, N. Kyriazis, and A. Argyros. Full dof tracking of a hand interacting with an object by modeling occlusions and physical constraints. In *ICCV*. IEEE, 2011.
    * N. Kyriazis, I. Oikonomidis, and A. Argyros. Binding vision to physics based simulation: The case study of a bouncing ball. In *BMVC*. BMVA, 2011.

    * N. Kyriazis, I. Oikonomidis, and A. Argyros. A gpu-powered computational framework for efficient 3d model-based vision. Technical Report TR420, ICS-FORTH, July 2011.
    * I. Oikonomidis, N. Kyriazis, and A. Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC*. BMVA, 2011.
    * I. Oikonomidis, N. Kyriazis, and A. Argyros. Markerless and efficient 26-dof hand pose recovery. In *ACCV*, pages 744–757. Springer, 2010.
  - Extra publications
    * K. Tzevanidis, X. Zabulis, T. Sarmis, P. Koutlemanis, N. Kyriazis, and A. Argyros. From multiple views to textured 3d meshes: a gpu-powered approach. In *ECCV Workshop on Computer Vision on GPUs (CVGPU)*, pages 5–11, 2010.

- Awards
  - "Young Researcher Award" 2012 – 2013, awarded and sponsored by the University of Crete, Heraklion, Crete.
  - 1st Prize in ChaLearn Gesture Challenge 2012, sponsored by Microsoft, Redmond, USA, in conjunction with ICPR 2012, Tsukuba, Japan.

- Highlights[1]
  - More than 390 citations, since 2010, h-index: 6[2].
  - More than 35.000 downloads[3] of 3D single hand tracking software[4].
  - More than 80.000 views[5] of the videos which supplement the thesis-related publications.
  - Contributions to several European projects:
    * GRASP (FP7-215821)
    * RoboHow.cog (FP7-288533)
    * WEARHAP (FP7-ICT-2011-9).

[1] The figures presented were captured at June 2014.

[2] Source is Google Scholar©.

[3] Sources are Google Analytics© and the Apache© server which hosts the web-page of the software.

[4] This is the part of our software implementation which has been made public. This software can be found at `http://cvrlcode.ics.forth.gr/handtracking`.

[5] Source is YouTube©, channels `Antonis Argyros` and `forth3DTracking`.

References

[1] Opengl mathematics. http://glm.g-truc.net/0.9.5/index.html.

[2] Cgal, Computational Geometry Algorithms Library. http://www.cgal.org.

[3] A. Andriyenko, K. Schindler, and S. Roth. Discrete-continuous optimization for multi-target tracking. In *CVPR*, 2012.

[4] Antonis A Argyros and Manolis IA Lourakis. Real-time tracking of multiple skin-colored objects with a possibly moving camera. In *Computer Vision-ECCV 2004*, pages 368–379. Springer, 2004.

[5] A. Armenti. *The physics of sports*. Copernicus Books, 1992.

[6] P.J. Aston and R. Shail. The dynamics of a bouncing superball with spin. *Dynamical Systems*, 22(3):291–322, 2007.

[7] ASUS. Xtion, 2014.

[8] Vassilis Athitsos and Stan Sclaroff. Estimating 3d hand pose from a cluttered image. *CVPR*, 2:432, 2003.

[9] L. Ballan, A. Taneja, J. Gall, L. Van Gool, and M. Pollefeys. Motion capture of hands in action using discriminative salient points. In *ECCV*, 2012.

[10] K. Bhat, S. Seitz, J. Popović, and P. Khosla. Computing the physical parameters of rigid-body motion from video. In *Computer Vision□ECCV 2002*, pages 551–565. Springer, 2002.

[11] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[12] M. Brand. Physics-Based Visual Understanding* 1. *Computer Vision and Image Understanding*, 65(2):192–205, 1997.

[13] M. Brand, P. Cooper, and L. Birnbaum. Seeing physics, or: Physics is for prediction. In *In Proceedings, Workshop on physics-based modeling in computer vision*. Citeseer, 1995.

[14] M Brubaker and Leonid Sigal. Physics-based human motion modeling for people tracking: A short tutorial. *Image (Rochester, N.Y.)*, pages 1–48, 2009.

[15] MA Brubaker and DJ Fleet. The Kneed Walker for human pose tracking. In *Computer Vision and Pattern Recognition, IEEE Conference on*, pages 1–8. IEEE, 2008.

