DOCTORAATSPROEFSCHRIFT

2010 | School voor Informatietechnologie
Kennistechnologie, Informatica, Wiskunde, ICT

Video Manipulation using External Cues

Proefschrift voorgelegd tot het behalen van de graad van
Doctor in de Wetenschappen: Informatica, te verdedigen door:

Bert DE DECKER

Promotor: prof. dr. Philippe Bekaert

Maastricht University

universiteit hasselt

D/2010/2451/7

Abstract

When it comes to traditional 2D video editing, there are many video manipulation techniques to choose from, but all of them suffer from the limited amount of information that is present in the video itself. When more information about the scene is available, more powerful video manipulation methods become possible.

In this dissertation, we examine what extra information about a scene might be useful and how this information can be used to develop powerful yet easy-to-use video manipulation techniques. We present a number of novel video manipulation methods that improve on the way the scene information is captured and the way this information is used.

First we show how a 2D video can be manipulated if the scene is captured using multiple video cameras. We present an interactive setup that calculates collisions between virtual objects and a real scene. It is a purely image based approach, so no time is wasted computing an explicit 3D geometry for the real objects in the scene, all calculations are performed on the input images directly. We demonstrate our approach by building a setup where a human can interact with a rigid body simulation in real-time.

Secondly, we investigate which manipulation techniques become possible if we track a number of points in the scene. For this purpose, we created two novel motion capture systems. Both are low cost optical systems that use imperceptible electronic markers. The first is a camera based location tracking system. A marker is attached to each point that needs to be tracked. A bright IR LED on the marker emits a specific light pattern that is captured by the cameras and decoded by a computer to locate and identify each marker. The second system projects a number of light patterns into the scene. Electronic markers attached to points in the scene decode these patterns to obtain their position and orientation. Each marker also senses the color and intensity of the ambient light. We show this information can be used in many applications, such as augmented reality and motion capture.

Many existing video manipulation methods require images of a scene under multiple lighting conditions. We present a new light multiplexing method to capture a dynamic scene under multiple lighting conditions. A general method is presented that uses both time and color multiplexing. We show how our approach can improve existing methods to relight or obtain depth and normal information of a scene. This information can be used to render the scene from novel viewpoints or to change material properties.

Acknowledgments

Making a PhD thesis is not possible without the help and support of many people. This is why I would like to thank all the people that were invaluable to realize this work.

I would like to thank my advisor, Phillipe Bekaert, for his supervision and guidance, for believing in me and for making sure I had all the resources I needed to do my research. For giving me the opportunity to cooperate with researchers all over the world and for giving me the chance to visit many interesting conferences.

I also express my gratitude to the people of my PhD committee, Prof. dr. Chris Raymaekers, Prof. dr. Philippe Bekaert, Prof. dr. Frank Van Reeth and Prof. dr. ir. Philippe Dutré, for monitoring my progress and for their much appreciated comments. I would also like to thank the chair of the jury, Prof. dr. Marc Gyssens, and the other members of the jury Prof. dr. Jan Kautz, Prof. dr. ir. Christophe De Vleeschouwer and Prof. dr. Peter Schelkens for evaluating my work.

I would also like to thank the managing director of the EDM, Prof. dr. Eddy Flerackers, and the other professors at the EDM, Prof. dr. Frank Van Reeth, Prof. dr. Karin Coninx, Prof. dr. Wim Lamotte, Prof. dr. Kris Luyten and Prof. dr. Chris Raymaekers, Prof. dr. Fabian Di Fiore along with Peter Vandoren for creating the productive working environment the EDM is today.

The people from the administration department, Ingrid Konings and Roger Claes, also deserve my gratitude for helping me with all the formalities associated with doing research and preparing a PhD.

I acknowledge financial support from the Flemish institute for broadband communication (IBBT) and the transnationale Universiteit Limburg.

I would also like to thank all my coauthors from whom I learned a lot and with whom it was a pleasure to work with, in particular, Tom Mertens, Ramesh Raskar, Jan

Kautz, Martin Eiseman, Yuki Hashimoto, Dylan Moore, Jonathan Westhues, Hideaki Nii, Masahiko Inami and Philippe Bekaert.

I would like to express my gratitude towards Ramesh Raskar for giving me the opportunity to experience the fascinating time I had during my stay at the Mitsubishi Electronic research Laboratories (MERL).

I thank all my colleagues for the many insightful discussions and for making the EDM a fun and exciting place to work. I would also like to thank the current and former members of the graphics group, Codruta Ancuti, dr. Cosmin Ancuti, dr. Koen Beets, Tom Cuypers, Maarten Dumont, Prof. dr. Fabian Di Fiore, Yannick Francken, dr. Jan Fransens, Karel Frederix, Mark Gerrits, Patrik Goorts, Tom Haber, Chris Hermans, dr. Erik Hubo, Steven Maesen, dr. Tom Mertens, Johan Nulens, Sammy Roggemans, Johannes Taelman, dr. William Van Haevre, dr. Tom Van Laerhoven and Cedric Vanaken. I especially thank Mark Gerrits for proofreading this dissertation.

I would also like to thank my friends and family for their support and encouragement. Finally, I would like to thank my parents, for supporting me in everything I want to do and for encouraging me to pursuit my dreams.

Diepenbeek, 2010.

Contents

Chapter 1

Introduction

Contents

1.1 Problem Statement

Traditional video editing packages offer a wide range of techniques to manipulate footage. Most of these methods only use the input video and user interaction to steer the manipulation. However, if additional information about the scene is available, more advanced manipulation techniques become possible. For example, when the 3D geometry of the scene is known, one can visualize the video from new viewpoints. Or if the position, orientation and incident illumination is known for a number of points in the scene, the scene can be augmented with virtual objects which can be lit by the same light that is present in the real scene. In this thesis, we investigate several methods to capture and use extra information about a dynamic scene in order to manipulate a video.

Until a short while ago, manipulating videos on a computer was not feasible, because the computers were not fast enough and did not have enough memory. In the early days, video editing was done by physically cutting the film into pieces and attaching

the pieces back together in a different order. When magnetic tapes came into play and electronics became more powerful, dedicated hardware became available to edit video. But this was only for VHS, high quality movies were still on film and had to be edited by hand. Greenkey also came into play, a specialized hardware box that was able to replace the background of a scene where a person is standing in front of a green cloth. As they steadily grew in power, computers were used more and more, first only for editing movies, later for all kinds of video manipulations and special effects. These days, more and more film makers are making the transition to a completely digital pipeline. From the cameras, to the processing, to the projectors in the cinemas that finally show the movie to a wide audience, everything is becoming digital.

Not only has this trend made more and more special effects feasible, they have also become significantly cheaper. Because digital copying is lossless, the quality of the final product has increased. Advanced manipulation techniques which only the largest studios could afford a few years ago, have now become available for everyone with a personal computer.

Video manipulation is used in a very wide range of applications:

- **Motion Pictures** Video manipulation plays a very important role in many motion pictures as special effects are used to convey the story.

- **Commercials** Commercials need to catch the attention of the audience and deliver a message in a very small amount of time. Stunning special effects are a good way to achieve this goal.

- **Music Videos** Music video clips are used as a marketing device for the music industry. The budget of many of these clips is very large, and no effort is too great to make them look as impressive as possible. Video manipulation and special effects clearly have a place here.

- **Augmented Reality** Augmented reality applications mix videos of the real world with virtual objects.

  - **Maintenance** During machine maintenance, virtual labels with information and instructions can be shown in crucial places to help the technician.

  - **Sports** During television broadcasts of sports games, information can be shown on the field. In soccer for instance, the offside position can be displayed.

  - **Architecture** In a street video, the design of a future building can be superimposed over the old house it is going to replace. This gives a feel of how the new building will fit into its environment.

- **Home Videos** Because video manipulation is becoming affordable for a wide audience, more and more people are using their pc to edit their home videos.

- **Teleconferencing** Until a short while ago, in teleconferencing, a video of the people in the meeting was just sent without any video manipulation. Now more and more techniques become available to enhance the teleconferencing experience. 3D geometry is used to correct eye-gaze and relighting techniques are used to illuminate the participants.

- **Computer Games** Computer games have also started focusing on video. A good example is the playstation eyetoy where a camera is used as an input device by the player.

In this thesis, we focus on video manipulation methods, so all the scene data needs to be captured in real time. But processing the data in real-time is only needed for some applications. We will investigate applications that need to generate results in real-time as well as applications that can process their data off-line.

The following is a short non-exhaustive overview of manipulation techniques that become possible when extra information about a scene is captured.

- **Relighting** In relighting applications, it is possible to change how a scene is lit, even after it is captured. The decision on how to light a scene can be postponed from the filming phase to post-processing. This is especially useful when combining real and cg-elements in the same shot because it becomes easier to match the lighting of the real and virtual objects.

- **Changing Material Properties** It is possible to change the material properties of filmed surfaces. For example making a surface more specular or changing the albedo of materials. To do this, extra information such as incident illumination or surface geometry is needed.

- **Augmented Reality** To add virtual objects to a scene, some 3D information is needed to handle occlusions. An extension to this is to calculate interactions between the virtual objects and the real filmed scene. If a number of real objects in the scene is tracked, it is possible to attach virtual objects to them. This way virtual labels can be attached to real objects or real objects can be replaced by virtual ones.

- **Novel Viewpoints** If some 3D geometry of a scene is known, the scene can be visualized from novel viewpoints. There are many ways to capture this 3D information, such as depth cameras, multi-camera systems and structured light approaches.
- **Fog and Depth of Field** If depth information of the scene is known, we can add fog to a scene or change the depth of field and point of focus of the lens in post-production.
- **Matting** The foreground of a video can be segmented out and composited on a new background. This can be used to place a person in a virtual environment.

1.2 Contributions

We present a number of contributions to the type of extra information that can be captured, the way this extra information is captured and how this information is used to manipulate a video.

- We present a real-time technique to calculate collisions between virtual objects and objects in the real world without calculating an explicit proxy for the geometry of the real objects.
- A camera based location tracking system is presented. An electronic marker is attached to a number of points in the scene and a LED on each marker emits a strobe pattern that is used by a pc attached to a number of cameras to locate and identify each marker.
- We describe a method to capture position, orientation and incident illumination for a number of scene points. By projecting well chosen light patterns onto the scene, electronic markers with optical sensors can calculate their own position and orientation. The markers are also able to sense the incident ambient illumination.
- A generic method to multiplex light patterns is presented. Both time and color multiplexing is used and a way to handle dynamic scenes is described.

1.3 Overview

In this section we give a short overview of the matter that will be covered in this thesis.

- Chapter 2 presents a method to calculate collisions between real and virtual objects. These collisions are calculated using the input images directly, so we do not need to obtain an explicit proxy of the geometry of the real objects and therefore save computation time. As a proof of concept, we implemented a system where a person, filmed in a multi-camera setup, is able to interact with a rigid body simulation in real time.

- In Chapter 3 we discuss two motion capture systems. The first is a camera based tracking system that locates and identifies markers in the scene. The second system captures the position, orientation and incident illumination for a number of points in the scene. The scene is lit by multiple imperceptible infrared light patterns, from which markers attached to points in the scene can calculate the desired information. We show experiments to determine the accuracy of the system and show applications such as motion capturing and augmented reality.

- Chapter 4 presents a method to capture a dynamic scene for a number of lighting conditions using time and color multiplexed illumination. This is a generic method that can be used to improve many existing video manipulation techniques. As a proof of concept we apply our method to relighting, depth from structured light and photometric stereo.

Chapter 2

Image Based Collision Detection

Contents

Many applications require a user to interact with virtual 3D objects. Most 3D interaction techniques require some training, eg. using a mouse to interact with a 3D environment is not always very easy. Some applications require a more intuitive interface. In the real world, we interact with objects by touching and grasping them with our hands and body. The most intuitive interaction method is therefore a method that allows the users to interact with virtual objects in the same way.

In this chapter we show how this can be achieved. We present an algorithm to cal-

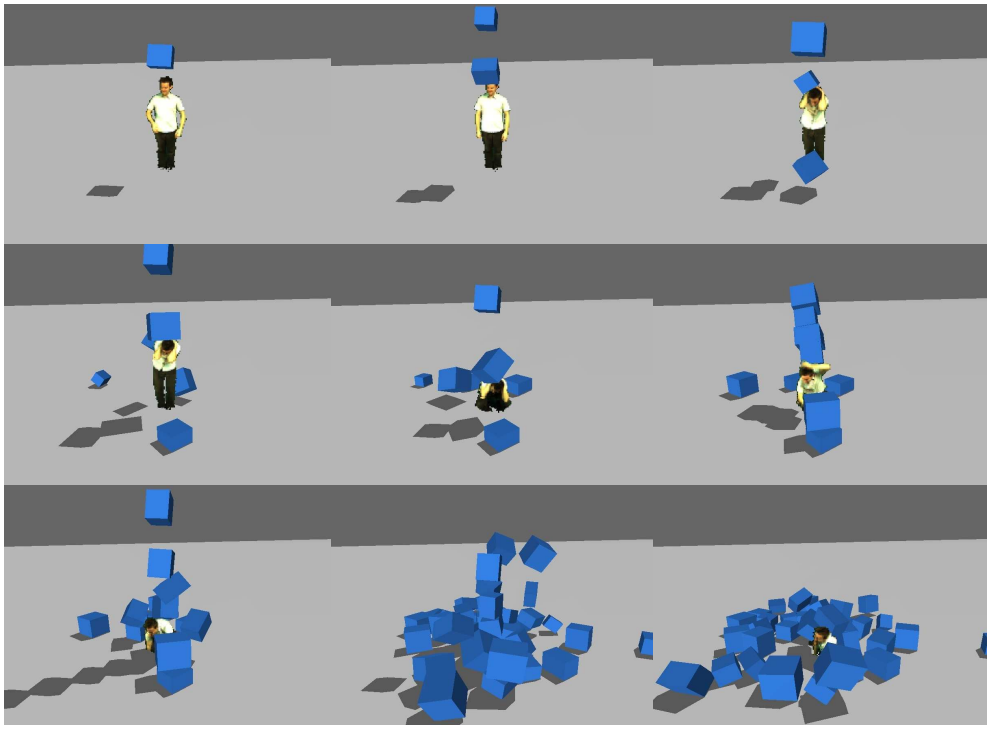


Figure 2.1: A person, captured by multiple calibrated digital video cameras, interacts with a rigid body simulation at interactive speeds.

culate collisions between real and virtual objects in real time using a multi-camera setup. We perform collision computations directly on the image data, as opposed to reconstructing the full geometry of the real objects. This reduces implementation complexity, and moreover, yields interactive performance. We demonstrate the effectiveness of our technique by incorporating it in a rigid body simulation. A person can interact with virtual objects and observe his or her actions, all in real-time. We render both the person and the virtual objects in a single view.

This is an example of a video manipulation technique that would have been very hard to achieve when only a single 2D video of the scene is captured. Occlusions have to be handled and intersections between the real and the virtual objects have to be dealt with. If no extra information about the scene is known, lots of user interaction would be required to obtain the desired result.

A first example is shown in figure 2.1. It shows a person interacting with a large number of virtual rigid bodies at interactive speeds.

2.1 Introduction

We present a collision detection system that calculates collisions between real and virtual objects in real time. A straightforward approach to tackle this problem would be to apply your favorite 3D reconstruction technique to obtain a triangle mesh of the scene, to which standard collision detection schemes can be applied. However, it would be better to avoid this intermediate reconstruction step, for the sake of simplicity and computational efficiency. Our technique is therefore designed to work directly on the information in the video frames.

In the classic collision detection problem, contact is determined from a full 3D description of the participating shapes (i.e. a triangle mesh). In our case, each camera only provides a discrete, 2D description of the real-world shapes. First, the information of the set of 2D frames acquired from each camera has to be aggregated to derive a collision test in 3 dimensions. We implement this based on the concept of a visual hull, i.e. the volume extruded from the silhouette of an object or subject. This information is readily available as a result of fast and simple foreground segmentation. The visual hull allows us to omit a costly 3D reconstruction step. Since camera registration is inherently a discrete process (granularity = one pixel), we have to be careful to avoid aliasing artifacts when determining the exact collision location. Finally, the analysis has to process data at a high bandwidth, since frames from multiple cameras have to be processed at each instant, while maintaining interactive performance. We therefore introduce a simple acceleration scheme to efficiently test the visual hull data. These issues will be explored in the remainder of this chapter.

We set out to make a collision detection method that conformed with the following goals

- faster than real time
- able to handle many types of virtual objects (meshes, point clouds, isosurfaces, ...)
- easy to incorporate into existing physically based simulation systems.

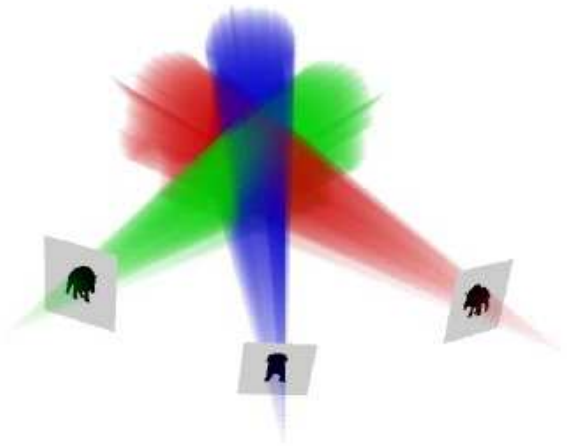


Figure 2.2: The concept of the visual hull [Matusik 00]. The visual hull as the intersection between the silhouette cones.

2.2 Related Work

2.2.1 Image Based Modeling and Rendering

Because the geometry of the real objects is unknown, it will have to be approximated when calculating collisions with virtual objects. For this purpose, we chose to use the visual hull, because very little computation time is needed to verify whether or not a point is inside a visual hull. Thus the collision detection is not performed between the virtual objects and the real objects, but is instead done between the virtual objects and the visual hull of the real objects.

The Visual Hull [Laurentini 94] describes the maximal volume compatible with a set of silhouettes. This shape can be used to obtain an approximate geometry of a real object, when a number of silhouettes of the object is available (See figure 2.2). It is a bounding volume of this object that is compatible with its silhouettes, so the real object is always smaller or of equal size than its visual hull.

There are a number of approaches to calculate the visual hull. Hasenfratz et al. [Hasenfratz 03] present a technique to calculate a voxel representation of the visual hull in real time. Another approach to calculate the visual hull is presented by Matusik et al. [Matusik 01]. They calculate a mesh of the visual hull from a number of silhouettes in real time.

2.2.2 Collision Detection between Virtual Objects

Many techniques have been developed to detect collisions between virtual objects such as rigid bodies, [Guendelman 03, Baraff 92, Pauly 04, Heidelberger 04] , cloth [Bridson 02, Govindaraju 05] , deformable objects [Debunne 01, Teschner 05, Dewaele 04] , articulated objects [Redon 04] and fluids [Losasso 06] .

2.2.3 Interaction Techniques

The technique presented in this chapter can be seen as a way to interact with a virtual environment [Hand 97, Bowman 97, Grossman 04]. The system by Kitamura et al. [Kitamura 03] calculates interactions between real and virtual objects by defining a constrained set of physical laws. Xiyong et al. [Wang 05] present a system where users can manipulate scanned, articulated virtual representations of real objects.

2.2.4 Force Feedback

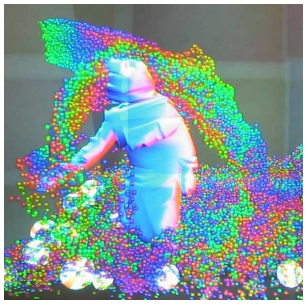
In our work, we perform a one way interaction: the real world can interact with the virtual world, but not vice versa. There are also some techniques that provide interaction in 2 directions. An example of such a technique is presented by Lindeman et al. [Lindeman 04]. They describe a system that gives haptic feedback to a user who walks around in a virtual environment.

2.2.5 Collision Detection between Real and Virtual Objects

Allard et al. [Allard 06b, Allard 06a] present a physically-based animation system in which users can interact with objects in a scene using a visual hull (See figure 2.3). First a system is presented where a polygonal visual hull is calculated in real time [Allard 06b]. The visual hull was calculated using 6 camera's. To obtain real-time frame rates, a distributed approach was used. In another paper by the same authors, a software framework for developing distributed physically based simulations in a VR-environment is presented[Allard 06a]. In this system they show how multiple physically based simulation systems can work together using a modular design, and how a polygonal visual hull of a person can interact with it in real-time on a cluster of machines.

Hasenfratz et al. [Hasenfratz 04] describe a system where a voxel visual hull of a real scene is calculated. To visualize this voxel visual hull, marching cubes is used to generate a smooth surface. 3D virtual buttons can be pressed by checking how many

(a)



(b)

Figure 2.3: Allard et al. [Allard 06b, Allard 06a] present a system that calculates collisions between real objects and the visual hull of a person.

voxels are penetrating the 3D shape of the button and virtual objects can bounce off the visual hull using standard collision detection because a mesh representation is present. Breen et al. [Breen 96] model part of a real static scene manually or using computer vision techniques. Standard collision detection can now be used to calculate collisions between real and virtual objects. Stam [Stam 00] presents a method where a depth map is calculated by filming a person with a special camera. Using this depth map a person can interact with a fluid simulation.

In contrast to all the methods we discussed so far, we bypass the mesh generation and work directly on the images which results in higher performance. Most related to our work is the system by Lok et al. [Lok 03] (See figure 2.4). Instead of using an intermediate mesh representation of the real world, they rasterize every virtual triangle to all the input cameras to determine collisions using graphics hardware. They assume there is at most one collision between one virtual object and the visual hull of the real object at a time. This is true in some cases (for example a virtual ball bouncing off a wall). However, when a virtual box lies on a real table, for instance, this assumption breaks down. The main advantage of our algorithm , is that we take into account multiple simultaneous collisions.

2.3 Overview

Our collision detection algorithm is based on the visual hull. We define a collision when a virtual object intersects the visual hull of the real object. This is only an approximation, since the shape of the visual hull is not exactly the same as the shape of the real object. But we found that it is a good compromise between speed and

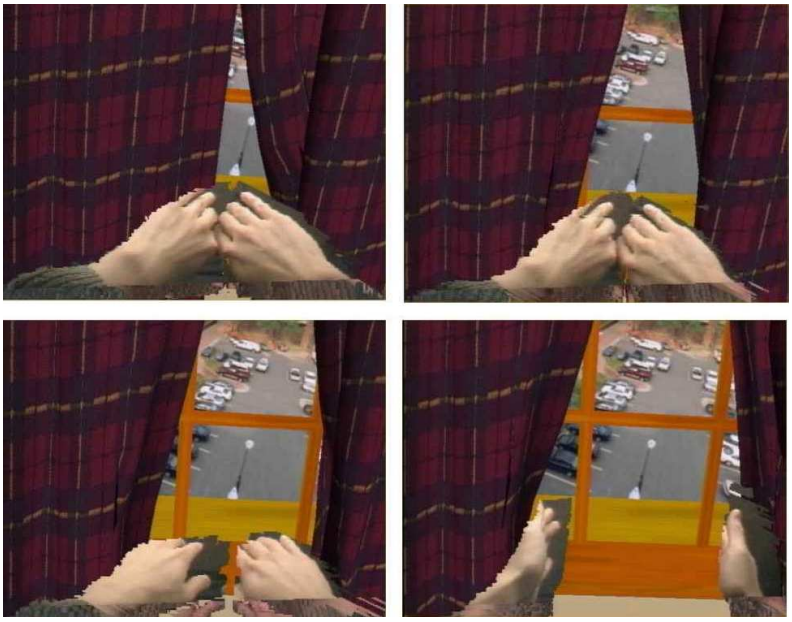


Figure 2.4: Lok et al. [Lok 03] present a system that calculates interactions between real and virtual objects without calculating an explicit 3D geometry of the real objects. A sequence of images is shown where a real person opens the virtual curtains of a virtual window.

accuracy. We obtain convincing simulations and need only little computation time.

A naive way to calculate collisions between virtual objects and the visual hull of the real object, would be to explicitly calculate the visual hull with for example the polygonal visual hull algorithm [Matusik 01]. We will show that we do not need to do this to perform our collision detection. We do not even have to segment our input images. In fact, we will show that when only a few virtual objects are present in the scene, running our collision detection is faster than segmenting the input images. Since segmenting the input images is only the first step in calculating an explicit visual hull, our algorithm clearly is a lot faster than the naive way to calculate collisions.

Determining whether an arbitrary virtual object intersects the visual hull of the real object is not a straightforward task. Especially since one of our goals was to make a faster than real time collision detection algorithm. On the other hand, determining whether a single point is inside the visual hull is a very cheap test and can be performed thousands of times a second. This is why we convert our problem of finding collisions between arbitrary virtual objects and the visual hull to a problem where we only need to calculate point-visual hull intersection tests. We do this by point sampling our virtual objects. Now, the virtual object intersects the real object, if one of the sample points intersects the real object. One advantage of this is that our method works with any virtual object that can be point sampled, such as meshes, point clouds, isosurfaces and NURBS. A disadvantage is that if we sample the virtual objects too sparsely, thin features of the visual hull can intersect the virtual object without touching any of the sample points. But since the point collision tests are very cheap, we can sample the virtual objects very densely, so this is not an issue in practice.

We also want to make sure our collision detection system can be easily incorporated into existing physically based simulation systems such as a rigid body system. Our collision detection algorithm calculates the collisions between the virtual and the real objects and the physically based simulation system calculates the required collision responses to make sure the virtual objects bounce off the real objects in a realistic way.

Most rigid body systems for virtual objects do not allow objects to intersect. When an intersection is found, the system searches the exact moment the collision occurred, and prevents the intersection. In order to do this, they assume they can calculate the state of the system at any time. In our case however, this is not possible. We only know the shape of the real objects at discrete time steps — the moments when the images are captured. We could approximate the shape of the real object in between these discrete time steps using for example optical flow techniques, but those are too time consuming for our purposes. So we have no choice but to let these intersections

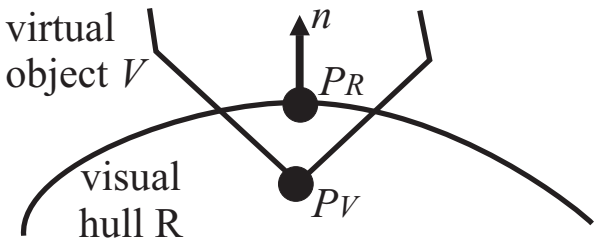


Figure 2.5: For each collision, we calculate the collision information. This consists of the surface normal n and the position of the collision on the real and on the virtual object.

happen. Luckily, modern physically based simulation systems are robust against this. They can recover from the intersections if the penetration depth is not too large.

