SANDIA REPORT
SAND2002-0801
Unlimited Release
Printed April 2002

Volumetric Video Motion Detection for Unobtrusive Human-Computer Interraction

Daniel E. Small, Jason P. Luck, and Jeffrey J. Carlson

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND2002-0801
Unlimited Release
Printed April 2002

Volumetric Video Motion Detection for Unobtrusive Human-Computer Interraction

Daniel E. Small, Jason P. Luck, and Jeffrey J. Carlson
Intelligent Systems Sensors Department
Sandia National Laboratories
PO Box 5800
Albuquerque, NM 87185-1003

Abstract

The computer vision field has undergone a revolution of sorts in the past five years. Moore's law has driven real-time image processing from the domain of dedicated, expensive hardware, to the domain of commercial off-the-shelf computers. This thesis describes our work on the design, analysis and implementation of a Real-Time Shape from Silhouette Sensor (RTS^3). The system produces time-varying volumetric data at real-time rates (10–30Hz). The data is in the form of binary volumetric images. Until recently, using this technique in a real-time system was impractical due to the computational burden. In this thesis we review the previous work in the field, and derive the mathematics behind volumetric calibration, silhouette extraction, and shape-from-silhouette. For our sensor implementation, we use four color camera / framegrabber pairs and a single high-end Pentium III computer. The color cameras were configured to observe a common volume. This hardware uses our RTS^3 software to track volumetric motion. Two types of shape-from-silhouette algorithms were implemented and their relative performance was compared. We have also explored an application of this sensor to markerless motion tracking. In his recent review of work done in motion tracking, Gavrila states that results of markerless vision based 3D tracking are still limited. The method proposed in this paper not only expands upon the previous work but will also attempt to overcome these limitations.

This page intentionally left blank

Contents

Figures

This page intentionally left blank

1 Data Acquisition Introduction

1.1 Overview

Applications such as virtual reality, telepresence, smart rooms, human robot interaction, automatic surveillance, gesture analysis, movement analysis for sports (and many others) require real-time motion-tracking. Because of this enormous number of applications requiring human-computer interaction, real-time 3D motion-tracking is an important problem. Much work has already been done in this field, but has used systems where images are acquired in real-time and processed later [30, 31]. However, offline processing does not permit real-time interaction. Off-line systems provide greater accuracy, but we are willing to sacrifice accuracy in order to achieve real-time performance.

There are a wide range of real-time motion tracking sensors currently in use. The most common ones require the object or person being tracked to be instrumented in some fashion, either through magnetic field sensors [33], wired optical emmiters [32] or passive optical targets [35]. As 3D user interfaces become commonplace, it becomes imperitive to eliminate encumbering wires, markers or sensors. We propose a technique that, until quite recently, had computational and memory requirements that exceeded the capabilities of commonly available computing systems. Our approach is called "Real-Time Shape-from-Silhouette." This approach will be described in detail in the first half of this report. The second half of the report will focus on a unique capability that we have developed with this sensor to track a kinematic model of a human using only the volumetric data derived from the sensor.

In his recent review of work done in human tracking, Gavrila states that results of markerless vision based 3D tracking are still limited. In conclusion he lists several challenges that must be resolved before visual tracking systems can be widely deployed [1].

1. The model acquisition issue: the majority of previous work assumes the 3D model is known apriori.

2. The occlusion issue: most systems cannot handle significant occlusion and do not have mechanisms to stop and restart tracking of individual body parts.

3. The modeling issue: few body models have incorporated articulation constraints or collision constraints.

4. The ground truth issue: no systems have compared their results to ground truth.

5. The 3D data issue: few systems have used 3D data as direct input to their tracking system. Using 3D data relieves the problems associated with retrieving 3D information from a 2D-view [1]. In addition to Gavrila's challenges we believe that there are two other requirements for a tracking system to be readily deployed.

6. A system must also perform tracking in real-time to be useful for most applications.

7. Calibration of the data acquisition device must be simple and fast.

The method proposed in this paper not only expands upon the previous work but will also attempt to meet these challenges. In this section we will cover the implementation of our tracking system in detail. Our tracking system went through two major developments.

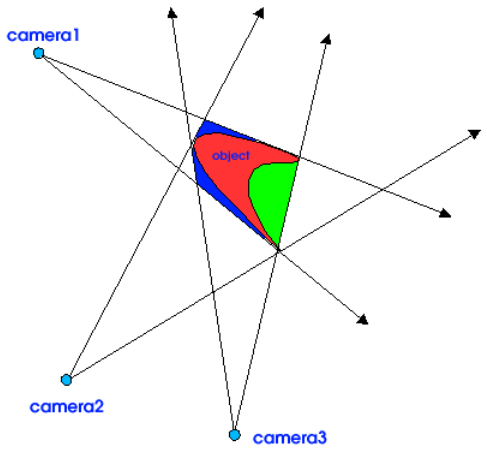

Figure 1: ***Reconstruction from three 2D views.*** *The intersection of three views (in a 2D plane) results in a reconstruction that includes 1) the original object (red); 2) points in concavities that are not visible from any viewpoint outside the convex hull of the object (green); and 3) points that are not eliminated in at least one of the three given views (blue). Variations in the estimated geometry can occur when approximating the shape with a visual hull. In general, the visual hull is a super-set of the actual geometry, and is not equivalent to the true shape, due to the inability of the shape-from-silhouette technique to resolve concavities. All of the algorithms we investigated return the union of all of the colored regions as the reconstructed shape.*

The first development only tracked the upper body dynamics of the subject. The second development tracked the full-body kinematics of the subject. Accordingly this summary is divided into two sections covering the development of each system. Within each system there exist two main components. The first component initializes our model and the second tracks the model. After discussing the development of our algorithm we will show how our system provides visual feedback of the tracking to the user. Following that a brief summary of the results for the current state of the project is provided. Lastly a synopsis of the tracking and its contributions to the field of human tracking are listed.

1.2 Shape-from-silhouette Techniques

There has been much work in the area of shape-from-silhouette. One of the goals of this thesis is to thoroughly investigate all aspects of using this method for real-time data acquisition. The implementation of a Real-Time Shape- from Silhouette Sensor (RTS^3) is contingent upon three main concepts:

- camera calibration,
- silhouette extraction, and
- silhouette projection and intersection.

The problem of mapping 3D points in the world to a pixel in the image plane of the camera is addressed by camera calibration. There are two sets of parameters that must be estimated.

