REAL-TIME 3D PERSON TRACKING AND DENSE STEREO MAPS USING GPU ACCELERATION

A Thesis

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Kendall Merriman

June 2014

COMMITTEE MEMBERSHIP

TITLE:	Real-time 3D Person Tracking and Dense Stereo Maps Using GPU Acceleration
AUTHOR:	Kendall Merriman
DATE SUBMITTED:	June 2014
COMMITTEE CHAIR:	Zoë Wood, Ph.D. Assistant Professor of Computer Science
COMMITTEE MEMBER:	Chris Lupo, Ph.D. Assistant Professor of Computer Science
COMMITTEE MEMBER:	Chris Buckalew, Ph.D. Professor of Computer Science

Abstract

Real-time 3D Person Tracking and Dense Stereo Maps Using GPU Acceleration

Kendall Merriman

Interfacing with a computer, especially when interacting with a virtual three dimensional (3D) scene, found in video games for example, can be frustrating when using only a mouse and keyboard. Recent work has been focused on alternative modes of interactions, including 3D tracking of the human body. One of the essential steps in this process is acquiring depth information of the scene. Stereo vision is the process of using two separate images of the same scene, taken from slightly different positions, to get a three dimensional view of the scene. One of the largest issues with dense stereo map generation is the high processor usage, usually preventing this process from being done in real time. In order to solve this problem, this project attempts to move the bulk of the processing to the GPU. The depth map extraction is done by matching points between the images, and using the difference in their positions to determine the depth, using multiple passes in a series of openGL vertex and fragment shaders. Once a depth map has been created, the software uses it to track a person's movement and pose in three dimensions, by tracking key points on the person across frames, and using the depth map to find the third dimension.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Interfacing with a computer, especially when interacting with a virtual three dimensional (3D) scene, found in video games for example, can be frustrating when using only a mouse and keyboard. A mouse provides only two dimensions of input, and a keyboard can only provide a series of on and off switches, which is not the best way to work with three dimensional data. While we as users have gotten used to using these devices for three dimensional input, they are not ideal, and the mapping of multiple two dimensional inputs to three dimensions is not exactly intuitive.

Modern user interfaces have begun moving beyond the simple mouse and keyboard. Alternative methods of interacting with computer systems have become more common, from voice recognition [7] to multi-touch gesture recognition [5, 9] to motion sensing [5, 12], different technologies and ideas are being explored, both academically and commercially. The best selling game console on the market uses a simple motion sensing controller, instead of the normal game pad, to make interacting with games a more physical, immersive, and interactive experience [10]. Multitouch feels like such a natural way to interact with a screen that it has been widely adopted on newer high end smartphones [5, 11].

One of the more interesting solutions to the problem of user interfaces is free-form gesture recognition, where the user uses their body as the input device. This area has been explored for many years, and example systems range from applications like video games [46], device control [24], web browsing [46], or artistic applications [46].

One of the difficulties to this type of application is that there is a large amount of data to process from the camera(s) in real time, to get accurate data. If the system makes use of only one camera to reduce the amount of data to deal with, accurately determining the depth of objects is more difficult. This can be dealt with by implementing the gesture system to not require depth.

Another option which has recently become available is a time of flight camera, which is custom hardware that uses an infrared radar to get depth information [14]. However, these cameras have difficulties with bright background lights. They also have lower resolutions then are possible with other approaches, and require the separate purchase of this specialty hardware.

This thesis describes a free form gesture based interactive experience using only commonly available commercial hardware. By using computer vision techniques and inexpensive and widely available web cams, it is possible to track a user's arm position in three dimensions, and use that position to provide interactivity. This system addresses only the first part of this problem, extracting 3D positional data for the user's arm, using inexpensive commercial hardware, without overburdening the main processor. Applying this data to a user interface is left for later projects.

Our approach consists of three basic steps: preprocessing, which prepares the web cam images for use in the remaining steps; depth map extraction, which uses image pyramids to find stereo correspondences from the processed images; and user tracking which uses a combination of blob and point tracking, plus the generated depth map, to track the movements of a users arm in three dimensions.

The main contribution of this paper is this real-time three dimensional user interface, using the GPU as its primary processor, which generates dense stereo maps in real time, while tracking a user's arm movements. We succeeded in demonstrating the feasibility of extracting these depth maps in real time, and using them for user interactivity. This could be used for many applications and control systems, while still leaving the main processor free for other tasks.

The following chapter (2) will provide information on computer vision in general, stereo vision, and person tracking and modeling. The chapters after that cover the algorithms used (3), the results (4), and areas for improvement and expansion (5). Lastly, some concluding thoughts are presented (6).

Chapter 2

Related Work

Computer vision has been an area of interest for many years, with constantly evolving areas of study. Some of the key areas to this project are reviewed below. First, however, a brief discussion of GPU programming is provided.

2.1 GPU Programming

In recent years, the computer graphics industry has been moving away from fixed function processing of vertices and pixels to a programmable model. The side effect of this change is that it is now possible to use the graphics processing unit (GPU) as a general purpose co-processor for executing arbitrary code. There have been numerous efforts to apply this to a wide range of projects, including encryption [32], vision [25], and simulation [22].

The GPU can be used to implement any programs that can be run as a series of Vertex and Fragment shaders, which perform processing per point and per pixel. The processing steps are shown in Figure 2.1. This processing can be used to do any operation that can be expressed through calculations and texture sampling. Newer

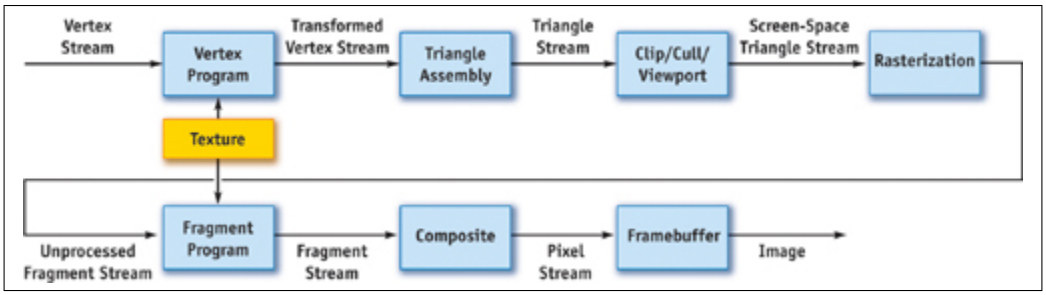

Figure 2.1: The processing steps on a GPU. Image from [35].

GPUs support processing in excess of 512 pixels simultaneously [13], and are therefore well suited to doing certain types of calculations on lots of data in parallel.

2.2 Image Processing and Computer Vision

There are a few computer vision techniques which are important for this project. These are object tracking, distortion correction, and noise removal, which is also called denoising.

2.2.1 Object Tracking

There are two basic types of object tracking: blob tracking and point tracking. Point tracking is based on finding key points, often called features, in an image, and matching them up with the features found in the previous frame. One method for this is called Scale Invariant Feature Transform, or SIFT [30]. This algorithm has already been implemented on a GPU [48], and as such was available for use in this project. An example of some detected features from a SIFT implementation is shown in Figure 2.2, and a matching to a second image is shown in Figure 2.3.

Blob tracking works by finding an area of a particular color, which is then tracked over multiple frames [23]. The center of the blob is used as the tracked object. This can work well, but if the tracked blob overlaps with another object of the same

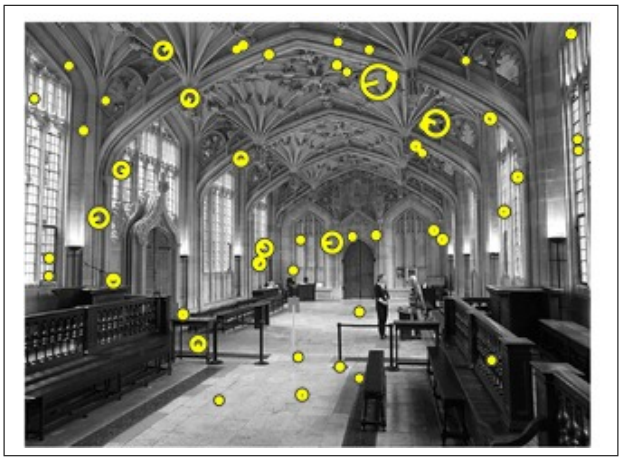

Figure 2.2: An example of point detection. The located points are highlighted with a yellow circle. Image from [15].

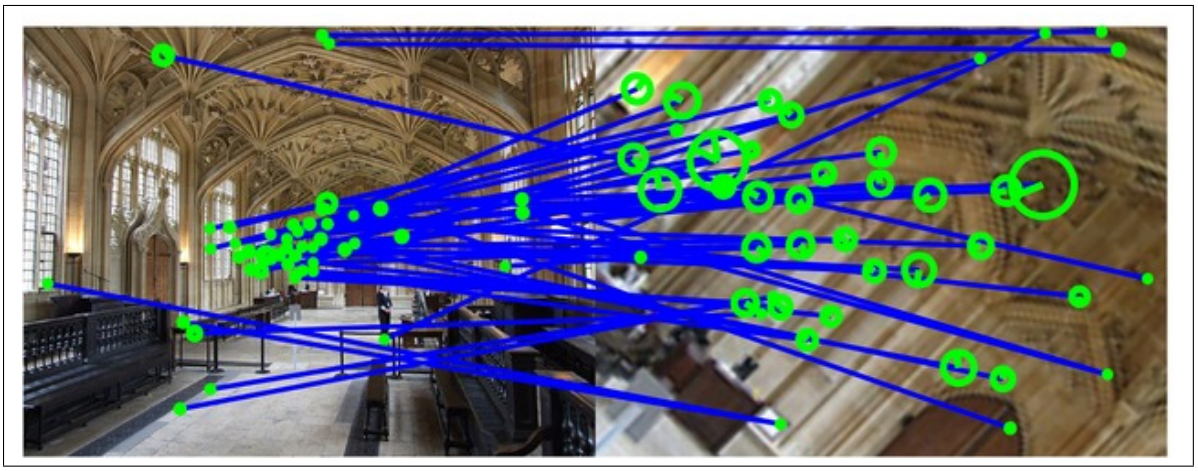

Figure 2.3: An example of point matching. The found matches are shown by blue lines. Images from [15].

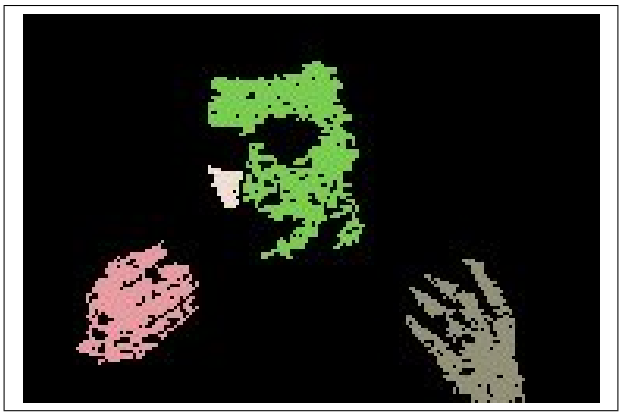

Figure 2.4: An example of blob tracking. There are three blobs being tracked in the image. Image from [1].

color, this algorithm can become confused. Also, the blob can be restricted to an area immediately surrounding the last known point, in order to eliminate data from outside of the tracked area. An example of blob tracking is shown in Figure 2.4.

2.2.2 Undistortion

One other computer vision method which is key to this project is lens distortion correction, often called radial undistortion. This usually involves calibrating the system to the distortion in a particular camera setup, determining the coefficients of the distortion, then correcting that distortion through a texture coordinate transformation. This process is well understood, and implemented in several computer vision libraries, including openCV [4], among others.