Our collision detection system calculates the position of the collision and the surface normal at the collision and we let the physically based simulation system handle the rest. Since we let the virtual and the real objects intersect, we have two positions for each collision, one on the virtual object and one on the visual hull (see figure 2.5). From now on we will call these two positions and the surface normal the collision information.

When calculating the collision information, we need to be careful since at a small level, each pixel generates a sharp corner in the visual hull. When we look at the visual hull at a small scale, it is quite rough. When using the visual hull directly, we do not obtain smooth animations because of this. To alleviate this problem, when a collision is detected, we locally approximate the visual hull by a plane and calculate the collision between this plane and the virtual object instead. This way the visual hull is locally smoothed out and we obtain more realistic animations.

2.4 Detecting Collisions

In this section we will show how we find collisions between the real and the virtual world. First we explain how we find collisions and later in this section we will show how we obtain the collision information once a collision is found.

2.4.1 Identifying Collisions

As explained in the overview section, finding collisions between real objects and arbitrary virtual objects is not easy, so we first point sample the virtual objects. A virtual object collides with the real world if one or more of its sampled points are inside the visual hull.

There is no special requirement on how the point sampling has to be done. The only requirement is that it has to be dense enough so that no collisions are missed and it should be sparse enough so that the system remains interactive.

Once the virtual objects are point sampled, we iterate over all the points and check if they intersect the real object. To check if a point intersects the visual hull of the real object, we project it to all the input cameras. If it projects inside the silhouette for all the cameras, it intersects the visual hull. To check if a projected point is inside the silhouette for a given camera, we use simple background subtraction for this single pixel. To make this more robust, we cover the walls with green cloth.

2.4.2 Point - Real World Collisions

Once a collision is identified, we need to calculate its collision information : the collision normal (N) and the position of the collision on the virtual object (P_V) and on the visual hull of the real object (P_R) (see figure 2.5).

Find the relevant camera Note that if a point lies on the surface of the visual hull it projects to the border of the silhouette for one camera and inside the silhouette for all the other cameras (e.g. point $P_{surface}$ in figure 2.6 is a point on the surface of the visual hull and it projects to the border of the silhouette for camera C_1 and inside the silhouette for camera C_2). If a point lies on an edge or on a corner of the surface of the visual hull, it projects to the border of the silhouette for multiple cameras. (e.g. point P_{corner} in figure 2.6 is a point on a corner of the surface of the visual hull and it projects to the border of the silhouette for both camera C_1 and camera C_2). In the case that the point does not lie on an edge or on a corner of the visual hull, there is only one camera that provides information about the surface of the visual hull near that point. When we want to calculate the collision information for a virtual point that is intersecting the visual hull, we only need one camera — the camera for which the virtual point projects the closest to the border of the silhouette. By doing this, we assume the virtual point is not near a corner or an edge of the visual hull. If it is near an edge or a corner of the visual hull, we pick the first camera for which it

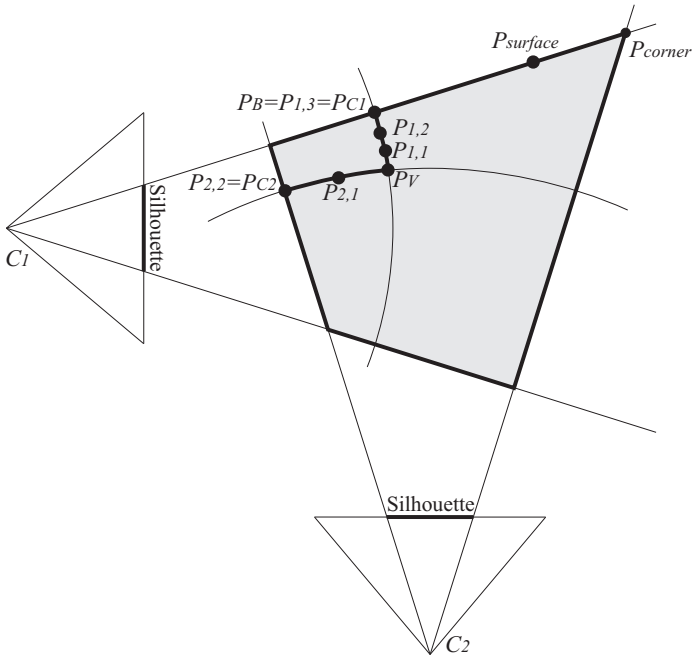


Figure 2.6: The grey quadrangle represents the visual hull defined by the silhouettes of two cameras C_1 and C_2. For a given virtual point P_V, we look for the nearest point P_B on the surface of the visual hull.

projects to a point close to the border of the silhouette to keep the computations fast. We found that we still generate convincing simulations, even though we make this approximation.

The grey quadrangle in figure 2.6, represents the visual hull defined by the silhouettes for cameras C_1 and C_2. Point P_V is a point that is intersecting the visual hull for which we want to calculate the collision information. We would like to find the point P_B on the surface of the visual hull that is the closest to P_V.To find P_B, we first calculate, for each camera C_i, the nearest point P_{Ci} that is the closest to P_V and that lies on the surface of the cone defined by the camera and its corresponding silhouette. Once P_{Ci} is calculated for each camera, P_B can be found by looking for the point P_{Ci} that is the closest to P_V (see figure 2.6). The corresponding camera C_i is the one relevant camera that will be used to calculate the collision information.

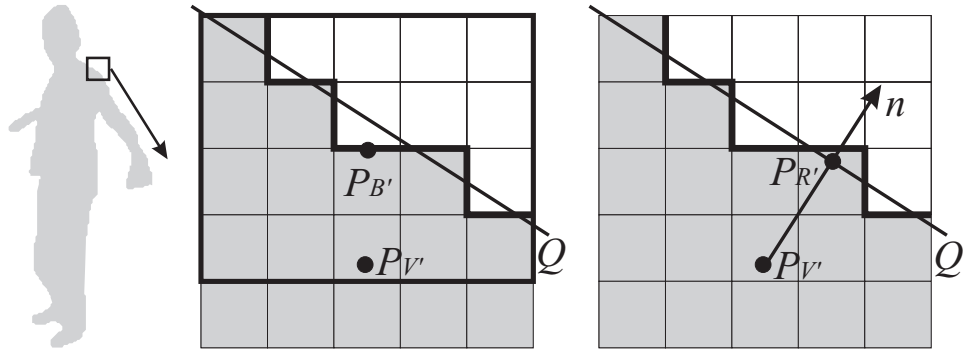
To calculate the point P_{Ci} for camera i, we look at points in a growing region around P_V until we find a point that does not project inside the silhouette for camera C_i. This point will be P_{Ci}. We define d_i as the distance between camera C_i and P_V. In the first

iteration, we construct all points that are at a distance d_i away from C_i and project one pixel away from the projection of P_V in camera C_i. In figure 2.6, these points are represented by $P_{1,1}$ and $P_{2,1}$ for the first and the second camera respectively. If one of these points project outside the silhouette,we found the border of the cone and we stop searching, otherwise we keep iterating. For the second iteration, we construct all points that are at a distance d_i away from C_i and project two pixels away from the projection of P_V in camera C_i ($P_{1,2}$ and $P_{2,2}$ in the figure). Then we check if any of these points project outside the silhouette for camera C_i. We keep iterating until we find a point that projects outside the silhouette. So we look at a growing window around P_V until we find a point that projects outside the silhouette. This way we can find the point P_{Ci} for each camera C_i. In figure 2.6, $P_{1,1}$, $P_{1,2}$ and $P_{1,3}$ depict the points that were generated when looking for P_{C1} during the first, second and third iteration respectively. $P_{2,1}$ and $P_{2,2}$ depict the points that were generated when looking for P_{C2} during the first and second iteration respectively.

By first calculating P_{Ci} for each camera C_i and then finding which of these points is the closest to P_V, we lose a lot of time. Typically there is one camera for which P_{Ci} lies very close to the P_V while in the other cameras, there is a large distance between P_V and P_{Ci}. So instead of first calculating P_{Ci} for each camera separately, we can calculate P_{Ci} for all cameras simultaneously. For each iteration of this improved algorithm, we select a couple (i, j) that has not been selected before and for which $P_{i,j}$ lies the closest to P_V. We check whether this point projects inside the silhouette for camera i. If it does project inside the silhouette, we keep iterating. Otherwise, we found the surface of the visual hull. Since for each iteration, the point $P_{i,j}$ is further away from P_V than the point selected during the previous iteration, we know that the first point we find that projects outside the silhouette, will be P_B, the point we were looking for.

Plane fitting Once P_B and the relevant camera is found, we can calculate the collision information using this camera. We now have the situation shown in image (b) of figure 2.7.

We have the projection P'_V of our virtual point P_V and the projection P'_B of the closest point P_B on the surface of the visual hull. As can be seen in this image, the surface of the visual hull is quite rough, each pixel introduces a sharp corner. Using this rough surface directly does not generate a smooth simulation as will be shown in the results section. To alleviate this problem, we locally approximate the visual hull by a plane Q, as shown in image (b). To obtain this plane, we first fit a line Q' through the border of the silhouette. To fit this line, we only consider a window of 4x5 pixels around P'_B, shown in image (b) as the black rectangle. The position of the line is the average of



(a) Silhouette for the relevant camera

(b) A 5x5 zoomed in window of the silhouette. The surface of the visual hull is smoothed by fitting a plane though it

(c) The position and the normal of the collision is calculated using this plane.

Figure 2.7: Once the relevant camera is identified, the collision information is calculated by fitting a plane through the visual hull.

all the points on the silhouette border and the normal of the plane is the average of all the normals of the points on the silhouette border. The plane Q is now defined by this line Q' and the centre of the camera. The normal of the collision surface is the normal of this plane Q. The position of the collision on the surface of the virtual object is P_V and the position on the real object is the point P_R on the plane Q the closest to P_V.

2.4.3 Bounding Volumes

The point - real world collision test is very fast, but using bounding volumes, we can prune the number of tests that need to be performed and increase the total performance. Finding bounding volumes for the virtual objects can easily be done with standard computer graphics algorithms. Defining a bounding volume for the real objects on the other hand is more difficult, especially since we do not know anything about them — we did not even segment the input images.

Constructing bounding volumes

To calculate a bounding volume for the real objects, we need to make some assumptions and/or we need to do some extra calculations to find out a bit more about their shape.

Here we present two possible ways to calculate a bounding volume.

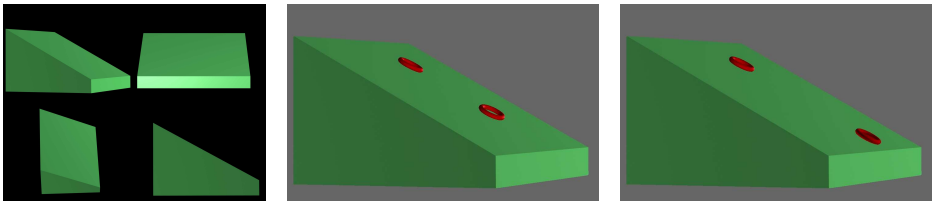
Assume all cameras see the entire object If we assume the real object is always inside the view frustum of each camera, we can use the intersection of all these frustums as a bounding volume of the real object. This bounding volume might be a lot bigger than the real object, but it only has to be calculated once and can provide a big speedup if the real object only represents a small portion of the virtual world. This bounding volume has only a little overhead, but it might be a lot bigger than the real object.

Segmenting the input images To construct a smaller bounding volume more information about the real scene has to be calculated. One way to do this would be to segment the input images and calculate in image space, a 2D bounding rectangle of the silhouette for each input image. For each camera, such a 2D bounding rectangle defines a frustum than encloses the real object. The intersection of these frustums defines a bounding volume for the real object. The disadvantage of this method is that it requires quite a lot of overhead — all the input images need to be segmented. But if there are a lot of virtual points in the scene, this extra work might pay off.

Note that creating a bounding volume is only valuable if there are a lot of virtual points that need to be tested. If we only have a few virtual objects in the scene, the overhead of calculating the bounding volumes does not outweigh the speedup we get from using them. In the results section we show the impact of the bounding volumes on the performance of our algorithm in a number of scenarios.

Using bounding volumes

Until now, we explained how the bounding volumes can be calculated. In this paragraph we describe how the bounding volumes are used to speed up the collision tests. For each virtual object we construct a kdtree containing its sample points. For each node of this kdtree we calculate the axis aligned bounding box of its children. This can all be done during a preprocessing step. This kdtree and the bounding boxes are in the local coordinate system of the virtual object and rotate when the virtual object rotates. To calculate the intersection between this virtual object and the real object, we traverse the kdtree and prune away the nodes for which the bounding box does not overlap with the bounding volume of the real object. To reduce the overhead of this pruning, we do not use the bounding volume of the real objects from the previous section directly. Instead we calculate an axis aligned bounding box of this volume in the local coordinate system of the virtual object. This way we only have to perform very fast intersection tests between axis aligned bounding boxes when pruning.



(a) input (b) without fitting a plane (c) with fitting a plane

Figure 2.8: Experiment that shows that a plane has to be fitted to obtain a smooth animation.

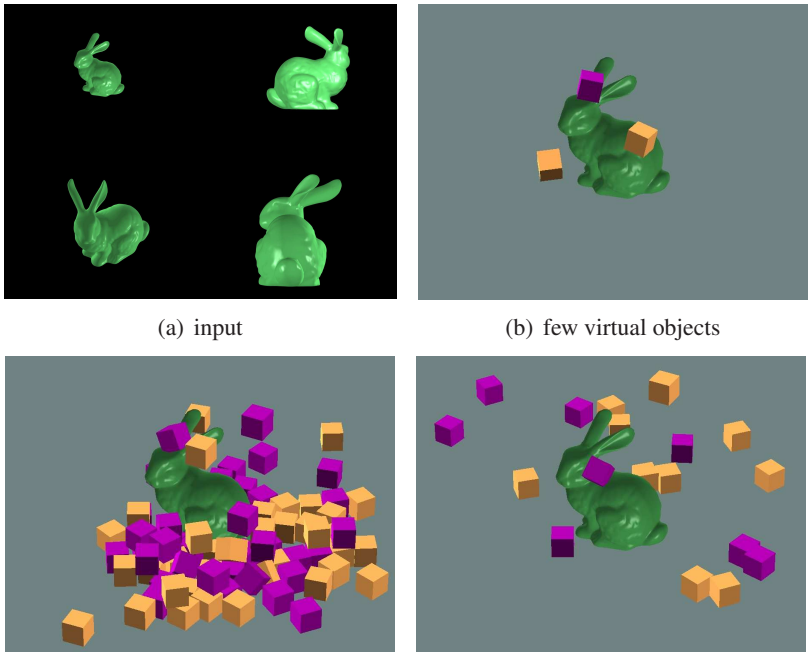
scene	few	many close	many far
no bounding volume	0.47ms 375	76.0ms 229918	9.66ms 54396
bounding frustum of cameras	0.51ms 375	76.7ms 180427	6.24ms 10793
bounding frustum of silhouettes	36.1ms 275	52.1ms 23230	42.4ms 2015

Table 2.1: The effect of the use of bounding volumes on the performance of our system. The three columns correspond to the three experiments shown in figure 2.9. The total time it takes to calculate the collisions and perform the rigid body simulation is shown, along with the number of points for which a visual hull intersection test was performed.

2.5 Results

As explained in section 2.3, we locally approximate the visual hull by a plane to calculate good collision information. Figure 2.8 shows an experiment where we demonstrate that fitting this plane is really necessary. In this experiment we used an artificial static scene that consists of a single slope. The four input images are shown in image (a). We drop a number of virtual objects (toruses) onto this plane and they should slide smoothly down the slope. Image (b) shows a frame of the resulting animation when we do not fit a plane but use the visual hull directly. In this case the animation is not smooth and the virtual object bounces up and down. In these still frames, this effect is not very obvious, but in the accompanying video it is very clear. Image (c) shows our plane based approach. Now the collision information is more accurate which results in a better simulation.

We tested the effect of the use of bounding volumes on the performance of the system. An artificial static scene of a rabbit is used for this experiment. The input images are shown in figure (a). We used the three different situations shown in figure 2.9 to test the performance. The first scene contains only a few virtual objects, the second scene



(a) input

(b) few virtual objects

(c) many virtual objects close to the real object

(d) many virtual objects at a larger distance from the real object

Figure 2.9: Three experiments to test the effect of the use of bounding volumes on the performance of our system.

contains many virtual objects that are close to the visual hull, the third scene contains many virtual objects that are spread out over a large distance. Table 2.1 shows the effect of using bounding volumes on the performance in the situations shown in figure 2.9. It is clear that whether and which bounding volumes should be used depends on the situation. Using the silhouettes to construct a bounding volume generates lots of overhead because segmenting the images is computationally intensive, but when a lot of virtual objects are near the visual hull, it pays off. When only a few virtual objects are present, using no bounding volume is the best choice, because of the overhead when using bounding volumes. When many virtual objects are present in the scene, but the virtual objects are far away from the visual hull, using a bounding frustum of the cameras is the best choice.

As can be seen in table 2.1, when only a few virtual objects are present in the scene, calculating the collisions is very fast, a lot faster than segmenting the input images. As said in the introduction, a naive way to calculate intersections between real and virtual objects is to first calculate a polygonal visual hull of the real objects and then use traditional collision detection algorithms to obtain the intersection between the visual hull and the virtual objects. Since segmenting the input images is only the first step in the naive method, our collision detection algorithm is a lot faster than the naive one.

Figure 2.10 and figure 2.11 show experiments where a person moves a real box and the virtual boxes on top of it move accordingly. In this experiment we show we can handle virtual objects that are lying in rest on top of real ones.

The experiment in figure 2.12 and figure 2.13 shows we can handle non-convex objects. A number of non-convex objects fall on a dancer. Figure 2.1 shows we have no problem handling many objects in real time.

2.6 Conclusions and Future Work

We presented a visual hull based real-time system for detecting collisions between real and virtual objects in a multicamera setting. We demonstrated the collision detection system could be coupled to a physically based simulation system. We had to locally approximate the real object by a plane to calculate accurate collision information for each collision. We also analysed the effect of bounding boxes on the performance of the algorithm.

Currently, the system can not handle objects that move very fast. If at one instant, the visual hull is in front of a rigid body and at the next frame, it is behind the rigid

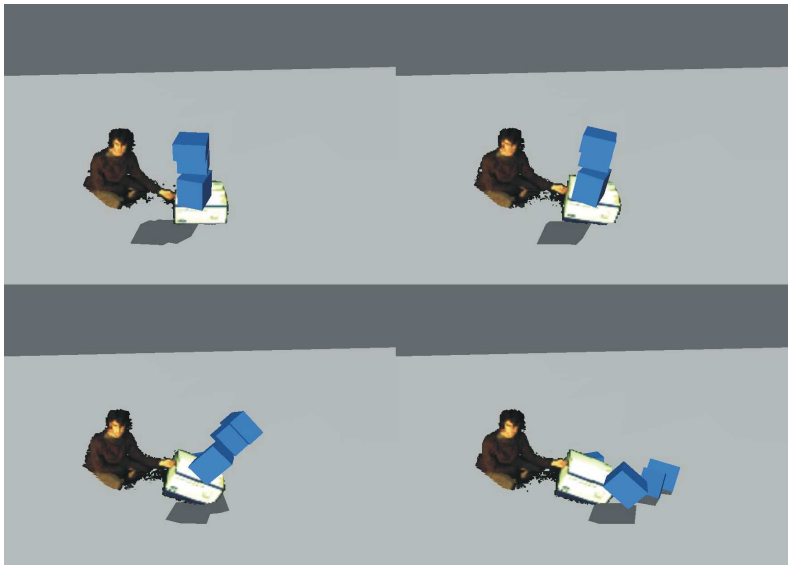


Figure 2.10: A stack of virtual boxes on top of a real box is tipped over by lifting the real box. This demonstrates how our technique is capable of modeling virtual objects that are lying in rest on top of real ones, due to correct treatment of multiple simultaneous collisions.

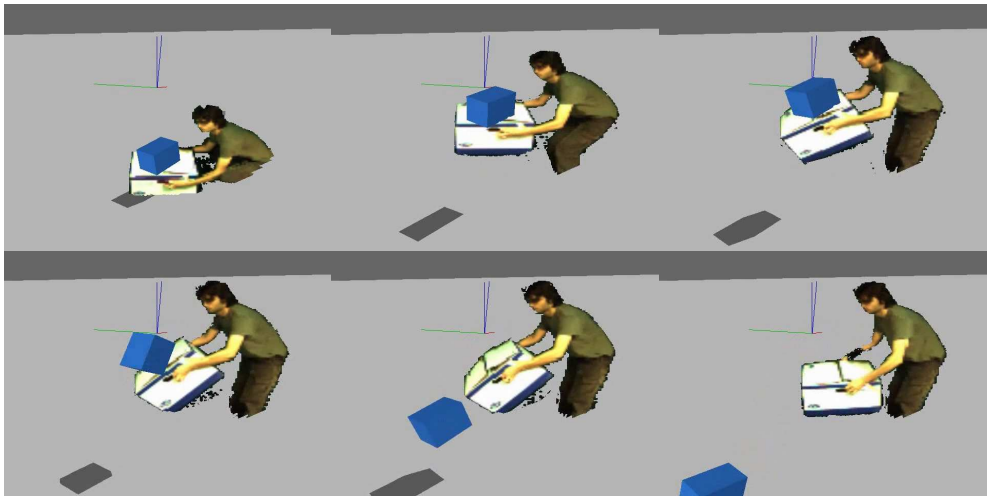
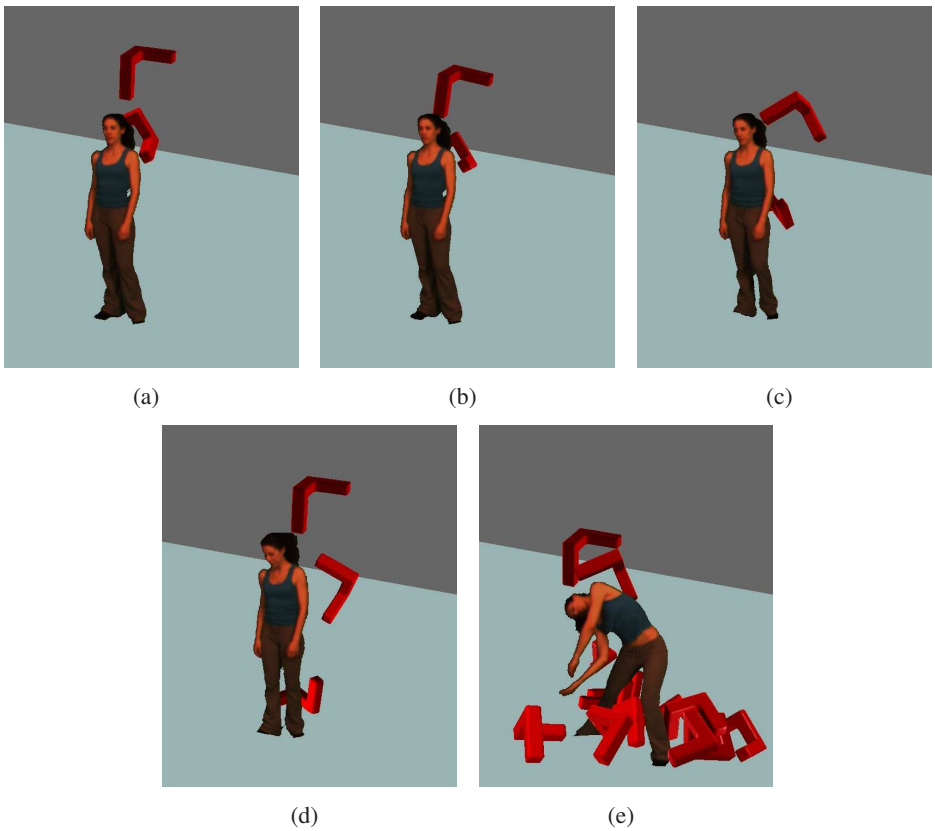
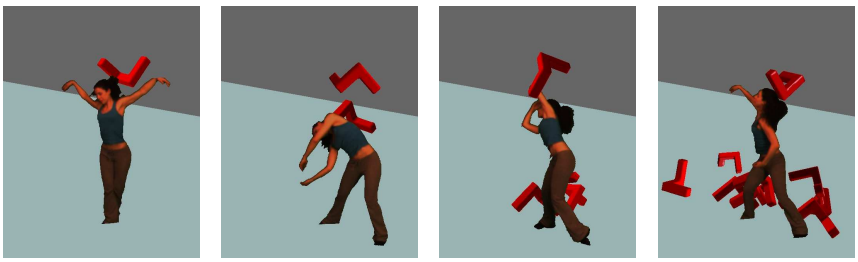


Figure 2.11: A virtual box on top of a real box. When the real box is lifted, the virtual box stays on top of the virtual box. When the real box is tilted, the virtual box falls from the real one.



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

Figure 2.12: A dancer interacts with concave objects.



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

Figure 2.13: A dancer interacts with concave objects.

body, the contact that should have been detected is missed. To solve this, we have to be able to query the visual hull in between two frames. For each camera, we would need some kind of optical flow between two consecutive frames of this camera to approximate the shape of the visual hull between these two frames. Since one of our goals is interactivity, calculating an approximation of the visual hull in between two frames is not an easy task.

Chapter 3

Motion Capture

Contents

In this chapter we present two new motion capture systems. They can be used to obtain information for a sparse set of scene points. We show how this information can be used in the context of video manipulation. Both systems are low cost optical

tracking systems that use imperceptible markers.

The first is a camera based system that locates and identifies a number of electronic markers that contain blinking LEDs.

The second system is called *Prakash*. In this system, light patterns are projected into the scene and electronic markers in the scene decode these patterns to calculate their position, orientation and incident illumination.

3.1 Related Work

3.1.1 Why Optical Tracking ?

The surveys by Welch and Foxlin [Welch 02] and Hightower and Borriello [Hightower 01] describe a large number of approaches to build a motion tracking system using magnetic, acoustic, optical, inertial or radio frequency signals.

We chose to create optical systems, because of the advantages over other approaches. Optical systems are typically able to achieve a high accuracy and low latency. The major disadvantage of optical systems is their need for line of sight. Ambient light can also interfere with such systems.

A typical optical system consists of a number of light sources and light sensors. Optical systems can be subdivided into two groups. Outside-in tracking systems have light sensors outside the scene and light sources in the scene. Inside-out tracking systems are the opposite, they have light sources outside the scene and light sensors in the scene.