The first are the intrinsic parameters, which model the optics of the camera. The second set of parameters are the extrinsic parameters, which model the position and orientation of the camera in three-space, relative to an external coordinate system. To accomplish the multi-camera implementation of a shape-from-silhouette sensor, all cameras must be calibrated to a common coordinate system.

Silhouette extraction is the process of identifying, in each image, those pixels which have changed with respect to a background image. This process is also called *background subtraction* or *background differencing*. We will describe commonly used methods, as well as some novel techniques.

The process of silhouette projection and intersection uses camera models to project the extracted silhouettes into a shared volumetric space, where their mutual intersection is computed. Figure 1 shows the problems inherent in approximating a shape by its visual hull. The visual hull is generally a superset of the actual shape due to the inability of the shape-from-silhouette technique to resolve concavities. To quote an eminent computer vision researcher, "You can't see what is not visible" -Takeo Kanade. Silhouette projection and intersection can be accomplished using a variety of techniques. We will review some of those techniques and will present an execution time analysis of two techniques which we implemented and tested.

1.3 Why Do This in Real-Time?

There are many reasons to pursue a real-time implementation of shape-from-silhouette. First, as computers begin to disappear into the walls and floors, new user interfaces will be needed. Human gesture is a natural interface that can be exploited to allow people to interact with these ubiquitous computer systems. However, human gesture is inherently three dimensional. Therefore, we need a system that is capable of measuring gestural data in three dimensions. A second reason for creating a system that works in real-time is the commercial demand for marker-less motion tracking. Such a system would represent a substantial savings in time and money and be more convenient for applications that currently use commercially available tracking systems (e.g. Ascension Technologies' "Flock of Birds" [32], the Polhemus Fastrak [34], or Northern Digital's Optotrak sensor [36]). Such a system could be applied to the areas of virtual reality, human and animal motion tracking for special effects work, and innovative video game interfaces.

With the advent of gigahertz computing and fast digital imaging interfaces, it is now possible to build an environment with embedded computer systems which can passively isolate and track the motion and gesture of the occupants. Using this technique, we demonstrate the ability to create a crude 3D approximation of the moving objects in an environment, and track their motion. We will present our work in applying this sensor to the problem of generating synthetic 3D views. We will also show results from a comparison of frame-rate vs. volumetric resolution for two shape-from-silhouette algorithms which we implemented.

At the time of this thesis, only one other team has attempted to implement a shape-from-silhouette sensor that functions in real-time, namely, Cheung and Kanade [7]. In their work, ellipsoids provide a coarse model of the segments of the body. The focus of this thesis is to reduce the hardware requirements of a system similar to the one they described, and to improve its efficiency. In Chapter 2 we will provide an overview of past work in the area of shape-from-silhouette. Technical details and early work in this area will also be presented.

Chapter 3 describes the implementation and architecture of RTS^3, and presents results from experiments in silhouette extraction. Chapter 4 gives a run-time analysis of two algorithms that were implemented on our system, and a brief overview of a novel application of this technology. Conclusions are given in Chapter 5 and areas for future work are described in Chapter 6.

2 History and Background of Shape-from-Silhouette

The background and history of shape-from-silhouette has its roots in three separate areas: camera calibration, background subtraction (silhouette extraction) and volume intersection.

2.1 Camera Calibration

Camera calibration is well studied in computer vision [7, 8, 9, 10, 13, 14, 15]. Before computer vision, it was extensively investigated in the field of photogrammetry [11, 12]. Calibration is a necessary step in extracting 3D information from 2D images. Calibration techniques can be classified into two categories: photogrammetric techniques and self-calibration techniques. Self calibration techniques are less mature than photogrammetric techniques [10]. For this reason, we used photogrammetric techniques.

Because we want to cover as large a volume as possible with the fewest number of cameras, the RTS^3 system uses cameras with extremely wide angle lenses. However, the 180° field of view of fish-eye lenses is too large because key features are projected onto a very small number of pixels. Typically, lenses in the systems we have reviewed have a field of view between 60° and 90°. This requires a calibration technique that includes estimating the coefficients of radial distortion. Tsai [7] showed how to analytically determine radial distortion parameters for machine vision systems. If a camera with a narrow field of view is used it may not be necessary to model radial distortion, as the camera will be well approximated by a pinhole. The pinhole model is commonly used in systems that do not use wide angle lenses.

Ganpathy [13] explores calibration for cameras on a robot. However, this method is limited because there is no way to achieve the matrix decomposition of the calibration if radial distortion is considered. This is due to the nonlinearities inherent in the modeling of the radial distortion. It is also desirable to use a calibration technique that relies on planar calibration targets rather than 3D calibration targets. This allows for simple calibration rigs (paper targets on glass or hard wood) that are easier to set up than 3D calibration rigs. Such methods were found first in Wei *et al.* [8] and further developed in Zhang's work on EasyCalib [9]. We used the camera calibration functions implemented in the OpenCV computer vision library [29], which are based on EasyCalib. Zhang demonstrated that the EasyCalib algorithm was robust and easy to use. As a result, the implementation used in the OpenCV library was also robust. A calibration filter application imlemented in the library uses feature extraction algorithms to locate the black/white intersections of a checkerboard in real-time, which speeds the process of collecting the data used in calibration.

Recovering three dimensional structure from images is very difficult without calibrated images. For our purposes, a camera is said to be calibrated if the mapping between image coordinates and directions relative to the camera center are known. However, the position of the camera in space (i.e. its translation and rotation with respect to world coordinates)

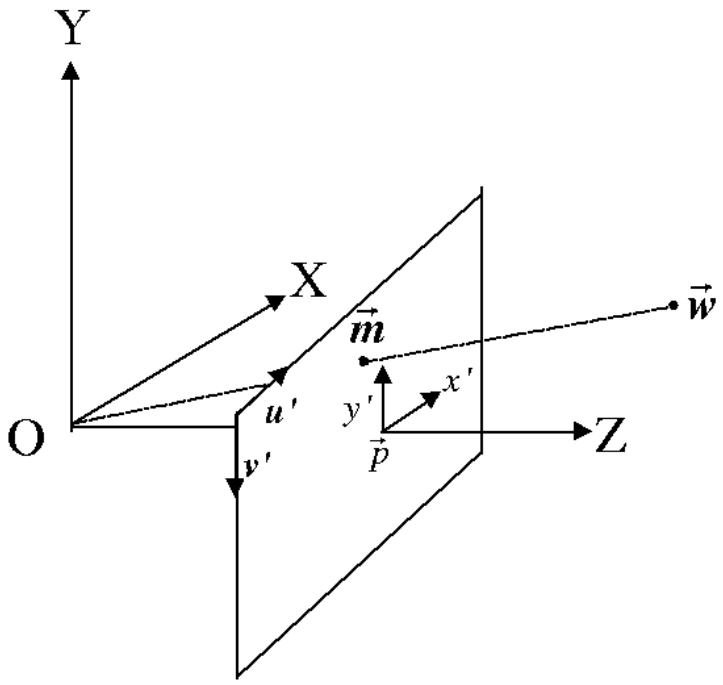

Figure 2: ***Central projection.*** *Projection of 3D point* $\vec{\mathbf{w}}=[X,Y,Z]^T$ *onto image plane at* $\vec{\mathbf{m}}=[x,y]^T$