There are two types of distortion in images: radial distortion and tangential distortion. Radial distortion is caused by the nature of the lenses, and is usually either barrel or pincushion distortion. This can be thought of as either pushing in or pulling out the corners uniformly, while leaving the edge centers anchored. An example of this can be seen in Figure 2.5. Tangential distortion is caused by the sensor not being precisely aligned with the lens, and is usually very minor [44].

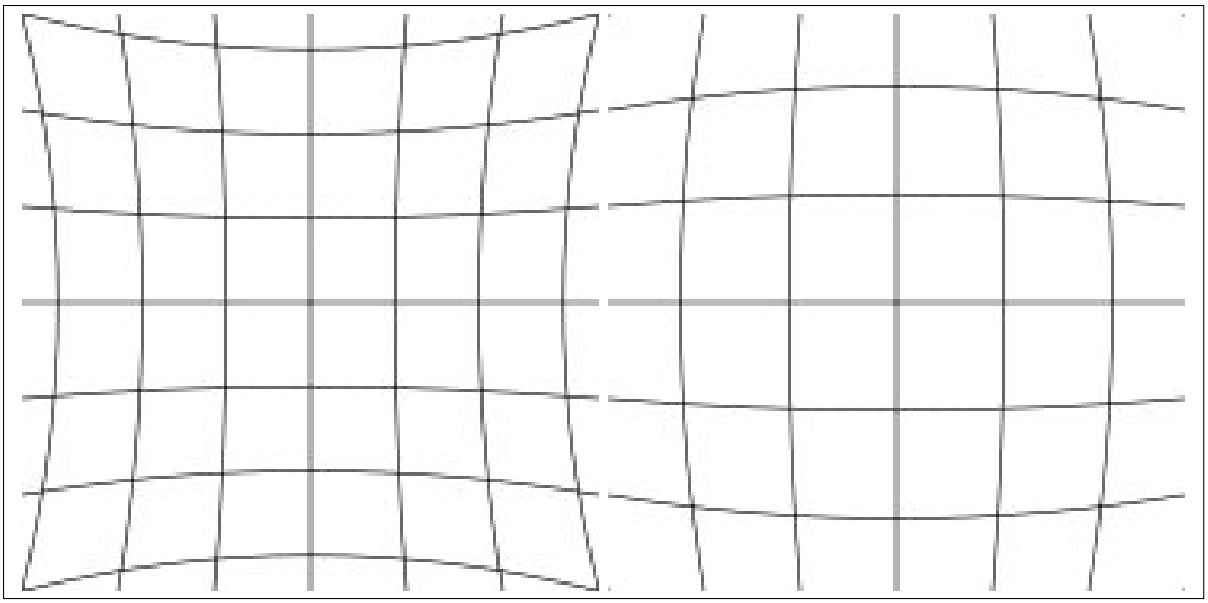

Figure 2.5: Typical lens distortion types. Left - Pincushion distortion. Right - Barrel distortion. Images from [6]

2.2.3 Image Denoising

The last computer vision technique which is important to this thesis is called denoising, and there are a few different approaches to this problem. In order to effectively remove the noise in images, it is important to understand the nature of the noise. There are two basic types of noise: additive noise, which happens when a pixel is adjusted from its correct value by some amount, and "salt and pepper noise", which usually involves a small number of pixels which are randomly set to high or low values [44].

There are three general approaches to dealing with image noise: convolution filters, such as a Gaussian filter, non-linear filters, such as a median filter, and anisotropic diffusion.

Gaussian filters are convenient, as they are easily implemented, they can be written as a two-pass, separable filter, allowing the filter to be run first vertically, then horizontally. The downside is that these filters can sometimes make detail harder to

make out, as these filters blur the image slightly. They also reduce noise, however, it is often only attenuated, not removed.

Median filters are very good at removing salt and pepper noise, however, they are not as good at removing additive noise. Median filters can be approximated separably, which makes them somewhat convenient, however, their limitation at removing other types of noise limits their usefulness in some cases.

Anisotropic Diffusion is a good method for removing noise, as it preserves much of the image detail, while still eliminating the noise, however, it is a more complicated method then others. By adjusting the smoothing along features, it causes the image to be filtered differently in different areas, which preserves features while eliminating image noise.

2.3 Stereo Vision

The idea of stereo vision is simple enough: by taking two different images of the same scene from slightly different positions, disparities between the images can be used to approximate the depth of objects in the scene. A complete map of the depths of all objects in the scene is generated by matching each region of one image with the second, and determining the distance these matched regions are from one another [44].

One major concept in stereo vision is the fundamental matrix. This matrix describes the relation between the two cameras of a stereo rig. There are several methods to extract this matrix, but the most straightforward method is to find matching points between the two images, and solving the resulting system of equations. Generally, there are many more matches found then needed, and this allows the solution to be optimized over the full set of matches [44].

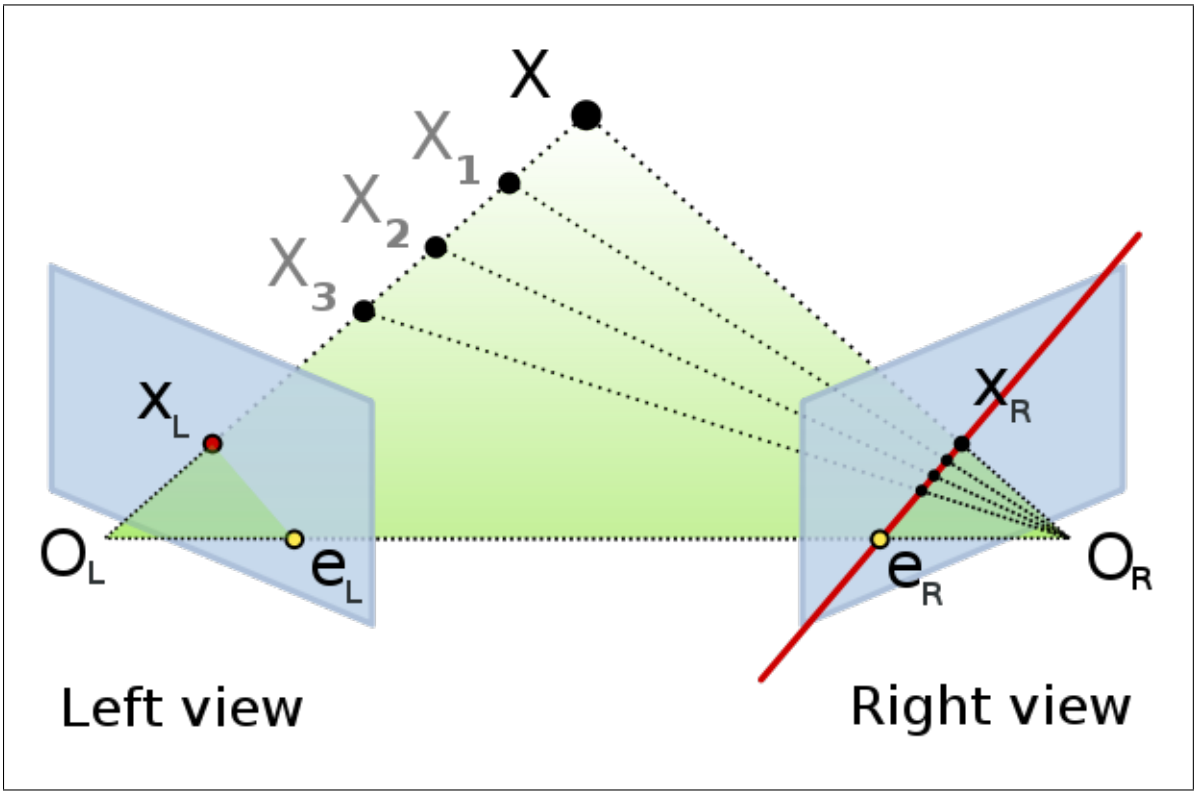

Figure 2.6: An example of epipolar geometry. The point X_L in the left view must lie along the line marked in red in the right view. Image from [8].

Figure 2.7: An example of a depth map. A. One of the images from the stereo pair. B. The extracted depth map. Images from [2].

Using the fundamental matrix, it is possible to narrow the search space for a point from one camera in the other. This is called the epipolar constraint, and is illustrated in Figure 2.6. In this example, the point marked in space, X, visible at X_R and X_L, in the right and left images. The epipolar line, marked in red in the right image, is the only place where the point X_R could be found in that image.

In order to make using the epipolar lines easier to use, the image can be rectified, which causes these lines to be parallel. Once rectification is done, the epipolar line for the point (x_L, y_L) will lie along the line at $y_R = y_L$, making it easier to find the corresponding point in the image. This process can be done fairly easily using a simple preprocessing step.

Generally, the product of a stereo vision system is a depth map. An example of one of the images from a stereo pair and the corresponding depth map is shown in Figure 2.7. The gray value of a pixel indicates the depth of the corresponding pixel in the image. Two different types of depth maps can be generated: Dense and Sparse. A dense depth map contains data for all pixels in the source image, whereas a sparse map only finds depths for the regions of interest in the image.

There have been several approaches to extracting these maps. The most basic

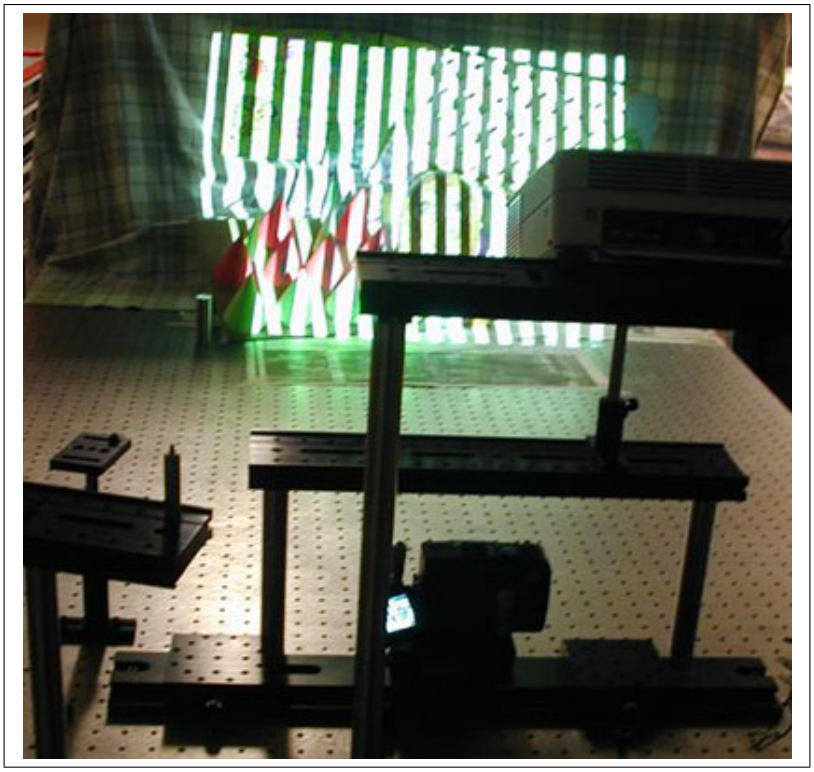

Figure 2.8: An example of a structured light pattern setup for depth map extraction. This setup provides good results, but not conveniently. Image from [2].

involve a simple search of all pixels in the other image to find the best match using an energy function, such as the sum of squared differences, usually searching along the epipolar lines to reduce the search space.

The easiest enhancement to this technique is to use a projection of structured light on the scene [42]. This simplifies the process of finding the correspondences between the images, by providing a known illumination pattern that can be viewed in both images. However, this does require using a projector to place a pattern onto the scene, which is not really feasible for use in a system for general use. However, this does provide a good depth map, and so is good for static scenes and academic applications. An example of this type of setup is shown in Figure 2.8.