[16] M.A. Brubaker, D.J. Fleet, and A. Hertzmann. Physics-based person tracking using the anthropomorphic walker. *International journal of computer vision*, 87(1):140–155, 2010.

[17] M.A. Brubaker, L. Sigal, and D.J. Fleet. Estimating contact dynamics. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2389–2396. IEEE, 2009.

[18] U.K. Chakraborty. *Advances in Differential Evolution*. Springer Publishing Company, Incorporated, 2008.

[19] H.T. Chen, M.C. Tien, Y.W. Chen, W.J. Tsai, and S.Y. Lee. Physics-based ball tracking and 3D trajectory reconstruction with applications to shooting location estimation in basketball video. *Journal of Visual Communication and Image Representation*, 20(3):204–216, 2009.

[20] Wongun Choi, Yu-Wei Chao, Caroline Pantofaru, and Silvio Savarese. Understanding indoor scenes using 3d geometric phrases. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 33–40. IEEE, 2013.

[21] Maurice Clerc and James Kennedy. The Particle Swarm - Explosion, Stability, and Convergence in a Multidimensional Complex Space. *Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[22] O. Cossairt, S. Nayar, and R. Ramamoorthi. Light field transfer: global illumination between real and synthetic objects. In *ACM SIGGRAPH 2008 papers*, pages 1–6. ACM, 2008.

[23] Erwin Coumans. Bullet game physics simulation, 2011.

[24] S. Das, A. Abraham, and A. Konar. Particle swarm optimization and differential evolution algorithms: technical analysis, applications and hybridization perspectives. *Advances of Computational Intelligence in Industrial Systems*, pages 1–38, 2008.

[25] M. de La Gorce, D.J. Fleet, and N. Paragios. Model-based 3d hand pose estimation from monocular video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[26] M. de la Gorce, N. Paragios, and D.J. Fleet. Model-based hand tracking with texture, shading and self-occlusions. In *CVPR*, pages 1–8, 2008.

[27] Martin de La Gorce, D.J. Fleet, and Nikos Paragios. Model-based 3d hand pose estimation from monocular video. *IEEE Trans. on PAMI*, pages 1–15, February 2011.

[28] Martin De La Gorce, Nikos Paragios, and David J. Fleet. Model-based Hand Tracking With Texture, Shading and Self-occlusions. In *CVPR*, pages 1–8. IEEE, Jun. 2008.

[29] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[30] Q. Delamarre and O. Faugeras. 3D Articulated Models and Multiview Tracking with Physical Forces* 1. *Computer Vision and Image Understanding*, 81(3):328–357, 2001.

[31] P. Douvatzis, I. Oikonomidis, N. Kyriazis, and A. Argyros. Dimensionality reduction for efficient single frame hand pose estimation. In *International Conference on Vision Systems, (ICVS)*, July 2013.

[32] P. Douvatzis, I. Oikonomidis, N. Kyriazis, and A. Argyros. Dimensionality reduction for efficient single frame hand pose estimation. In *International Conference on Vision Systems, (ICVS)*, July 2013.

[33] D.J. Duff, J. Wyatt, and R. Stolkin. Motion estimation using physical simulation. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1511–1517. IEEE, 2010.

[34] Ricardo Fabbri, Luciano Da F Costa, Julio C Torelli, and Odemir M Bruno. 2d euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys (CSUR)*, 40(1):2, 2008.

[35] Andrea Fossati, Helmut Grabner, and Luc Van Gool. Exploiting physical inconsistencies for 3d scene understanding. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 136–143. IEEE, 2012.

[36] A. Gupta, A. Efros, and M. Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. *Computer Vision–ECCV 2010*, pages 482–496, 2010.

[37] Peter Hachenberger. Exact minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra in convex pieces. In *Algorithms–ESA 2007*, pages 669–680. Springer, 2007.

[38] H. Hamer, K. Schindler, E. Koller-Meier, and L. Van Gool. Tracking a hand manipulating an object. In *ICCV*, 2009.

[39] Henning Hamer, Konrad Schindler, E. Koller-Meier, and Luc Van Gool. Tracking a Hand Manipulating an Object. In *ICCV*, 2009.

[40] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Pošík. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proceedings of the 12th annual conference comp on Genetic and evolutionary computation*, pages 1689–1696. ACM, 2010.