is not necessarily known. For an ideal pinhole camera delivering a true perspective image, this mapping can be characterized completely by just four numbers, called the *intrinsic parameters* of the camera. In contrast, a camera's *extrinsic* parameters represent its location and rotation in space relative to the world coordinate system. We can represent these two sets of parameters with three coordinate systems:

- image coordinate system
- camera coordinate system
- world coordinate system

The camera coordinate system can be represented by a perspective projection, which can be written as a linear mapping between two points in projective coordinates:

$$\begin{bmatrix} x \\ y \\ s \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \qquad \text{where } x' = \frac{x}{s}, \quad y' = \frac{y}{s} \tag{1}$$

and f is the *focal length*. This 3×4 projection matrix represents a map from 3D to 2D in projective coordinates.

As shown in Figure 3, the two coordinate systems, (x', y') and (u', v'), are related by a translation and a reflection in one axis. We need to translate and invert the camera coordinate system, (x', y') so that the origin of the image coordinate system (u', v') is transformed

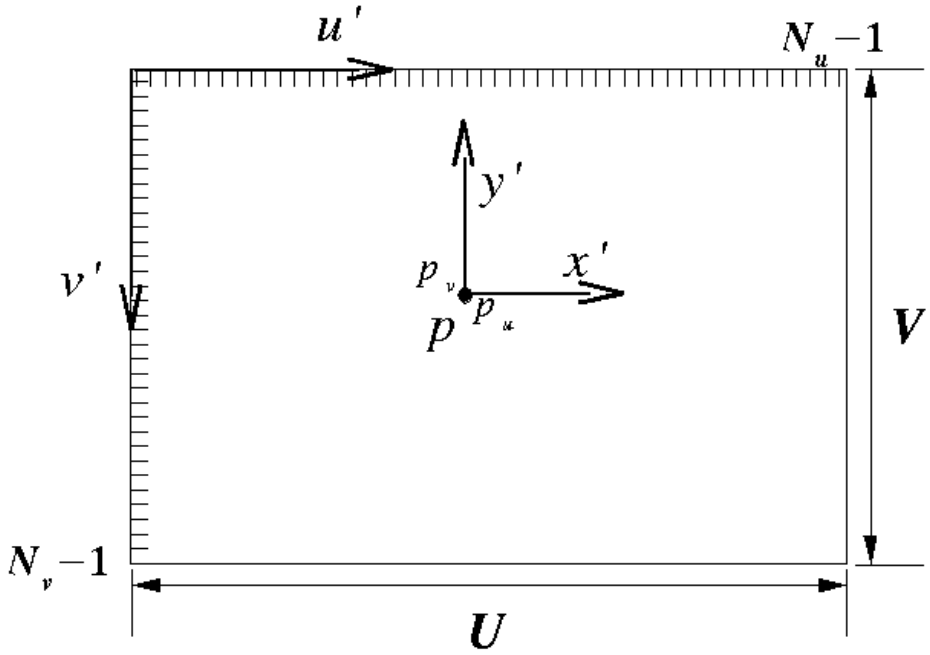

Figure 3: ***Camera coordinate space.*** *p is the point where the optical axis intersects the camera plane, and is often referred to as the* principal point. *This is the origin of the projective coordinate system. The upper left corner is the origin of the image coordinate system*

to the upper left corner. The image coordinate system needs to be scaled such that maximum values in the u and v axes correspond to the maximum pixel coordinates that can be accessed in the video framegrabber. To accomplish this we modify the parameters of the matrix in Equation 1.

To derive a new offset and scale for the image coordinate system, we need to define the following terms:

N_u - the number of pixels in the u axis of the camera image.

N_v - the number of pixels in the v axis of the camera image.

U - the physical width of the CCD sensor element.

V - the physical height of the CCD sensor element.

These parameters can be combined into two parameters, (k_u, k_v), which are scale factors which take into account the size and resolution of the CCD.

$$(k_u, k_v) \;=\; (\frac{N_u}{U},\; \frac{N_v}{V})$$

Using these scale factors, we can relate the origin of the camera coordinate system, $[x_0, y_0]^T$, to the image coordinate system, $[u_0, v_0]^T$, using the following relationship:

$$(u', v') \;=\; (k_u x' + p_u, \quad k_v y' + p_v) \tag{2}$$

The optical axis intersects the camera plane at p. The point (p_u, p_v) is the coordinate location of p in image coordinates, and is often referred to as the *principal point*. We now have enough information to define the projection in terms of the image plane with an origin at $[u, v]^T$ and a maximum coordinate value of (N_u, N_v). We define four parameters to model the projection:

(p_u, p_v) - the coordinates of the the center of projection, in image coordinates with origin (u_0, v_0).

fk_u - the scaled focal length for u in pixels.

fk_v - the scaled focal length for v in pixels.

Once these four parameters (i.e the *intrinsic calibration parameters*) are known, the camera is said to be *calibrated*. A calibrated camera can map pixels in the image to rays in three dimensional space and three dimensional points to pixels in the image. From these values we can form the intrinsic projection matrix, $\mathbf{A}$, which is often called the *camera calibration matrix*.

$$\mathbf{A} = \begin{bmatrix} fk_u & 0 & p_u & 0 \\ 0 & -fk_v & p_v & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3}$$

The last coordinate transformation relates the camera and world coordinate systems. The extrinsic parameters can be represented by a rotation matrix $\mathbf{R}$ and a translation vector $\vec{\mathbf{t}}$:

$$\mathbf{R} = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix}, \qquad \vec{\mathbf{t}} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{4}$$

which can be combined into a homogeneous 4×4 transform,

$$\mathbf{T} = \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

We can now represent the projection of an arbitrary three dimensional point in the world coordinates, $\vec{\mathbf{w}}$, to a two dimensional point in camera coordinates, $\vec{\mathbf{m}}$, using the following equation:

$$\underbrace{\begin{bmatrix} u \\ v \\ s \end{bmatrix}}_{\vec{\mathbf{m}}} = \underbrace{\begin{bmatrix} fk_u & 0 & p_u & 0 \\ 0 & -fk_v & p_v & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\vec{\mathbf{w}}} \tag{6}$$

where $(u', v') = (u/s,\ v/s)$.