Another enhancement is to use several versions of the image at different sizes [29]. This allows an approximate depth to be found in a smaller image, and then that

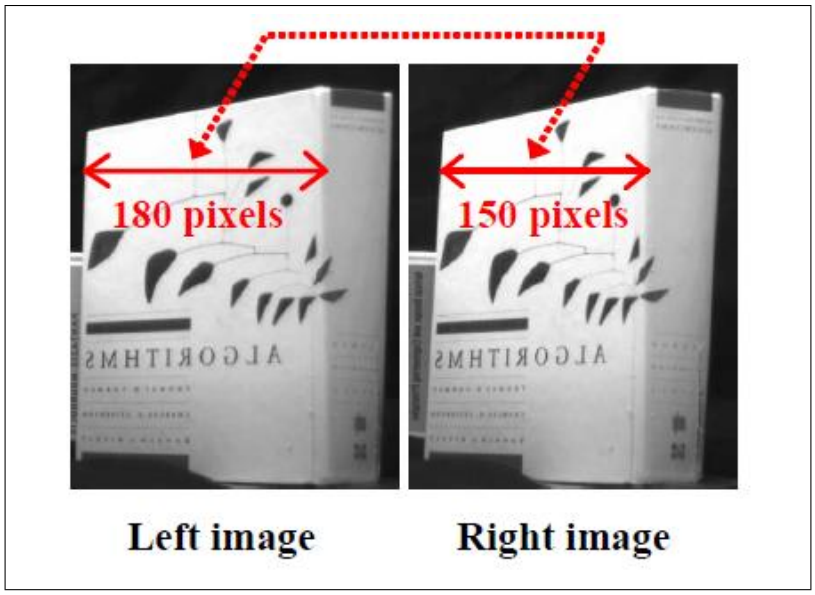

Figure 2.9: An example of perspective distortion caused by viewing an object from two different points of view. Image from [33].

estimate can be used as a baseline to find a better match in the next size, within a smaller search space. This method reduces the required search space for the match for each pixel by several orders of magnitude.

A completely different approach to stereo vision was proposed by Ogale, et al. [33]. In this method, the rectified images are offset and compared, and regions in the two images which represent the same object will overlap. The depth is determined by the offset at which the largest blob is present at any given pixel.

The most challenging aspect of this approach is dealing with the changed perspectives. If an object is not directly parallel to the view plane, the object will be slightly longer in different views, as seen in Figure 2.9. This can be corrected by stretching one of the images. By stretching the image, the matching can be thought of as more of a process of matching segments or regions then individual pixels, which allows the matches to be more continuous. This method is highly effective, and provides very good accuracy, however, it is too inefficient for use in any kind of time sensitive application.

2.4 Person Tracking and Modeling

The basic premise of person tracking is that it is possible to extract additional information about the state of a person from images of that person. This information includes:

- Position of the person in the scene.
- The orientation of the person's head relative to the screen.
- The orientation of the person's body.
- The position and orientation of the person's appendages, relative to the rest of the body.

While some of this information is relatively easy to obtain (the position and orientation of the body, for instance, can be solved using standard computer vision techniques), some of it is extremely difficult to extract from just the video images.

The rest of this section will focus on the techniques used to extract this information from images of people.

2.4.1 Shortcuts: Marking your model

Some of the earliest work in this area attempted to shortcut the hard problem of locating and tracking the features of the user by requiring them to wear a specially marked glove [21], or have markers attached at the various joints [40]. While these techniques place more of a burden on the user, and are therefore less desirable then methods which do not require these things, they provide the advantage of not requiring large amounts of upfront work on image processing to build an interface, instead

requiring only relatively simple image processing and object tracking of small, easily identified blobs.

Davis and Shah [21] put together one of the earliest systems to use this style of shortcuts, using a dark glove with bright fingertips to highlight the key features. Images of this glove could be easily analyzed for the bright spots which signified the fingertips, and the relationships between these spots could be used to determine the gesture the user was performing. This system used various hand configurations as "gestures" to control a robotic arm, such as holding the hand in an American sign language "L" sign to move the arm left. The hand position was determined when the hand did not move between 3 consecutive frames, and the "gesture" was treated as active so long as the hand did not move. This system recognized specific gestures by using a set of vectors and magnitudes, which represented the movement direction and amount for each finger. These were compared to a library of gestures to determine the command.

Ringer and Lasenby [40] marked the joints of interest on their subjects, to allow them to ignore the feature extraction problem and focus on the gestures. This system uses the positions of the markers in each of multiple cameras to determine the state of a model of the limbs in question. Note that this model is also used to determine some of the possible occlusions of the markers on the subject. The system uses two different filtering systems to determine the most likely correspondence between markers and joints.

While both of these methods are interesting, the use of markers on the subjects to facilitate tracking makes them very different from the other methods described below, as well as less desirable, due to the extra step of having the user put on and take off the markers.

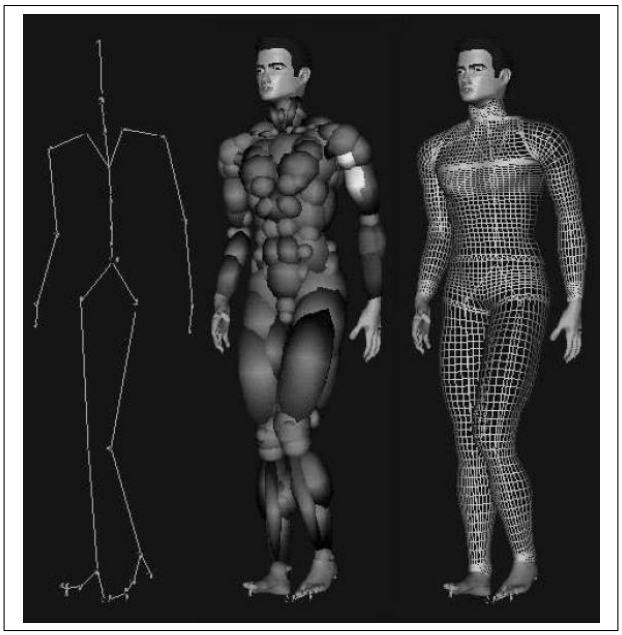

Figure 2.10: Example body models. From left, a skeleton, a ball based model, and a mesh based model. Image from [36].

2.4.2 Extracting 3D model state from video

Most of these systems work with a 3D model of the human body, such as the ones in Figure 2.10. These model-based systems work by building a model of the portion of the body of interest, then using the images of the body to determine the state of that model. Once this is done, examination of the state of the model, as well as past states, can give some indication of what actions the user is attempting to initiate.

The earliest such system was developed by O'Rourke and Badler [34]. This system worked through a cycle of stages:

- Image processing: The image is analyzed for the features of interest within the areas the system thinks they should be in.
- Parsing: The locations of the features from the image processing stage are used to determine the motion of the feature over time.
- Prediction: The system uses the motion over time to predict the change in the

model of the person for the next frame.

- Simulation: The model is projected into image space to determine where to look for the features in the next frame.

This style of observe, model, predict, repeat is common to many of the later algorithms based on a 3D model of the person.

To determine the position of the various body parts, the system uses certain knowable constraints on the motion of the human body, such as distance limitations, joint angle restrictions, and acceleration limits, to determine the possible locations of the feature in question within the image. After this is done, standard vision techniques can be applied to find the item in the image. The biggest issue with this system is that it was not tested with real human input data. It was instead tested with synthetic images of humans.

A later system, developed by Akita [16], was designed to work on some of the shortcomings of the above system. It worked with images taken from film of a gymnastics exercise. The system first initialized by determining which parts of the image corresponded to which parts of the person. These parts were then tracked as they moved through the images, and the model was used to predict the location of the objects in the case of occlusions. This system dealt with some of the shortcomings in the system by O'Rourke, mainly by simplifying the process and the constraints.

One more modern implementation of this is the DigitEyes system, developed by Rehg and Kanade [37]. This system works by building a kinematic model of the hand, then estimating the angle of each joint from video frames being given to the system. However, one of the major limitations of this system was its inability to handle self-occlusion. Overall, this system is just a modernized version of the earlier techniques, with a focus on hand modeling, instead of full body motion.

An improved version of this system, also by Rehg and Kanade [38] handles self-occlusion through using the kinematic model to predict the occlusions and determine the visibility order. The system uses knowledge of the model and knowledge of how it can move to determine which parts can block which other parts and in what way, and can use this to derive the state of the model even in cases where this occlusion occurs. This is one of the earliest systems to handle self-occlusion in a reasonable way.

An alternative system for handling occlusion was presented by Kakadiaris and Metaxas [28]. This system works by using three mutually orthogonal cameras, which have their input all fed to a single system. This system then uses the occlusion contours of the human body to determine which cameras will have relevant information about the body part in question. Once the cameras are determined, the views from each camera are used to construct a model. This system has an initialization step which is used to construct the model of the person, allowing it to more accurately determine the occlusions.

Plänkers and Fua [36] proposed a system for extracting the state of what they call an "articulated soft object", such as a human or animal body. The system works by using a simple skeletal model, to which a number of metaballs, representing various bits of the body, are attached. A smooth surface is then built over these metaballs, and this surface is used for the state. Because this model is built from the observed subject, it can be used to more accurately predict occlusions. The model is built using a combination of standard stereo vision techniques and the silhouette of the user.

While most of the above systems use the model as a predictor of where to look for the features in the image, the system by Kakadiaris uses the model to decide which images to look at. Otherwise, most of the model-based systems handle the

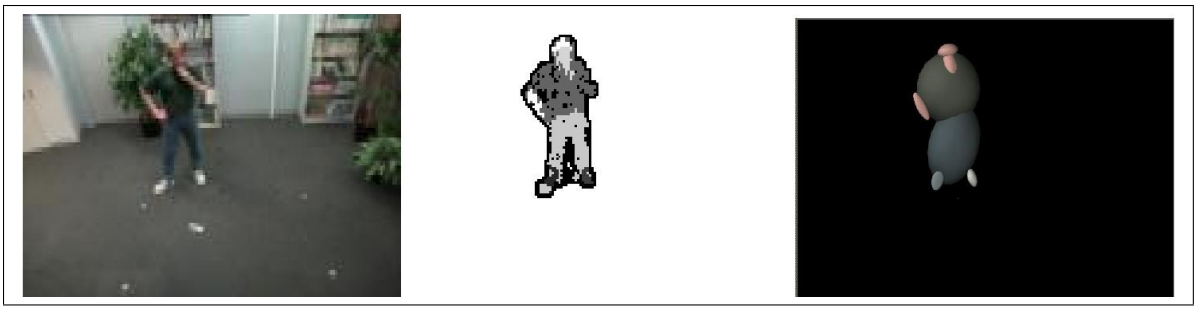

Figure 2.11: A strictly blob-based person tracker. Pfinder identified the regions of the image where a user was and tracked them. Image from [47].

general approach the same way. Plänkers and Fua, however, did not use the model for this, as their primary goal was just in building the articulated model. The model articulation parameters were defined based on the movements after this, not the other way around, as the other systems do.

Also, the Kakadiaris and Plänkers systems are the only ones which explicitly initialize the model from the actual subject, instead of fitting the subject to a general case model. While both of these systems use this initialization for essentially the same purpose, namely improving the ability to determine the occlusions, both handle it in a different fashion. While the model in Kakadiaris is initialized as a preprocess, the Plankers system handles the refinement of the model over the course of the tracking, bypassing an initialization phase.

2.4.3 Non-3D methods

Another class of systems used for this task do not really work with a 3D model as the above systems do, but rather stick closer to the input images. These systems work by any number of other techniques, but generally do not require or build a model of a person in order to interpret the motion.