3.1.2 Outside-in Tracking

In outside-in tracking systems, the light sensors are located outside the scene, and the markers that are being tracked emit or reflect light to the sensors. Most outside-in optical tracking systems film a number of markers with high-speed cameras. To each interesting point in the scene, a marker is attached. The 2D position of the markers is determined by each camera. Because the cameras are calibrated, a 3D position can be obtained from the multiple 2D positions captured by the cameras. The markers can reflect light from their environment onto the cameras or they can emit light themselves.

In most systems that use reflecting markers, retroreflective materials are used. These materials have the special property that they reflect most of the incident light back

Figure 3.1: In the PhaseSpace system [inc 07], a number of LEDs are attached to points of interest. These points are tracked using multiple high speed cameras to obtain their 3D position.

to the light source it was emitted from. To maximally profit from the retroreflective effect of the markers, a light source is placed near each camera[Viconpeak 06, Corporation 06].

In systems that use markers that emit light themselves, each marker has to be equipped with a power source and a light source [inc 06, Optotrak 07, Codamotion 07, inc 07]. LEDs are mostly used as light sources because of their small size and energy efficiency. Using markers with LEDs instead of reflective markers has one big advantage. If each marker is given a unique ID, the LEDs can be turned on and off in a pattern corresponding to this ID. The cameras capturing these LEDs can determine the iID of each LED from the way it turns on and off. This way two markers will never get swapped — even after occlusions the system will be able to identify each marker.

Because high frame rates are desirable in a tracking system, high speed cameras are often used. The disadvantage of high speed cameras is their cost and their high bandwidth requirements. To alleviate these problems, the PhaseSpace motion tracking system [inc 07] uses two 1D line sensors instead of a normal 2D camera (See figure 3.1). In their system, the LED of only one of the markers is turned on while the others are turned off. When the scene is captured using two 1D line sensors that are orthogonal to each other, the 2D position of the marker can be obtained. The 3D position can be obtained if even more 1D line sensors are used.

The disadvantage of outside-in tracking is that the contrast between the markers and

Figure 3.2: Using the HiBall system [Welch 97], a marker which contains multiple light sensors senses the amount of light arriving from a number of LEDs attached to the ceiling to calculate its position.

the rest of the scene has to be high. Otherwise the markers can not be found in the camera images, and the tracking fails. When tracking a human actor using such systems, the person has to wear dark clothes.

3.1.3 Inside-out Tracking

The opposite of outside-in tracking systems are Inside-out Tracking systems. In these systems, the light sources are placed outside the scene while the light sensors are located on the markers that are being tracked.

One way to construct an inside-out tracking system is to sweep the scene with a fast rotating beam of light. The markers that are being tracked contain a cheap light sensor. By sensing the moment in time when the sensor is lit by the beam of light, its position can be obtained. Multiple rotating light sources are needed to obtain a 3D position. Examples of such methods are Indoor GPS [Kang 04, Sorensen 89], Spatiotrack [Iltanen 98] and Shadow Track [Palovuori 00].

In the UNC HiBall system [Welch 97], multiple light sensors are placed on the marker (See figure 3.2). Each of these sensors faces in a different direction. The ceiling is covered with LEDs that are turned on one after the other. The amount of light arriving at the light sensors from each of the LEDs depends on, among other things, the angle and the distance between the sensor and the light source. When the amount of light

(a) sketch of the system (b) a person using the system

Figure 3.3: Using the RFIG system [Raskar 04] a person can find out information about boxes in a warehouse by aiming a projector at them.

arriving at the sensor is known for a number of LEDs, it becomes possible to calculate the position and orientation of the sensor.

The performance of most inside-out tracking systems does not depend on the number of markers. This is an advantage over outside-in tracking systems, where it becomes increasingly difficult to identify each marker when more markers are added to the system. With inside-out systems, the markers calculate their own position and identifying markers is trivial. However, in outside-in tracking systems all the markers can be very simple devices consisting of a retroreflective patch or a single LED, whereas in inside-out tracking systems, the markers are more complex because they have to sense light and process their measurements.

3.1.4 Projecting Data

A video projector can be used to project more than just images. Extra information, invisible to the human eye, can be embedded in the projected images. Or by projecting special patterns instead of normal images onto a scene, a light sensor attached to a marker in the scene can calculate its position.

Raskar et al. [Raskar 04] present a system where a hand-held projector illuminates a scene containing a number of objects which have markers attached to them (See figure 3.3). The markers are RFID markers that contain a light sensor. When the markers are lit by a number of binary light patterns, they can calculate their own position. Their position is sent back to electronics attached to the projector. The projector can then project additional information onto the objects because their positions are now known. This is useful in a warehouse where a person with a portable

projector can find out information about boxes by aiming a projector at them.

Nii et al. [NII 05] present a system where a projector is used to send position dependent information (e.g. audio) to receivers in a scene. They do this in such a way that it seems to a human observer that the projector is projecting a normal image. They use the observation that the human visual system averages out patterns that are projected onto a scene at a very high speed. The projected patterns are chosen in such a way that they are averaged out by the human visual system to produce a normal image, while special devices with optical sensors are able to decode the invisible high frequency data that is embedded in these patterns. Cotting et al. [Cotting 04] present a similar system where they use a DLP projector to project the patterns at a high speed. Lee et al. [Lee 05] transmit low-perception grey code patterns encoded in an image with a projector onto a scene containing markers with light sensors attached to them. The markers that receive this light can calculate their positions while a human observer sees a normal image. The least significant bits of the projected patterns are used to transmit data to the markers, while the most significant bits of the projected patterns are used to let a user view a normal image.

3.1.5 Projectors for Tracking

One could use projectors designed to project video images for the purpose of transmitting (location) information to optical sensors in the scene, as explained in the previous section. Because the amount of information that can be sent depends on the speed of the projector, high speed projectors are desired. High speed projectors designed for projecting video images rely on Digital Light Processing (DLP) or on Gated Light Valves (GLV) [Machines 06]. Instead of using a general purpose projector to project data onto the scene, one could devise projectors that are only able to project patterns for location tracking. With only this application in mind, choices can be made that result in a low cost and a high frame rate as we will show in section 3.3.

3.2 Locating And Identifying Active Markers using a Multi-Camera Setup

In this section we describe a camera based tracking system that can identify and locate a number of electronic markers. Each marker contains an IR LED that emits a unique time coded light pattern. The 3D position of each marker can be calculated using images of the scene captured by multiple calibrated cameras. The time coded

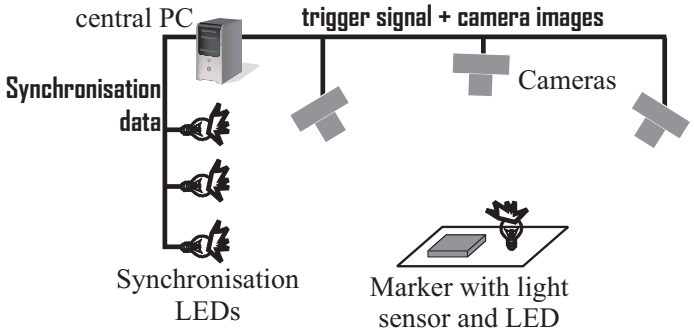


Figure 3.4: A sketch of the hardware components of our system.

light pattern is used to uniquely identify each marker so there are no reacquisition problems after occlusions.

3.2.1 Introduction

Figure 3.4 shows a sketch of the system. The light emitted by a LED on the marker is captured by the cameras. The captured images are sent to a PC that identifies the marker and calculates the 3D position of the marker. The PC is responsible to keep everything synchronised. To synchronise the cameras to the PC, the PC sends a trigger signal to the cameras. To synchronise the markers to the PC, synchronisation data is optically send to the markers using the synchronisation LEDs. To capture this synchronisation data, the markers contain a binary light sensor.

The system is applicable in a wide range of situations, because its accuracy and working volume depend entirely on the cameras and the lenses used. Simply by changing the cameras or lenses, the system can fit specific budget and accuracy constraints. Because the cost of cameras is dropping, the presented system is not expensive. Because it is an optical system, occlusions can cause the system to fail. We address this issue by the use of a redundant set of cameras. The light emitted by the markers is invisible to the human observer because infra red LEDs are used.

In figure 3.5, we show how the system is used to create peephole displays. Robots with laptops on top of them drive around on a stage during a theatre play. Because the robots are being tracked using our system, the laptops can display images to create the illusion of a peephole display.

(a) (b)

Figure 3.5: Laptops attached to robots are tracked by our system to create peephole displays during a theatre play.

3.2.2 Strobe Pattern

Because the cameras are calibrated, calculating the 3D position of a marker is easy once its 2D position is found for each camera. When multiple markers are being tracked, the hard part is to find the corresponding LED in an image captured by one camera given the 2D position of this LED for another camera. To solve this problem, each marker is given a unique ID. This is a number that uniquely identifies each marker. This ID is embedded in a time coded light pattern emitted by the LED on the marker. By decoding this pattern, the PC can identify each marker using the images captured by the cameras. For each frame the cameras capture, the LED is turned on/off according to this pattern.

Figure 3.6 shows a light pattern that can be used to identify 8 markers. The even frames of the pattern are used to locate the markers and to determine occlusions. For these frames, the LED of all markers are turned on. The odd frames are used to identify the markers. During each odd frame, the LED emits a single bit of the ID of the corresponding marker. In the example pattern shown in the figure, the ID consists of 3 bits and one parity bit for error detection, thus $2^3 = 8$ markers can be identified using this pattern. In our prototype setup we used 64 markers, in that case the marker ID consists of 6 bits and the strobe pattern is 14 frames long.

frame nr	1	2	3	4	5	6	7	8
LED on/off	bit1	on	bit2	on	bit3	on	parity	on

Figure 3.6: The strobe pattern of a LED is used to find the position and the ID of the corresponding marker. In this example, the ID of the marker consists of three bits. These three bits are encoded in the strobe pattern.

3.2.3 Tracking Algorithm

Overview

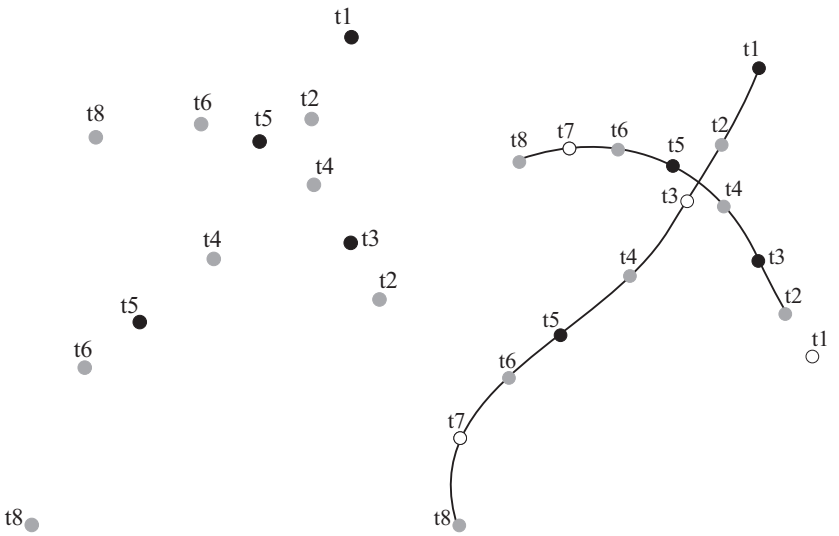
In this section we give an overview of the algorithm to locate and identify the markers using the images captured by the cameras. In the next sections each step of the algorithm will be described in detail.

To find the 3D position and ID of a marker, we first find its 2D pixel position and ID for each camera. From this we can calculate its 3D position

To find the 2D position in image space and the ID for each marker for a given camera we perform a number of steps. Note that we need multiple frames to decode the ID of the markers visible in these frames (see figure 3.6). First we use simple thresholding to find the 2D position of the markers in these camera images. Image (a) of figure 3.7 shows an example where two markers are filmed by a single camera. It shows the 2D pixel positions where a LED was located for 8 consecutive frames, each position is labeled by the frame number where the LED was found. Note that for most frames, we found more than one LED. Next we find correspondences between the marker positions over time. For each marker that is visible in the camera images, we find its trajectory (See figure 3.7 (b)). This way we know, for a given marker position and frame, what the position of this marker was for the other frames. To find the trajectories, we only use the even frames, the moments for which we know all the LEDs are turned on (see figure 3.6). Once the trajectory of each marker is known, its corresponding ID can be calculated. To find the ID, we need to determine for which odd frames the LED of the marker was turned on (see figure 3.6). Because we calculated the trajectory of the markers for the even frames, we can estimate their positions for the odd frames. To determine whether the LED is turned on for a given odd frame, we simply check if we found a LED in the camera image near the estimated position.

So in summary

- Find 2D positions of LEDs for each camera image using thresholding.



(a) All the LEDs we found over 8 frames for a single camera. Each point is labeled with the frame number it was found in. The LED positions found in even frames are colored gray, those found in odd frames are colored black.

(b) Each line represents the trajectory of a single marker and connects the positions of this marker. These trajectories were estimated using the even frames. The white circles depict the estimated position of the marker for odd frames where the LED was turned off.

Figure 3.7: Finding the position and ID of multiple markers for multiple frames of a single camera.

- Find trajectory of each LED over a small number of even frames.
- Find the corresponding ID for each trajectory using the odd frames.
- For each camera, we now have the 2D position and the ID for each marker. Now we combine the information from multiple cameras to obtain the 3D positions.

Now we will describe each step in detail.

Segment images

Given a single image of a camera, we need to find the 2D position of each LED in this image. We use two observations to do so. The LEDs are brighter than most objects in the scene and they are very small, so the gradient of the image is large near a LED. To find the LEDs, we segment out each pixel that is brighter than a given threshold and

for which the image gradient is larger than another threshold. Now we have multiple image segments where a LED might be. For each segment we calculate the size. If it is too big to be a LED, it is removed. We calculate the centre of the remaining segments and these are passed on to the next step of the algorithm as the potential 2D positions of the markers.

Find trajectory

The input of this step of the algorithm is a number of potential marker positions for a number of even frames. An example of such a dataset is shown in image (a) of figure 3.7. For each point in the current even frame, we need to determine which point in the previous even frame corresponds to it. We only use the even frames for this, because we know all LEDs are turned on for these frames (see figure 3.6). We also have to make sure this step is robust against errors made during the previous step. The previous step might have found false positives.

We use the observation that we are designing a tracking system for a large working volume. So in between two even frames, the markers will not have moved many pixels. If we want to know, for example, which point in frame 4 corresponds to a given point in frame 2, we simply look for the point in frame 4 that is the closest to the given point of frame 2. If this distance is not too large, we assume these two points correspond to the same marker. If the distance is larger that a given threshold, we assume there is no point in frame 4 that corresponds to the given point. This could be due to occlusions or the given point might be a false positive from the previous step of the algorithm.

Find ID

Once the trajectories are found, we know the position of each marker for a number of even frames. But we do not know the ID of each marker yet. To find the ID for a given marker, we need to know for a number of odd frames whether its LED was turned on or off (See figure 3.6). To do this, we estimate the position of the marker for each odd frame by interpolating the position of the marker at the even frames. For a given odd frame, we assume the LED of the marker is turned off if no LED is found near the estimated position. If a LED is found in the camera image near this position, we assume the LED of the marker was turned on. We use linear interpolation to estimate the position.

Once all the bits of the ID and the parity bit are obtained this way, we check the ID against the parity bit to determine whether an error has occurred. One reason for

errors might be occlusions We found that one parity bit was enough to obtain a robust system, but more rigid error detection can be incorporated if needed.

Find 3D position

Now we know the 2D position and ID of each marker for each camera and frame. Because the cameras are calibrated, we can find the 3D position of a marker for a given frame, using the 2D positions if the marker is visible in at least 2 cameras.

3.2.4 Hardware Details

Overview

Figure 3.4 shows the different hardware components of our system. The cameras that capture the scene are connected to a PC. This PC processes the images and broadcasts the position of each marker. The marker contains a microcontroller and a LED to signal its position to the cameras. Because the markers are battery powered, we want them to use as little power as possible. That is why we turn on the LEDs only for the very short time the shutter of the cameras is open. This is why the markers have to be synchronised to the cameras. The PC also needs to know which bit of the strobe pattern (see figure 3.6) the markers are sending for a given frame, so the markers also have to be synchronised to the PC. For this purpose, the PC is responsible to send trigger signals to the cameras and synchronisation signals to the markers. The PC sends a trigger signal to the cameras using an external trigger cable. To send a synchronisation signal to the markers, a number of synchronisation LEDs are placed above the scene. The PC sends a synchronisation signal to the synchronisation LEDs over a synchronisation cable, these LEDs optically send this signal to the markers that contain a light sensor to capture the signal. The technology used to optically send this signal is the same as that used in the remote control for a television. In the synchronisation signal the PC sends to the LEDs, the current frame number is encoded. This way, the markers know which bit of their strobe pattern they need to send to the cameras.

Figure 3.8 shows a block diagram of the hardware components of our tracking system. Each rectangle with a dashed border represents a component of our system. Each subcomponent is represented by a solid rectangle. The arrows represent the flow of data. Solid arrows represent wires and dashed arrows represent wireless connections. In the following sections, we will describe each component in detail.

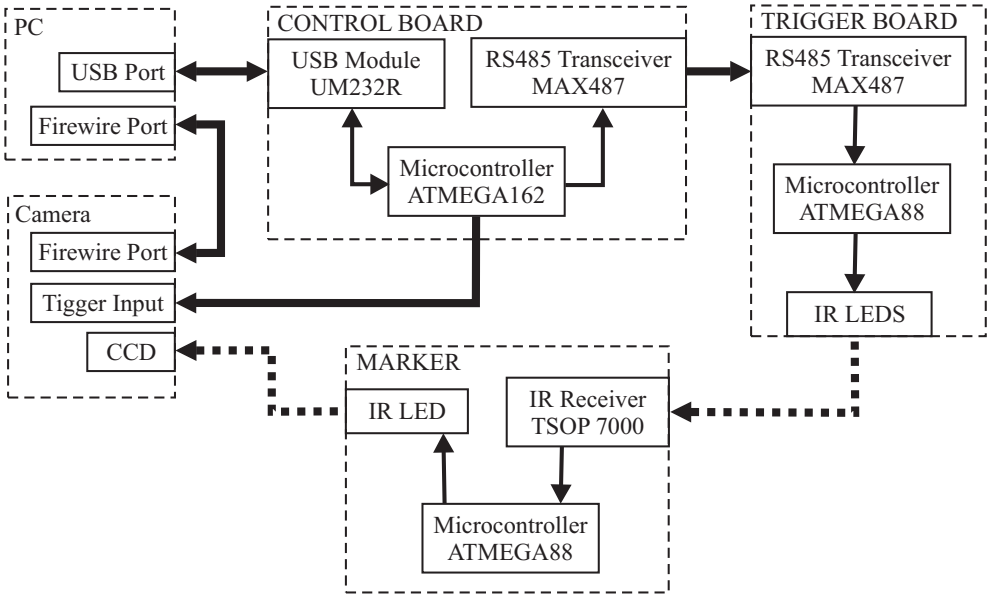


Figure 3.8: Block diagram of the hardware components of the system.

PC

The PC runs the tracking software that interprets the images from the cameras. Firewire is used to connect the cameras to the PC. The tracking software has to be synchronised to the markers and the cameras. To this end the PC is connected to the Control board. This is a custom designed printed circuit board responsible for keeping all the components in the system synchronised.

Each time the tracking software receives a frame from the cameras, it tells the Control board a new image has to be captured while the tracking software processes the current one. To do this, the PC sends the current frame number to the Control board (also see figure 3.6). The Control board will make sure that when the cameras capture the next image, the LED of each marker is turned on or off according to this frame number and the strobe pattern of the marker. This way the PC stays synchronised with the other components in the system.

Cameras

The cameras send the images they capture to the PC. They are triggered when a signal is applied to their trigger input. This input is connected to the Control board. So the

Control board controls when the cameras capture an image. The cameras capture the position and ID of each marker through their CCDs.

Control board

This is a custom designed printed circuit board. It is connected to the PC and is responsible for synchronising the other components in the system. It contains a microcontroller with two USARTs (Universal Synchronous-Asynchronous Receiver/-Transmitter) used for serial communication with the PC and the Trigger board.

One of these USARTs is used to communicate with the PC. This USART is directly connected to a USB module. The USB module is connected to the PC and acts as a virtual com port. Using this virtual com port, the tracking software on the PC can communicate with the microcontroller.

The other USART is used to send synchronisation data to the markers. To this end, it is connected to a Trigger board which optically sends data to the markers. Because the distance between the Control board and the Trigger board can be quite large, RS485 is used to connect them. The USART of the microcontroller is connected to a RS485 Transceiver which in its turn is connected to the Trigger board.

To trigger the cameras, a general purpose IO pin of the microcontroller is connected to the trigger input of the cameras.

Trigger board

The Trigger board is also a custom designed printed circuit board. It is responsible for relaying synchronisation data from the Control board to the markers. This data is optically transmitted to the markers using a number of LEDs. The light the LEDs emit is modulated with a carrier frequency of 455 kHz to make the system more robust against ambient light. The markers only sense light that is modulated by this frequency. The ambient light, which is not modulated by this frequency, is ignored.

A microcontroller on the Trigger board receives synchronisation data from the Control board using a RS485 link. To this end, the USART of the microcontroller is connected to a RS485 transceiver which in turn is connected to the PC Board.

The microcontroller receives synchronisation data from the Control board and checks it for transmission errors. If the data is OK, it is transmitted to the markers using the LEDs. A general purpose IO pin of the microcontroller is used to turn the LEDs on and off.

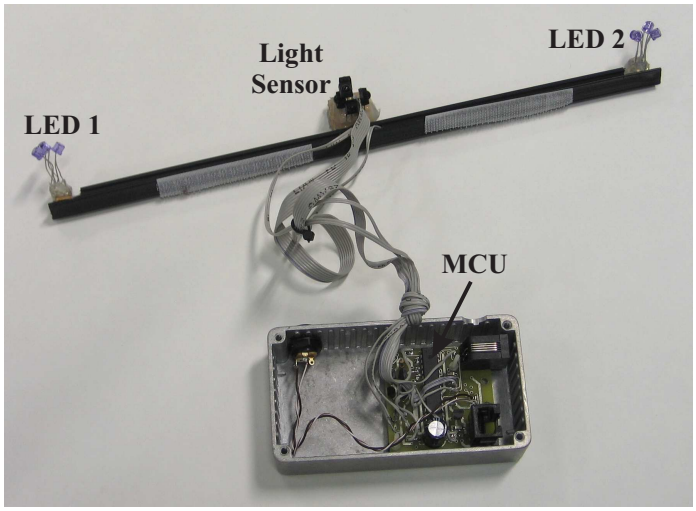


Figure 3.9: One of the markers that is tracked by our system.

The Control board is typically located near the PC while the Trigger board is located above the scene where its LEDs can illuminate the markers below.

Marker

Each marker contains an IR receiver to receive the synchronisation signal from the Trigger board, a microcontroller and a LED to signal its position and ID to the cameras. The IR Receiver is a binary chip that filters out light that is not modulated by a carrier frequency of 455 kHz It is connected to the USART of the microcontroller. The microcontroller interprets the synchronisation data and turns the LED on or off according to the strobe pattern of the marker. The ID of the marker is programmed in the memory of the microcontroller together with the firmware.

Figure 3.9 shows a picture of one of the markers that is tracked by our system. Note that this marker contains two LEDs that are being tracked, and consequently it also has two IDs, one for each LED. The light sensor, used to capture the synchronisation signal, and the microcontroller (MCU) are also visible in this picture.

Timings

Figure 3.10 shows the time it takes for the synchronisation signal to travel from the PC to the markers. When the PC receives an image from the cameras, it sends a

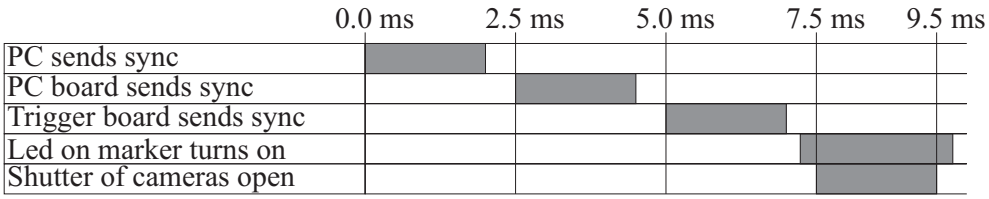


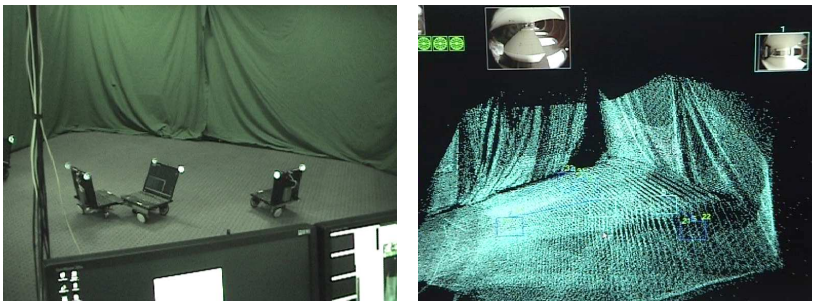
Figure 3.10: The timings of the synchronisation signal.

synchronisation signal to the Control board. The Control board receives this, checks for transmission errors and sends this signal to the Trigger board. The Trigger board also checks for transmission errors and then sends the signal to the markers. The markers then turn their LEDs on or off according to the synchronisation signal and their ID. The Control board sends a synchronisation signal to the cameras at the same moment that the markers turn on their LEDs. The time it takes to send the synchronisation signal from the Control board to the markers is hard coded in the firmware of the Control board, so the Control board knows how long it has to wait after sending the synchronisation signal before triggering the cameras. The shutter of the camera is opened for 2 ms and the LEDs of the markers are turned off once the shutter of the camera closes. The LEDs are turned on a little longer than the camera shutters are open to allow for a margin of error in the timings. The shutter time of the cameras is hard coded in the firmware of the markers, so the markers know how long their LEDs should be turned on. This way the LEDs of the markers are turned on for only a very small amount of time and battery power is not wasted. Once the images are captured, the cameras wait for the PC until it finishes processing the previous image. Then the images are sent to the PC and the PC signals the Control board to start capturing the next image.