To derive the pixel coordinates on the image plane, u and v must be scaled by dividing by s, the third element of the vector $\vec{m}$. This is a scale factor which is derived from the distance to the point, $\vec{w}$, in camera coordinates.

The calibration of real cameras is often approximated with this four-parameter mapping, though rarely is such an approximation accurate to within one pixel [9]. For most images taken with standard lenses, the center of projection is at (or near) the coordinate center of the image. However, small but significant shifts are often introduced in the digitizing process. Such an image shift has the most impact on camera calibration for lenses with shorter focal lengths (such as the ones we are using in our system). The focal lengths of the camera in pixels can be estimated by dividing the marked focal length of the camera lens by the width of the image on the imaging surface (the film or CCD array), and then multiplying by the width and height of the final image in pixels. This scaling is modeled in Equation 2.

Real cameras deviate from the pinhole model in several respects. First, in order to collect enough light to expose the film, light is gathered across the entire surface of the lens. The most noticeable effect of this is that only a single surface in space, called the focal plane, will be in perfect focus. In terms of camera calibration, each pixel corresponds not to a single ray from the camera center, but to a set of rays from across the front of the lens which all converge on a particular point on the focal plane. The second, and most significant effect, is radial distortion caused by the lens. Because of constraints in the lens manufacturing process, straight lines in the world imaged through real lenses generally become somewhat curved on the image plane. However, since each lens element is radially symmetric, and the elements are typically centered with high precision on the optical axis, this distortion is almost always radially symmetric. As a result, the center of radial distortion also corresponds to the intersection of the center of optical axis with the image plane. This point is the principal point, (p_u, p_v).

We can model the distortion with a nonlinear geometric transform. Let $(\hat{u}, \hat{v})$ be the true position of pixel, (u, v), in the distorted image. Let (p_u, p_v) be the center of the radial distortion. Zhang [9] demonstrated that representing the radial distortion using a second degree polynomial is good enough for most purposes and that using a higher order approximation actually causes numerical instability. Therefore, we use a second order approximation here. We can represent the geometric transform between the distorted and undistorted image spaces as follows:

$$\hat{u} = u + (u - p_u)[k_1(u^2 + v^2) + k_2(u^2 + v^2)^2] \quad (7)$$

$$\hat{v} = v + (v - p_v)[k_1(u^2 + v^2) + k_2(u^2 + v^2)^2] \quad (8)$$

The coefficients k_1 and k_2 can be determined by measuring the curvature of straight lines in sample images. Because our system uses very wide angle lenses, we have found intrinsic camera calibration and estimation of radial distortion parameters to be a straightforward process that considerably simplifies the problem of three dimensional reconstruction. By using Equations 7 and 8 in conjunction with Equation 6, we can get pinhole projection model that is corrected for radial distortion.

The calibration method presented by Zhang[9] estimates the intrinsic matrix and radial distortion parameters by using a series of checkerboard patterns as shown in Figure 4. The checkerboard represents a three dimensional rigid body that lies on the plane, $Z = 0$. The interior corners of the checkerboard are located to sub pixel accuracy in a series of images.

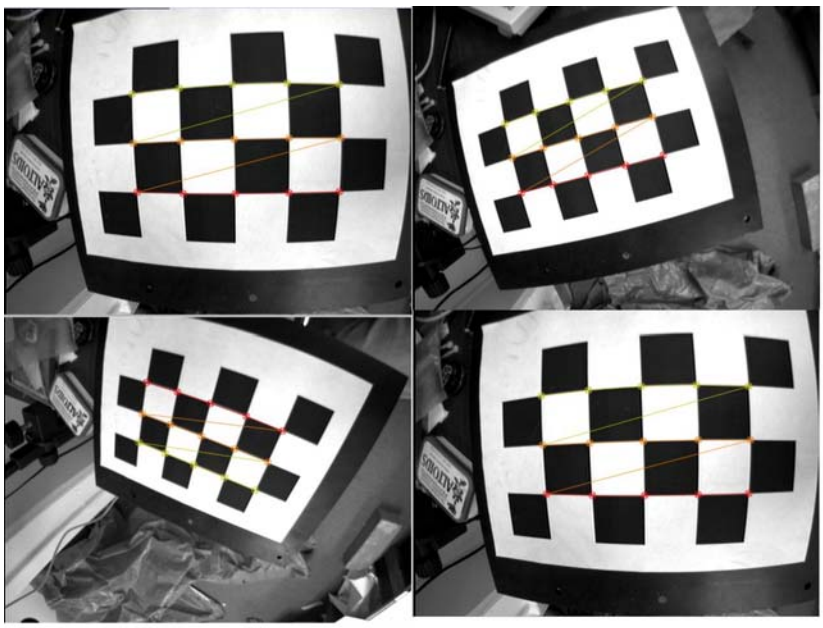

Figure 4: ***Intrinsic calibration checkerboards with features automatically detected.*** *This figure shows the data that is used to estimate the intrinsic parameters. These are four consecutive images of the checkerboard calibration object. The intersections are automatically located to sub-pixel accuracy. This is the output from the calibration filter, which is implemented in the Open Source Computer Vision library (OpenCV) [29].*

Given the dimensions and the size of the checkerboard, the intrinsic parameters are solved for using an iterative non-linear least squares method that successively estimates the intrinsic matrix and the relative position and orientation of the checkerboards with respect to the camera. During this process the extrinsic parameters of the camera are fixed at zero.