An example of this type of system is the Pfinder system, developed by Wren, et al. [47]. This system works in a strictly 2D environment, finding the blobs of color which

correspond to the head, hands, torso, arms, and legs of the user. These blobs give the position and movement of the user over time, and this can be used to determine user actions and gestures. This system is shown in figure 2.11.

Another interesting system in this category is based on Temporal Templates, and was developed by Bobbick and Davis [20]. A Temporal Template is simply a description of the motion of an object over time, represented as a sequence of "motion images". These images define the portion of an image which has movement occurring, such as all the pixels that change when a hand is waved at the camera. These are accumulated over several frames, with the most recent motion being more intense then the older motion. Based on a series of these images for a particular action, it can then be identified from a video feed by building a motion image in real-time, and comparing that to the known gestures in the library. An example of these templates is shown in Figure 2.12. This system, and others like it, are less concerned with the absolute position of the user, and more with what they are doing.

Another system, which operates entirely on the source images, was developed by Jepson, et al. [27]. This system works by defining regions of the image which represent the same object. These are called "polybones". They will be shaped as irregular octagons, and will be reshaped as needed to match the region of the object. These shapes are stored in a layered fashion, allowing the system to deal with occlusion of objects in the scene.

Additionally, a system developed by Ricquebourg and Bouthemy [39] uses "Spatio-temporal image slices" to derive information about the motion of a person, without the need for a 3D model. This system works by taking a series of images and determining the contours of the moving objects within them. These contours can then be used to determine the movement of the people of interest within the video frame.

Among these systems, there are few similarities. The temporal template system

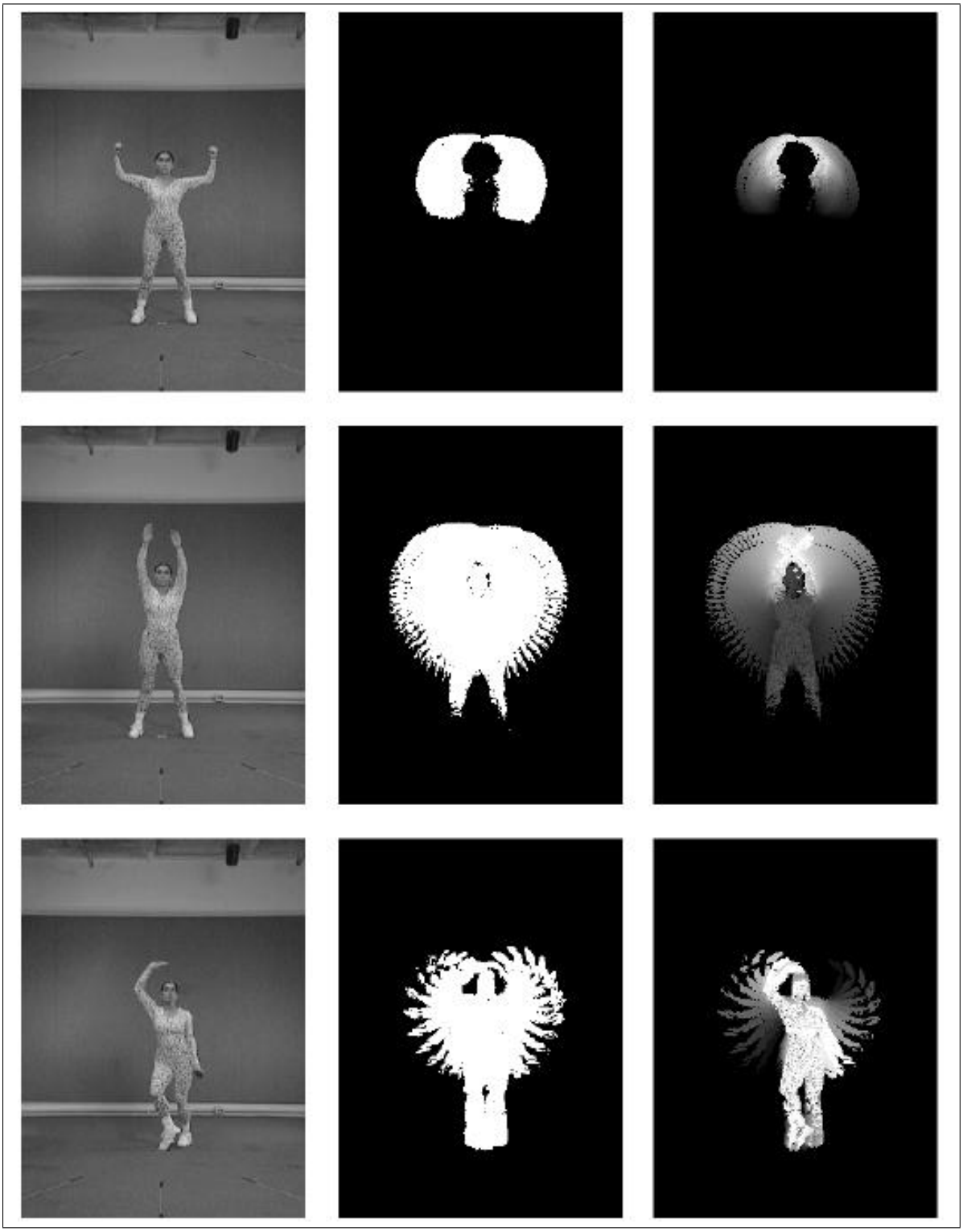

Figure 2.12: Temporal Templates - A gesture identification system.

and the spatio-temporal image slices systems are similar in that they are primarily concerned with determining where things are moving and how that affects the scene, however the approach to recognizing actions is very different. Where temporal templates compares the current state of motion to known templates, the spatio-temporal compares the actual motion to known patterns.

Pfinder and the polybones system discussed here also have some similarities. Because of the focus on isolating blob features in both and using those to determine user motion, they both share some of the same computer vision algorithms. However, the use of the information, and even the blobs of interest are very different. Also, pfinder does not do well with occlusions, unlike polybones, which can handle partial occlusions thanks to the layered approach it uses.

2.4.4 Hybrid Approaches

A hybrid of the 2D and 3D approaches discussed above was built by Mori and Malik [31]. This system works by attempting to match the input image to a set of predefined images with marked "key points", representing the joints of interest. Once matches are obtained with the least error, the system uses these points in the image, and the distance between them to construct a model of the users pose. This approach works very well with enough premarked data for the system to use in determining pose, however it sometimes will mix up the right and left sides of the body.

A second hybrid approach, developed by Azarbayejani and Pentland [18] uses the blob based approach of pfinder, but uses the located blobs to determine position in 3D with a stereo-vision based system. The system works by first identifying the blobs for the head and hands in two separate 2D images, then, through a series of projections and rotations, determining the correspondence. Using this method, the system also self-calibrates the stereo setup, meaning that the user is not required

to provide any additional input about the locations of the cameras in order for the system to accurately place the user in three dimensions.

Jennings [26] approached the problem by simply doing everything, and trying to get all the approaches to agree. He built a system which uses 7 distinct methods to try and identify the finger, then tracks it in real-time with the 7 methods. By combining stereo vision, edge detection, color segmentation, and other techniques, he was able to easily track the finger in 3d space. This system combines many different 2d techniques to produce a 3d model.

An additional system by Segen and Kumar [43] uses a single camera and a point light to determine the state of the hand. It does this by examining both the hand and the shadow it casts on the table, and determining the state of the hand from these two inputs. It can place the hand in 3 dimensions by using the distance from the shadow to the hand to determine the distance from the tabletop. This system is simple, if somewhat constrained by the need for the bright light over the table. However, it is effective, operating at 60 frames per second and tracking the state of the fingers easily.

These hybrid approaches have little in common with the other approaches listed above, for the most part. Azarbayejani's system, while similar to pfinder, is very different because of its use of stereo vision. It does, however, have some similarities in the self-calibration methods to some of the other model based approaches. Mori's system is a significant departure from most of the other systems discussed here, and the use of multiple key frame images with manual marking would lead to it being a less popular approach to the problem, in spite of the good performance and accurate modeling. Jennings' system, while effective, is somewhat computationally expensive to find widespread adoption, but does provide excellent results.

2.5 Comparison to this project

Our approach to depth map extraction is the most basic approach, using some methods adapted from Koschan and Rodehorst [29], however, we decided against using a structured light system, as that approach requires multiple images of a static scene for good results, and would greatly inconvenience a user who had to stand in the projected light pattern. With a dynamic scene this is not feasible, so we decided to only use the image pyramids. Aside from that, our implementation is surprisingly basic, using the GPU to make this basic approach work fast.

Our person tracking system is probably most similar to pfinder [47], with the addition of 3d information extracted from the depth map. We are not using a full 3d model, as this was decided to be too complex for this project. However, there is no reason that the depth maps generated by our implementation could not be used in support of a model based user tracking system, like those described above.

Chapter 3

Algorithms

Our approach consists of three basic steps, which are shown in Figure 3.1. The first step, preprocessing, prepares the images for use in the remaining steps, which is described in section 3.1. The depth map step does the hard work of the stereo vision, and is described in section 3.2. The final step is the tracking step, which is described in section 3.3.

3.1 Preprocessing

The preprocessing step uses several passes to do cleanup of the images. These are basic denoising, undistortion, and rectification.

The basic denoising process eliminates some of the noise in the images, to allow for better matching and tracking of objects. This process, however, is not perfect. One of the biggest downsides of using the commercial web cams, as we have done here, is that these typically have lower image quality than more expensive cameras, meaning that the noise is harder to eliminate, especially in low light situations. An example of this noise is shown in Figure 3.2, which should show an object of uniform

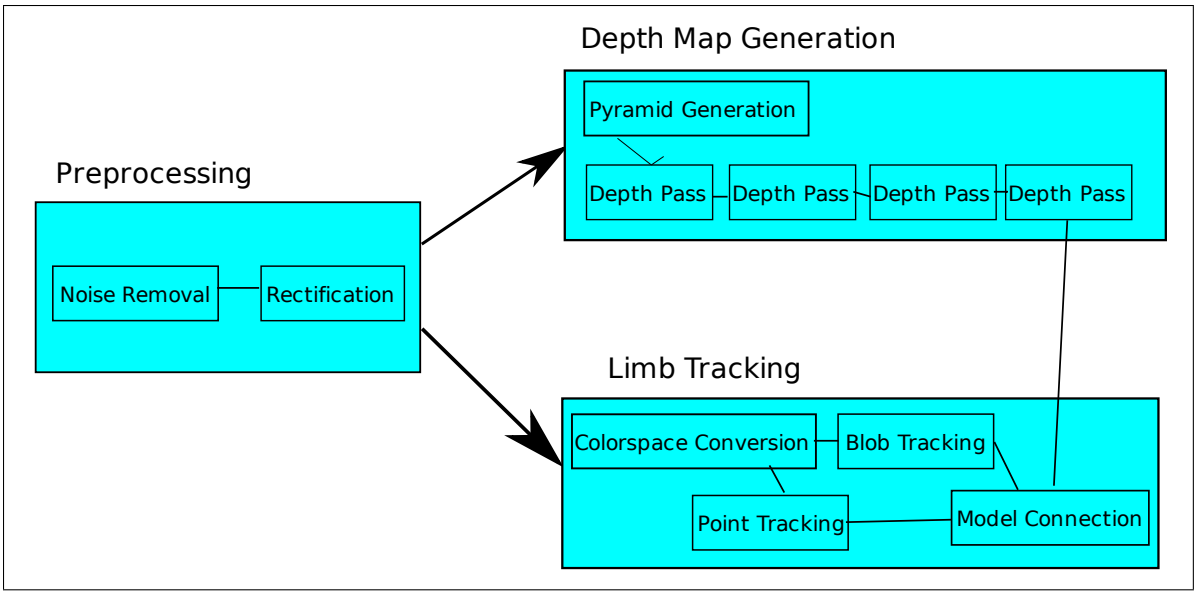

Figure 3.1: The data flow. The three basic parts are shown, along with their sub-components.

color, with a small amount of shadowing. However, you can see the pixels which are off color easily in the resulting image. As a result of this high noise level, multiple passes are made to try to eliminate this noise.