3.2.5 Results

Peephole Display

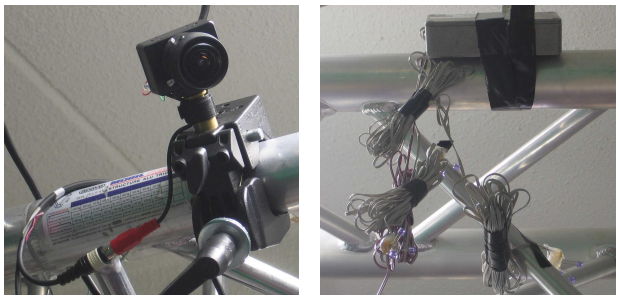
We tracked a number of robots with laptops on top of them using our tracking system to build peephole displays. In this setup we used four cameras with a resolution of 640x480. The area across which the robots are tracked is 10 by 8 meters, which results in an accuracy of approximately 10 m / 640 $\approx$ 8 m / 480 $\approx$ 2 cm. The cameras are calibrated using the calibration toolbox by Svoboda [Svoboda]. In image (a) of figure 3.11 we show a number of laptops that are being tracked. Each laptop is equipped with one markers containing two LEDs (see figure 3.9) to estimate the



(a) Robots being tracked

(b) Estimated 3D position of the robots along with a point cloud representation of the scene.

Figure 3.11: Tracking robots to create peephole displays.



(a) Camera

(b) The synchronisation LEDs

Figure 3.12: One of the IR Cameras and the LEDs which send the synchronisation signal to the markers.

position and 2D orientation of each robot. Because the camera used to capture image (a) is sensitive to infrared light, the light emitted by the markers can be seen as the white dots at the top of the laptop displays. Note that this IR light is invisible to the human observer. Image (b) shows a 3D point cloud representation of the scene. The squares represent the estimated positions of the robots.

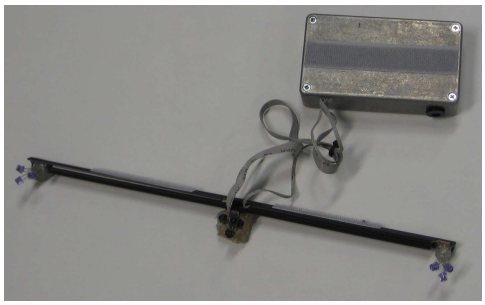
Image (a) of figure 3.12 shows one of the IR cameras that are used to capture the light from the LEDs on the markers. Image (b) shows the LEDs used to optically send the synchronisation data to the markers.

Verification Experiments

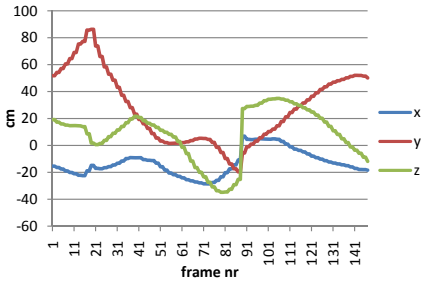
In this section we perform a number of experiments to determine the accuracy of our system. Four cameras with a resolution of 640x480 pixels are attached to the ceiling (2.5m above the floor) of a square room that measures 4m x 5.5m. All the experiments are performed near the floor, in the middle of the room. The distance between this point and the cameras is 4.5m.

In the first experiment, two markers are attached to a stick, approximately 30 cm apart from each other. The tracking system is used to measure the distance between these two markers while the stick is moved around in a box of approximately 1m x 1m x 1m. The variation in the calculated distance is a measure for the accuracy of our system. The results of this test are shown in figure 3.13. Image (a) shows the two markers (LEDs) attached to a stick. Image (b) shows the 3D position of the first marker calculated by our system. The position looks jagged because it is only updated during the even frames (see figure 3.6). Image (c) shows the distance between the markers calculated by our system for each frame. The average distance is 30.94 cm. For each frame, the difference between the calculated distance and the average distance is shown in image (d). The average difference is 0.17 cm and the maximum difference is 0.51 cm.

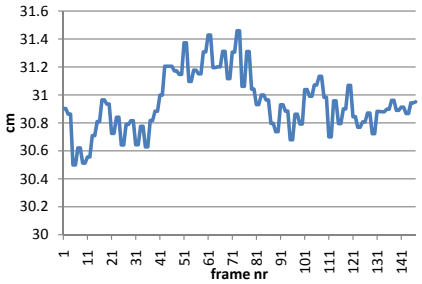
In the second experiment we move a marker along a line and capture its position every 10 cm. The results of this test are shown in figure 3.14. Image (a) shows a picture of the setup used to perform this experiment. A marker is shown next to a ruler that is used to move the marker exactly 10 cm between two measurements. Image (b) shows the 3D position of the marker relative to its position at the first frame. Image (c) shows the distance between the position of the marker and the position of the marker at the first frame along with the ground truth. The difference between the ground truth and the calculated distance is shown in figure (d). The average difference is 0.53 cm and the maximum difference is 1.24 cm.



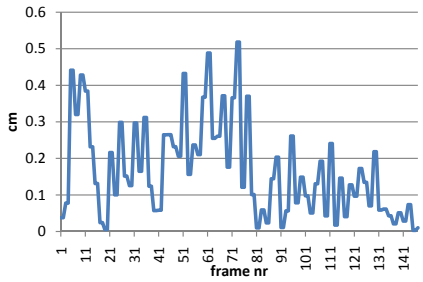
(a) Setup



(b) XYZ position of the first marker

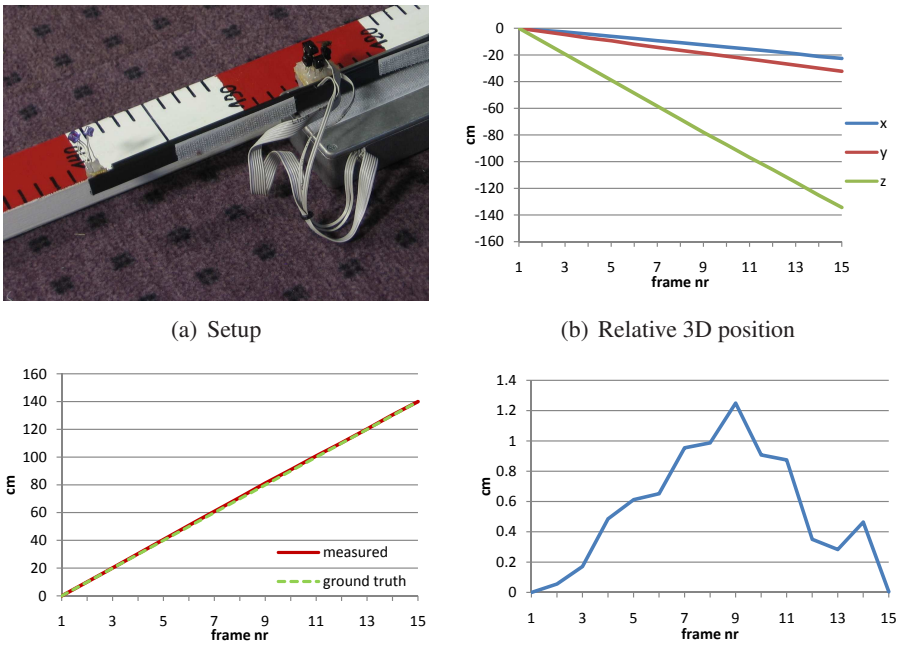


(c) Distance between the markers



(d) Error

Figure 3.13: The tracking system is used to calculate the distance between two markers attached to a stick that is moved around. The error is the difference between the average distance and the distance for a single frame.



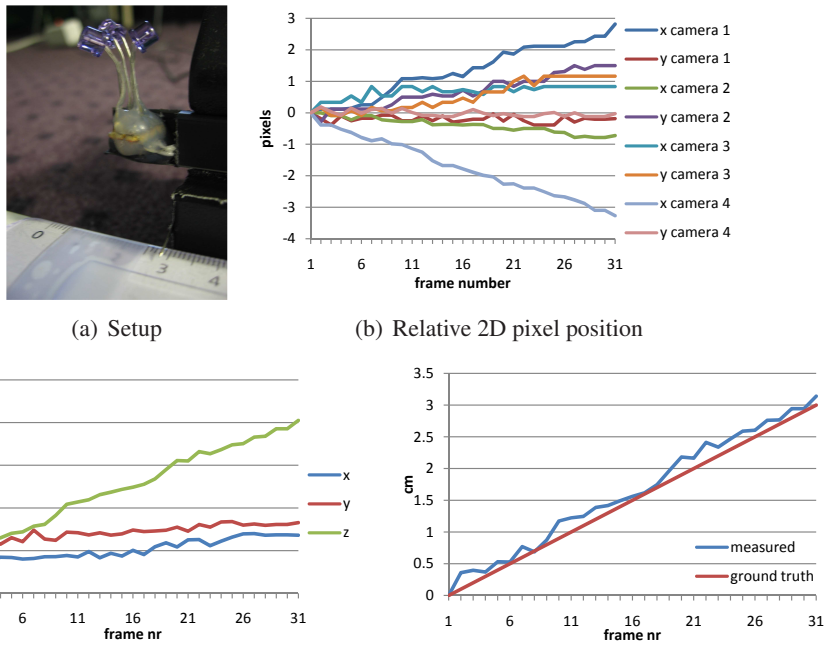
(a) Setup

(b) Relative 3D position

(c) Distance

(d) Error

Figure 3.14: The position of a marker is calculated every 10 cm.



(a) Setup

(b) Relative 2D pixel position

(c) Relative 3D position

(d) distance

Figure 3.15: The position of a marker is calculated every millimeter.

In the final experiment we measured the spatial resolution of the system by sliding a marker along a line and capturing its position every millimeter. The results of this test are shown in figure 3.15. Image (a) shows a marker next to a measure. Image (b) shows the 2D pixel coordinate of the marker for each of the four cameras. The coordinate is relative to the coordinate of the marker at the first frame. The coordinates are calculated with subpixel accuracy. This is possible because the marker is larger than one pixel in the camera images, the larger the marker in the camera image, the higher the accuracy. This can be seen in the graph, camera 4 is closer to the marker and therefore the marker is larger in the images it captures and thus the subpixel accuracy of camera 4 is higher than that of, for example, camera 3 which is further away. Image (c) shows the 3D position of the marker and image (d) shows the distance traveled along with the ground truth. The average difference between the distance measured by our system and the ground truth is 0.13 cm and the maximum difference is 0.31 cm.

Conclusions

From these experiments, we can conclude that our tracking system is able to accurately measure both small local movements and the absolute marker position. This is an advantage over many other methods that are only able to determine local movements (such as inertia based systems) or global movements (such as GPS based systems). Our system is able to determine the absolute position of each marker accurately because current camera calibration methods are very accurate. Because the markers are larger than one pixel in the images, we are able to determine their position with subpixel accuracy. This way our system is able to accurately measure very small movements.

When we assume a 2D accuracy of one pixel (no subpixel accuracy), the theoretical spatial resolution of the setup is approximately 5.50 m / 640 pixels $\approx$ 4 m / 480 pixels $\approx$ 0.8 cm. This matches the results from our experiments because we can conclude from the first two experiments that the spatial accuracy of this setup is between 0.5 cm en 1 cm.

3.2.6 Conclusions

We presented a real time camera based system to locate and identify a number of electronic markers. Because the price of cameras and computational power is quickly decreasing, the hardware cost of the system is low. Because the accuracy and working volume of the system completely depends on the cameras and lenses used, one can

easily adjust the system to comply with certain cost and performance requirements. To demonstrate the usefulness of our system, we build peephole displays by tracking robots with laptops attached to them. In this setup, a large working volume (10m by 8m) was used with an accuracy of approximately 2 cm.

The system could be improved by combining multiple steps of our algorithm using probabilistic methods. One could devise a system that finds the 3D positions, trajectories and IDs all at once. By doing this, more prior information will be available to each step of the algorithm resulting in a more robust system. The system could be made more robust against ambient light by placing an IR bandpass filter in front of the cameras to filter out light with a different wavelength than that of the IR LEDs.

3.3 Prakash

In this section we describe *Prakash*, a high speed motion tracking system that captures, not only the position, but also the orientation and incident illumination for a number of scene points. In contrast to the system presented in the previous section, this is not a camera based system. Instead, multiple patterns of light are projected onto the scene and electronic markers calculate their own position and orientation by sensing this light. The result is a highly scalable system which is not bound by the number of markers being tracked.

3.3.1 Introduction

Many high speed motion tracking systems rely on high speed cameras to obtain the position for a number of scene points that are marked by special reflective markers [Robertson 06]. A disadvantage of this approach is the need for a large amount of bandwidth and computational power to handle the vast amount of data generated by the high speed cameras.

In contrast to camera based systems (such as the one presented in the previous section), the method presented in this section is an inside-out tracking system. An electronic marker containing a number of light sensors is attached to each scene point we want to track. Time multiplexed light sources project well chosen illumination patterns onto the scene. By sensing these illumination patterns, the markers can calculate their position, orientation and incident illumination.

Unlike with camera based approaches, decoding the light patterns is not computationally intensive, nor are there any high bandwidth requirements. The system can

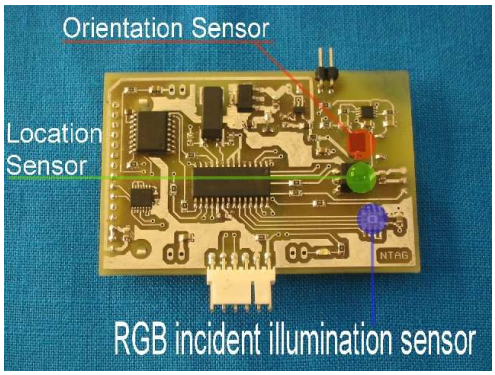


Figure 3.16: A marker is attached to each scene point of interest. It consists of a microcontroller and a number of sensors to calculate its position, orientation and incident illumination.

be built using off the shelf components, resulting in a low cost system. Because we use an optical system, interreflections of light and very bright ambient illumination can cause the system to fail. Later in this section we will describe which measures were taken to make the system more robust against ambient light. We show this by tracking a person outside in daylight.

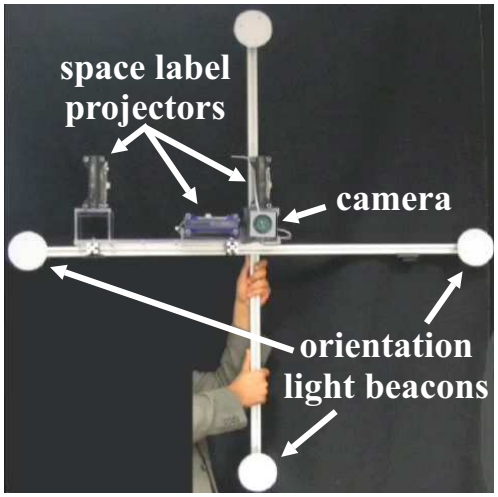
The main contributions of our system are:

- High speed location and orientation tracking by decoding multiplexed light patterns.
- Capturing incident illumination.
- The used markers are imperceptible under clothing and work in normal lighting environments.
- New video manipulation methods that use the measured properties.

3.3.2 Estimating Scene Parameters

In this section, we describe how each property — position, orientation and incident illumination — is captured and we analyse the accuracy of the system.

An electronic marker is attached to each point in the scene we need to track. As is shown in figure 3.16, such a marker consists of a printed circuit board, a microcontroller and a number of optical sensors. For each property we capture (position,



(a)

Figure 3.17: This is an image of the light sources that optically label the scene. The marker uses the light from the projectors for position tracking, the intensity of the light from the orientation beacons for orientation tracking. A camera is added to show tracking results on top of a video of the scene.

orientation and incident illumination), a sensor is present on the marker. The micro-controller processes the information from the sensors and transmits the result via an RF system.

Figure 3.17 shows the light sources that label the scene. Three one dimensional projectors are used to label the 3D space, more details will be given in the section about location sensing. Four light beacons are used for the orientation tracking. And also a camera is present so the tracking results can be used to augment a video of the scene.

Position

First we will describe 1D position tracking, later we show how we can extend this to 2D and 3D tracking.

For 1D position tracking, a number of binary patterns are projected onto the scene. These patterns are shown in figure 3.18. The patterns are projected onto the scene one after the other using our custom projector. A sketch of a 4-bit projector is shown in figure 3.19. It consists of a number of LEDs, and in front of each LED is a mask.

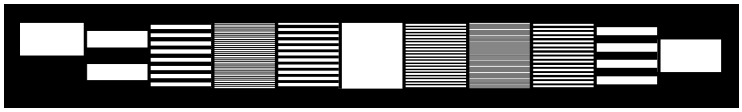


Figure 3.18: The light patterns that are projected onto the scene for 10-bit position tracking.

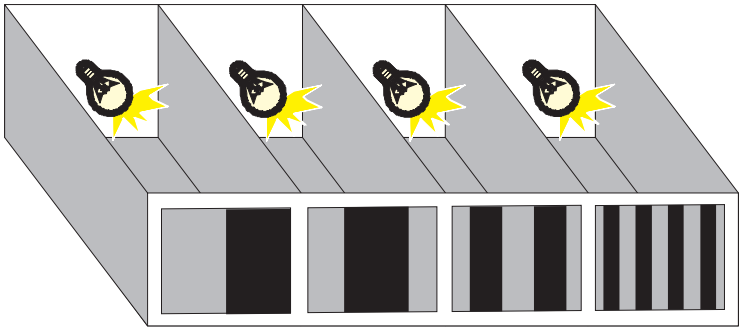
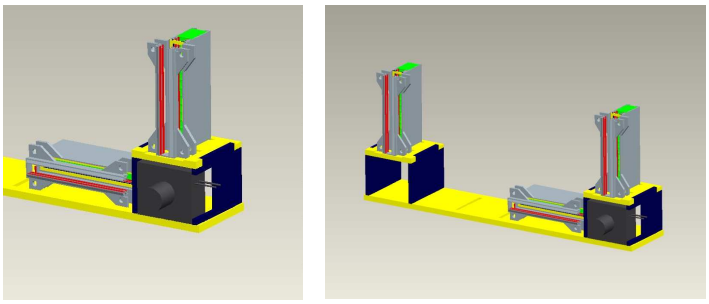


Figure 3.19: A 4-bit high speed pattern projector for location tracking. By turning the 4 LEDs on one after the other, the patterns in front of each LED is projected.

When a LED is turned on, the corresponding pattern is projected onto the scene. When the LEDs are turned on one after the other, a marker in the scene can calculate its position from the sequence by which it is illuminated by the light patterns. The first (leftmost in figure 3.19) pattern is used to obtain the most significant bit of the coordinate of the marker, it decides whether the marker is in the left or in the right side of the scene. The second pattern is used to obtain the next bit of the coordinate. The last pattern is used to obtain the least significant bit of the coordinate. So, each mask that is projected improves the accuracy of the system by one bit. In our setup we used projectors with 8 LEDs, so the coordinates have a precision of 8 bits. To decode the light patterns, the marker has a binary light sensor that senses whether or not it is illuminated. Infrared LEDs were used to make the system invisible to the human observer.

To make sure the system is robust against ambient illumination, the projectors emit light modulated with a carrier frequency of 455 kHz. The binary light sensor only responds to light with this carrier frequency. Ambient infrared light (e.g. from the sun) is not modulated with this frequency and is thus ignored. The technology used is the same as that in a remote control for a television.

Because we use LEDs as light sources we are able to switch them on and off very



(a) 2D setup (b) 3D setup

Figure 3.20: The setups of the projectors for 2D and 3D position tracking are shown.

rapidly and thus are able track the markers at high sample rates. Our 1D location tracker works at 1000 Hz.

To extend this approach to 2D, we simply put two 1D projectors orthogonal to each other and operate them one after the other. One projector is used to obtain the X-coordinates and the other one is used for the Y-coordinates. For 3D location tracking, the 2D system is extended by a third 1D projector in stereo with the first one. Using the first and the third projector, the distance between the marker and the projectors can be calculated using triangularization. Figure 3.20 shows a sketch of the setup for 2D and 3D location tracking.

The field of view of the system is 1.5m x 1.5m at a distance of 3m from the projectors, or 27 degrees. So, in theory, the X-Y resolution of our 8-bit system is 1.5m/255=5.8mm at 3 meters.

The resolution in de Z-direction depends on the distance between the first and the third projector, and it also depends on the distance between the marker and the projectors. If d is the distance between the projectors, p is the distance between the projectors and the marker and α is the angular resolution of the projector, then the theoretical Z-resolution of the system is equal to

$$\alpha \frac{d^2 + p^2}{d - \alpha p}$$

We have a baseline of $d = 1.3$m and an angular resolution of $\alpha = 0.0017$ rad $= 0.1$ degrees, so at a distance of $p = 3$m from the projectors, the theoretical resolution in the Z-direction is 22mm.

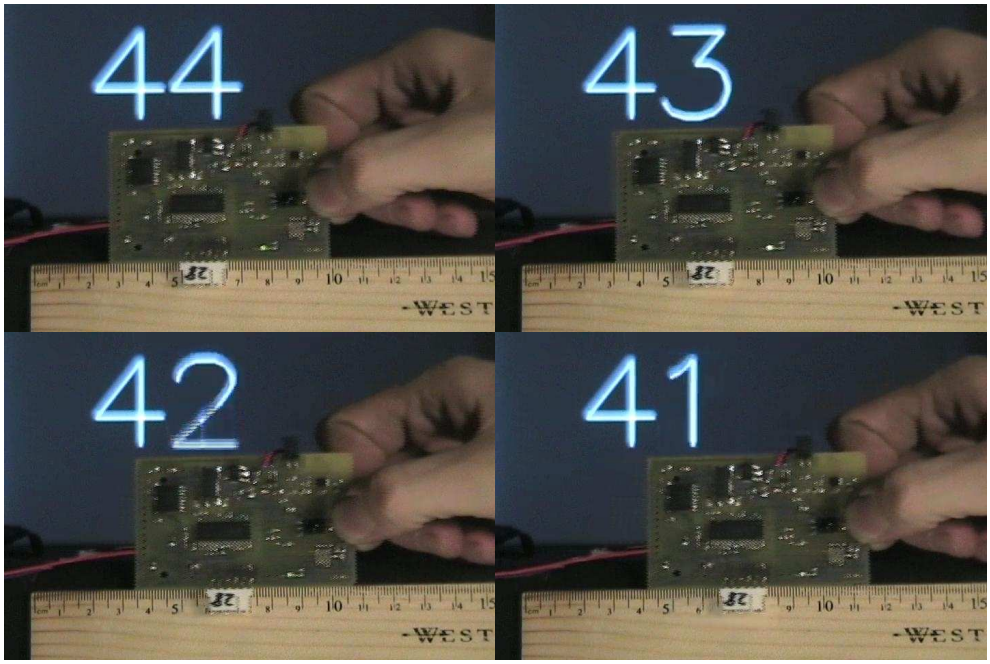


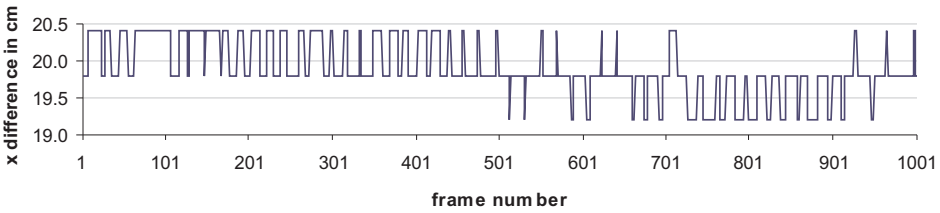
Figure 3.21: When moving a single marker in front of a measure, we show that in practice the accuracy of 2D the position tracking is as high as predicted in theory. In the top left corner of each frame, we show the x-coordinate that was calculated by our system.

Verification Experiments To verify the accuracy of the position tracking system, we performed three tests.

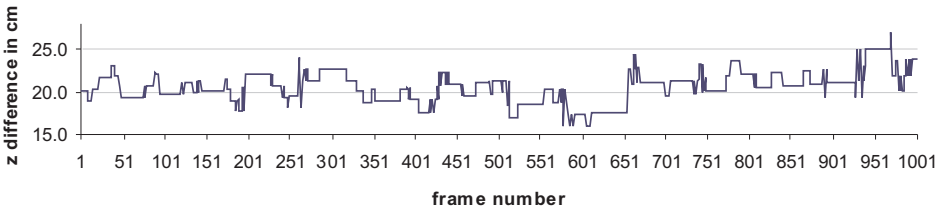
In the first test, we moved a marker along the x-axis in front of a measure as can be seen in figure 3.21. The purpose of this test is to verify whether the actual X-Y resolution of the system is the same as we predicted in theory. We positioned the marker 3m away from the projectors, so the resolution should in theory be 5.8 mm. In figure 3.21 we see the marker in front of the measure, and in each successive image, the marker is moved to the right over a distance of approximately 5.8 mm. From the figure, it is apparent that our system is indeed precise enough to measure these movements, so the 2D position tracking is as accurate as we predicted in theory.

The second test is also designed to test the accuracy of the system in the X-Y direction. The setup of the experiment is shown in figure 3.22 (a). It consists of two markers that are attached to the two ends of a stick, facing in the direction of the projectors parallel to each other. This way the two markers will not move with respect to

(a) A photo of the position verification experiment



(b) Distance measured by our system between two markers that are attached to a rigid body along the x-axis



(c) Distance measured by our system between two markers that are attached to a rigid body along the Z-axis

Figure 3.22: The results of the tests to verify the accuracy of the position tracking.

each other. In this experiment, we will wave this stick around in a plane parallel to the X-Y plane without rotating it at a distance of 3m from the projectors. The distance measured by our system between the two markers should remain nearly constant. The variation of the measured distance between the two markers is a measure of the accuracy of the system. In figure 3.22 (b) we show the results of this experiment. The average distance between the markers is 199.01 mm, the standard deviation is 3.70 mm and the maximum deviation is 7.01 mm. As in the previous test, this test also confirms that the actual accuracy of the 2D tracking system corresponds to the theoretical accuracy, which is, as said before, 5.8 mm at a distance of 3m using our 8-bit system.

The third experiment is very similar to the second one. But instead of the accuracy in the X-Y direction, we test the accuracy in the Z direction. As in the previous experiment, we once again attach two markers at the ends of a stick, but this time we hold the stick in the direction of the Z-axis. Now we move the stick back and forth and once again we look at the relative distance between the markers as calculated by our system. Figure 3.22 (c) shows the results of this experiment. The average distance between the markers is, 205.58 mm, the standard deviation is 18.33 mm and the maximum deviation is 64.29 mm.

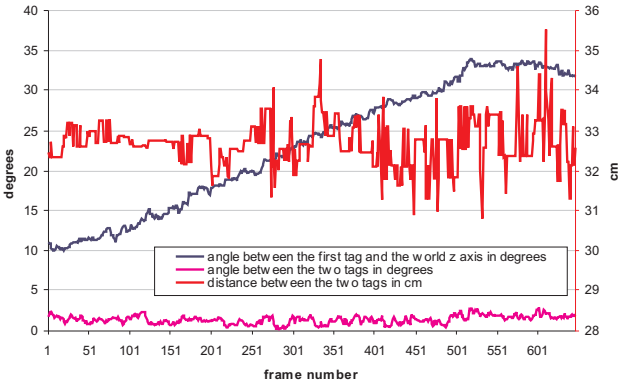
Orientation

For the orientation tracking, we have a number of infrared light sources (orientation beacons) that light up one after the other. These light sources are shown in figure 3.17. For each of these light sources, the marker measures the amount of incident illumination. The orientation of the marker can be calculated from these measurements using the fact that the amount of incident illumination depends on the angle between the sensor and the light source.