Once the intrinsic parameters have been determined, the same process is used to solve for the extrinsic parameters. The intrinsic parameters are fixed at their best estimates, and the extrinsic parameters are solved for by observing a common set of planar targets and associating each 2D point with its 3D world coordinate. The number of points in the planar calibration target must exceed the number of parameters that are being solved for. In our case this is six: $X, Y, Z, \Omega_x, \Omega_y$ and Ω_z. This yields a set of linear equations where the number of equations is greater than the number of unknowns. The targets we used are the square black and white 2×2 checkerboards on the wooden platform in Figure 5. This series of images shows a set of points in the xy plane (separated at a $\frac{1}{4}$ inch resolution) which are back-projected into the image at increasing heights through the volume. You can see the radial distortion present in the images, as the grid appears to bow heavily at the top of the volume. This type of back-projection can be accomplished only when the extrinsic parameters have been determined. The extrinsic calibration process requires only one image per camera. If all of the cameras extrinsic parameters are estimated by observing the same planar points, we say the the cameras are calibrated with respect to the same *world coordinate system*. If we use a checkerboard to calibrate the extrinsic parameters, then the checkerboard must remain in the same static position while images from all of the cameras

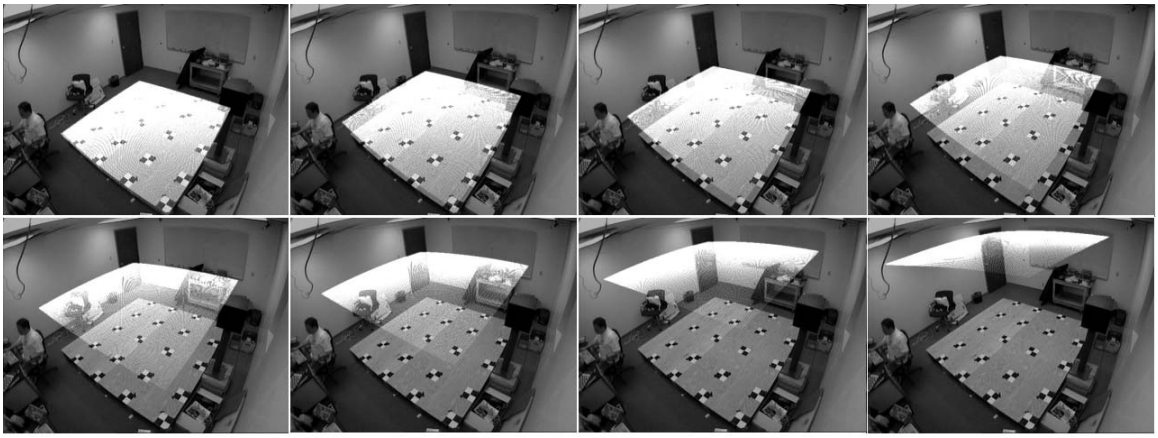

Figure 5: ***Planar targets for extrinsic calibration.*** *This series of images shows a set of points in the xy plane (separated at a $\frac{1}{4}$ inch resolution) which are back-projected into the image at increasing heights through the volume. You can see the radial distortion present in the images, as the grid appears to bow heavily at the top of the volume.*

are acquired. However, the targets should span a large portion of the field of view for each camera [9]. This means that the calibration checkerboards make poor targets for calibration of multiple wide-angle cameras, unless the baselines between the cameras are small enough that the checkerboard can occupy a large portion of all the camera images.

The only drawback to this calibration technique is that the numerical method for computing the intrinsic parameters requires that there be no two images in the data set where the checkerboards are related by a pure translation. In cases where this happens, the iterative process is numerically unstable and does not converge. A complete description of this process can be found in Zhang [9], and an implementation can be found in the Open Source Computer Vision Library [29], with optimized shared libraries for the Win32 and Linux operating systems. The OpenCV library was used for the work in this thesis.[1]

2.2 Background Subtraction

Background subtraction is a commonly used method for motion detection and silhouette extraction. We describe a method that combines the best features of existing techniques which are compatible with real-time performance.

Carlson [19, 20] and Davis [21] used only monochrome cameras in their systems. Depending on the environment, monochrome cameras can be adequate. Davis [21] designed his imaging system to be used as a perceptual user interface (PUI) that monitors the user performing a series of exercises. A projection screen shows the image of a drill sergeant who yells at the user if he or she slows down. The user's silhouette appears against the projection screen. Interference with the projected image is mitigated through the use of infrared filters on the cameras. This enables back-lighting using infrared light. This multispectral approach is appropriate for these circumstances. In the first two papers by Carlson [19, 20], the back-

[1] We highly recommend the distribution and use of the OpenCV library in both the academic and commercial computer vision communities due to the ease with which we were able to implement high level functionality.

ground subtraction algorithm first filters all images with a low-pass filter, and saves the first image as a reference image. A threshold image, consisting of a separate threshold value for each pixel, is initialized to a constant value. Both the reference image and the threshold image are updated slowly over time. The reference image incorporates new image data by averaging new images into the reference image at a rate specified by the user. The threshold image is updated by incrementing or decrementing the initial threshold pixel values based on their rate of change with respect to the current image. After a period of time, the slowly updating threshold image will approximate the standard deviation of the video images over time. In Davis' paper [21], the image statistics are calculated during a training period during which a mean background reference image and a standard deviation image are computed directly. In all three papers the foreground images are differenced as shown in Algorithm 1.

Algorithm 1: Algorithm: Background Subtraction using Standard Deviation

```
for j ← 1 to N_v do
    for i ← 1 to N_u do
        if (|current[i,j] - mean[i,j]|) ≥ stdDev[i,j]) then
            silhouette[i,j] = 1
        else
            silhouette[i,j] = 0
        end
    end
end
```

In Davis' work [21], because of the use of back-lighting, the problem of shadows was negligible. Carlson [19, 20] did not address the problem of shadows. More specifically, their Volumetric Video Motion Detector (VVMD) did not attempt to construct volumetric data near the floor, where the problems of cast shadows is most severe.

Cheung and Kanade's [4] goal was to accurately reconstruct human motion, so shadows were a much larger concern for them. In addition to intensity they used color for extra discrimination power. Rather than using image statistics such as the mean and standard deviation, they used a complicated scheme which involved manually partitioning each image into separate regions, which were then assigned different upper and lower intensity thresholds. They also used color as a secondary means of differentiating foreground and background. Their algorithm uses a lower intensity threshold (I_{lower}), an upper intensity threshold (I_{upper}), and a hue angle difference threshold (ΔH_{thresh}). It is described in pseudocode in Algorithm 2.