These noise removal passes consist of a median filter followed by a Gaussian filter. A median filter sets each pixel to the value of the median pixel in the 3x3 neighborhood surrounding a pixel. This type of filter very effectively removes salt and pepper noise, which is very prevalent in the images generated by most web cameras. It is also a good method as the output color is guaranteed to be from the input, unlike averaging or smoothing filters. The downside to this method is that extremely noisy areas still tend to have noise.

A Gaussian filter works well to attenuate noise in an image, by applying a smoothing function which puts extra weight on the pixels closest to the center of the filter. This type of filter, while it does reduce noise, has the unfortunate side effect of smoothing out features in the images, such as edges. However, since this project does not use edge detection for anything this downside is acceptable. The other downside to

Figure 3.2: Image noise. This is a blown up section on a shirt, and should be largely the same color. However, there is a significant amount of variation.

this filter is that it allows noise to affect nearby pixels, spreading the noise somewhat. However, since this noise is also attenuated, this is less of a concern.

Following the cleanup, the images are rectified and undistorted, as described in the background section and literature. This step corrects for the lens distortion and the camera positioning, making the process of searching for matches and generating the depth map easier. The output from this process is used as the input for the next steps, as seen in Figure 3.1. While thought of as one step, this is actually two separate processes.

Undistortion is done to correct for the lens distortion. On most lenses, this distortion is a simple radial distortion, usually barrel or pincushion distortion, as described in the background. This type of distortion causes straight lines in the scene to not appear straight in the final image, and must be corrected for the depth map extraction to work correctly. The coefficients for this correction can be determined offline, and simply stored for later use. This is implemented as a simple transformation of the texture coordinates per pixel, based on a well known model, producing a transformed image for use in later steps. More information on correcting lens distortion and distortion models can be found in Wang, et al[45].

Rectification, as described in the background, is used to make the epipolar lines in the two views parallel to the x axis, and arranged such that the epipolar line for the point (x_R, y_R) lies along the line $y_L = y_R$. This process is important as it simplifies the search for matching regions in the depth map extraction step, by making it a simple linear search, instead of a search on the whole image. This process, like the undistortion, is also implemented as a simple transformation of the texture coordinates per pixel, based on the fundamental matrix. As such, both of these steps are easily implemented in a single pass.

3.2 Depth Map Extraction

Normally, with rectified images, the process of generating a depth map consists of searching along the epipolar line for the match for each pixel. In 640x480 images, this consists of checking each of 307,200 pixels against all 640 pixels in the matching line, for a total of 196,608,000 checks, where each check requires comparing a neighborhood of 25, 49, or more pixels, centered on the pixel being searched for. Obviously, this would not be feasible to do in real time with current hardware, though it may be in a year or two.

To deal with this, our depth map extraction step consists of a pyramid generation pass, which generates smaller textures for processing, followed by 4 depth map extraction passes, each working on successively larger textures, and using the previous pass to narrow the search window for the match.

The pyramid generation step produces 3 textures, sized at 1/2, 1/4, and 1/8 of the original texture along each dimension, by averaging the pixels in a neighborhood. Each of these is used as input to the successive depth passes. By generating these smaller textures, searches for matches can be done on smaller amounts of data in the smaller images, then refined in the larger images.

This is based on a simple concept, called image pyramids[17], which was first proposed by Anderson et al in 1984. This technique has been applied to many different image processing problems, and it works well in this case for reducing the search space and improving the results. By matching on the smaller images first, larger amounts of the input affect the resulting match for each pixel. Thus, if two similar structures, such as the human eyes, would match one another in the full size image are surrounded by different structures, their matches in the smaller images will be affected by those surrounding structures, causing them to be matched to the correct places. This creates a more continuous depth map, which is more useful.

In order to find a match for a pixel, the neighborhood surrounding the pixel will be compared to the neighborhood around a potential match. As multiple pixels on any given scan line are likely to have similar colors, larger neighborhoods are used to get better match results, and also to reduce the effect of noisy single pixels on the results. Based on experimentation, 7x7 pixel regions were found to give the best trade off between accuracy and speed for this project.

The two regions are compared for similarity, using a method proposed by Birchfield [19], which compares the images in such a way that the sampling does not hamper the matching process.

Given any 2 images, the pixel sampling between them is unlikely to match up exactly. As a result, any comparison must check against the colors in between the pixel being checked against and its neighbors. So, when comparing the pixel at x_L to the pixel at x_R, it is a good idea to also check against the position between x_R and $x_R - 1$, and the position between x_R and $x_R + 1$. Of course, these in between values don't exist in the image, so they are determined by linear interpolation, giving the values x_R^- for the lower value and x_R^+ for the higher value. The best match, among the colors of x_R, x_R^- and x_R^+, is then used as the match value. In this way, if x_L would not have a good match because the sampling in the other image does not include it, a good match will still be found.

This approach to pixel similarity is good for this project since it provides better results then a simple sum of squared differences, however, it does have some disadvantages. It can still be influenced by noise, which could give bad results. It can also match a pixel to any similarly colored pixel anywhere on the row, which is why neighborhood checks are still required.

The first depth pass deals with the smallest image, finding the best match for a small window surrounding each pixel along the entire width of the image. The position

of this match is then used as an input to the next depth pass, which will search a neighborhood of $2^{n-1} * D_T$ pixels surrounding the position found on the previous pass, where n is the pass number (from 2 to 4) and D_T is an adjustable parameter, controlling the size of the window. An example of the match search space is shown in Figure 3.3. This method has been adapted from Koschan and Rodehorst[29], and has proved to be very effective and efficient.

This pyramid based approach was chosen because it significantly reduces the search space for each match, making it possible to find a match for each pixel in the constraints of real-time. The number of matches to check for a 640x480 image in this fashion is only 9,196,800, which is a significant improvement over the 196,608,000 required for an implementation that does not use image pyramids. The downside to this approach is that errors propagate. A mistaken match in the first pass will cause bad results in later passes, and over larger areas.

The biggest downside to this approach to generating a depth map is that it is not completely accurate. For example, one pixel can not be matched to two points in the other image, however this constraint is not enforced. In cases of two similar points being matched to the same point, one of them must be wrong. Another constraint which should be enforced is the continuity constraint, which causes flat objects to be matched up consistently, by ensuring that pixels which are part of the same continuous object will be matched up to the same continuous chunk of pixels in the other image. While we do not attempt to enforce this constraint, the pyramid method we used for finding the matches helps with this problem. Because the depth found with the smaller image is used as an estimate for the next largest one, objects which are successfully matched in the smaller image have a good estimate, which needs only minor refinement in the following passes. Since this estimate covers a few pixels of the next largest image, these pixels are already continuous.

Figure 3.3: The search space for matches, at each level.

3.3 Limb Tracking

Our approach to limb tracking is handled through 3 passes on the GPU, followed by a small amount of post processing on the CPU to put the new positional data in the arm model. These passes each serve a different purpose.

The first pass is a color space conversion. This pass converts the image to a HSV image. The HSV color space defines a color as three values: Hue, Saturation, and Value. Hue is the shade of color, Saturation is the amount of the color, where lower values are more white, and Value is the brightness of the color, where lower values are more black. One of the most important properties of this model is that colors which appear to be similar shades have similar hues, as opposed to in RGB, where small shifts in one value can cause the shade to change drastically.

This pass is important due to the properties of HSV. Skin tone, and many other colors, cluster better in HSV compared to RGB across varying lighting conditions. This allows the blob tracking to track the user better when the light across the scene is not even. An example of this color conversion is shown in Figure 3.4.

The second and third passes work in tandem by tracking blobs and key points from the user across multiple images to keep track of the arm. The blob tracker consists of the following steps:

- Mark all pixels which are within the blob, by color. The marked data consists of: (1, xpos, ypos, hue).
- Marked pixels are eroded, which eliminates random noise pixels which happen to match the blob color, so they do not affect the results.

- Sum the buffer containing all marked pixels. This gives a single pixel containing (number of blob pixels, $\sum xpos$, $\sum ypos$, $\sum hue$).
- Divide these values by the first value, giving (1, average x, average y, average hue). These give you the (x, y) of the blob and the average hue, allowing the system to track the users hand across dynamic lighting by getting an updated color for the blob, along with the position.

The erosion process used in this process is fairly straightforward: every pixel with a non-matching neighbor is discarded. In this way, if a single pixel, or even a small group of 4 or 5, show up in the blob by mistake, they are removed. This eliminates bad data from the blob, but it also removes the edges of the blob, as these pixels have neighbors which are not part of the blob. This is not a problem, however, as the blob still has the same shape and center with or without these pixels, giving a good position for the tracked object.

In order to efficiently track a users arm, the three blobs are tracked in parallel, rather then in series. We decided to do this in one pass with the three different colors, which produces three separate blob textures, each containing one of the tracked blobs. Once these three textures have been processed, the data concerning the blobs can be read back from the textures.

An example of using this blob tracking on a hand is shown in Figure 3.4. The blob is drawn in white in the bottom left of the image. Sections that are white that are not part of the hand are excluded by a continuity constraint. Only those parts that are continuous with the tracked blob are used for determining it's size and position. The rest are ignored by the summation.

We decided to use SIFT for the point tracking, which has a GPU implementation already available at [48]. SIFT produces a set of feature vectors, which contain information about the features which is not affected by translation, scaling, or rotation,

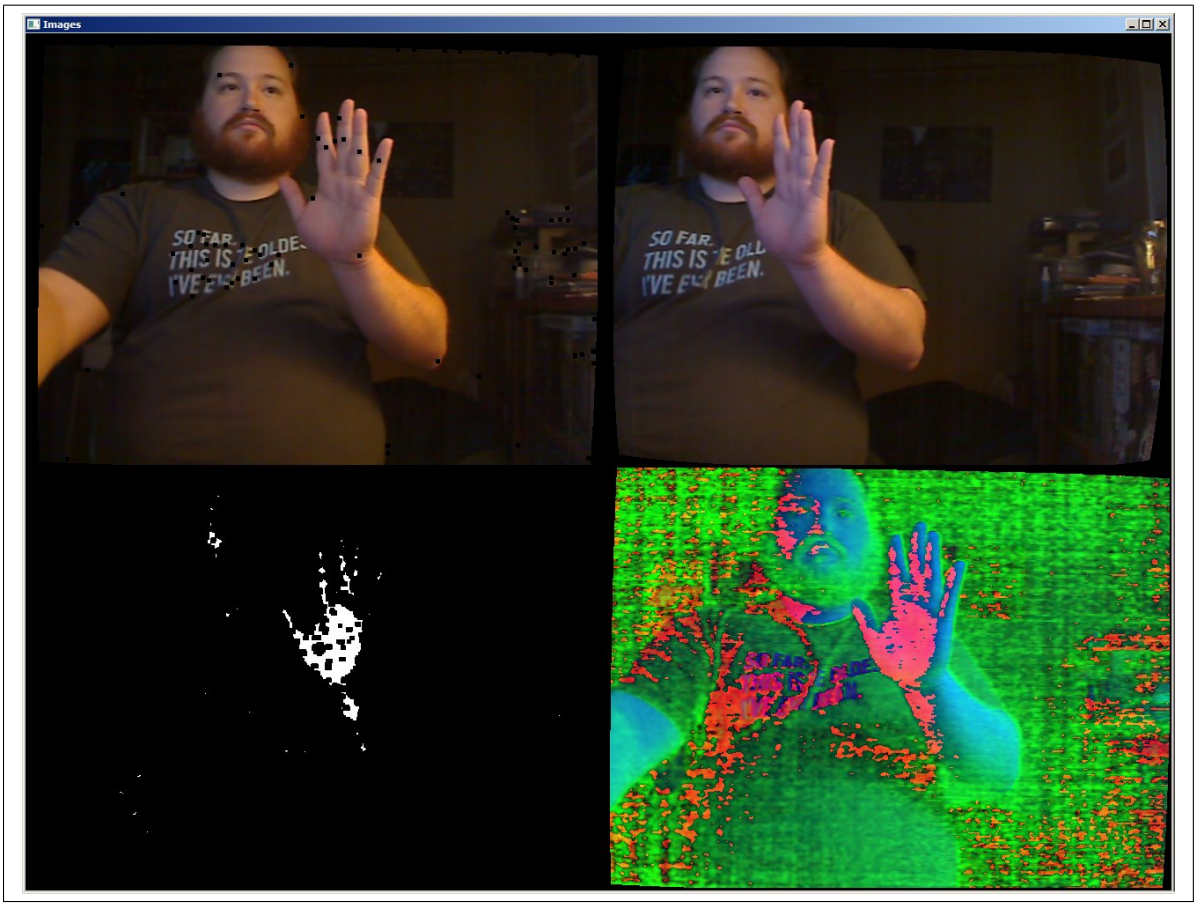

Figure 3.4: Using the blob tracker to track a hand. The hand shows in white in the image in bottom left, while the bottom right shows the color converted image, with HSV being drawn as RGB.

and which is only somewhat affected by illumination changes. Further details on this algorithm can be found in Lowe[30].