We assume the location and the brightness of each light source is known. We also assume the position of the marker is captured using the method described in the previous section. Each light source L_i is turned on one after the other, and for each light source, the sensor on the marker measures the amount of incident illumination I_i from this light source. We can estimate the normal N of the marker by solving the following set of equations:

$$I_i = k(V_i \cdot N)\frac{P_i}{d_i^2} \quad \text{for } i = 1 \text{ to \#light sources} \tag{3.1}$$

With k the gain of the sensor, d_i the distance between the marker and the i-th light source, P_i the power of the i-th light source and $V_i = [V_{ix}, V_{iy}, V_{iz}]$ the normalized



(a) (b)

Figure 3.23: A photo of and a graph with the results of the orientation verification experiment. It shows the angle between two markers that are attached parallel to each other on a rigid box that is being rotated.

vector from the marker to the i-th light source. At least three light sources are needed to obtain the normal N, but to increase the accuracy, more light sources are used in our setup.

Verification Experiments To verify the orientation tracking, we performed the experiment shown in figure 3.23. Image (a) shows the setup used to perform the experiment. Three markers are shown that are attached to a plank approximately parallel to each other. When the plank is tilted, the angle between the markers stays the same. The variation of the angle between two markers measured by our system is an indication of the accuracy of our system. In this experiment we used 8 orientation beacons, and the beacons were spread over 2.5m. The graph in figure 3.23 (b) shows the difference of the angle between two markers while the plank is rotated. As is visible in the graph, the box was rotated from 10 to 30 degrees and the angle between the two markers has a mean of 1.32 degrees with a standard deviation of 0.46 and a maximum variation of 1.49 degrees. So the accuracy of the orientation of the system is within 1 degree.

Incident Illumination

A sensor on the marker measures the incident visible light. It returns the color of the light that illuminates the marker. This information can be used in two ways. A first application is to color virtual objects as if they were lit by light present in the real scene. The information can also be used to calculate the reflectance of a surface in the scene. The irradience near the surface can be captured by placing a marker nearby and the light reflected by the surface can be captured using a camera. From this irradiance E_λ and radiance B_λ, the lambertian reflection ρ_λ can be calculated.

$$\rho_\lambda \approx \frac{B_\lambda}{E_\lambda} \approx \frac{CameraReading_\lambda}{\Gamma(TagReading_\lambda / dA)} \tag{3.2}$$

$\Gamma(.)$ represents the function that maps colors from the marker color space to the camera color space.

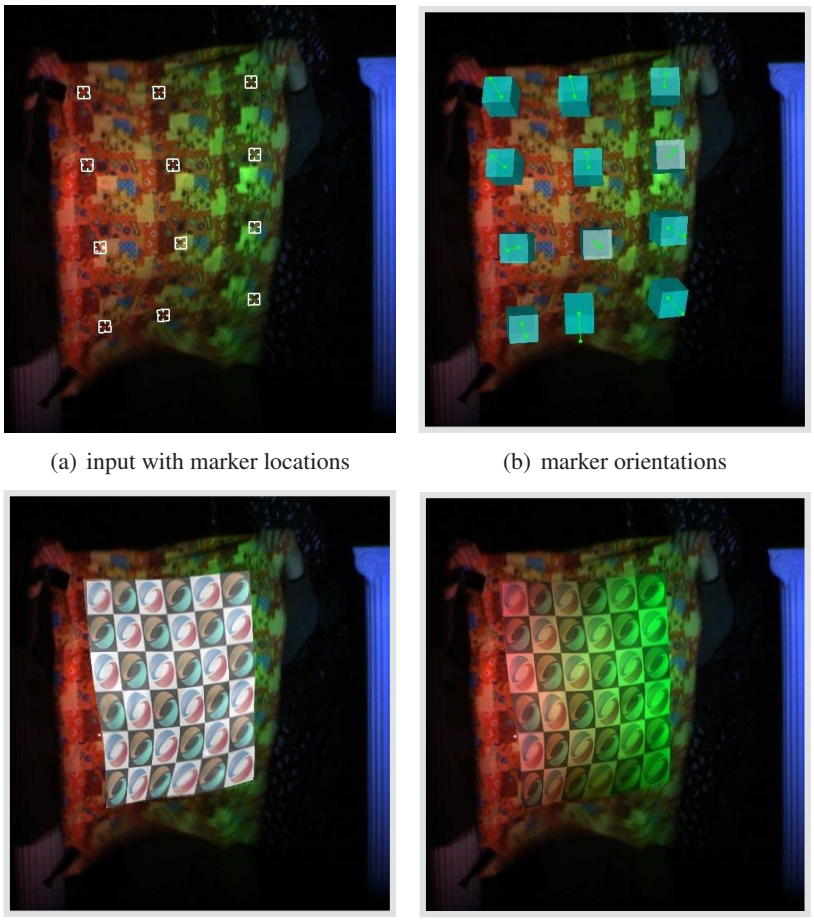
3.3.3 Applications

In this section we show a number of video manipulation techniques that use the motion capture system presented in this chapter.

Virtual Objects

In figure 3.24 we show an example where a virtual label is added to a piece of cloth. The scene consists of a person holding a piece of cloth illuminated by a video projector projecting a red-green-blue rainbow pattern. To track the cloth, twelve imperceptible markers are attached to it. We fit an openGL NURBS through the position and normal information of the markers as shown in image (c) of figure 3.24. But as we did not use the illumination information of the markers for this image, the cloth is lit in a different way than the rest of the scene. As shown in image (d), realism can be added by coloring the virtual label using the incident illumination captured by the markers. The illusion is created that the rainbow pattern projected onto the real scene also illuminates the virtual label.

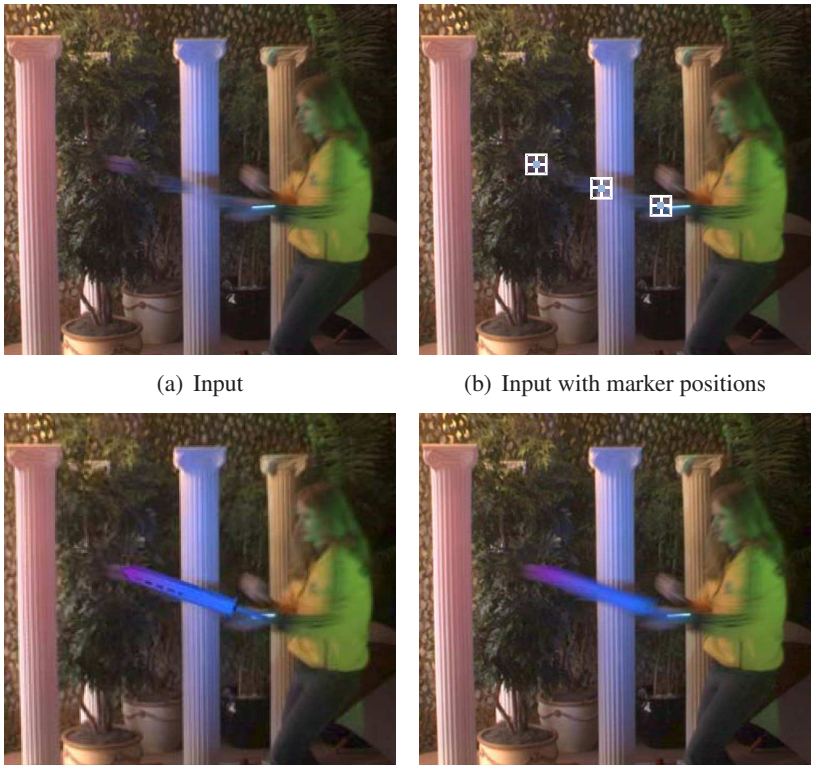
With camera based tracking systems, similar results can be obtained. But for such systems the markers will no longer be imperceptible. In applications where transparent labels have to be shown on top of the real objects, this might be intolerable.



(a) input with marker locations (b) marker orientations

(c) virtual label (d) virtual label with incident illumination

Figure 3.24: A virtual label is added to a piece of cloth using the position, orientation and incident illumination of twelve markers attached to a piece of cloth.



(a) Input (b) Input with marker positions

(c) With virtual sword (d) With motion blur

Figure 3.25: A stick, with three markers attached, is replaced by a virtual sword. The scene is lit by a rainbow pattern, and the virtual sword is lit accordingly. Because we have a high speed tracking system, realistic motion blur can be added.

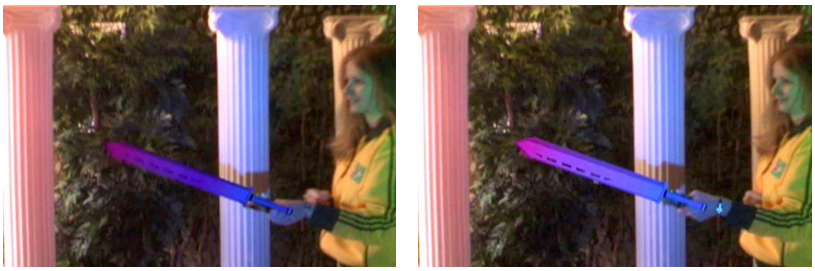
Motion Blur In figure 3.25 we show an example where a tracked real object is replaced by a virtual object. The scene consists of a person holding a plastic stick. The performance is filmed by a camera that is calibrated with respect to the tracking system. Three markers are attached to the stick in order to track it. Using the position and orientation information we obtain from the markers, we can render a virtual model of a sword on top of the stick in the input video. This example shows another application of capturing the incident illumination. The scene is lit by a LCD projector that projects a red-blue-green rainbow pattern. If we did not capture the incident illumination, the rendered sword would be uniformly lit, while the other objects in the video would be lit by the rainbow pattern, which reduces the realism of the final result. The markers attached to the stick capture the local lighting information, and by using this information while rendering the sword, we create the illusion that the sword was lit by the rainbow pattern. The top images of figure 3.25 show a frame of the input video. The white squares shown in image (b) indicate the position of each marker as calculated by our system. The bottom images shows the virtual sword composited with the input video.

The tracking system runs at a much higher frame rate than the input video. We can use this to add realistic motion blur effects to the final rendering result. Instead of rendering the sword once for each video frame, we render it once for each sample we obtain from the tracking system and blend the results together. This way we get correct motion blur.

Achieving this result using a pure video based approach is not very easy. Tracking the stick using a pure video based approach can be quite hard for a number of reasons. If the sword moves fast, a lot of motion blur is generated, which makes tracking it using video based approaches very difficult. The material of the stick the markers are attached to, is not important in our approach. Video based techniques will have trouble handling specular or transparent materials. Because the scene is lit by light sources with arbitrary colors, video based approaches can not easily use color information to track the stick. Estimating the motion blur using video based approaches is also not very straightforward.

Orientation Tracking

In figure 3.26 we show the strength of our single point orientation tracking method. Many tracking systems only obtain position information, orientation is derived from the position of multiple tracked points. In order to obtain an accurate estimate of the orientation of a surface, it has to be large enough to attach multiple markers to it and the further away the markers are from each other, the higher the accuracy. It is not



(a) (b)

Figure 3.26: A stick, with markers attached to it, is rotated. Our system is able to track this movement as is shown by rendering a virtual sword in place of the tracked stick. Motion tracking systems that calculate the orientation of surfaces from position information have problems to track this kind of motion of thin objects, because there is not enough space to place the required markers.

possible to obtain the orientation of a single point or of a very thin stick using such an approach.

In figure 3.26 we show an example where the orientation of a stick is required. It shows a person holding a stick that is being tracked. The position and orientation information of this stick is used to replace the stick by a virtual model of a sword. Using only position tracking, the subtle rotation of the sword would be hard to capture — the multiple markers needed to derive the orientation, would have to be placed very close to each other resulting in a low accuracy.

In our system however we can track the orientation of a single marker directly. We do not have to place multiple markers to obtain the orientation of a surface. This is why our system has no problems with tracking the motions and objects shown in figure 3.26.

Robust Tracking

In this section we will describe experiments to test the robustness of our system. We show it works in daylight and has no problem handling specular objects.

A first experiment is shown in figure 3.27. It shows a person who is being tracked outside in daylight. The person is wearing a jacket equipped with a number of markers. The position of the markers is tracked using a portable version of our 2D tracking system. The 2D system consists of two perpendicular 1D projectors and a camera to film the scene that is being tracked. A person in a car uses this hand held 2D system

(a) setup (b) tracking a running person

Figure 3.27: An experiment that shows our system is able to work in daylight conditions using imperceptible markers. A person who runs on a street is tracked from a car using a portable version of our tracking system.

to track a person wearing the jacket with the markers attached to it.

As shown in the top left image, the markers are hidden under clothing with only the small sensors exposed. This demonstrates the markers are imperceptible under clothing.

In contrast to camera or vision based tracking methods, our system is insensitive to specular objects in the scene. Figure 3.28 shows how three markers behind a sheet of tin foil are tracked without a problem. In this figure we also show that we are able to track markers behind materials that are opaque for visible light but transparent for infrared light such as coke.

3.3.4 Conclusions

In this section, we presented *Prakash*, a high speed tracking system that obtains the position, the orientation and the incident illumination for a number of points. Using a number of verification experiments, we analysed the accuracy and robustness of this system. We also demonstrated how this system can be used to manipulate footage and augment it with virtual objects.

Figure 3.28: We can track markers behind highly specular surfaces and behind materials that are opaque for visible light but transparent for infrared light such as coke.

3.4 Conclusions

In this chapter we presented two tracking systems and demonstrated their use in video manipulation tasks. The first system is a camera based position tracking system. It is a flexible system because its field of view and accuracy can easily be changed by using different cameras or lenses. The second system tracks position, orientation and incident illumination at high speeds. It is a little less flexible because changing its field of view is less straightforward, but there are no theoretical limitations to the field of view.

Both systems are low cost optical systems that are quite robust against ambient light and both use imperceptible electronic markers. Line of sight is needed in both systems and they are both able to uniquely identify each marker. In the camera based tracker, a PC calculates the position of each marker, so this information is easily broadcast to other devices using ethernet. In the *Prakash* system, the markers themselves calculate their position and orientation. If another device needs this information, it has to be transmitted by the markers using an RF channel which might cause problems due to limited bandwidth.

Because the markers calculate their own position and orientation in the *Prakash* system, it supports an unlimited number of markers. The camera based tracker is less scalable because the PC calculates the position of each marker, which means that if more markers are added, more computational power is needed.

In the camera based tracking system, we currently use cameras of 640x480 pixels,

so the accuracy of the position tracking in this system is about 9 bits. The position tracking in the *Prakash* system is 8 bit because we use 8 LEDs in the position projectors.

Prakash uses LEDs to project the patterns into the scene which results in very high frame rates, the position tracking system works at 1000Hz for 1D tracking and at 300Hz for the 3D tracking while the camera based tracking system only runs at 30Hz.

Chapter 4

Time and Color Multiplexed Illumination

Contents

Multiplexing light is important for many video manipulation techniques. By projecting patterns of light onto a scene, depth information can be acquired. Normals of surfaces in a scene can be obtained by turning a number of point light sources on and off. Relighting techniques also rely on multiplexing light. In this chapter we present

a new method to multiplex light and show this can be used for a wide range of video manipulation methods.

4.1 Introduction

Many vision and graphics problems such as relighting, structured light scanning and photometric stereo, need images of a scene under a number of different illumination conditions. It is typically assumed that the scene is static. To extend such methods to dynamic scenes, dense optical flow can be used to register adjacent frames. This registration becomes inaccurate if the frame rate is too low with respect to the degree of movement in the scenes. We present a general method that extends time multiplexing with color multiplexing in order to better handle dynamic scenes. Our method allows for packing more illumination information into a single frame, thereby reducing the number of required frames over which optical flow must be computed. Moreover, color-multiplexed frames lend themselves better to reliably computing optical flow. We show that our method produces better results compared to time-multiplexing alone. We demonstrate its application to relighting, structured light scanning and photometric stereo in dynamic scenes.

There is a wide variety of algorithms that need images of a scene under a number of different lighting conditions. Structured light scanning computes a depth map by illuminating the scene with different coded patterns [Scharstein 03]. Photometric stereo [Barsky 03, Woodham 80] obtains a per pixel normal of a scene by capturing the scene under a number of different point light source positions. In computer graphics, image-based relighting is performed by recording a scene under a number of basis illumination patterns [Debevec 00].

In all of these methods, each illumination condition requires a separate image, and it is assumed the scene remains static. Specific methods have been developed in order to deal with dynamic scenes [Zhang 02, Zhang 03, Davis 03, Hernandez 07, Wenger 05]. In this paper, we seek a general technique which is applicable to all methods requiring multiple illumination conditions.

An obvious way to extend the aforementioned methods to handle dynamic scenes, is to use a dense optical flow to register dynamic content between frames. Even for high frame rates, this registration is required. This registration becomes inaccurate if scene elements move too fast, or equivalently, if the frame rate is too low. Also, as more illumination conditions are needed, optical flow must be computed over a longer span of frames. The results will therefore be more prone to occlusion errors.

Figure 4.1: Demultiplexing illumination using our method. Top row: 9 input frames of a dynamic scene. A person is lit by only **2** different colored illumination patterns using 4 differently colored light sources (indicated by the squares). Bottom row: The output images show the demultiplexed lighting. I.e., each image contains the result of having illuminated the scene by each source separately.

In this paper, we propose to extend time multiplexing with color multiplexing. With color multiplexing we can encode more illumination conditions per frame, which reduces the number of required input images over which optical flow must be computed. This relaxes the requirement of having a sufficiently high frame rate, making it possible for using (cheaper) cameras with a lower frame rate. Estimating optical flow under varying illumination conditions is hard. Even though state-of-the art optical flow algorithms can deal with varying illumination, the aforementioned methods typically employ significantly different illumination patterns from frame to frame. This causes wildly varying intensities and even image features (e.g., due to moving shadow boundaries). An additional benefit of color-multiplexing is that optical flow can be computed more reliably, because all light sources are enabled in every frame and only their color varies.

Figure 4.1 shows a typical use of our light multiplexing approach in the context of relighting. In the top row, 9 frames of the input sequence are shown. They depict a dynamic scene consisting of a face and the scene is lit by four colored light sources.

The colors of these light sources change each frame and the four squares on top of each input frame depict the color of each of the four light sources for this frame. In the bottom row of Figure 4.1 there are four output images that show our algorithm is able to demultiplex the lighting. For each of the 4 light sources, there is an output image that shows the scene as lit by only this light source.

Our contributions are as follows:

- We demonstrate the use of both color and time multiplexing to capture a scene under a number of lighting conditions.
- We show how we can compensate for the motion in a dynamic scene even when each frame is lit differently.
- We demonstrate that our light multiplexing scheme is generic and can be used to improve many existing algorithms.

4.2 Related Work

There is a wide range of techniques that need images of a scene under varying lighting conditions. In this section we will give a short overview of these algorithms. Many of them only use time multiplexing to obtain the needed information. Using the algorithm described in this chapter, it becomes possible to alter these techniques so they use both time and color multiplexing.

4.2.1 Relighting

Model Based There are a number of model based relighting techniques that are specialized in relighting a subset of scenes, for example they can only relight faces. Peers et al. [Peers 07] capture a reference reflectance field of the face of an actor and they transfer it to a dynamic performance of someone else. Hawkins et al. [Hawkins 04] capture the face of a person for different facial expressions, head poses, view directions, and lighting conditions. From these images, they can render the face under new poses and new lighting conditions by warping, scaling, and blending data from the original images. Wen et al. [Wen 03] show a method to approximate the radiance environment map for any given image of a face using spherical harmonics. By changing the spherical harmonics coefficients they are able to change the lighting. In contrast to these methods, in this chapter we will focus on a technique to relight arbitrary scenes.

Lightstage Of course we are not the first to present a relighting method for arbitrary scenes. Debevec et al. [Debevec 00] show a system that is able to relight static scenes. They capture images of a scene under a large number of lighting conditions, and by blending the images, they can relight the scene. This system is extended to

Figure 4.2: A lighting pattern used by Wenger et al. [Wenger 05] to relight a human actor.

dynamic scenes by Wenger et al. [Wenger 05]. They use a high speed camera and LEDs to capture the scene under different lighting conditions at a very high frame rate. Their setup consists of a camera that is able to capture 2160 images a second and 156 LEDs positioned in a sphere around a person. LEDs are used as light sources for their ability to switch on and off almost instantly. Different LEDS are turned on for each frame, this way they obtain the scene for a large number of lighting conditions. If they assumed the scene was static, they could relight simply by blending the different input images. But since they consider dynamic scenes, they have to compensate for the motion. To do this, a tracking frame is added to their lighting pattern 120 times a second. This tracking frame is an image of the scene lit under uniform lighting. The optical flow is calculated between these tracking frames. Using this optical flow, the displacements between consecutive frames is known and the motion can be compensated for. They also added matting frames to allow for easy foreground-background subtraction. For these frames a backdrop is lit, but the actor is kept in the dark. Figure 4.2 shows an actor lit by one sequence of the lighting pattern. The tracking and matting frames are shown in the two left columns.

Another extension of this idea is presented by Wang et al. [Wang 08]. They relight a person that is lit by ambient light and time multiplexed IR light patterns. They show how to transfer the information from the IR domain to the visible domain. Until now, we described techniques that only use time multiplexing to obtain the scene under a number of lighting conditions. In this chapter we present a technique to use both time and color multiplexing to obtain this information.

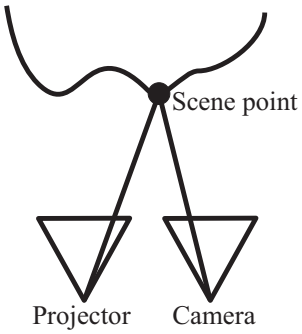


Figure 4.3: If correspondences between the projector and the camera are established using the structured light approach, the 3D position of the corresponding scene point can be calculated.

White balancing Hsu et al. [Hsu 08] show they can relight a scene from a single image using their white balancing technique. They are able to change the colors of the light sources in the scene under the assumption that there are only two light sources present in the scene. Our method can work with any number of light sources.

4.2.2 Light Patterns

Depth from Structured Light Depth from structured light [Scharstein 03, Sato 85] is an example of a method where images of a scene are needed under a number of lighting conditions. It is a method where binary patterns are projected onto a scene and from images of the scene, lit by these patterns, the depth of the scene can be obtained.

In its simplest form it works as follows. An LCD projector and a camera are calibrated with respect to each other and are aimed at a scene. The scene is completely dark, the only light source present is the LCD projector. The LCD projector turns on each of its pixels one at a time, this way a single point in the scene is illuminated. When the camera takes an image of the scene, only a single pixel of this image will be illuminated. This way, a stereo correspondence between the projector and the camera is found and the 3D position of the illuminated scene point can be obtained as shown in figure 4.3.

This is a robust way to obtain a depthmap of a scene, but it is also very tedious since the scene is scanned, one pixel at a time. Fortunately the speed can be increased using a simple extension to this approach. Instead of turning on only one pixel of the projector at a time, the patterns shown in figure 4.4 are projected onto the scene.

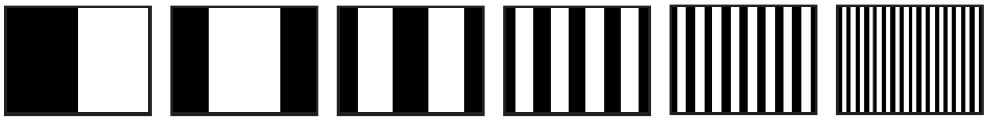
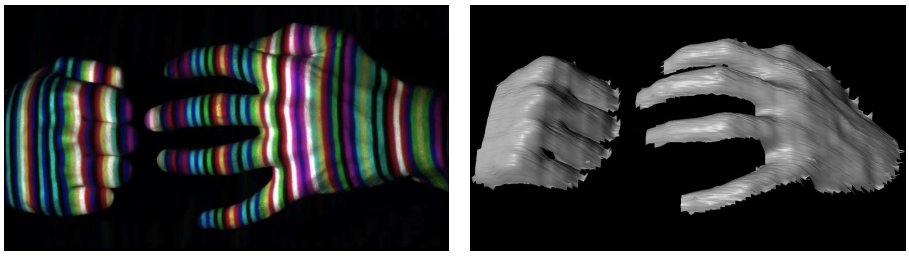


Figure 4.4: When these patterns are projected onto a scene one after the other, each point in the scene is labeled with a unique code that corresponds to a given column in the projector. This can be used to establish camera-projector correspondences.



(a) Single input image (b) Resulting 3D model

Figure 4.5: Zhang et al. [Zhang 02] is able to acquire the 3D geometry of a scene from a single image using structured light patterns.

When these patterns are projected onto a scene one after the other, each point in the scene is labeled with a unique code that corresponds to a given column in the projector. This can be used to establish camera-projector correspondences. From these correspondence, we can calculate the 3D shape of the scene using triangularization.

This approach uses only time multiplexing to project the black-white patterns onto the scene. Using our approach, we can extend this method so it uses both time and color multiplexing. We are not the first to use colored light in the context of depth from structured light. Caspi et al. [Caspi 98] project a number of colored stripe patterns onto a static scene, and for each pattern an image is taken. This way each pixel is given a unique label over time and from this a depthmap can be calculated. They also need images of the scene under ambient illumination and uniform white illumination to decode the patterns. The system calculates the colors of the stripes in such a way that the resulting depthmap has as much accuracy for as little input images and projected patterns as possible, taking into account the noise of the projector and the camera. But since they need multiple images of the scene, they can not handle dynamic scenes.

The method presented by Zhang et al. [Zhang 02] can calculate the depth of a scene from a single image. They also project one or more colored stripe patterns onto the

scene, but they use the color transitions between stripes to label the space. This way they can encode a lot more depth information in one single image. This allows them to generate a good depth map from a single image as shown in figure 4.5. The R, G and B patterns were generated using de Bruijn sequences. A k-ary de Bruijn pattern is a pattern $d_0, d_1, ..., d_{k^n-1}$ for which each n consecutive elements appear only once. In this case, k was 7 and n was 3, so each three consecutive color transitions in the lighting pattern is unique. The correspondences are calculated using a multi-pass dynamic programming algorithm.