In Davis and Bradski [22], the algorithms are simple enough to run in real-time, but are still interesting in the sense that they adapted a new technique to solve an old problem. Like Carlson [19, 20], they used simple image statistics as reference images, but they also used a combination of morphological processing, contour extraction and filling algorithms to fill the small holes that are pervasive in silhouette images. This approach is interesting because it substantially eliminates the problems that occur when holes in the silhouette are projected through the visual hull. This problem, in particular, causes large gaps in the construction of the visual hull. Because the calculation of motion template gradients (first defined in Davis' paper [21]) is not affected by changes in the total area of the silhouette, they were free to use morphological filtering to fill small holes in the silhouette. For our work morphological

Algorithm 2: Algorithm: Background Subtraction using Hue Difference

```
for j ← 1 to N_v do
    for i ← 1 to N_u do
        if (|current[i,j].intensity - mean[i,j].intensity|< I_lower) then
            silhouette[i,j] = 0;
        else
            if (|current[i,j].intensity - mean[i,j].intensity|> I_upper) then
                silhouette[i,j] = 1;
            else
                c1 = [current[i,j].red,current[i,j].green,current[i,j].blue]^T
                c0 = [mean[i,j].red,mean[i,j].green,mean[i,j].blue]^T
                ΔH = cos^-1 [ c1·c0 / (||c1|| ||c0||) ]
                if (ΔH > ΔH_thresh) then
                    silhouette[i,j] = 1
                end
            end
        end
    end
end
```

filtering was a key area of image processing that we intentionally avoided. Erosion and dilation filters change the overall shape of the silhouette and will seriously affect the overall quality of the reconstructed visual hull. Dilation and opening filters increase the area of the silhouettes which gives the 3D volumetric reconstructions a bloated look. Erosion and closing filters can decrease the area of the silhouette and open larger holes in the silhouette, which in turn causes holes in the 3D reconstruction.

In our work, we examined different techniques for background subtraction, adopted those that fit our goals, and adapted those that did not. We used a hybrid algorithm that combined statistical image differencing, hue angle differencing, and contour analysis. This will be described in Section 3.3.

2.3 Shape-from-silhouette

Reconstructing an object's 3D shape from a set of images collected using multiple cameras is a classic computer vision problem. In the last few years, this problem has generated considerable interest, partly due to a number of new applications (e.g. Rander and Kanade [23]) that require good volumetric reconstructions. Shape-from-silhouette is a well-known approach to this problem, dating back at least to the early 1980's [27]. In the volumetric approach to shape-from-silhouette, the scene is represented as a set of three dimensional voxels, and the task is to label the individual voxels as occupied or empty. These methods work by first computing the object's silhouettes from each camera image, typically using background subtraction. In the geometric approach to shape-from-silhouette, every silhouette pixel from every camera image is projected into 3D space as a conic solid. After these solids are intersected, what is left is a geometric representation of the visual hull. The boundary of the visual hull can then be represented using a triangular mesh.

The shape-from-silhouette problem that we address is significantly simpler than the re-

lated problem of voxel coloring [26]. In voxel coloring, the task is to label every voxel with its color plus its transparency. Voxel coloring requires handling difficult issues, like visibility relationships. One advantage of voxel coloring is that the shape of the object can be estimated more accurately, since a match in camera intensities can be used to prune away empty voxels. In addition, the resulting voxel colors can be directly used to generate new views.

Shape-from-silhouette is, relatively speaking, an easier problem to solve. It is also much less sensitive to the errors in the photometric calibration of the camera system. It is important to note that the views of an object from multiple cameras do not uniquely determine the shape of a non-convex object. This is true even with an infinite number of cameras. The visual hull of an object [17] is defined to be the maximal object that is consistent with the object's silhouette from any viewpoint. It is impossible to distinguish two different objects with the same visual hull purely by their silhouettes.

Many researchers have been experimenting with various algorithms to implement shape-from silhouette. Matusik *et al.* [18] used a geometric approach to silhouette intersection which required the use of a distributed computer system to solve for the intersections in real-time. They show that their intersection algorithm parallelizes linearly using a divide-and-conquer approach. However, this process is computationally intensive. Cheung and Kanade [4] use a volumetric method on one computer, but distributes the silhouette extraction among five others. The volumetric methods tend to be simpler, but use more memory and do not scale as well as the geometric methods. In Cheung's more recent work, he uses a high resolution volumetric approach. This is performed entirely off-line, and uses a marching cubes algorithm to extract an isosurface [5].

Recently, researchers have begun to experiment with systems that can both generate and interpret volumetric data in real-time. The system developed by Cheung *et al.* [4] uses six computers and five color cameras. It supports a voxel resolution of $64 \times 64 \times 64 \times 1\text{in}^3$. The main purpose of their system was 3D reconstruction of human motion. Simple fitting of ellipses was performed to track extremities. Carlson *et al.*[19, 20] used a real-time sensor as well with a lower resolution volume size of about 40^3 voxels with a voxel size of 3in^3. The main purpose of their system was real-time 3D interaction with a commercial robotic manipulator. A key difference between these two systems is that Cheung's was implemented on a distributed system that used color cameras, while Carlson's system was implemented using monochrome cameras and a single computer. Such a system is also used by Luck *et al.*[28], where the goal was also human motion tracking. The thing which differentiates Cheung and Kanade [4] and Luck *et al.* [28] is the resolution of the kinematic model that is tracked. The system described by Cheung and Kanade [4] used single ellipses to model and track the extremities, whereas the system described by Luck *et al.* [28] used a fully articulated model of the arms and shoulders. Both systems track in real-time, but because it is very desirable to reduce the amount of hardware required for such a system, the second approach is better suited to real-time volumetric sensing. However, because the second system used monochrome video frame grabbers, the volumetric data computed by the Cheung and Kanade [4] looks qualitatively better. This is due, in part, to their use of color for background subtraction. One of the goals of this thesis is to combine the two approaches by using four color video framegrabbers in a single computer. In this way we produce a system with qualitatively better volumetric data than that computed by Luck *et*