SIFT was chosen because it is a reliable method for extracting points, and because this GPU implementation was already available. This implementation works well for finding points in the image, which can be used for tracking, and is very efficient.

The reasoning behind using two separate tracking methods, as done here, is to allow the tracking to perform better. Point tracking is effective, however, blob tracking is usually much easier, and is somewhat more reliable for smooth objects, such as a shoulder, which don't have any interesting points. Also, point tracking can sometimes identify image noise as a key point, which is a problem not found with blob tracking. As a result, the blob is used first, and the points are used to double check the results or adjust when dramatic light changes throw off the blob tracker. The downside to this is the time spent on it, however, the time spent on this is small compared to the time spent in the rest of the system.

Lastly the points are matched with their previously tracked positions, and the model is updated. At the same time, the depth of each point on the model is read back from the depth buffer, providing a full three dimensional position for each part of the limb. The area of the limb which is tracked, when first selected, has a color and a set of points associated with it. When the tracking is updated, the points and blob for each portion of the limb are updated. These updated values are averaged to find the location of the point on the model.

An example of this modeled arm can be seen in Figure 3.5. In the top left, the three points of the hand, elbow, and shoulder are marked with green boxes, and the red lines represent the bones connecting these points. In the bottom left, the three tracked blobs for these key points are seen in three different shades, along with other areas of similar colors throughout the image. Also shown is the HSV color converted

Figure 3.5: The tracked arm model, as seen by the user. The points of the model are marked in green, with the connections marked in red.

image, in the bottom right, as discussed earlier. Comparing these bottom two images provides a good illustration of the blob detection process as well.

Chapter 4

Results

The test hardware rig for this system consists of an Intel core i7 processor, an nVidia GTX 260, and two Logitech Webcam Pro 9000 cameras, positioned 5 inches apart, running under Windows Server 2008. While running the application, the CPU showed almost no usage, as most of the processing was being done on the GPU. The web cams were set to run at 640x480 at 30fps, and the application ran at a solid 30fps, matching the input rate from the web cams. One setup for the web cams used in this project is shown in Figure 4.1. This rig was assembled so the system did not require recalibration whenever it was moved, although any other setup could work equally well. Our assembly of the rig with masking tape and pencils helped keep the costs of this project low.

Figure 4.2 shows the depth map and tracked arm, along with the source images that generated this depth map. As can be seen in the image, the user's left arm is being tracked. The user can select either arm by a few mouse clicks, and the arm will be tracked until the tracking is reset. Also seen in this image are the blobs being tracked for the arm, in the bottom right. Each of the three shades represent a different color being tracked, and each shade corresponds to a different blob being

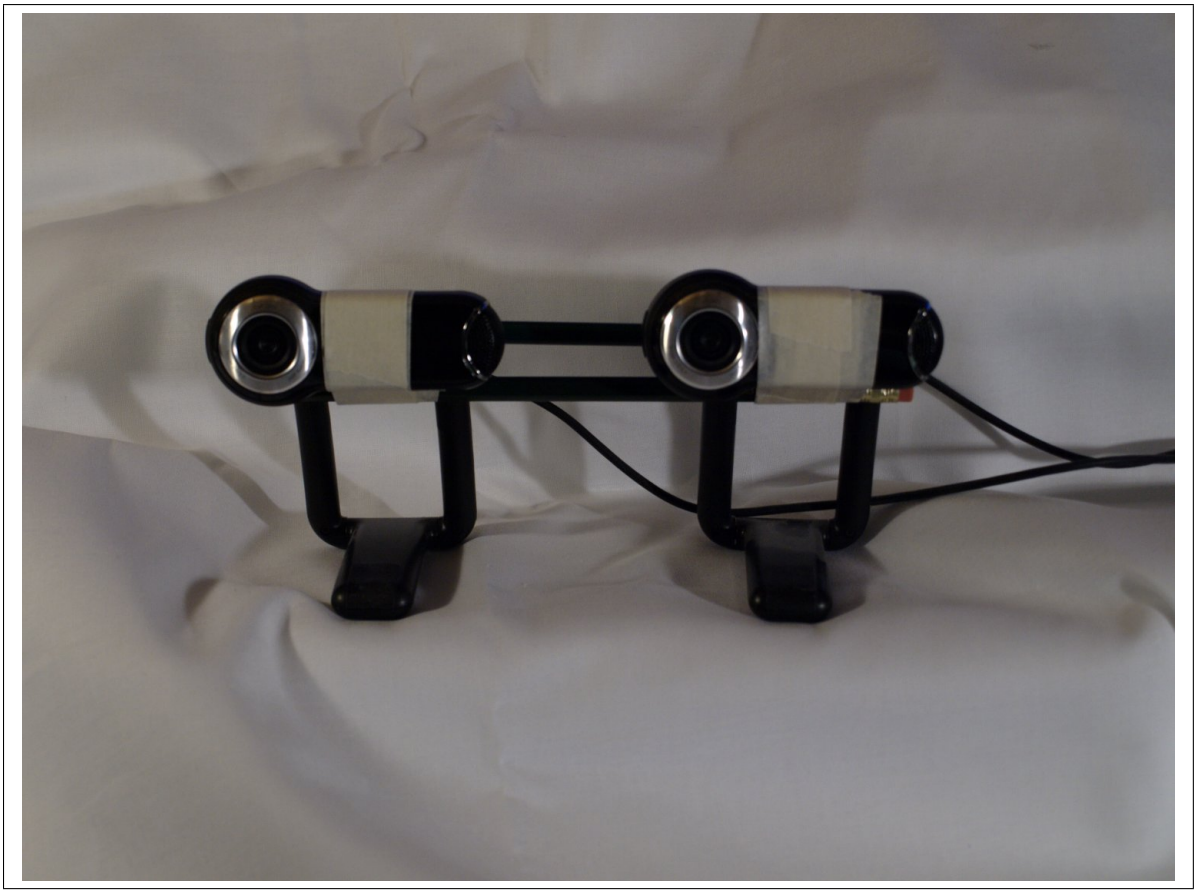

Figure 4.1: A stereo camera rig built out of two web cams, masking tape, and pencils.

Figure 4.2: An example from the system, showing the depth map and tracked arm.

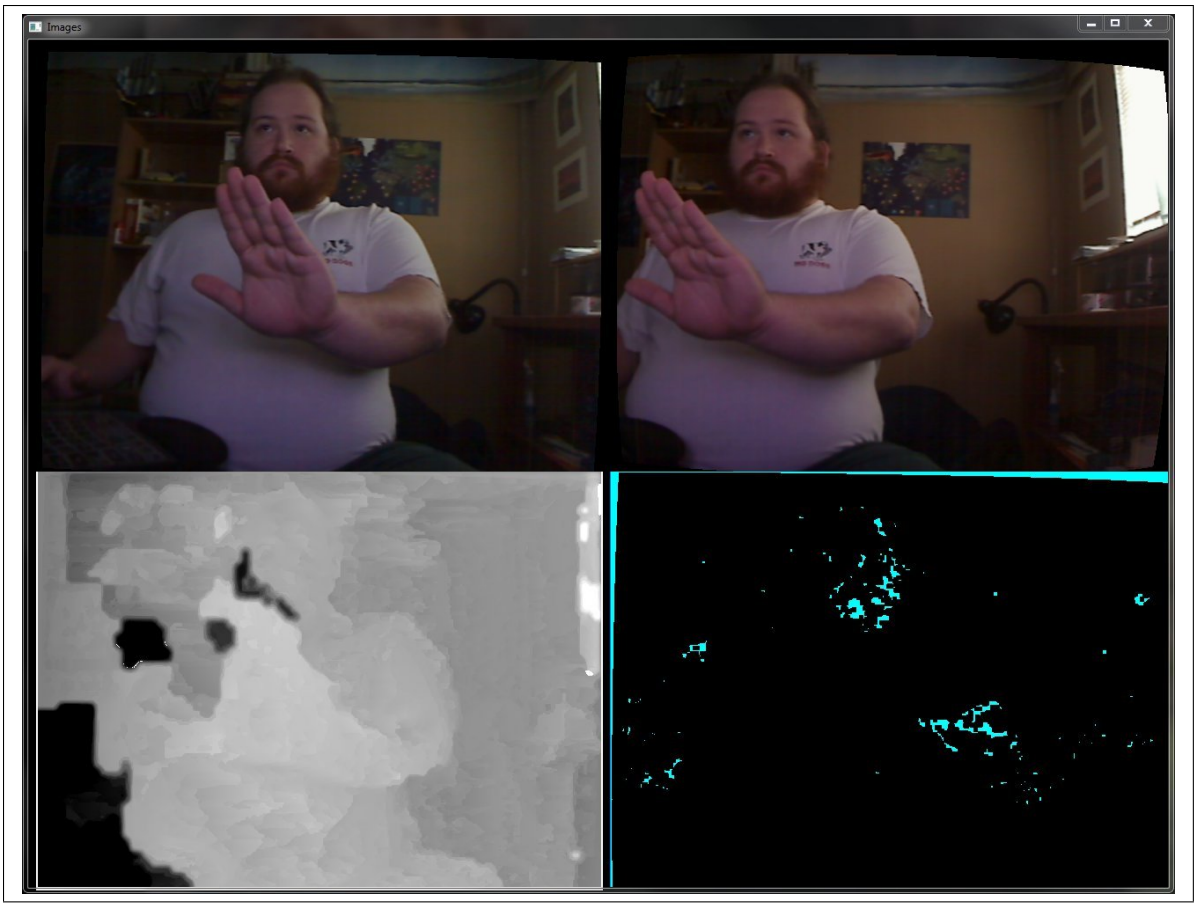

Figure 4.3: Another example from the system. Areas which didn't match due to being out of the frame or due to occlusions are marked in white.

tracked.

Figure 4.3 shows the depth map with some bad areas showing. These areas are shown in white. If there is no match found, the area gets filled in with an arbitrary color, in this case white. There is no match found for these areas because the appropriate area to match is not visible in the second image. In the case of the bottom corner, this is due to that area not being in the image area. On the face, this is caused by the occlusions caused by the positioning of the arm.