Lee et al. [Lee 07] proposed a method to address some problems that arise when using colored stripes to obtain a depthmap from a scene. The first problem they address is that the color of the objects in the scene can interfere with the color of the stripes. Especially high frequency components of the texture of an object can have a harmful effect on the final result. Lee et al. [Lee 07] solve this by using a log-gradient filter to diminish the effect of the surface color. Artefacts can occur near depth discontinuities when using color transitions to label space since the pattern of consecutive colors is broken by the discontinuity. This is solved by using multiple colored stripe patterns with different orientations. They present a method to combine multiple patterns in one image and they are able to restore the different patterns afterwards.

Chen et al. [Chen 97] use colored structured light patterns to diminish the ambiguity when calculating depth from stereo. They do not calculate correspondences between camera image and the projected pattern as the other methods discussed here do. Instead they use two or more cameras to capture the scene, and correspondences between these cameras are calculated. The sole purpose of the projected pattern is to diminish the ambiguity and to ease the calculation of the correspondences between the camera images.

Note that these methods that can calculate the geometry of a dynamic scene using structured light patterns are not able to estimate the surface color at the same time as the geometry.

Photometric Stereo Another method that can be improved using the method presented in this chapter is Photometric Stereo [Barsky 03, Woodham 89]. It is a technique that is able to calculate a dense normal map of a scene. It works by taking a number of pictures of a static scene that is lit by a single point light source. The only thing that differs between these pictures is the position of the point light source. If the camera and the point light sources are calibrated, a normal can be calculated for each pixel independently, see figure 4.6. The surface normal for a single pixel can be calculated by solving the following system of equations:

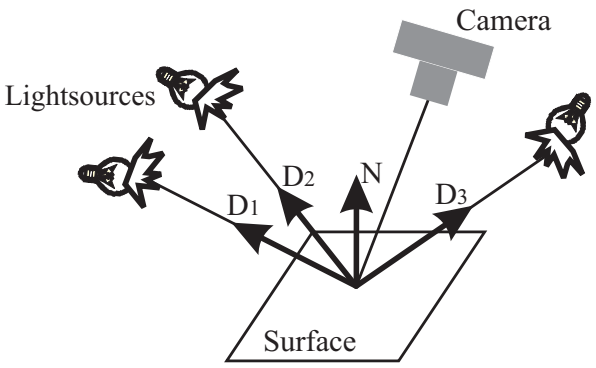


Figure 4.6: The normal N of a surface point can be calculated if it is lit by a number of point light sources and captured by a camera. Given the amount of the light from the point light sources that is reflected off the surface towards the camera, the normal N can be calculated if the positions of the camera and the light sources are known.

$$I = \rho L_i (D_i \cdot N) \tag{4.1}$$

with I the intensity captured by the camera, ρ the albedo of the material, D_i the direction of the i-th light source, L_i the intensity of the i-th light source and N the surface normal.

Since multiple images are needed to perform photometric stereo, it is not straightforward to apply it to dynamic surfaces. Hernandez et al. [Hernandez 07] present a method that performs photometric stereo using colored lights (see figure 4.7). By using a red, green and blue light source, they can multiplex the lighting of three light sources using color multiplexing. So they can perform photometric stereo with three light sources using a single frame and are able to handle dynamic surfaces. The drawback of their approach is that they assume the material of the surface is known. They can not handle arbitrary scenes, with multiple surfaces with different materials. We will show our method is able to perform photometric stereo on dynamic scenes while we do not constrain the number of materials present in the scene.

4.2.3 General Light Multiplexing

Schechner et al. [Schechner 07, Schechner 03] present a general technique to multiplex light without the use of color multiplexing. The lights are turned on/off using Hadamard-based patterns. For each frame they turn on multiple lights at once.

(a) Setup (b) Input image (c) Result

Figure 4.7: Hernandez et al. [Hernandez 07] present a method to use photometric stereo with colored light sources to estimate a normal map of a dynamic surface.

This way they can capture low intensity lighting, without the need for long exposure times. Also the technique of Wenger et al. [Wenger 05] can be seen as a general light multiplexing technique. They compare multiple light bases, among which the Hadamard-based patterns.

4.2.4 Discussion

While there are specific extensions to some of the basic structured light and photometric stereo algorithms that use both color and time multiplexed light [Woodham 80, Lee 07, Chen 97], these extensions are not as general and have their own advantages and disadvantages. Existing photometric stereo algorithms [Woodham 80, Hernandez 07, Hernandez 08] suppose the scene is static or that all the objects in the scene have the same albedo. Our method on the other hand is able to handle dynamic scenes without constraining the number of albedos. When our method is applied to depth from structured light, we obtain the depth map and the scene colors simultaneously. Existing depth from structured light algorithms for dynamic scenes are unable to do this, they only obtain a depth map [Caspi 98, Zhang 02, Hall-Holt 01, Lee 07, Chen 97].

4.3 Time and Color Multiplexing

In this section, we present our light multiplexing algorithm for static scenes. The extension to dynamic scenes will be described in section 4.5.

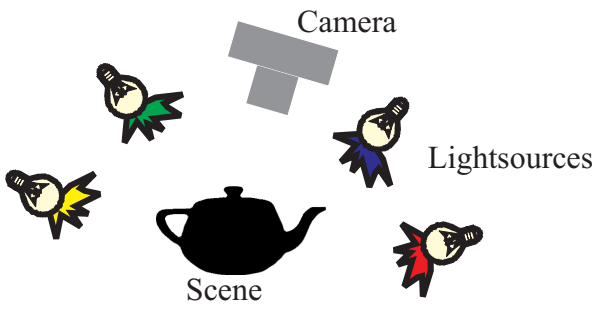


Figure 4.8: Our setup consists of the scene to be captured, filmed by one camera and lit by a number of light sources with changing colors.

4.3.1 Model

The aforementioned applications all require that an object be captured under varying illumination. We address the common scenario in which illumination is varied by changing the color of otherwise fixed light sources (Figure 4.8).

We make the following assumptions:

- Only the color of the light sources varies from captured frame to frame. Position and directional emission distribution remain constant during the capture, but are otherwise free to choose. In particular, the light sources are not assumed to be diffuse, neither need they be point sources;
- Surfaces are Lambertian;
- Indirect illumination (reflections, refractions and color bleeding effects) is negligible;
- Negligible motion blur;
- Object motion relative to light sources affects only reflected light intensity, not its color.

Since we only deal with static scenes in this section, for now we also assume the position of the camera and the objects is fixed. In the next section we will show how we can get rid of this extra constraint.

It will be shown in the results section 4.6, that our time and color multiplexing scheme still yields acceptable results, even if these assumptions are violated to some (practical) extent.

If only a single light source illuminates the object, the RGB color $\mathbf{C}^i = (C^i_r, C^i_g, C^i_b)$ of a pixel in any captured image $i, i = 0 \ldots n$ with n the number of illumination scenarios, will be the product

$$\mathbf{C}^i = \mathbf{A} \odot \mathbf{L}^i I$$

of three factors:

- The RGB color $\mathbf{A} = (A_r, A_g, A_b)$ of the object seen in the pixel. $\mathbf{A}$ is independent of illumination;
- The RGB light source color $\mathbf{L}^i = (L^i_r, L^i_g, L^i_b)$ in frame i;
- A scalar form factor I that indicates how intensely the object at the pixel would be lit by a unit strength monochrome light source. I depends on object and light source relative position, orientation and visibility. Since neither move (in this section), it is constant. (It is allowed to vary in the next section.)

The symbol $\odot$ stands for component-wise multiplication of RGB color vectors. For clarity, we do not index pixels: We use the same symbol $\mathbf{C}$ for the color of any single pixel and for a complete RGB image. We use the same symbol I for the form factor of a single pixel, and for the set of form factors corresponding to all pixels in an image. We will refer to the latter as "intensity images" in the sequel.

Since only the product counts, any of these factors can be scaled to wish, provided the scaling is compensated in the other factors. In our multiplexing scheme, we normalize material colors $\mathbf{A}$ such that the sum of square components is unity. Our choice of Euclidean metric is convenient, but otherwise arbitrary: any normalization metric would do. Compensation is in the intensity value I.

When m light sources $j, j = 0 \ldots m$, with colors $\mathbf{L}^i_j$ are used, pixel colors $\mathbf{C}^i$ will satisfy the following matrix equation:

$$\mathbf{C} = \mathbf{M}\,\mathbf{I}, \text{ with} \tag{4.2}$$

$$\begin{bmatrix} C^1_r \\ C^1_g \\ C^1_b \\ C^2_r \\ C^2_g \\ \vdots \\ C^n_b \end{bmatrix} = \begin{bmatrix} A_r L^1_{1r} & A_r L^1_{2r} & \ldots A_r L^1_{mr} \\ A_g L^1_{1g} & A_g L^1_{2g} & \ldots A_g L^1_{mg} \\ A_b L^1_{1b} & A_b L^1_{2b} & \ldots A_b L^1_{mb} \\ A_r L^2_{1r} & A_r L^2_{2r} & \ldots A_r L^2_{mr} \\ A_g L^2_{1g} & A_g L^2_{2g} & \ldots A_g L^2_{mg} \\ \vdots & \vdots & \vdots \\ A_b L^n_{1b} & A_b L^n_{2b} & \ldots A_b L^n_{mb} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_m \end{bmatrix}$$

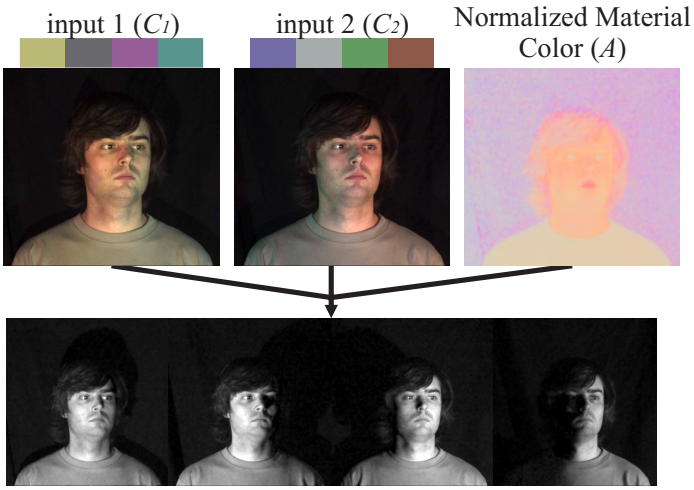


Figure 4.9: Given images $\mathbf{C}^1$ and $\mathbf{C}^2$ of an object captured under varying illumination, and assuming the normalized material color image $\mathbf{A}$ is known, our algorithm calculates intensity images I_j for each light source $j = 1\ldots4$. The colored squares on top of the input images indicate the (user chosen) color of the (otherwise fixed) light sources during capture.

4.3.2 Algorithm outline

Capturing an object under n illumination scenarios $i, i = 1\ldots n$, realized with m light sources $j, j = 1\ldots m$, as explained above, comes down to measuring $\mathbf{C}^i$ as a function of $\mathbf{L}^i_j$. If the normalized material colors $\mathbf{A}$ are known, and provided the rank of $\mathbf{M}$ is greater or equal than the number of light sources m, the intensity image I_j of each light source j can be computed by solving the linear system of equations (4.2). Since this system is in general over-determined, least squares techniques are used. The normalized material colors are usually not a priori known, however. We explain in 4.3.3 how to obtain them.

Example Application: Relighting Given the normalized material colors $\mathbf{A}$ and the intensity images I_j for all light sources j, a relighted image C' corresponding to freely chosen light source colors $\mathbf{L}'_j$ can be obtained directly from (4.2) as

$$\mathbf{C}' = \sum_{j=1}^{m} \mathbf{A} \odot \mathbf{L}'_j I_j. \tag{4.3}$$

Selection of Light Source Colors The light source colors $\mathbf{L}$ are chosen using grid

search techniques, as to ensure good conditioning of the system (4.2). The ideal light source colors depend on the normalized material colors **A**. For dynamic scenes, the normalized material color images vary from frame to frame, as will be explained in section 4.5. Non-optimal, but still good, choices for the light colors are obtained by assuming that all material colors are white.

4.3.3 Acquisition of Normalized Material Colors

The normalized material color image **A** corresponds to the normalized RGB image that would be captured if all light sources are set to unit strength white color. Colors are normalized by scaling them such that the sum of squared components is unity in our implementation, but any other normalization metric would do as well. We now present two ways to obtain **A**.

Dedicated Image Method One way to obtain the normalized material colors, is to actually capture, and normalize, an image **W** with all light source colors set to full intensity white. The disadvantage of this approach is the need to capture an extra image only to calculate **A**.

Given $m = 3$ coloured light sources, a single image ($n = 1$) yields **M** as a 3×3 matrix, and it is trivial to see that it is possible to choose the colors of our light sources in such a way that this matrix becomes rank 3 as needed to solve (4.2). A second image is needed in order to obtain **A**.

Complementary Colors Method Another way of calculating the normalized material color is by choosing the colors of each light source for each frame in such a way that these colors add up to white. So for example if we only take 2 images, the color of one light source for the first frame is the complement of the color of this light source for the second frame. Or in general $\forall j : \sum_i (L^i_{jr}, L^i_{jg}, L^i_{jb}) = (1,1,1)$ Normalizing the average of these images also yields **A**.

If we have n frames, for which the colors of each light source add up to white, one would expect that it is possible to demultiplex $3n$ light sources since **M** has $3n$ rows, but as shown below, in this case it is only possible to demultiplex $(3n-2)$ light sources. This is because we constrained the colors of our light sources to add up to white, and **M** is not of full rank. The advantage of this method compared to the *Dedicated Image Method* is that we can demultiplex one extra light source for the same number of images, but this is at the cost of some extra noise.

In the case where we take 2 images, 6 light sources and take no extra image to calculate the normalized material color, so the colors of one light source for each frame add up to white, **M** in equation (4.2) is a 6×6-matrix but the rank of **M** is only 4, so it is

impossible to calculate the light intensity image of 6 light sources, only 4. The rank is only 4 instead of 6 for the following reason. For simplicity assume that $A = (1,1,1)$. Define $V1$ the first row of $\mathbf{M}$, $V2$ the second row and $V3$ the third row. Because the colors of the first frame are the complement of the colors of the second frame, we have that the fourth row is $V4 = k - V1$, with k some number. The fifth row is $V5 = k - V2$, and the sixth row is $V6 = k - V3$. Now we have that the fifth row is a linear combination of the other rows: $V5 = k - V2 = (k - V1) + (V1) - (V2) = V4 + V1 - V2$. And in the same way is $V6$ also a linear combination of the first 4 rows, so the matrix $\mathbf{M}$ is rank 4.

In general, if we have n frames, $\mathbf{M}$ is rank $(3 * m - 2)$. If we have m light sources in our scene, we need to take $(m+2)/3$ images.

For example, if one needs to demultiplex seven light sources, there are two options. If one uses the *Dedicated Image Method*, we need to take 1 dedicated image to obtain the normalized material color and 3 images for estimating the light intensities of the seven light sources. If the *Complementary Colors Method* is used we only need $(m+2)/3 = (7+2)/3 = 3$ images. In that case the colors of the seven different light sources have to be chosen in such a way that the color of one light source adds up to white for all three frames. In this case it is also possible to combine the *Dedicated Image Method* and the *Complementary Colors Method*. You can choose the colors in such a way that the colors of one light source add up to white for the first two images. This way the *Complementary Colors Method* is used for the first two frames. From the first two images, we now know what the scene looks like under white illumination, and now we can freely choose the colors for last frame. Which of these techniques is the best suited, depends on the application. The results of these methods will be the same, but they are not equally robust to noise and it is future work to determine which method generates the least amount of noise.

Dealing with Sensor Noise

By using color multiplexing, each pixel of our input images contains more information than when using only time multiplexing. So our algorithm trades color depth for fewer input frames. When using a normal digital video camera, the input images are only low dynamic range images and also contain some noise. If we used our input images directly, the noise in the input images would be amplified, which would result in noisy output images as shown in Figure 4.10 (a).

In order to reduce noise, we can blur the input images when calculating the intensity images $\mathbf{I}$; or in other words, we trade resolution for dynamic range. The disadvantage

(a) Without noise reduction (b) Post-process bilateral blur (c) Our noise reduction method

Figure 4.10: Noise reduction is needed to generate high quality results. Blurring the noisy result during a post processing step generates less good results than our noise reduction method.

of this approach is that we remove the high frequency components of the image. However, it is possible to add these high frequency components back to the final result as follows:

$$I_{\text{res}} \quad = \quad I_{\text{blur}}\, W/W_{\text{blur}}, \tag{4.4}$$

where I_{blur} is the blurred light intensity image using a usual Gaussian blur kernel, I_{res} is the resulting light intensity image with the high frequencies put back in, W is the lightness of the RGB image $\mathbf{W}$ of the object under white light, introduced in the previous section. W_{blur} is W blurred with the same kernel as I_{blur}. The main approximation here, is that the high frequencies of W, which contains little noise as it is fully lit (W it is derived from $\mathbf{W}$), should also be present in the resulting intensity image I_{res}. Multiplication with W/W_{blur} puts some of these frequencies back into I_{blur}.

This way at least the high frequency components of the texture of the input images are preserved, even though the high frequency components of the lighting are lost. As a consequence we cannot reproduce very hard shadows. We demonstrate in the results section that this effect is not very significant.

4.4 Automatic Calculation of Light Sources Colors

The algorithm presented in the previous section assumes that the color of each light source is known. As discussed previously, the best results are obtained if the colors of the light sources can be chosen according to a number of constraints. In this section we show that if we are not able to manually set the color of each light source to the exact color we want and we are unable to measure the color of the light sources during

a preprocessing step, we can calculate the color of the light sources from the input images if we have at least one light source less than the strict maximum as derived in section 4.3. Or in other words if rank(M) is larger than the number of light sources.

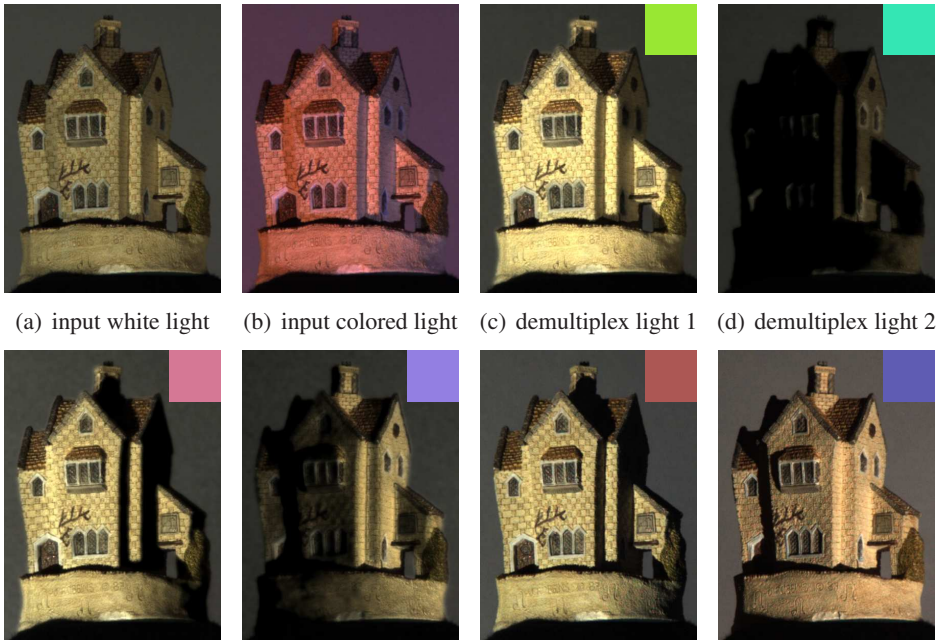
From equation (4.2) follows that if the normalized material color A and the input images C are known, and the rank of M is one less that the number of light sources, we can calculate the color of the light sources and the intensity images, by solving this non-linear set of equations for at least $3 * n$ pixels at the same time, with n the number of light sources. We use the interior-reflective Newton method [Coleman 94] for this purpose and use 1000 randomly selected pixels of the image. This non-linear system of equations is well defined if the chosen pixels have different normalized material colors and different colors in the input images. It is not guaranteed that the global optimum of the problem is obtained. We need an initial estimate of the colors of the light sources, and if this estimate is not chosen correctly, the solver will return a local optimum instead of the solution we are looking for. But as we will show next, the initial guess can be very crude while still converging to a good solution.

In figure 4.11 we show an example where we calculate both the intensity images and the color of the light sources. The scene consists of a house lit by two light sources. Image *a* shows the first input image, for this image the color of each light source was white. Image *b* shows the second input image, for this image the color of each light source was unknown. The top right corners of images *c* and *d* show the initial user defined guess of the colors of the two light sources we use to start the non-linear solver. Images *c* and *d* show the result of our demultiplexing algorithm when using these bad initial color estimates directly. Images *e* and *f* show the result of our algorithm after automatically calculating the colors of the light sources. The colors of the light sources estimated by our algorithm are show in the top right corners of images *e* and *f*. Images *g* and *h* show the ground truth and the real colors of the light sources. As we can see, even though the initial color estimates were not good, the solver can calculate the color of the light sources quite well. However, note that the intensity is not completely correct. According to equation (4.2), this is due to the fact that we can multiply M with any value, as long as we divide I by this same value. So we can calculate M up to a scale factor.

4.5 Dynamic Scenes

So far we have shown how time and color multiplexing works in the case of a static scene. In this section, we show how we can extend this method to dynamic scenes.

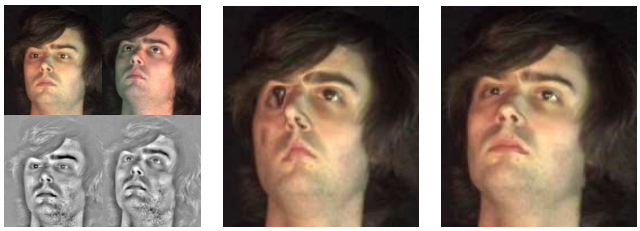
In dynamic scenes, the objects in the scene will have moved in two consecutive im-



(a) input white light (b) input colored light (c) demultiplex light 1 (d) demultiplex light 2

(e) demultiplex light 1 (f) demultiplex light 2 (g) ground truth light 1 (h) ground truth light 2

Figure 4.11: a,b: input images of a scene lit by two light sources with unknown colors. c,d: result of our algorithm when the wrong colors are used, the square in the top right corner represents the color of the light source used to demultiplex the image. e,f: result of our algorithm after the colors of the light sources were calculated automatically. g,h: ground truth.



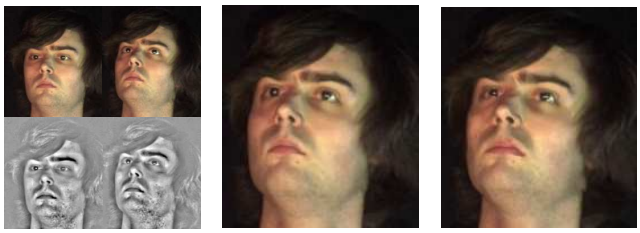
(a) (b) (c)

Figure 4.12: Calculating the optical flow *with* and *without* removing lighting from the image. (a) Input images and filtered input images. (b) Result when calculating the optical flow from the input images directly. (c) Result when calculating the optical flow from the filtered input images. When comparing the left eye and the nose in images b and c, it is apparent that calculating the optical flow from the input images directly does not work very well.

ages and we cannot directly apply our proposed algorithm. To compensate for this motion, we calculate the optical flow between the images and warp the images to a reference image, for which we use the optical flow algorithm by Brox et al. [Brox 04] with improvements by Sand et al. [Sand 06].

Since the color of the light sources are not the same for the each image, naïvely applying optical flow on the images does not work very well as shown in Figure 4.12. Instead we first apply a filter to the images that removes the lighting, but keeps the texture [Sand 04]. For each pixel we calculate the local brightness and the local contrast of a small neighborhood, and the filtered pixel value is the original pixel value subtracted by the local brightness and divided by the local contrast. We perform the optical flow on these filtered images. We also choose the color of the light sources in such a way that no dark shadows are created in any of the color channels. By doing this, all the texture information of the scene is visible in every image, and nothing gets fully obscured by shadows. As can be seen in Figure 4.13, the use of the filter does not degrade the quality of the optical flow, even when we calculate the optical flow between two images that are lit in the same way.

If we want to demultiplex the light for a given frame, we warp the required number of nearby frames to this frame. This way, we can assume we have a static scene for the other steps of the algorithm.



(a) (b) (c)

Figure 4.13: Calculating the optical flow with and without filtering when the scene is lit identically for two consecutive frames. (a) Input images and filtered input images. (b) Result when calculating the optical flow from the input images directly. (c) Result when calculating the optical flow from the filtered input images. The quality of the optical flow does not degrade when we use the filtered input images instead of the unfiltered input images.

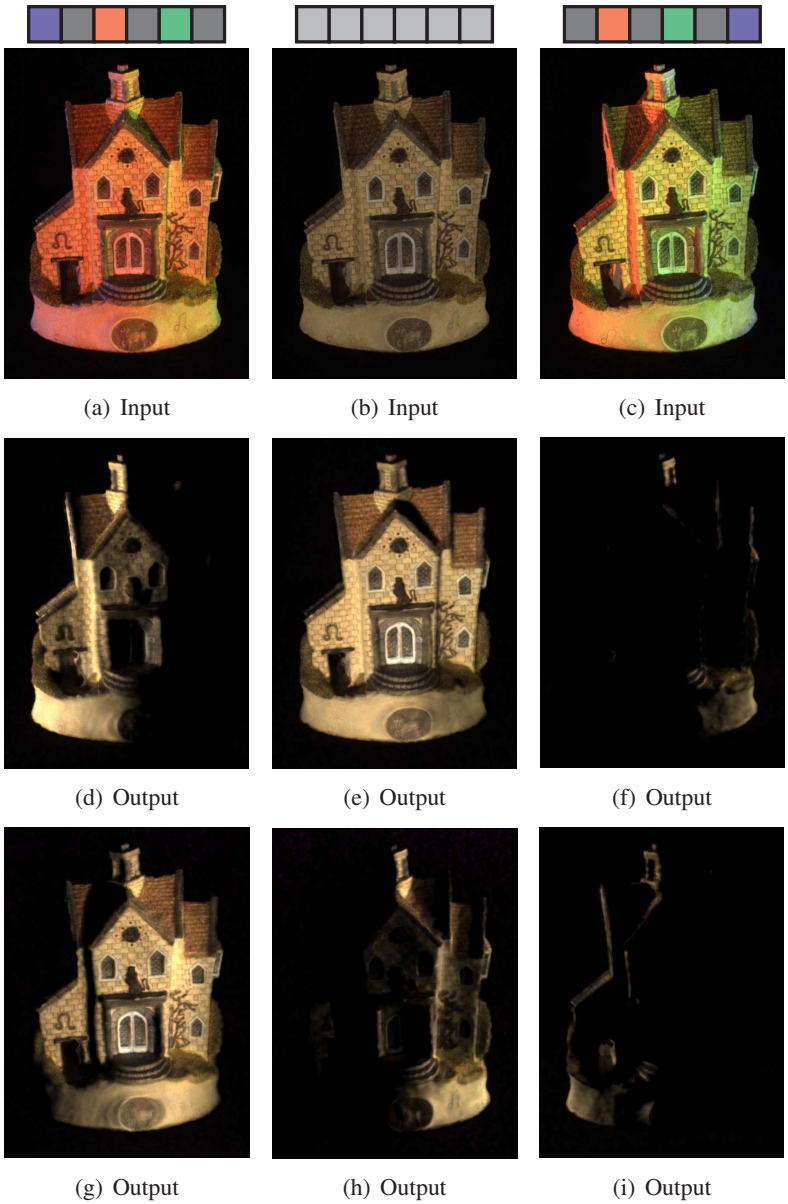
4.6 Results

In our experimental setup we use projectors as light sources as they are easy to control and it is possible to switch their colors rapidly. In a practical setup, using LEDs as light sources might be a better option, since they are cheap and can switch on and off rapidly. PointGrey Flea and Grasshopper cameras have been used to capture video sequences at 20 fps.