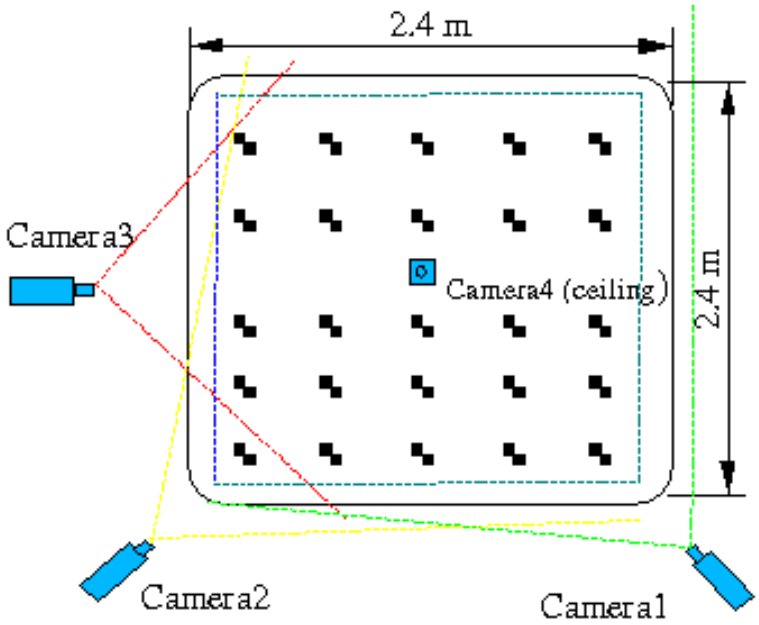

Figure 6: ***Layout of the* RTS^3 *lab.*** *This is a view from above. The cross-hatches are the extrinsic calibration targets. The colored lines represent the approximate fields of view of the cameras.*

al. [28] without the major hardware requirements of Cheung and Kanade [4].

3 Implementation of RTS^3

3.1 System Architecture

RTS^3 is the system we built to acquire volumetric data. This sensor uses a combination of industry standard components including a high-end PC, four analog color video cameras, and four PCI-bus color frame grabber cards. Using this hardware, we create a time-varying volumetric image of the visual hull of whatever object is moving in the space observed by all four cameras. Figure 6 shows the positions of the four cameras from above. An additional goal was to design the software with a standard application programming interface (API) to the cameras, so that new types of cameras could be added with relative ease. To date, we have used the system with analog color framegrabbers, RS-170 monochrome framegrabbers, and IEEE 1394 (FireWire) cameras.

We implemented RTS^3 using a software framework developed at Sandia National Labs [37] called Umbra. It is a cross-platform development environment (Windows, Irix, and Linux operating systems) written in C++, Tcl/Tk [38], OpenGL [39] and the Standard Template Library (STL). A screen-shot of the run-time GUI for RTS^3 is shown in Figure 7. To better illustrate the 3D nature of the volumetric data, the voxels are rendered as spheres. Spheres of a single color occupy planes of constant Z value. This method is only used for

illustration, as it creates a graphics bottleneck at higher resolutions. Normally the voxels are displayed as simple points. [2] A second pop-up window displays the real-time silhouette information.

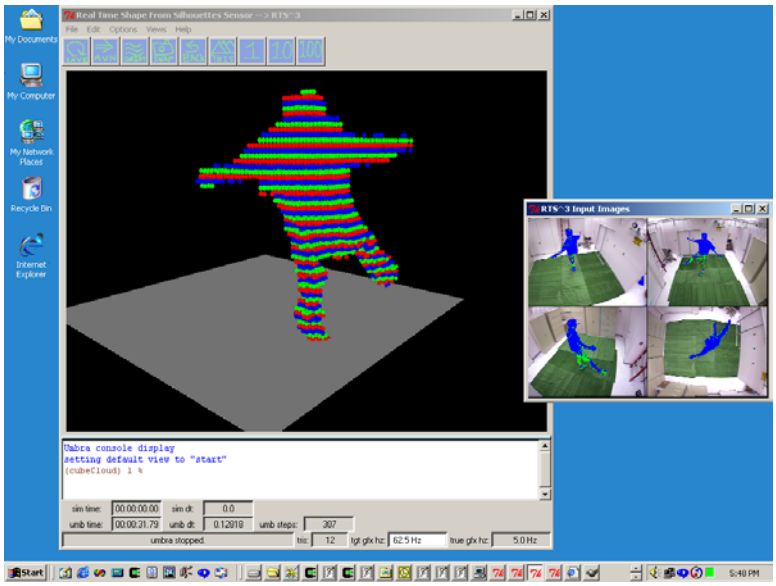

Figure 7: ***Graphical user interface for*** $\mathbf{RTS^3}$***.*** *This is the graphical user interface for RTS^3. The GUI allows for a large OpenGL 3D display, reconfigurable buttons and menus whose functionality can be set using the Tcl scripting language, and a Tcl shell interface at the bottom of the GUI. The smaller window displays the images and overlays the silhouette information in real-time. To better illustrate the 3D nature of the volumetric data, the voxels are rendered as spheres. Spheres of a single color occupy planes of a constant Z value. This method is only used for illustration, as it creates a graphics bottleneck at higher resolutions. Normally the voxels are displayed as simple points.*

The RTS^3 system was rapidly designed and implemented using the GUI tools and data flow architecture provided by this framework. The individual software modules can be described as; 1) data sources, 2) filters, and 3) sinks. In our system, the cameras act like data sources and the shape-from-silhouette algorithm acts as a data filter. The data sink is a rendering module in OpenGL. A data flow diagram is shown in Figure 8.

3.2 Camera Calibration Methodology

Our implementation of camera calibration involved two stages. As mentioned in Section 2.1, a series of checkerboard images was used to estimate the intrinsic parameters. The checkerboard image yields a set of 2D points that corresponds to points on a 3D rigid body. Next, each camera was calibrated relative to a common world coordinate system. This

[2] Tcl/Tk is used to implement the GUI. This is a scripting language that allows the user to parameterize RTS^3 at run time using scripts. Individual modules in the system (background subtraction, volume computation, and rendering) use input and output connectors to define the flow of data through the system. Modules are fully decoupled from the other modules in the system except through the language interpreter itself.