[41] I. Ihrke, K.N. Kutulakos, H. Lensch, M. Magnor, and W. Heidrich. Transparent and Specular Object Reconstruction. In *Computer Graphics Forum*. Wiley Online Library, 2010.

[42] J Jarez and Alain Suero. Newton game dynamics, 2007.

[43] J. Kennedy, R.C. Eberhart, and Yuhui. Shi. *Swarm intelligence*. Morgan Kaufmann Publ., 2001.

[44] James Kennedy and Russ Eberhart. Particle Swarm Optimization. In *International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, January 1995.

[45] James Kennedy, Russ Eberhart, and Shi Yuhui. *Swarm intelligence*. Morgan Kaufmann, 2001.

[46] Cem Keskin, Furkan Kıraç, Yunus Emre Kara, and Lale Akarun. Real time hand pose estimation using depth sensors. In *Consumer Depth Cameras for Computer Vision*, pages 119–137. Springer, 2013.

[47] K. Kim, V. Lepetit, and W. Woo. Keyframe-based modeling and tracking of multiple 3d objects. In *ISMAR*, 2010.

[48] H. Kjellstrom, D. Kragic, and M. Black. Tracking people interacting with objects. In *CVPR*, 2010.

[49] N. Kyriazis and A. Argyros. Physically plausible 3d scene tracking: The single actor hypothesis. In *CVPR*, 2013.

[50] N. Kyriazis and A. Argyros. Physically plausible 3d scene tracking: The single actor hypothesis. In *CVPR*, 2013.

[51] N. Kyriazis and A. Argyros. Scalable 3d tracking of multiple interacting objects. In *CVPR*, 2014.

[52] N. Kyriazis, I. Oikonomidis, and A. Argyros. Binding vision to physics based simulation: The case study of a bouncing ball. In *BMVC 2011*. BMVA, 2011.

[53] N. Kyriazis, I. Oikonomidis, and A. Argyros. Binding vision to physics based simulation: The case study of a bouncing ball. In *BMVC*. BMVA, 2011.

[54] N. Kyriazis, I. Oikonomidis, and A. Argyros. A gpu-powered computational framework for efficient 3d model-based vision. Technical Report TR420, ICS-FORTH, July 2011.

[55] R. Mann, A. Jepson, and J.M. Siskind. The computational perception of scene dynamics* 1. *Computer Vision and Image Understanding*, 65(2):113–128, 1997.

[56] D. Metaxas and D. Terzopoulos. Shape and nonrigid motion estimation through physics-based synthesis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(6):580–591, 1993.

[57] Microsoft. Kinect for windows, 2014.

[58] S.K. Nayar and S.G. Narasimhan. Vision in bad weather. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 820–827. IEEE, 1999.

[59] NVidia. Physx, 2007.

[60] CUDA Nvidia. Cuda c programming guide, July 2013.

[61] I. Oikonomidis, N. Kyriazis, and A. Argyros. Markerless and efficient 26-dof hand pose recovery. In *ACCV 2010*, pages 744–757. Springer, 2010.

[62] I. Oikonomidis, N. Kyriazis, and A. Argyros. Markerless and efficient 26-dof hand pose recovery. In *ACCV*, pages 744–757. Springer, 2010.

[63] I. Oikonomidis, N. Kyriazis, and A. Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC 2011*. BMVA, 2011.

[64] I. Oikonomidis, N. Kyriazis, and A. Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC*. BMVA, 2011.

[65] I. Oikonomidis, N. Kyriazis, and A. Argyros. Full dof tracking of a hand interacting with an object by modeling occlusions and physical constraints. In *ICCV 2011*. IEEE, 2011.

[66] I. Oikonomidis, N. Kyriazis, and A. Argyros. Full dof tracking of a hand interacting with an object by modeling occlusions and physical constraints. In *ICCV*. IEEE, 2011.

[67] I. Oikonomidis, N. Kyriazis, and A. Argyros. Tracking the articulated motion of two strongly interacting hands. In *CVPR*. IEEE, June 2012.

[68] I. Oikonomidis, N. Kyriazis, and A. Argyros. Tracking the articulated motion of two strongly interacting hands. In *CVPR*. IEEE, June 2012.

[69] I. Oikonomidis, N. Kyriazis, and A.A. Argyros. Tracking the articulated motion of two strongly interacting hands. In *CVPR*, 2012.