Computation times are in the order of 25 minutes for the calculation of the optical flow between two images and approximately 5 minutes for the rest of the calculations using our Matlab implementation on a 3 GHz CPU.

4.6.1 Relighting

We first apply our method to relighting, see Figure 4.14. The scene consists of a house on a rotating platform and it is lit by six light sources. The house rotates 5 degrees between two consecutive images, or in other words, it completes one entire turn in 72 images. Images of 1024x768 pixels were used in this example. We used stop motion to capture this sequence in order to capture additional ground truth images and to compare our results with those of Wenger at al. [Wenger 05]. From this large set of captured frames, we generate image sequences that are given to the algorithms we are comparing. To these algorithms, the input looks as if it was captured in real time using a video camera. Images (a)-(c) in Figure 4.14 show the input images for our



(a) Input (b) Input (c) Input

(d) Output (e) Output (f) Output

(g) Output (h) Output (i) Output

Figure 4.14: Top row: input images of a dynamic (rotating) scene lit by six light sources with changing colors, the squares on top of each image depict the colors of each of the six light sources for this frame. Two bottom rows: the output images show the scene lit under new lighting conditions, each image shows the scene as lit by only one of the six light sources.

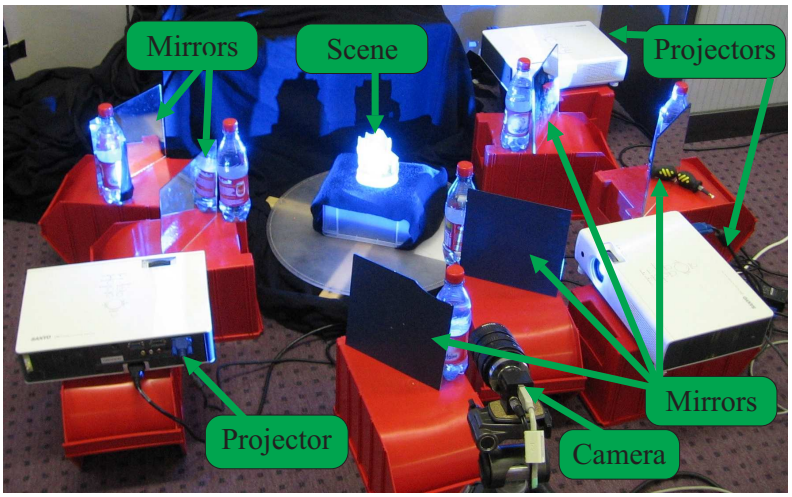
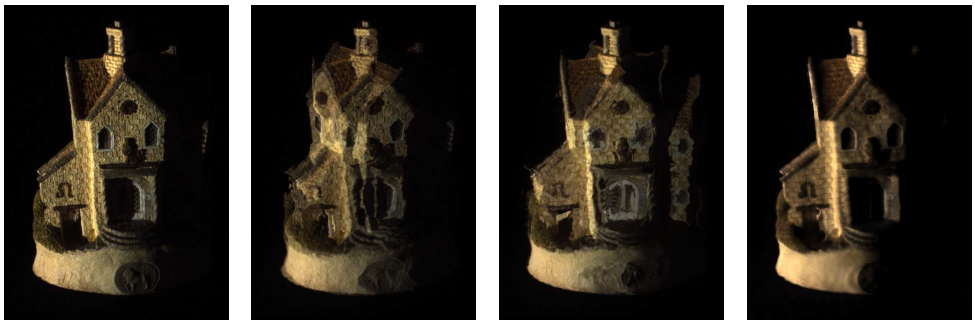


Figure 4.15: The setup used to capture the house example.



(a) Reference (b) Wenger et al. (c) Wenger et al. (d) Our method

Figure 4.16: From left to right: ground truth, output of Wenger et al. with a tracking frame every 7 frames, output of Wenger et al. with a tracking frame every 3 frames, our result.

(a) Optical flow for Wenger at al.

(b) Optical flow for our algorithm

Figure 4.17: The top row shows the input for Wenger et al. The bottom row shows the input for our algorithm. The solid arrows show the images between which the optical flow has to be calculated. The dashed arrows show the distance between two frames with identical lighting conditions.

algorithm. The squares on top of each input image depict the colors of each of the six light sources for the frame. Images (d)-(i) show the output of our algorithm, the scene is shown under a number of new lighting conditions. In this example we used the *Dedicated Image Method* to obtain the normalized material color A, for which we use image (b). To demultiplex the illumination, we run our algorithm twice. First we demultiplex the three light sources that were not grey in image (a). Image (a) is influenced by all 6 light sources, the 3 colored ones and the 3 grey ones. We remove the influence of the three grey light sources by subtracting a scaled version of image (b) from image (a). The resulting image now only depends on the three colored light sources we are trying to demultiplex. Using this image and the normalized material color A we are now able to demultiplex the three light sources. Since we need to solve for 3 light sources, $\mathbf{M}$ is a full rank 3×3 matrix, so equation (4.2) is easy to solve. To demultiplex the other three light sources, the ones that are not grey in image (c), we use a similar approach. We also could have demultiplexed the six light sources at once, but those results would be less robust against errors in the optical flow. To relight the scene, we use all six light sources. In the accompanying video, some small artifacts are visible, this is because of occlusion problems when calculating the optical flow.

Figure 4.15 shows the setup that was used to obtain the example shown in figure 4.14. The scene consists of a house on a rotating platform. There are markings on the rotating platform to make it possible to rotate the platform exactly 5 degrees between two images. We used three LCD projectors and six mirrors to simulate six light sources. In front of each LCD projector, two mirrors are placed to simulate two different light sources. This way, using three projectors with two mirrors in front of each, we can simulate six light sources. The camera at the bottom of figure 4.15 captures the images.

In Figure 4.16 we show a comparison between our method and the relighting method of Wenger et al. [Wenger 05]. Note that the method of Wenger et al. is a high quality relighting system that generates very good results but only uses time multiplexing. I.e., one light source is turned on in each frame and optical flow is used to merge the information from several frames. In this particular example we push their method and our algorithm to the limit by running it on a scene that has a lot of movement between two successive frames. Image (a) shows the ground truth, only one of the light sources is turned on in this image; (b) and (c) show the same image, but obtained by the method of Wenger et al.; (d) shows the same image obtained by our method. To deal with dynamic scenes, Wenger et al. inserts tracking frames. These are frames for which the scene is uniformly lit with white light. For image (b) a tracking frame was inserted every seven and for (c) every three frames. In (b) it is clear that the optical flow breaks down. This is because the house rotated 35 degrees between two consecutive tracking frames, which results in occlusion problems. For very fast moving scenes our algorithm suffers from the same problem. In (c) the house rotates only 15 degrees between two tracking frames, so the optical flow works quite well, but the lighting information still needs to be warped over a distance of 8 frames (6 light sources and 2 tracking frames). So shadows will not move over the object as they should. Our algorithm has the same problem, but since the lighting information only has to be warped over 2 frames, this problem is far less pronounced. These artifacts are most visible when comparing the door of the house.

In figure 4.17 we compare the optical flow between our method and the method of Wenger et al. when using a tracking frame every three frames. Solid arrows show the images between which the optical flow has to be calculated. As can be seen, in our method the optical flow has to be calculated between frames that are very close to each other, and thus very similar. In the method of Wenger et al., the optical flow needs to be calculated between the tracking frames which are farther apart from each other and thus less similar, which results in a worse optical flow. The dashed arrows show the distance between two frames with identical lighting conditions. For the method of Wenger et al., the lighting information needs to be warped over a large

(a) Ground truth (b) Our result

Figure 4.18: Comparing ground truth and our method when relighting a static scene. Our approach generates convincing results, even though the assumption that surfaces are Lambertian, is slightly broken. The results might not be physically correct, yet they are very convincing. Our method generates high quality results, indicating that our noise reduction works effectively.

distance. This means that the optical flow needs to be very accurate. This distance can also become so large, that the lighting information is no longer valid, as we saw on the door in the previous example. In our method the lighting information needs to be warped over a much smaller distance. For that reason the optical flow does not have to be as good as compared to Wenger et al. and the effect of invalid lighting information is less pronounced.

Since we use the three color channels, one might expect that our system has a speedup of 3 compared to the method of Wenger et al. [Wenger 05], but it is more nuanced than that. For example, if we have 10 lights, we need 4 frames to capture all the information. Wenger et al. needs 15 frames, if a tracking frame has to be inserted every 2 lighting frames. In this case, our method has a speedup of 3.75.

For the example shown in Figure 4.1 on page 69, we used four light sources. The top row shows the input images. The result of our algorithm is shown in the bottom row, they show the scene lit under new lighting conditions. In this example we used the *Complementary Colors Method* to obtain the normalized material color. As explained in section 4.3, we need to have at least 2 frames to do this so AL is a 6×4 matrix of rank 4.

To test the robustness of our algorithm, we ran it on two scenes that break our assumptions. A first scene consists of a person that is being relit, see Figure 4.18. In this scene, the objects are not perfectly Lambertian, there is a small amount of motion

(a) Input (b) Ground truth (c) Our result

Figure 4.19: A test case where our scene assumptions are violated. False color bleeding is visible (e.g., on the wall) and artifacts appear near specularities.

blur since it is a dynamic scene and some interreflections are present. When comparing the ground truth with our result, we can conclude that our algorithm delivers convincing results, even though scene assumptions are violated slightly. In a second scene, which consists of yellow transparent soda in a glass and a specular apple, we clearly break our assumptions, see Figure 4.19. As expected, some artifacts appear. A false specularity is visible on the apple, near the bottom left of the apple there is false color bleeding, and the wall on the right of the glass obtains the yellow color of the soda. Our method also breaks down when applied to a scene with considerable motion blur.

4.6.2 Depth from Structured Light

In Figure 4.20 we show how we can calculate depth from structured light [Scharstein 03, Sato 85] using both time and color multiplexing. We want to acquire a depth map with 6-bit precision, so we need to project 6 structured light patterns. We use the *Dedicated Image Method* in this example, so since we need to multiplex 6 light sources, we need to capture 3 images. One under white illumination to obtain the material colors, and two for the lighting conditions. In figure 4.21 we show how the 6 binary patterns of figure 4.4 are color multiplexed. The images at the bottom row are the ones that are projected. In figure 4.20, image (a) shows the scene lit by three light sources, the red one is the most significant structured light pattern, the green one the second and the blue one the third most significant pattern. Image (b) is used to obtain the normalized material color A. And image (c) shows the scene lit by the three least significant structured light patterns. Extracted structured light patterns are shown in (d) and the resulting depth map along with images of the scene under novel viewpoints is shown in (e). In contrast to previous methods for dynamic scenes

(a) input: three MSB (b) input: albedo image (c) input: three LSB

(d) Extracted structured light patterns

(e) Extracted depth map and novel viewpoints

Figure 4.20: Depth from structured light using time and color multiplexing.

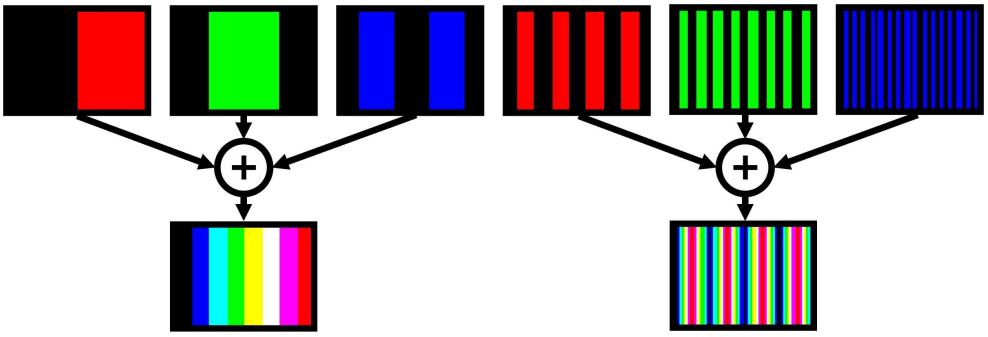


Figure 4.21: Constructing the patterns for depth from structured light using time and color multiplexing. The tree most significant patterns are combined using color multiplexing as can be seen on he left. Similarly, the three least significant patterns are multiplexed to form the pattern shown on the right.

[Caspi 98, Zhang 02, Hall-Holt 01, Lee 07, Chen 97], scene colors do not interfere in the process and we can capture the scene colors and depth map simultaneously.

4.6.3 Photometric Stereo

Figure 4.22 shows how our technique can be used to reduce the number of images needed to perform photometric stereo [Barsky 03, Woodham 80] for acquiring normal maps. With few exceptions (below), previous photometric stereo algorithms are restricted to static objects. We can handle dynamic scenes. Photometric stereo algorithms usually need three images, one for every light source. We need only two. With more light sources, the reduction in images would even be bigger.

Figure 4.22 (a) shows two input images of a scene lit by three light sources. Image (b) shows the demultiplexed lights, these images are the input of the photometric stereo algorithm. Image (c) shows the calculated normal map of the scene. Normals can then be used for instance to add specular highlights using an environment map, as seen in images (d) and (e). Images (f), (g), (h) and (i) show some more results where specularities were added to a diffuse scene.

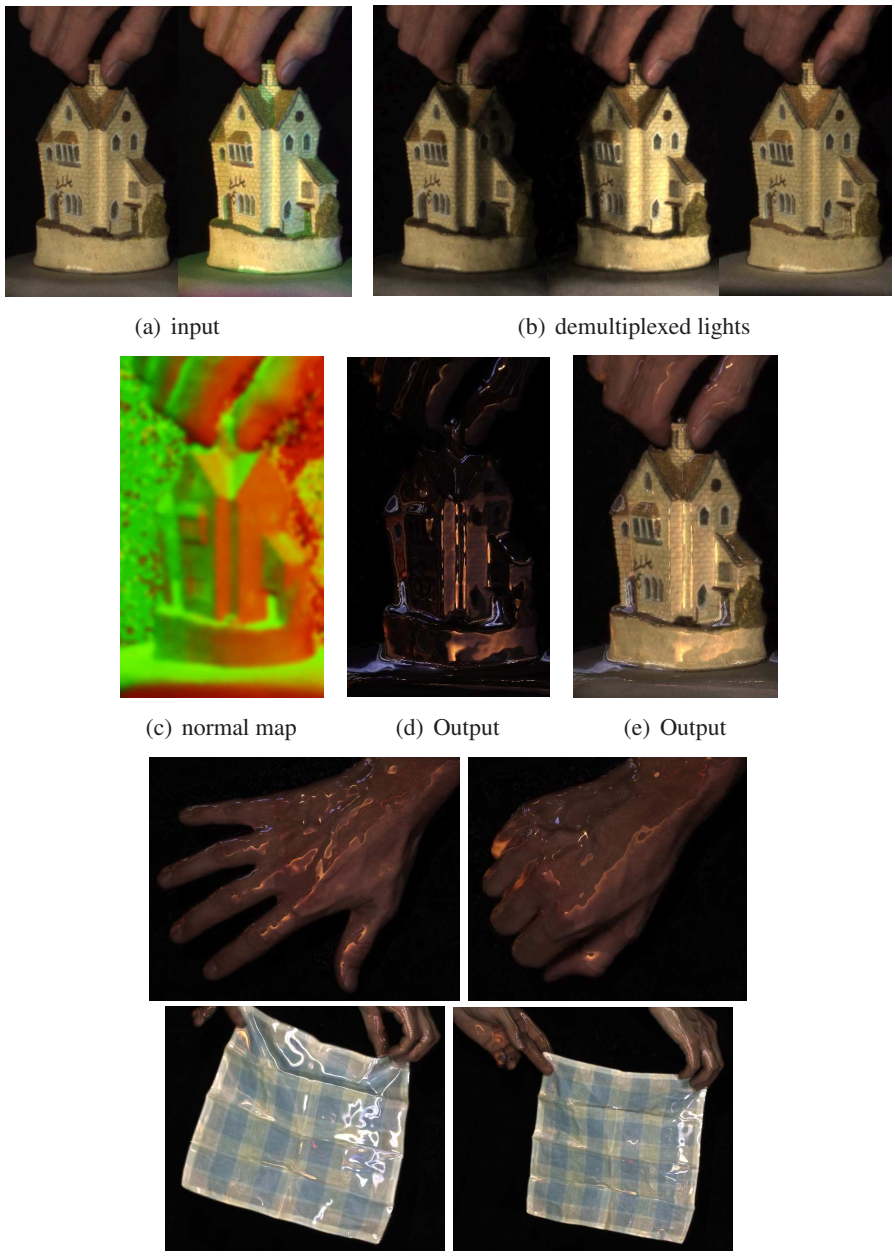
Previous photometric stereo algorithms for dynamic scenes are restricted to scenes in which all objects have the same albedo [Hernandez 07, Hernandez 08]. As shown in Figure 4.22, our method handles multi-coloured dynamic scenes. Apart from the normal map, our method also recovers material colors.

4.6.4 Demultiplexing Moonlight

In figure 4.23 we show how our algorithm can demultiplex moonlight and street light. The top left image was taken at night, it is lit by moonlight and street lights. Because the moonlight has a different color than the street lights we can demultiplex them. The top right image was taken on a cloudy day, we use this image to obtain the normalized material color of the scene. The bottom row shows the demultiplexed result, the bottom left is the scene with only moonlight, the bottom right with only street light.

4.7 Conclusion and Future Work

We proposed a generic approach for capturing dynamic scenes under both color and time multiplexed varying illumination. Our approach is well suited for a broad range



(a) input (b) demultiplexed lights

(c) normal map (d) Output (e) Output

Figure 4.22: A normal map can be calculated using photometric stereo, which allows us to add additional specular highlights for instance.

(a) input night (b) input day

(c) demultiplexed moonlight (d) demultiplexed street lights

Figure 4.23: a,b: input images. a: scene illuminated with grey moonlight and orange street lights. b: scene under cloudy conditions. c,d: output images. c: scene lit by moonlight. d: scene lit by street light.

of applications, and was demonstrated in the context of relighting, structured light depth capturing and photometric stereo. Compared to previous light multiplexing methods, our method allows to extract more information from fewer images. In contrast to previous photometric stereo methods, our technique is able to handle multi-coloured dynamic scenes. Applied to structured light scanning, our method allows to simultaneously acquire depth and scene colors.

Instead of using an RGB camera we might use a camera that is able to distinguish many more light frequencies. This way we can trade fewer frames for resolution.

At present, the computation time with our implementation is rather high. We believe there is plenty of room for improvement, perhaps up to near real time speed.

Some avenues for future research include the noise filtering explained in section 4.3.3, investigating the relation between noise in the input images and noise in the output images. The here presented approach assumes direct-only illumination and Lambertian surfaces. The core idea of this paper may also be applicable to other multiplexing techniques.

Chapter 5

Conclusions

5.1 Summary

In this dissertation, we investigated a number of new methods to manipulate an image sequence in the presence of information about the recorded scene that can not be deduced from the image sequence itself. We looked for new methods to capture additional information about a scene and we searched for innovative ways to use this information to manipulate a video.

5.1.1 Collision Detection

In the first part of this dissertation, we presented a system that allows a person to interact with virtual rigid bodies in real time. A multi-camera setup was used to obtain the 3D information needed to augment the image sequence of the person with the rigid bodies. A method to calculate collisions between the real and the virtual objects was presented. One could obtain these collisions by calculating an approximate geometry for the real objects and determining the collisions between this approximate geometry and the virtual objects. We demonstrated that it is better to calculate the collisions directly from the images themselves, eliminating the need for an approximate geometry of the real objects and thus saving computation time.

5.1.2 Motion Capture

If some extra information is recorded for a number of points in the scene, a number of powerful video manipulation techniques, such as augmented reality, become possible. For this purpose, we presented two novel motion tracking systems.

The first system calculates the 3D position for a number of points in the scene. Electronic markers containing a LED are attached to interesting points in the scene. A computer calculates the 3D position of each marker using the images from multiple calibrated cameras that capture the light emitted by the LEDs. By letting each LED emit a unique pattern, each marker can be uniquely identified. This way, markers never get mixed up. Since the working volume and working precision of the system can easily be adjusted simply by changing the resolution of the cameras or the field of view of the lenses, it is a very flexible system and since we only use off the shelf components and cameras are becoming cheap, the cost of the system is low. As an example application, 3D peepholes were built using a number of robots with laptops attached on top of them. These 3D peepholes were used during a live art performance and our camera based tracking system was used to capture the position of each laptop while the robots drove around on stage.

The second system captures, apart from the position, also the orientation and the incident ambient illumination for a number of scene points. As in the previous system, we attach electronic markers to interesting points in the scene. But unlike the previous system, this system does not use cameras to locate the markers. Instead, a number of binary light patterns are projected onto the scene one after the other and light sensors on the markers decode these patterns to obtain the position of each marker. Identifying the markers to make sure they never get mixed up is not an issue in this system, since the markers calculate their own position. A sensor attached to the marker measures the amount of light arriving from a number of calibrated light sources. This can be used to estimate the orientation of the marker. A final sensor measures the intensity and the color of the ambient illumination. This system can for example be used to add virtual labels and virtual models to a filmed scene. We can illuminate the virtual objects as if they were lit by the light sources present in the real scene using the captured ambient illumination and we can render the virtual objects with realistic motion blur since the system runs at a high frame rate.

5.1.3 Time and Color Multiplexed Illumination

As a final contribution, we presented a generic light multiplexing scheme for dynamic scenes. We show it can be used to improve many existing methods that require

images of a scene under a number of lighting conditions. Examples of such methods are depth from structured light, photometric stereo and relighting. For static scenes, capturing images of a scene under a number of lighting conditions is very straightforward using time multiplexing. For dynamic scenes, this is less trivial, since the objects in the scene move between two successive frames. Optical flow techniques could be used to solve this problem, but since the illumination between successive frames changes dramatically, calculating a good optical flow is not feasible. We present a method that uses both color and time multiplexing to solve this problem. For each frame the camera captures, the color of the light sources are changed. By choosing these colors in a smart way, and by using a filter that removes the illumination from an image, we can use optical flow to warp the images to a reference frame. This way we can obtain a single frame of a dynamic scene for a number of lighting conditions.

5.2 Contributions

The main contributions of this thesis are

- Collision Detection
  - A new image based algorithm to calculate collisions between real and virtual objects.
  - A method to calculate surface information for the real objects near the collisions.
- Camera Based Motion Capture
  - A new camera based system to capture the location of a number of electronic markers.
  - A new method to identify each of the markers using a unique flashing pattern for each LED on the markers.
- Prakash: Capturing Position, Orientation and Incident Illumination using Lighting Aware Motion Capture
  - A system to capture the position, orientation and incident illumination for a number of electronic markers.
  - New methods to use this captured information for video editing.

- Time and Color Multiplexed Illumination
  - An algorithm that uses both color and time multiplexing to capture a scene under a number of lighting conditions. This algorithm is generic and can be used to improve many existing algorithms.
  - A method to compensate for the motion in a dynamic scene even when each frame is lit differently.

5.3 Future Work

Some avenues for future work include

5.3.1 Collision Detection

We represent the virtual objects using point clouds, which has the advantage that our method supports all types of virtual objects that can be represented by point clouds. A disadvantage of this is that the objects may be sampled too sparsely. Especially when thin features are present in the real scene, some collisions will not be found. One avenue for future research is to solve this by using an adaptive approach that samples the virtual objects more densely near potential collisions.

Currently our method cannot handle real objects that move very fast. This could be solved using real time optical flow techniques to estimate the position and shape of the real objects in between two frames.

Another avenue for future work is to eliminate the need for a multi-camera setup. One might be able to estimate an approximate shape of the real scene from a single view and use this to calculate the collisions.

5.3.2 Motion Capture

The robustness of the camera based motion tracking system could be further improved by changing the way the strobe patterns of the LEDs are decoded. By calculating the 3D position, identifying the markers and decoding the strobe pattern all at once, more prior information can be used during each step of the algorithm. It might also be possible to change the strobe pattern in such a way that a marker is identified faster after a long occlusion or the pattern might be changed in a way that short occlusions can be handled more robustly.

Using smart cameras, the position of each LED in image space could be calculated by the camera itself, freeing up resources on the computer and increasing the scalability of the system.

The Prakash motion capture system that captures position, orientation and incident illumination can also be improved in a number of ways. For position tracking, a line of sight is needed between the markers and the light sources that project the binary patterns. If one of the projectors is occluded, the position cannot be calculated. Using a redundant set of projectors, this problem can be solved.

In order to further increase the accuracy and robustness, both systems can be improved by extending them with additional sensors. For example, inertial sensors can be used to increase the accuracy and to handle occlusion problems.

5.3.3 Time and Color Multiplexed Illumination

Some avenues for future research include the noise filtering and investigating the relation between noise in the input images and noise in the output images. At present, the computation time with our implementation is rather high. We believe there is plenty of room for improvement, perhaps up to near real time speed. Instead of using an RGB camera we might use a camera that is able to distinguish more than three light frequencies. This way we can trade fewer frames for resolution.

Apart from color and time multiplexing, other light multiplexing techniques might be used. One might investigate the possibility of using a different carrier frequency for each light source and a special camera to decode these different frequencies. Or we might be able to devise a method where each light source projects a specific spatial pattern that can be demultiplexed afterwards.

Appendices

Appendix A

Publications

This dissertation contains parts of the the following work.