Figure 4.4 shows the user with an arm that crosses the depth gradient. With the shoulder towards the back of the gradient, and the arm held straight out, the color in the depth map changes. The hand, which is closest to the camera, is a dark gray,

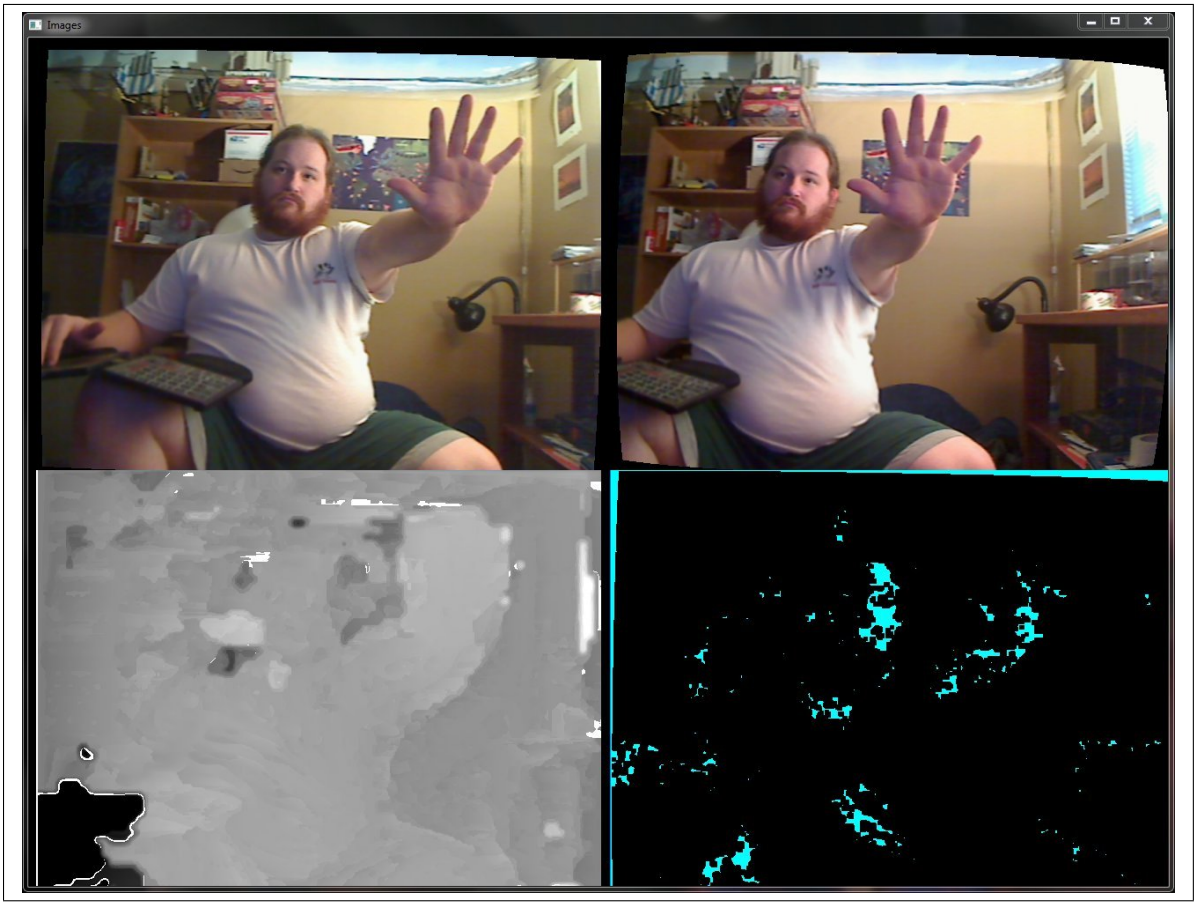

Figure 4.4: Another example from the system. The depth gradient over the length of the arm is visible.

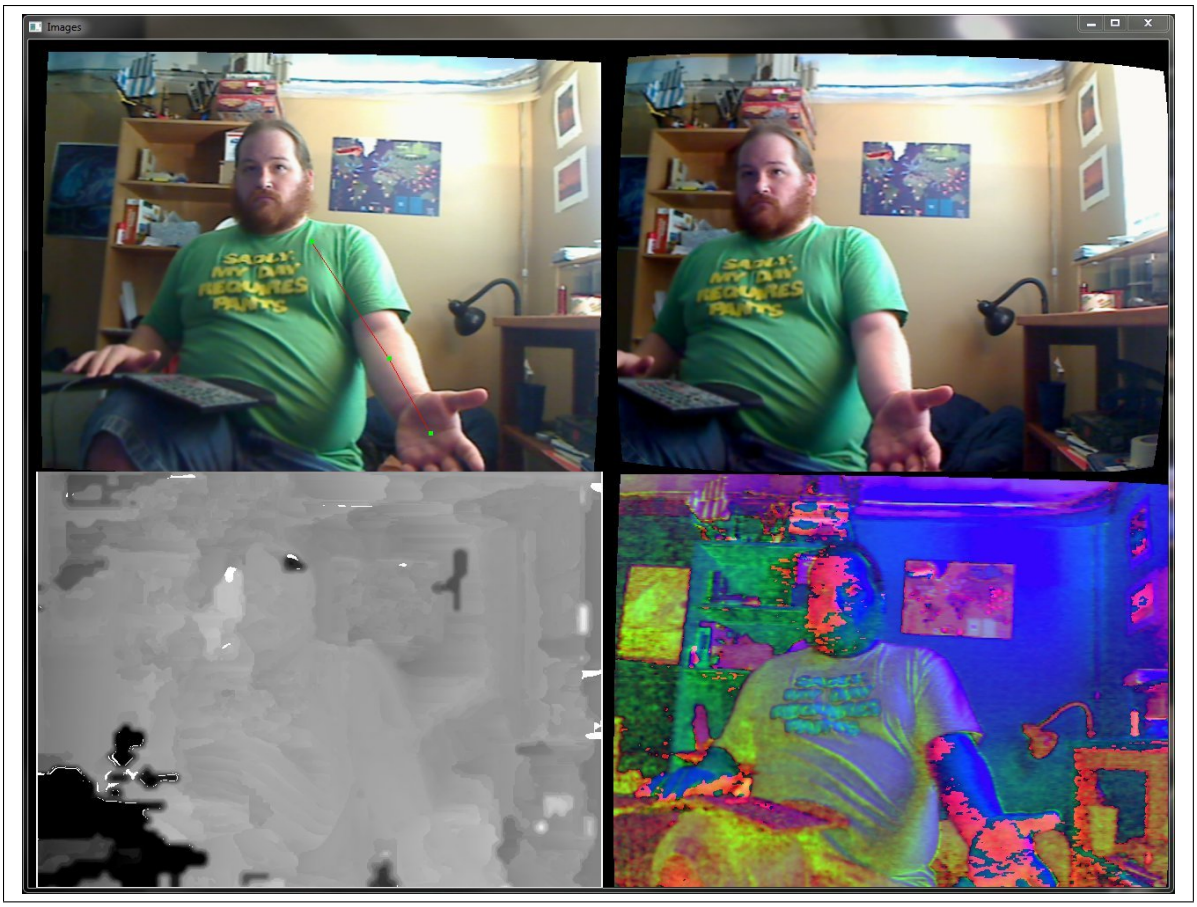

Figure 4.5: Another example from the system. The hand shows in a darker color than the shoulder.

where the shoulder has a lighter shade. The wall behind is an even lighter shade.

One other thing this example demonstrates well is one of the biggest disadvantages of the tree approach we used: fine structures sometimes have bad depth values. For example, the left hand in the image with the fingers spread shows this very well. In between the fingers, where the wall can be seen, should have a different depth then the fingers, but in the depth map, this whole area is a large, dark blob. Because these finer features don't take up much space, they are not matched properly at the lower resolutions.

Figure 4.5 also shows the depth gradient over the arm. In this image, the left hand, which is being held much closer to the camera then the rest of the body, is

Figure 4.6: Another example from the system. The occluded hand has left the system somewhat confused.

shown in a darker color in the depth map. This does not have as much of an issue as the previous example, since the fingers are not visible. Notice that even the three wrinkles in the shirt show up in the depth map, giving a good approximation of the shape of the user. Also, the color change along the shoulder is visible in this image, showing where the user ends and the wall begins in the image.

Lastly, Figure 4.6 shows the state of the system after the user has placed his hand behind his body. This leaves the system somewhat confused, making the data useless. This is a known limitation to our approach to tracking, and can't really be solved without significant additional work on the model. This is discussed in more detail in section 5.

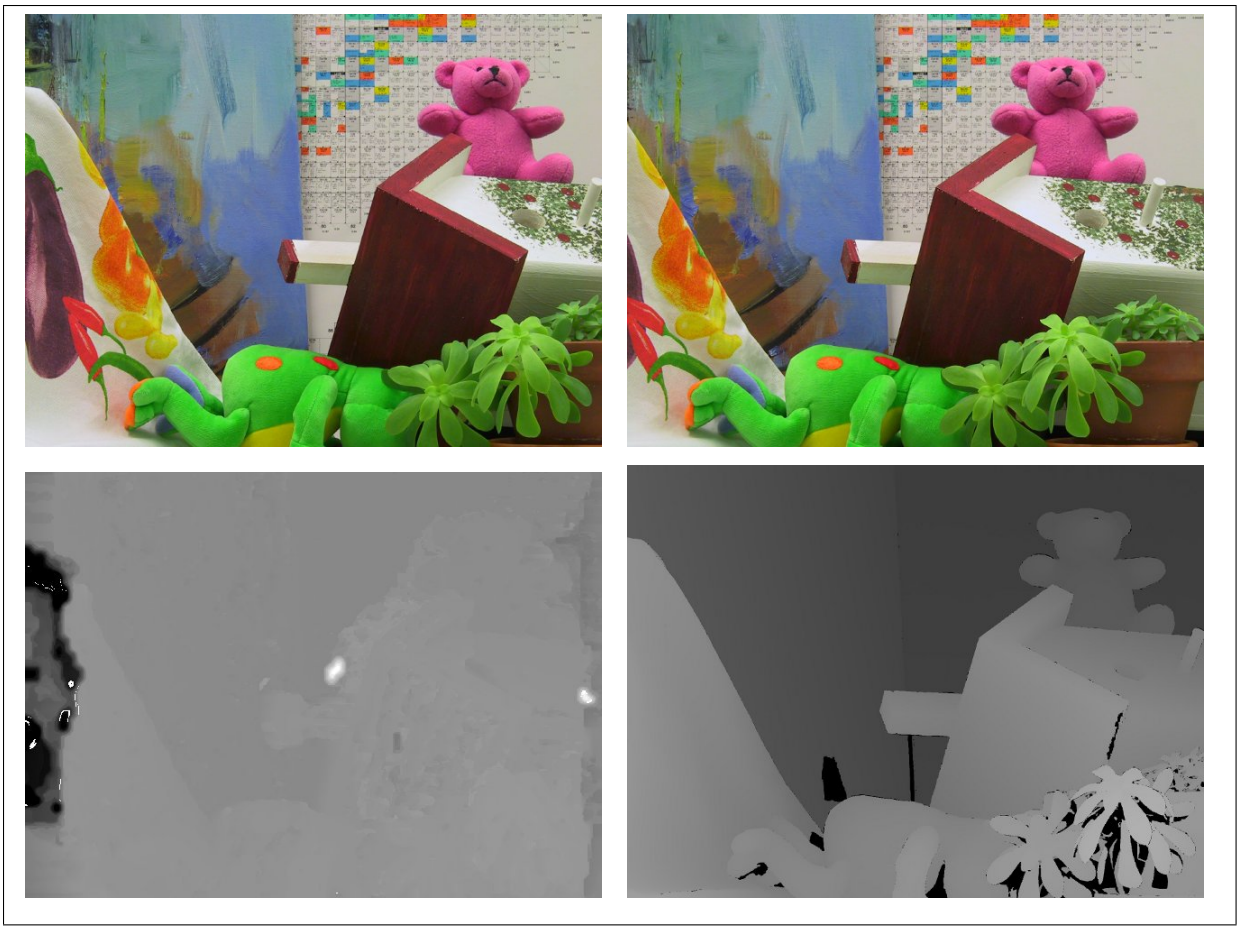

Figure 4.7: A comparison with ground truth. Top row: the original stereo pair. Bottom row: on the left is the extracted map, and on the right is the ground truth map.