[70] I. Oikonomidis, N. Kyriazis, K. Tzevanidis, and A.A. Argyros. Tracking hand articulations: Relying on 3d visual hulls versus relying on multiple 2d cues. In *Proceedings of the IEEE International Symposium on Ubiquitous Virtual Reality 2013 (ISUVR 2013)*, July 2013. By invitation.

[71] I. Oikonomidis, N. Kyriazis, K. Tzevanidis, and A.A. Argyros. Tracking hand articulations: Relying on 3d visual hulls versus relying on multiple 2d cues. In *Proceedings of the IEEE International Symposium on Ubiquitous Virtual Reality 2013 (ISUVR 2013)*, July 2013. By invitation.

[72] Iason Oikonomidis. *Markerless visual full hand pose estimation and tracking*. PhD thesis, University of Crete, 2014.

[73] Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A. Argyros. Tracking the articulated motion of two strongly interacting hands. *CVPR*, pages 1862–1869, June 2012.

[74] V. Papadourakis and A. Argyros. Multiple objects tracking in the presence of long-term occlusions. *Computer Vision and Image Understanding*, 114(7):835–846, 2010.

[75] C. Papazov and D. Burschka. Stochastic global optimization for robust point set registration. *CVIU*, 2011.

[76] M. Patel, C. H. Ek, N. Kyriazis, A. Argyros, J. Valls Miro, and D. Kragic. Language for learning complex human-object interactions. In *ICRA*, 2013.

[77] Mitesh Patel, Carl Henrik Ek, Nikolaos Kyriazis, Antonis A. Argyros, Jaime Valls Miro, and Danica Kragic. Language for learning complex human-object interactions. In *ICRA*, pages 4997–5002. IEEE, 2013.

[78] Matt Pharr and Randima Fernando. *Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley Professional, 2005.

[79] Z. Popović and A. Witkin. Physically based motion transformation. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 11–20. ACM Press/Addison-Wesley Publishing Co., 1999.

[80] S. Ramalingam, J.K. Pillai, A. Jain, and Y. Taguchi. Manhattan junction catalogue for spatial reasoning of indoor scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[81] James M. Rehg and Takeo Kanade. Visual tracking of high dof articulated structures: An application to human hand tracking. In *ECCV*, pages 35–46. Springer-Verlag, 1994.

[82] J. Romero, H. Kjellström, C.H. Ek, and D. Kragic. Non-parametric hand pose estimation with object context. *Image and Vision Computing*, 2013.

[83] Javier Romero, H. Kjellström, and Danica Kragic. Hands in action: Real-time 3D reconstruction of hands in interaction with objects. In *ICRA*, 2010.

[84] Javier Romero, Hedvig Kjellström, and Danica Kragic. Monocular Real-time 3D Articulated Hand Pose Estimation. In *International Conference on Humanoid Robots*, pages 87–92. IEEE, Dec. 2009.

[85] Romer Rosales, Vassilis Athitsos, Leonid Sigal, and Stan Sclaroff. 3d hand pose reconstruction using specialized mappings. In *ICCV*, pages 378–385, 2001.

[86] M. Salzmann and R. Urtasun. Physically-based motion models for 3d tracking: A convex formulation. In *ICCV 2011*. IEEE, 2011.

[87] Shubhabrata Sengupta, Mark Harris, and Michael Garland. Efficient parallel scan algorithms for gpus. *NVIDIA, Santa Clara, CA, Tech. Rep. NVR-2008-003*, (1):1–17, 2008.

[88] R.J. Sethi and A.K. Roy-Chowdhury. Physics-based activity modelling in phase space. In *Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing*, pages 170–177. ACM, 2010.

[89] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[90] Russell Smith. Open dynamics engine, 2008.

[91] D. Song, N. Kyriazis, I. Oikonomidis, C. Papazov, A. Argyros, D. Burschka, and D. Kragic. Predicting human intention in visual observations of hand-object interactions. In *ICRA*, 2013.

[92] Dan Song, Nikolaos Kyriazis, Iason Oikonomidis, Chavdar Papazov, Antonis A. Argyros, Darius Burschka, and Danica Kragic. Predicting human intention in visual observations of hand-object interactions. In *ICRA*, pages 1608–1615. IEEE, 2013.

[93] B Stenger, P.R.S. Mendonça, and R Cipolla. Model-based hand tracking using an unscented kalman filter. In *BMVC*, volume 1, page 63–72. Citeseer, 2001.