Papers in Proceedings

[Decker 09] *Bert De Decker, Jan Kautz, Tom Mertens and Philippe Bekaert* Capturing Multiple Illumination Conditions using Time and Color Multiplexing *To appear in the proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2009)*

[Decker 07a] *Bert De Decker, Tom Mertens, Philippe Bekaert* Interactive Collision Detection for Free-Viewpoint Video *GRAPP 2007: Proceedings of the Second International Conference on Computer Graphics Theory and Applications Barcelona, isbn 978-972-8865-72-6, pages 114-120, Spain, 2007*

Journal Papers

[Raskar 07b] *Ramesh Raskar, Hideaki Nii, Bert De Decker, Yuki Hashimoto, Jay Summet, Dylan Moore, Yong Zhao, Jonathan Westhues, Paul Dietz, John Barnwell, Shree Nayar, Masahiko Inami, Philippe Bekaert, Michael Noland, Vlad Branzoi, Erich Burns* Prakash: Lighting Aware Motion Capture using Photosensing Markers and Multiplexed Illuminators *SIGGRAPH 2007*

Sketches and Posters

[Decker 07b] *Bert De Decker, Hideaki Nii, Yuki Hashimoto, Dylan Moore, Jay Summet, Yong Zhao, Jonathan Westhues, Paul Dietz, John Barnwell, Masahiko Inami, Philippe Bekaert, Ramesh Raskar* Illumination sensitive dynamic virtual sets *Sketch*

at SIGGRAPH 2007

Exhibitions

[Raskar 07a] *Ramesh Raskar, Hideaki Nii, Bert De Decker, Yuki Hashimoto, Dylan Moore, Jay Summet, Yong Zhao, Jonathan Westhues, Paul Dietz, John Barnwell, Masahiko Inami, Philippe Bekaert* Prakash: Lighting-Aware Motion Capture for Dynamic Virtual Sets *SIGGRAPH 2007 Emerging Technologies*

The following publications are not part of this dissertation.

Journal Papers

[Eisemann 08] *M. Eisemann, B. De Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, A. Sellent* Floating Textures *Computer Graphics Forum (Proc. Eurographics EG'08), vol. 27, no. 2, pages 409-418, April 2008*

Sketches and Posters

[Decker 06] *Bert De Decker, Philippe Bekaert* Interactive Acquisition and Rendering of Humans *Proc. of Workshop on Content Generation and Coding for 3D-Television Eindhoven, Netherlands, 2006-06-02*

Bijlage B

Samenvatting (Dutch Summary)

Video manipulatie bestaat al zolang als er filmcamera's bestaan. Zo worden verschillende beeldsequenties in de juiste volgorde gemonteerd om tot een verhaal te komen en worden speciale effecten gebruikt om acties te filmen die in realiteit moeilijk na te spelen zijn. Voor de opkomst van computers werd film analoog voorgesteld met behulp van pellicule of magneetband. Het nadeel hiervan is dat de kwaliteit achteruitgaat bij elke manipulatie. Om deze reden en doordat computers steeds krachtiger worden, worden de analoge technieken hoe langer hoe meer verruild voor digitale. Terwijl enkele jaren geleden de film camera's, de projectoren in bioscopen en de televisies in de huiskamers nog analoog waren, worden ze nu vervangen door digitale technologie. Door een film digitaal voor te stellen en door gebruik te maken van computers, wordt video manipulatie ook veel gemakkelijker en goedkoper.

Vele video manipulatie methodes gebruiken enkel input van de gebruiker en de video zelf om de informatie te bekomen die nodig is om de video te manipuleren. Echter, als er meer informatie over de gefilmde scène beschikbaar is, wordt het mogelijk om de video te bewerken op een meer geavanceerde manier. Als de 3D geometrie van de scène bekend is, wordt het bijvoorbeeld mogelijk om de scène vanuit nieuwe standpunten te visualiseren. Of als de positie, oriëntatie en belichting bekend is voor een aantal punten in de scène, wordt het mogelijk om virtuele objecten toe te voegen aan een scène op een realistische manier. In deze thesis onderzoeken we wat voor informatie over een scène kan gebruikt worden om een video te bewerken, we ontwikkelen nieuwe methodes om deze informatie te bekomen en we bespreken nieuwe algoritmes om deze informatie te gebruiken voor video manipulatie.

Zo bespreken we een methode waarbij een persoon die gefilmd wordt door meerdere video camera's in reële tijd kan interageren met virtuele objecten. Om dit mogelijk te maken, presenteren we een nieuwe methode om intersecties te detecteren tussen echte en virtuele objecten. Een kortzichtige manier om deze intersecties te bekomen is door eerst een expliciete 3D representatie van de echte objecten te berekenen en vervolgens de intersecties tussen deze representatie en de virtuele objecten te detecteren. Om rekentijd te sparen, doet de hier gepresenteerde methode dit echter niet, ze berekent de intersecties rechtstreeks op de invoer beelden zelf. Om aliasing artefacten te vermijden, moet er wel rekening gehouden worden met het feit dat de invoer beelden discreet worden voorgesteld, ze zijn opgebouwd uit pixels. Als toepassing van deze techniek tonen we een systeem waarbij een gebruiker in reële tijd kan interageren met een simulator van onvervormbare objecten.

We onderzochten ook welke video manipulatie technieken mogelijk worden als extra informatie wordt opgenomen voor een aantal punten in de scène. Deze informatie kan gebruikt worden voor een heel aantal video manipulatie technieken, zoals het automatisch berekenen van de witbalans, het toevoegen van virtuele objecten aan een scène of het verwijderen van bewegingsonscherpte. Om deze extra informatie te bekomen presenteren we twee nieuwe systemen. Het eerste is een camera gebaseerd systeem dat de positie van een aantal punten in de scène berekend. Elk object in de scène waarvoor we de positie willen weten, wordt voorzien van elektronische kentekens. Op elk kenteken is er een infrarode lichtdiode aangebracht die knippert volgens een bepaald patroon dat uniek is voor elk kenteken. Doordat er meerdere gekalibreerde camera's deze lichtdiodes filmen, kan de 3D positie van elk kenteken bepaald worden. En doordat elke lichtdiode knippert volgens zijn eigen uniek patroon, worden de kentekens nooit onderling verwisseld. Zo kan het systeem op elk moment bepalen waar welk kenteken zich bevindt. We tonen de werking van dit systeem aan door de positie van een aantal robotten te berekenen tijdens een theater voorstelling.

Het tweede systeem besproken in deze thesis berekent buiten de positie ook de oriëntatie en inkomende belichting voor een aantal punten in de scène. Ook in dit systeem worden er elektronische kentekens aan bepaalde punten in de scène vastgemaakt. Maar in tegenstelling tot het vorige systeem, bevatten de kentekens in dit systeem geen lichtdiodes maar lichtsensoren. Een binaire lichtsensor op de kentekens wordt gebruikt om de 3D positie te berekenen aan de hand van verschillende binaire lichtpatronen die op de scène worden geprojecteerd. De oriëntatie kan berekend worden door de hoeveelheid licht te meten die invalt op een analoge sensor op het kenteken voor aantal gekalibreerde lichtbronnen. Ten slotte is er ook nog een sensor op het kenteken aangebracht die de kleur en intensiteit van het omgevingslicht meet. Dit systeem kan bijvoorbeeld gebruikt worden om virtuele objecten aan een scène toe te

voegen. Doordat het systeem aan een hoge snelheid werkt, kan het virtueel object met realistische bewegingsonscherpte getekend worden en doordat het omgevingslicht wordt gemeten, kan het virtueel object getekend worden alsof het werd belicht door de lichtbronnen aanwezig in de gefilmde scène.

Er bestaan een heel aantal video manipulatie technieken die als invoer foto's van een scène nodig hebben onder verschillende belichtingen. Zo kan men een dieptemap bekomen als men een aantal foto's neemt van een scène die telkens wordt belicht met een ander binair patroon. Als men de scène niet belicht met binaire patronen, maar met een aantal puntlichtbronnen, dan kan men een normaalmap van de scène berekenen gebruikmakende van de genomen foto's. Als de scène statisch is, dan is het zeer eenvoudig om deze verschillende foto's te bekomen, je belicht de scène telkens met een andere lichtbron en neemt een foto. Als de scène dynamisch is, dan is dit minder evident. Tussen twee opeenvolgende foto's is de inhoud van de scène immers gewijzigd. Men zou kunnen proberen de beweging in de scène te schatten, maar vermits de belichting tussen twee opeenvolgende foto's zo drastisch anders is, is dit niet voor de hand liggend. Wij presenteren een methode die gebruik maakt van gekleurd licht om dit probleem op te lossen. We nemen verschillende foto's van onze dynamische scène, en voor elke foto wijzigen we de kleur van de lichtbronnen. We gebruiken dus tijd en kleur om foto's van de scène onder verschillende belichtingen te bekomen. Doordat we geen van de lichtbronnen ooit volledig uit zetten, kunnen we een filter gebruiken die de belichting uit de beelden verwijdert. Uit deze gefilterde beelden kunnen we dan makkelijker de beweging in de scène schatten. Omdat we met gekleurd licht werken, kunnen we meer belichtingsinformatie in een foto steken en hebben we dus minder foto's nodig. We tonen aan dat het een algemene methode is die vele bestaande video manipulatie technieken kan verbeteren.

Bibliography

[Allard 06a] J. Allard, J. Franco, C. Menier, E. Boyer & B. Raffin. *The GrImage Platform: A Mixed Reality Environment for Interactions*. In ICVS, 2006.

[Allard 06b] J. Allard & B. Raffin. *Distributed Physical Based Simulations for Large VR Applications*. In IEEE Virtual Reality Conference, 2006.

[Baraff 92] D. Baraff. *Dynamic simulation of non-penetrating rigid body simulation*. PhD thesis, 1992.

[Barsky 03] Svetlana Barsky & Maria Petrou. *The 4-Source Photometric Stereo Technique for Three-Dimensional Surfaces in the Presence of Highlights and Shadows*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 10, pages 1239–1252, 2003.

[Bowman 97] D. Bowman & L. Hodges. *Techniques for Grabbing and Manipulating Remote Objects in Immersive Virtual Environments*. In ACM Symposium on Interactive 3-D Graphics, 1997.

[Breen 96] D. Breen, R. Whitaker & M. Tuceryan. *Interactive Occlusion and Automatic Object Placement for Augmented Reality*. In Computer Graphics Forum, Blackwell Publishers, 1996.

[Bridson 02] Robert Bridson, Ronald Fedkiw & John Anderson. *Robust treatment of collisions, contact and friction for cloth animation*. ACM Trans. Graph., vol. 21, no. 3, pages 594–603, 2002.

[Brox 04] T. Brox, A. Bruhn, N. Papenberg & J. Weickert. *High accuracy optical flow estimation based on a theory for warping*.

In European Conference on Computer Vision (ECCV), volume 3024 of *LNCS*, pages 25–36, Prague, Czech Republic, May 2004. Springer.

[Caspi 98] Dalit Caspi, Nahum Kiryati & Joseph Shamir. *Range Imaging With Adaptive Color Structured Light*. IEEE Trans. Pattern Anal. Mach. Intell., vol. 20, no. 5, pages 470–480, 1998.

[Chen 97] C.-S. Chen, Y.-P. Hung, C.-C. Chiang & J.-L. Wu. *Range data acquisition using color structured lighting and stereo vision*. Image and Vision Computing, vol. 15, no. 6, pages 445–456, 1997.

[Codamotion 07] Codamotion. *Charnwood Dynamics Ltd. http://www.charndyn.com/*, 2007.

[Coleman 94] Thomas F. Coleman & Yuying Li. *On the convergence of interior-reflective Newton methods for nonlinear minimization subject to bounds*. Math. Program., vol. 67, no. 2-17, pages 189–224, 1994.

[Corporation 06] Motion Analysis Corporation. *Hawk-I Digital System*, 2006.

[Cotting 04] Daniel Cotting, Martin Naef, Markus Gross & Henry Fuchs. *Embedding Imperceptible Patterns into Projected Images for Simultaneous Acquisition and Display*. In ISMAR '04: Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality, pages 100–109, Washington, DC, USA, 2004. IEEE Computer Society.

[Davis 03] J. Davis, R. Ramamoorthi & S. Rusinkiewicz. *Spacetime Stereo: A Unifying Framework for Depth from Triangulation*. In CVPR, pages II: 359–366, 2003.

[Debevec 00] Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin & Mark Sagar. *Acquiring the reflectance field of a human face*. In SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pages 145–156, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Debunne 01] G. Debunne, M. Desbrun, M.-P. Cani & A. H. Barr. *Dynamic realtime deformations using space and time adaptive sampling*. In ACM SIGGRAPH, 2001.

[Decker 06] Bert De Decker & Philippe Bekaert. *Interactive Acquisition and Rendering of Humans*. In Proc. of Workshop on Content Generation and Coding for 3D-Television Eindhoven, 06 2006.

[Decker 07a] Bert De Decker, Tom Mertens & Philippe Bekaert. *Interactive collision detection for free-viewpoint video*. In GRAPP (AS/IE), pages 114–120, 2007.

[Decker 07b] Bert De Decker, Hideaki Nii, Yuki Hashimoto, Dylan Moore, Jay Summet, Yong Zhao, Jonathan Westhues, Paul Dietz, John Barnwell, Masahiko Inami, Philippe Bekaert & Ramesh Raskar. *Illumination sensitive dynamic virtual sets*. Sketch at SIGGRAPH 2007, 2007.

[Decker 09] Bert De Decker, Jan Kautz, Tom Mertens & Philippe Bekaert. *Capturing Multiple Illumination Conditions using Time and Color Multiplexing*. In To appear in the proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009.

[Dewaele 04] Guillaume Dewaele & Marie-Paule Cani. *Interactive Global and Local Deformations for Virtual Clay*. In Graphical Models, 2004.

[Eisemann 08] Martin Eisemann, Bert De Decker, Marcus Magnor, Philippe Bekaert, Edilson de Aguiar, Naveed Ahmed, Christian Theobalt & Anita Sellent. *Floating Textures*. Computer Graphics Forum (Proc. Eurographics EG'08), vol. 27, no. 2, pages 409–418, 4 2008.

[Govindaraju 05] Naga Govindaraju, David Knott, Nitin Jain, Ilknur Kabul, Rasmus Tamstorf, Russell Gayle, Ming Lin & Dinesh Manocha. *Interactive Collision Detection between Deformable Models using Chromatic Decomposition*. In ACM SIGGRAPH, 2005.

[Grossman 04] Tovi Grossman, Daniel Wigdor & Ravin Balakrishnan. *Multi-finger gestural interaction with 3d volumetric displays*. In UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology, pages 61–70, New York, NY, USA, 2004. ACM Press.

[Guendelman 03] E. Guendelman, R. Bridson, & R. Fedkiw. *Nonconvex rigid bodies with stacking*. In ACM SIGGRAPH, 2003.

[Hall-Holt 01] Olaf Hall-Holt & Szymon Rusinkiewicz. *Stripe Boundary Codes for Real-Time Structured-Light Range Scanning of Moving Objects*. In IEEE International Conference on Computer Vision, pages II: 359–366, 2001.

[Hand 97] C. Hand. *A Survey of 3-D Interaction Techniques*. In Computer Graphics Forum, Blackwell Publishers, 1997.

[Hasenfratz 03] J.M. Hasenfratz, M. Lapierre, J.D. Gascuel & E. Boyer. *Real-Time Capture, Reconstruction and Insertion into Virtual World of Human Actors*. In Vision, Video and Graphics Conference, 2003.

[Hasenfratz 04] J.M. Hasenfratz, M. Lapierre & F. Sillion. *A Real-Time System for Full Body Interaction with Virtual Worlds*. In Eurographics Symposium on Virtual Environments, 2004.

[Hawkins 04] Tim Hawkins, Andreas Wenger, Chris Tchou, Andrew Gardner, Fredrik Göransson & Paul E. Debevec. *Animatable Facial Reflectance Fields*. In Rendering Techniques, pages 309–321, 2004.

[Heidelberger 04] Bruno Heidelberger, Matthias Teschner & Markus Gross. *Detection of Collisions and Self-collisions Using Image-space Techniques*. In WSCG, 2004.

[Hernandez 07] Carlos Hernandez, George Vogiatzis, Gabriel J. Brostow, Bjorn Stenger & Roberto Cipolla. *Non-rigid Photometric Stereo with Colored Lights*. In IEEE International Conference on Computer Vision, pages 1–8, 10 2007.

[Hernandez 08] Carlos Hernandez, George Vogiatzis & Roberto Cipolla. *Shadows in Three-Source Photometric Stereo.* In European Conference on Computer Vision, volume 5302, pages 290–303, 2008.

[Hightower 01] Jeffrey Hightower & Gaetano Borriello. *Location Systems for Ubiquitous Computing*. Computer, vol. 34, no. 8, pages 57–66, 2001.

[Hsu 08] Eugene Hsu, Tom Mertens, Sylvain Paris, Shai Avidan & Fredo Durand. *Light Mixture Estimation for Spatially Varying White Balance*. to appear in proc of Siggraph 2008, 2008.

[Iltanen 98] M. Iltanen, H. Kosola, K. Palovuori & Vanhala. *Optical Positioning and Tracking System for a Head Mounted Display Based on*

Spread Spectrum Technology. In 2nd International Conference on Machine Automation (ICMA), pages 597–608, 1998.

[inc 06] PTI inc. *VisualEyez VZ 4000*, 2006.

[inc 07] Phase Space inc. *Impulse Camera http://www.phasespace.com*, 2007.

[Kang 04] Seong-Ho Kang & Delbert Tesar. *Indoor GPS Metrology System with 3D Probe for Precision Applications*. In Proceedings of ASME IMECE 2004 International Mechanical Engineering Congress and RD&D Expo, 2004.

[Kitamura 03] Yoshifumi Kitamura, Susumu Ogata & Fumio Kishino. *A Manipulation Environment of Virtual and Real Objects using a Magnetic Metaphor*. In ACM Symposium on Virtual Reality Software and Technology, 2003.

[Laurentini 94] A. Laurentini. *The Visual Hull Concept for Silhouette-Based Image Understanding*. IEEE Trans. Pattern Anal. Mach. Intell., vol. 16, no. 2, pages 150–162, 1994.

[Lee 05] Johnny C. Lee, Scott E. Hudson, Jay W. Summet & Paul H. Dietz. *Moveable interactive projected displays using projector based tracking*. In UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology, pages 63–72, New York, NY, USA, 2005. ACM.

[Lee 07] K.H. Lee, C.S. Je & S.W. Lee. *Color-Stripe Structured Light Robust to Surface Color and Discontinuity*. In ACCV07, pages II: 507–516, 2007.

[Lindeman 04] R. W. Lindeman, R. Page, Y. Yanagida & J. L. Sibert. *Towards Full-Body Haptic Feedback: The Design and Deployment of a Spatialized Vibrotactile Feedback System*. In ACM Virtual Reality Software and Technology (VRST), 2004.

[Lok 03] B. Lok, S. Naik, M. Whitton & F.P. Brooks Jr. *Incorporating dynamic real objects into immersive virtual environments*. In Proceedings of the 2003 symposium on Interactive 3D graphics, 2003.

[Losasso 06] F. Losasso, T. Shinar, A. Selle & R Fedkiw. *Multiple Interacting Liquids*. In ACM SIGGRAPH, 2006.

[Machines 06] Silicon Light Machines. *Gated Light Valve. http://www.siliconlight.com*, 2006.

[Matusik 00] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler & Leonard McMillan. *Image-based visual hulls*. In SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pages 369–374, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Matusik 01] W. Matusik, C. Buehler & L. McMillan. *Polyhedral visual hulls for real-time rendering*. In Eurographics Workshop on Rendering, 2001.

[NII 05] Hideaki NII, Maki SUGIMOTO & Masahiko INAMI. *Smart Light-Ultra High Speed Projector for Spatial Multiplexing Optical Transmission*. In CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops, page 95, Washington, DC, USA, 2005. IEEE Computer Society.

[Optotrak 07] Optotrak. *NDI Optotrak Certus Spatial Measurement. http://www.ndigital.com/certus.php*, 2007.

[Palovuori 00] Karri T. Palovuori, Jukka J. Vanhala & Markku A. Kivikoski. *Shadowtrack: A Novel Tracking System Based on Spreadspectrum Spatio-Temporal Illumination*. Presence: Teleoper. Virtual Environ., vol. 9, no. 6, pages 581–592, 2000.

[Pauly 04] Mark Pauly, Dinesh K. Pai & Leonidas J. Guibas. *Quasi-Rigid Objects in Contact*. In Eurographics/ACM SIGGRAPH Symposium on Computer Animation, 2004.

[Peers 07] Pieter Peers, Naoki Tamura, Wojciech Matusik & Paul Debevec. *Post-production facial performance relighting using reflectance transfer*. ACM Trans. Graph., vol. 26, no. 3, page 52, 2007.

[Raskar 04] Ramesh Raskar, Paul Beardsley, Jeroen van Baar, Yao Wang, Paul Dietz, Johnny Lee, Darren Leigh & Thomas Willwacher. *RFIG lamps: interacting with a self-describing world via photosensing wireless tags and projectors*. ACM Trans. Graph., vol. 23, no. 3, pages 406–415, 2004.

[Raskar 07a] Ramesh Raskar, Hideaki Nii, Bert De Decker, Yuki Hashimoto, Dylan Moore, Jay Summet, Yong Zhao, Jonathan Westhues, Paul Dietz, John Barnwell, Masahiko Inami & Philippe Bekaert. *Prakash: Lighting-Aware Motion Capture for Dynamic Virtual Sets*. SIGGRAPH 2007 Emerging Technologies, 2007.

[Raskar 07b] Ramesh Raskar, Hideaki Nii, Bert De Decker, Yuki Hashimoto, Jay Summet, Dylan Moore, Yong Zhao, Jonathan Westhues, Paul Dietz, John Barnwell, Shree Nayar, Masahiko Inami, Philippe Bekaert, Michael Noland, Vlad Branzoi & Erich Bruns. *Prakash: lighting aware motion capture using photosensing markers and multiplexed illuminators*. ACM Trans. Graph., vol. 26, no. 3, page 36, 2007.

[Redon 04] Stephane Redon, Young J. Kim, Ming C. Lin & Dinesh Manocha. *Fast Continuous Collision Detection for Articulated Models*. In ACM Symposium on Solid Modeling and Applications, 2004.

[Robertson 06] B Robertson. *Big moves*. In Computer Graphics World 29, 11 (Nov)., 2006.

[Sand 04] Peter Sand & Seth Teller. *Video Matching*. ACM Transactions on Graphics (TOG), vol. 22, no. 3, pages 592–599, 2004.

[Sand 06] Peter Sand & Seth Teller. *Particle Video: Long-Range Motion Estimation using Point Trajectories*. In CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 2195–2202, Washington, DC, USA, 2006. IEEE Computer Society.

[Sato 85] K. Sato & S. Inokuchi. *Three-dimensional surface measurement by space encoding range imaging*. Journal of Robotic Systems, vol. 2, pages 27–39, 1985.

[Scharstein 03] Daniel Scharstein & Richard Szeliski. *High-Accuracy Stereo Depth Maps Using Structured Light*. cvpr, vol. 01, page 195, 2003.

[Schechner 03] Yoav Y. Schechner, Shree K. Nayar & Peter N. Belhumeur. *A Theory of Multiplexed Illumination*. In IEEE International Conference on Computer Vision, pages 808–815, 2003.

[Schechner 07] Yoav Y. Schechner, Shree K. Nayar & Peter N. Belhumeur. *Multiplexing for Optimal Lighting*. IEEE Trans. Pattern Anal. Mach. Intell., vol. 29, no. 8, pages 1339–1354, 2007.

[Sorensen 89] B.R. Sorensen, M. Donath, G.-B. Yang & R.C. Starr. *The Minnesota Scanner: A Prototype Sensor for Three-dimensional Tracking of Moving Body Segments*. In IEEE Transactions on Robotics and Automation 45 4, pages 499–509, 1989.

[Stam 00] J. Stam. *Interacting with smoke and fire in real time*. In Communications of the ACM, 2000.

[Svoboda] T. Svoboda. *Multi-Camera Self-Calibration*.

[Teschner 05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, M.-P. Cani A. Fuhrmann, F. Faure, N. Magnenat-Thalmann, W. Strasser & P. Volino. *Collision Detection for Deformable Objects*. In Computer Graphics Forum, 2005.

[Viconpeak 06] Viconpeak. *Camera MX 40 http://www.vicon.com/*, 2006.

[Wang 05] Xiyong Wang, Aaron Kotranza, John Quarles, Benjamin Lok & B. Danette Allen. *Rapidly Incorporating Real Objects for Evaluation of Engineering Designs in a Mixed Reality Environment*. In 3D User Interfaces Workshop, IEEE Virtual Reality, 2005.

[Wang 08] O. Wang, J. Davis, E. Chuang, I. Rickard, K. de Mesa & C. Dave. *Video Relighting Using Infrared Illumination*. Computer Graphics Forum (Proc. Eurographics EG'08), vol. 27, no. 2, pages 271–279, 4 2008.

[Welch 97] Greg Welch & Gary Bishop. *SCAAT: incremental tracking with incomplete information*. In SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pages 333–344, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[Welch 02] Greg Welch & Eric Foxlin. *Motion Tracking: No Silver Bullet, but a Respectable Arsenal*. IEEE Comput. Graph. Appl., vol. 22, no. 6, pages 24–38, 2002.

[Wen 03] Z. Wen, Z. Liu & T. Huang. *Face relighting with radiance environment maps*. In IEEE International Conference on Computer Vision and Pattern Recognition, pages II: 158–65, 2003.

[Wenger 05] Andreas Wenger, Andrew Gardner, Chris Tchou, Jonas Unger, Tim Hawkins & Paul Debevec. *Performance relighting and reflectance transformation with time-multiplexed illumination*. ACM Trans. Graph., vol. 24, no. 3, pages 756–764, 2005.

[Woodham 80] R. J. Woodham. *Photometric Method for Determining Surface Orientation from Multiple Images*. Optical Engineering, vol. 19, no. 1, pages 139–144, 1980.

[Woodham 89] Robert J. Woodham. *Photometric method for determining surface orientation from multiple images*. Shape from shading, pages 513–531, 1989.

[Zhang 02] Li Zhang, Brian Curless & Steven M. Seitz. *Rapid Shape Acquisition Using Color Structured Light and Multi-pass Dynamic Programming*. In The 1st IEEE International Symposium on 3D Data Processing, Visualization, and Transmission, pages 24–36, June 2002.

[Zhang 03] Li Zhang, Brian Curless & Steven M. Seitz. *Spacetime Stereo: Shape Recovery for Dynamic Scenes*. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 367–374, June 2003.