While using blob and point tracking makes it easy to track the arm, it does mean that occlusions can cause the arm to be completely lost. Full model based systems, as described in the background, would provide a better solution, however, these methods are more complex, and can not be implemented in the GPU as easily as this approach was. Because of our stated goal of implementing this system almost entirely on the GPU, we decided not to use a more complete user model.

Our results should also be evaluated in terms of accuracy, and so in Figure 4.7, we see a comparison of our results with ground truth. On the left is the extracted depth map, and on the right is the ground truth depth map. You can see there are still a few

nonocc	all	disc	
33.2	32.4	27.6	tsukuba
32.3	32.3	26.0	venus
34.7	34.6	34.7	teddy
34.4	34.3	33.8	cones
32.5			average

Table 4.1: Accuracy. Each column lists the percentage of bad pixels. *nonocc* is the unoccluded regions, *disc* is the areas near discontinuities, and *all* is the total for the entire image.

places where the images do not match. Lastly, a normalization could be done to give a higher contrast, however, we didn't feel this was necessary. For our purposes, these results are acceptable, as the key is speed, not accuracy. If this were intended for 3D model extraction, these inaccuracies would make this system unusable, however, this is good enough for the purpose.

Looking at Table ??, the actual accuracy, in terms of percent wrong, is 32.5%. While this is not a very accurate solution, especially compared to some of the better performing solutions of the last few years, which are as low as 4% [3], it works for our system.

Chapter 5

Future Work

One of the most obvious areas for improvement on this work is to expand the tracking to a full body model, rather then just an arm. This could be done as a simple continuation of the already completed work. This could be done using any full body model, such as the one in Plänkers and Fua[36]. By creating a full skeletal model, more complicated interfaces would be possible. This would likely require changing the tracking system significantly.

Additionally, it should be possible to use this method as a user interface mechanism, rather then a simple tracking project. By taking the data from the application through some mechanism, this system could be used to control a computer, for use in a video game or CAD application. While this was not the primary focus of this project, it would be a good extension. Some examples as in Rehg and Kanade[37] or Wren[46] would be good ideas to try.

One other potential use for this approach could be as a real-time estimate of incoming stereo data, which could then be further processed offline. This would operate similar to the system developed by Rusinkiewicz [41] for 3D scene acquisition. This may not be as useful as Rusinkiewicz's original system, however it could be

helpful when using some type of carriable stereo rig and trying to get models of a full environment. Using this, it would be possible to have a good sense of the quality of the scene data acquired using the rig, before doing the full processing. This would be especially important when trying to acquire data from somewhere far away from the site where the bulk of the processing will be done, and without reliable Internet access, such as in ancient ruins.

One small improvement would be to use the depth map as part of the tracking. By checking to make sure that there are no major depth changes in the blobs, background objects of a similar color to the users clothes or skin would not confuse the user tracking. This was especially noticeable in some of our earlier work, as the test area for this project has oddly beige walls, which are often picked up in the blob tracking for certain users.

One other thing would be to further improve the performance, to allow this system to be integrated into an application more easily. While the performance was good enough, many problem domains would have issues using the system with any significant amount of lag, which could be minimized by improving the performance. This would make using this system as a user interface much easier, as well as making it easier to extend to a full body model. It would also allow it to be used better on older computers. This could be done by experimenting with different implementations of various parts. It could also be done through use of a sparse depth map, finding the depth of only those points where the user has already been found in the tracking algorithm.

Some experimentation could also be done with various other matching techniques and similarity functions, to see if the accuracy could be improved enough for higher-precision work, such as model extraction. This would require only limited changes to the existing implementation, and could possibly provide significantly improved

results.

Lastly, a better occlusion handling system could be implemented. This would allow the system to handle cases where the user's hand has been removed from the frame momentarily, either due to leaving the field of view of the cameras, or moving behind the user's body. This would probably be best handled through use of a full person model, not just a skeleton, and any of the occlusion handling techniques described in Kakardiaris and Metaxas[28] or Plänkers and Fua[36], or any number of other projects. As such, it would be better to first expand the system to use a full body model, so that self-occlusions would be easier to identify.

Chapter 6

Conclusion

Overall, we were successful, in that the algorithms perform as expected, running at real time speeds and providing user arm tracking and dense depth maps. The depth extraction and limb tracking were successfully implemented on the GPU, leaving the CPU free for other work. This approach could easily be adapted to be used as a user interface.

While not as accurate as offline stereo vision systems, we found that this approach was good enough for our purposes, however, if the accuracy could be improved, this approach could be used for getting real time estimates of models. The particular issues encountered were somewhat expected, given the algorithms we chose, such as the occlusion issues in the tracking and the depth inaccuracies, as described in section 4 and 5.

We feel that further testing should be conducted to determine if this approach could be used on older graphics hardware, which could expand its usefulness in interfaces or applications. Along with other extensions discussed above, this approach to user tracking should certainly be further investigated.

While similar performance could be achieved through custom hardware, or even

a purely CPU based method for smaller image feeds, custom hardware is expensive, and a CPU solution limits the possibilities for applications to make use of this data, as they would have to share those CPU cycles.

While there is still room for improvement, we feel that this approach could serve as a good basis for any number of HCI projects, novel games, or simply further research, and as such can be considered a success. However, further work on improving the accuracy or tracking capabilities could make it a useful approach for other tasks, such as motion or model capture, or any number of other domains.

Bibliography

[1] Head and hand tracking. http://www1.cs.columbia.edu/~jebara/htmlpapers/ARL/node21.html, 1999.

[2] High-accuracy stereo depth maps using structured light. http://community.middlebury.edu/~schar/papers/structlight/, 2003.

[3] Middlebury stereo evaluation results. http://vision.middlebury.edu/stereo/eval/, 2009.

[4] Welcome - opencv wiki. http://opencv.willowgarage.com/wiki/, September 2009.

[5] Apple iphone. http://www.apple.com/iphone/, January 2010.

[6] Distortion (optics) - wikipedia. http://en.wikipedia.org/wiki/Distortion_(optics), January 2010.

[7] Dragon naturally speaking. http://www.nuance.com/naturallyspeaking/, January 2010.

[8] Epipolar geometry. http://en.wikipedia.org/wiki/Epipolar_geometry, January 2010.

[9] Fingerworks. http://en.wikipedia.org/wiki/FingerWorks, January 2010.

[10] Hardware sales from launch on vgchartz.com. http://vgchartz.com/hwlaunch.php, January 2010.

[11] Motorola droid. http://phones.verizonwireless.com/motorola/droid/, January 2010.

[12] Nintendo wii. http://www.nintendo.com/wii, January 2010.

[13] Tech arp - desktop graphics card comparison guide rev. 19.1. http://www.techarp.com/showarticle.aspx?artno=88&pgno=8, January 2010.

[14] Time-of-flight camera - wikipedia. http://en.wikipedia.org/wiki/Time-of-flight_camera, January 2010.

[15] Vlfeat - tutorials - sift. http://www.vlfeat.org/overview/sift.html, 2010.

[16] K. Akita. Image sequence analysis of real world human motion. *Pattern Recognition*, 17(1):73–83, 1984.

[17] C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. Pyramid methods in image processing. *RCA Engineer*, 29:33–41, 1984.

[18] A. Azarbayejani and A. Pentland. Real-time self-calibrating stereo person tracking using 3-d shape estimation from blob features. In *ICPR '96: Proceedings of the International Conference on Pattern Recognition (ICPR '96) Volume III-Volume 7276*, page 627, Washington, DC, USA, 1996. IEEE Computer Society.

[19] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:401–406, 1998.

[20] A. F. Bobick and J. W. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):257–267, 2001.

[21] J. Davis and M. Shah. Visual gesture recognition. *Vision, Image and Signal Processing*, 141:101–10, 1994.

[22] T. Ertl, W. Heidrich, M. D. (editors, M. J. Harris, G. Coombe, T. Scheuermann, and A. Lastra. Physically-based visual simulation on graphics hardware, 2002.

[23] A. R. François. Real-time multi-resolution blob tracking. Technical Report IRIS-04-422, Institute for Robotics and Intelligent Systems, University of Southern California, April 2004.

[24] W. Freeman and C. Weissman. Television control by hand gestures. *IEEE Intl. Wkshp. on Automatic Face and Gesture Recognition*, 1995.

[25] J. Fung and S. Mann. Computer vision signal processing on graphics processing units. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004*, pages 93–96, 2004.

[26] C. Jennings. Robust finger tracking with multiple cameras. In *RATFG99*, pages 152–160, 1999.

[27] A. D. Jepson, D. J. Fleet, and M. J. Black. A layered motion representation with occlusion and compact spatial support. In *ECCV (1)*, pages 692–706, 2002.

[28] I. A. Kakadiaris and D. Metaxas. Model-based estimation of 3D human motion with occlusion based on active multi-viewpoint selection. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition, CVPR*, pages 81–87, Los Alamitos, California, U.S.A., 18–20 1996. IEEE Computer Society.

[29] A. Koschan and V. Rodehorst. Dense depth maps by active color illumination and image pyramids. In *Advances in Computer Vision*, pages 137–148, 1997.

[30] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[31] G. Mori and J. Malik. Estimating human body configurations using shape context matching. In *ECCV (3)*, pages 666–680, 2002.

[32] A. Moss, D. Page, and N. Smart. Toward acceleration of rsa using 3d graphics hardware. *Cryptography and Coding*, pages 364–383, 2007.

[33] A. S. Ogale and Y. Aloimonos. Shape and the stereo correspondence problem. *Int. J. Comput. Vision*, 65(3):147–162, 2005.

[34] J. O'Rourke and N. Badler. Model-based image analysis of human motion using constraint propogation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(6):522–536, 1980.

[35] M. Pharr and R. Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005.

[36] R. Plänkers and P. Fua. Articulated soft objects for video-based body modeling. In *ICCV*, Vancouver, Canada, July 2001.

[37] J. Rehg and T. Kanade. Digiteyes: Vision-based hand tracking for human-computer interaction. *Proceedings of the workshop on Motion of Non-Rigid and Articulated Bodies*, pages 16–24, 1994.

[38] J. M. Rehg and T. Kanade. Model-based tracking of self-occluding articulated objects. In *ICCV*, pages 612–617, 1995.

[39] Y. Ricquebourg and P. Bouthemy. Real-time tracking of moving persons by exploiting spatio-temporal image slices. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):797–808, 2000.

[40] M. Ringer and J. Lasenby. Modelling and tracking articulated motion from

multiple camera views. *Proc. British Machine Vision Conf.*, pages 172–182, 2000.

[41] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-time 3d model acquisition. *ACM Trans. Graph.*, 21(3):438–446, 2002.

[42] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 195–202, 2003.

[43] J. Segen and S. Kumar. Shadow gestures: 3D hand pose estimation using a single camera. *Proc. IEEE International Conference on Computer Vision and Pattern Recognition*, pages 479–485, 1999.

[44] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, 1998.

[45] J. Wang, F. Shi, J. Zhang, and Y. Liu. A new calibration model of camera lens distortion. *Pattern Recogn.*, 41(2):607–615, 2008.

[46] C. Wren. Perceptive spaces for performance and entertainment: Untethered interaction using computer vision and audition, 1997.

[47] C. R. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.

[48] C. Wu. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). http://cs.unc.edu/~ccwu/siftgpu, 2007.