[94] Björn Stenger, P.R.S. Mendonça, and Roberto Cipolla. Model-based 3D Tracking of an Articulated Hand. In *CVPR*, pages II–310–II–315. IEEE, 2001.

[95] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[96] E.B. Sudderth, M.I Mandel, W.T Freeman, and A.S Willsky. Visual hand tracking using nonparametric belief propagation. In *CVPRW*. IEEE, 2004.

[97] Erik B. Sudderth, Michael I. Mandel, William T. Freeman, and Alan S. Willsky. Visual Hand Tracking Using Nonparametric Belief Propagation. In *CVPR Workshop*, pages 189–189, 2004.

[98] Martin Szummer and Rosalind W Picard. Indoor-outdoor image classification. In *Content-Based Access of Image and Video Database, 1998. Proceedings., 1998 IEEE International Workshop on*, pages 42–51. IEEE, 1998.

[99] K. Tzevanidis, X. Zabulis, T. Sarmis, P. Koutlemanis, N. Kyriazis, and A. Argyros. From multiple views to textured 3d meshes: a gpu-powered approach. In *ECCV Workshop on Computer Vision on GPUs (CVGPU)*, pages 5–11, 2010.

[100] K Tzevanidis, X Zabulis, T Sarmis, P Koutlemanis, N Kyriazis, and Antonis A. Argyros. From multiple views to textured 3d meshes: a gpu-powered approach. In *ECCV Workshops*, pages 5–11, 2010.

[101] Konstantinos Tzevanidis and Antonis Argyros. Unsupervised learning of background modeling parameters in multicamera systems. *Computer vision and Image understanding*, 115(1):105–116, 2011.

[102] R. Urtasun, D.J. Fleet, and P. Fua. Monocular 3d tracking of the golf swing. In *CVPR*, 2005.

[103] B.N. Vo, B.T. Vo, N.T. Pham, and D. Suter. Joint detection and estimation of multiple objects from image observations. *IEEE Trans. on Signal Processing*, 58(10), 2010.

[104] M. Vondrak, L. Sigal, and OC Jenkins. Physical simulation for probabilistic motion tracking. In *Computer Vision and Pattern Recognition, IEEE Conference on*, pages 1–8. IEEE, 2008.

[105] J.M. Wang, D.J. Fleet, and A. Hertzmann. Optimizing walking controllers. In *ACM Transactions on Graphics (TOG)*, volume 28, page 168. ACM, 2009.

[106] Robert Y. Wang and Jovan Popović. Real-time Hand-tracking With a Color Glove. *ACM Transactions on Graphics*, 28(3):1, Jul. 2009.

[107] Xiaoyu Wang, Gang Hua, and Tony X Han. Discriminative tracking by metric learning. In *Computer Vision–ECCV 2010*, pages 200–214. Springer, 2010.

[108] Wikipedia. Deferred shading, 2013. [Online; accessed 3-October-2013].

[109] Wikipedia. Discriminative model, 2013. [Online; accessed 29-JAnuary-2014].

[110] Wikipedia. Generative model, 2013. [Online; accessed 29-JAnuary-2014].

[111] Wikipedia. Jacobian matrix and determinant, 2013. [Online; accessed 29-JAnuary-2014].

[112] Wikipedia. Quaternion, 2013. [Online; accessed 3-September-2013].

[113] Daniel M Wolpert and Zoubin Ghahramani. Computational principles of movement neuroscience. *nature neuroscience*, 3:1212–1217, 2000.

[114] Chenglei Wu, Kiran Varanasi, and Christian Theobalt. Full body performance capture under uncontrolled and varying illumination: A shading-based approach. In *Computer Vision–ECCV 2012*, pages 757–770. Springer, 2012.

[115] Ying Wu and Thomas S. Huang. View-independent recognition of hand postures. In *CVPR*, pages 88–94, 2000.

[116] Y. Ye and C.K. Liu. Synthesis of detailed hand manipulations using contact sampling. *ACM Transactions on Graphics (TOG)*, 31(4), 2012.

[117] T.E. Zickler, P.N. Belhumeur, and D.J. Kriegman. Helmholtz stereopsis: Exploiting reciprocity for surface reconstruction. In *International Journal of Computer Vision*, volume 49, pages 215–227. Springer, 2002.

[118] Zoran Zivkovic. Improved adaptive gaussian mixture model for background subtraction. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 28–31. IEEE, 2004.