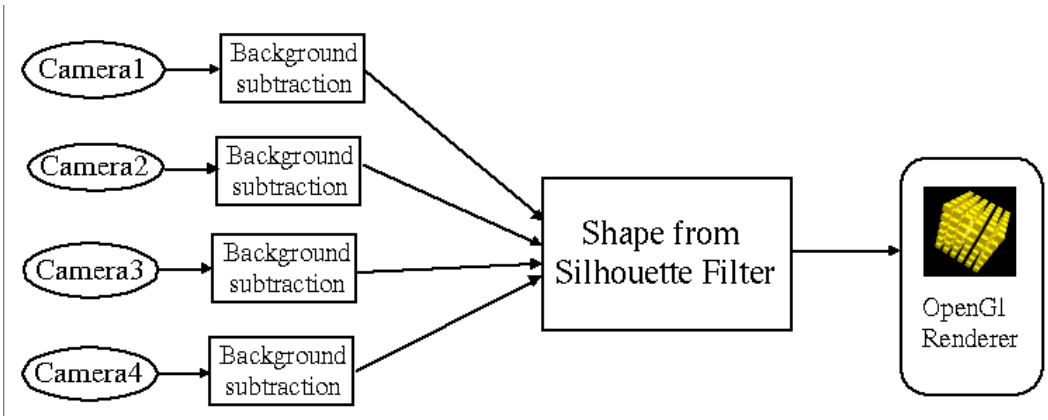

Figure 8: ***Data flow diagram for*** $\mathbf{RTS^3}$***.*** *This is the data flow diagram as it is connected in the Umbra framework*

was done with a GUI (as shown in Figure 9) that allows the user to pick specific planar target points on the ground plane and associate them with 3D points stored in a text file. The camera views have intrinsic calibration parameters associated with them. The manual identification of the targets is performed by the user. Inaccuracies are minimized by searching a 5×5 pixel neighborhood of the points around the selected coordinate. This provides for an optimal sub-pixel match for the center of the target. Once all targets are selected, the user invokes a function to find the extrinsic parameters. This GUI allows for multiple cameras to be selected and calibrated to the same world coordinate system.

Once the intrinsic and extrinsic parameters of each camera have been determined, the next step is to calibrate the common volume observed by all the cameras. Our system uses lookup tables to store precalculated index information. In brief summary, each voxel centroid position is back-projected into each of the camera images using the extrinsic and intrinsic calibration parameters for that camera. The algorithm to create the individual lookup tables is shown in Algorithm 3, where *backProject3DToCamera()* takes a point in 3D, (X, Y, Z), and a set of camera calibration parameters, and returns the pixel position, $(\hat{u}, \hat{v})$, that the point maps to using the intrinsic and extrinsic calibration parameters for that camera. In our system, the precomputation allows us to perform the silhouette intersection in real-time. Each voxel holds a single pixel position for each camera. In the lookup table file, this is represented as an ordered pair, (u, v). At run time, a pixel position is translated into a pointer to the specific memory location representing that pixel in the silhouette image. On 32-bit operating systems pointers use four bytes of memory. This leads to a voxel lookup table whose total size is $4 \times numCameras \times N_x \times N_y \times N_z$ bytes (where N_x, N_y and N_z are the number of voxels in their respective axes). The pseudocode for *backProject3DToCamera()* is shown in Algorithm 4. These calibration operations are all performed offline. The precomputation of the 3D to 2D associations allows the system to perform in real-time.

The algorithms to construct the volume calibration are $O(n)$, where n is the number of voxels in the volume. The function *backProject3DToCamera()* is $O(1)$. This assumes the number of cameras is a constant, and is *far* less than the number of voxels.

Figure 9: ***Extrinsic calibration GUI.*** *This figure shows the GUI that is used to calculate the extrinsic calibration parameters. The red numbers are positioned near the planar targets (within a* 5×5 *pixel window), and the 2D positions that are selected are correlated with 3D positions stored in a file. By using the intrinsic parameters, we can solve for the relative transform between the camera and the planar calibration grid.*

Algorithm 3: Algorithm: Create 3D to 2D Look Up Tables

input : A volume origin, voxelSize and resolution in three dimensions, and a set of camera calibration parameters

output: A look-up table that associates 3D points in the volume to 2D points in the camera image planes

```
CreateLookUpTable(origin, voxelSize)
i ← 0
for x ← 1 to N_x do
    for y ← 1 to N_y do
        for z ← 1 to N_z do
            //X, Y, Z are the actual points in 3D space
            X ← origin.x + (x × voxelSize.x)
            Y ← origin.y + (y × voxelSize.y)
            Z ← origin.z + (z × voxelSize.z)
            for Camera ← 1 to numCameras do
                //point2d is an ordered pair (u,v)
                point2D ← BackProject3DToCamera(X, Y, Z,Camera)
                voxelTable[i][Camera] ← point2D
            end
            i ← i+1
        end
    end
end
```

3.3 Background Differencing and Silhouette Extraction

The initial background subtraction algorithm was the simplest part of the sensor to implement, but creating an algorithm that performs robustly in many different lighting conditions

Algorithm 4: Algorithm: Back Project 3D Point to Camera

BackProject3DToCamera(X, Y, Z, Camera)
begin
$\vec{w} \leftarrow$ [X,Y,Z,1]
$T \leftarrow$ Camera.T // 4×4 homogeneous transform
$A \leftarrow$ Camera.A // 4×3 camera matrix
$k_1 \leftarrow$ Camera.k_1 // first radial distortion coefficient
$k_2 \leftarrow$ Camera.k_2 // second radial distortion coefficient
//put w into camera frame by multiplying it by the camera transform T
$\vec{w'} \leftarrow T\vec{w}$
// project $\vec{w'}$ onto the image plane via calibration matrix A
$\vec{m} \leftarrow A\vec{w'}$
//scale by m_2
$x \leftarrow \frac{m_0}{m_2}$
$y \leftarrow \frac{m_1}{m_2}$
//correct for radial distortion
radius $\leftarrow \sqrt{x^2 + y^2}$
radius$^2 \leftarrow$ radius $\times$ radius
$\hat{x} \leftarrow x + (x - c_x)[k_1(\text{radius}) + k_2(\text{radius}^2)]$
$\hat{y} \leftarrow y + (y - c_y)[k_1(\text{radius}) + k_2(\text{radius}^2)]$
return $(\hat{x}, \hat{y})$
end

is partly art. There were many problems to overcome, including noisy cameras, shadows, and separating silhouette pixels from background pixels.

Castleman [6] describes the basic technique of image differencing using a hard threshold, a reference image, and a current image. Sometimes the reference image is an average image (i.e. average of several images acquired *a priori*), sometimes the reference image is a low-pass filtered copy of the first image that is acquired, and sometimes it is the first image with no manipulation at all. Depending on the quality of the video signal, and the way in which it is digitized, the amount of ambient noise in a video image can vary widely.

The first problem we addressed was the elimination of shadows, especially those cast on the floor near the feet. In Cheung and Kanade [4] and Davis and Bradski [22] color cameras and framegrabbers were used to acquire 24-bit color data. Although these authors used slightly different techniques, their results were similar. Cheung [4] calculated a hue angle difference when the intensity difference values were within a specified range. Davis and Bradski [22] estimated a mean and a standard deviation image (over a series of the initial images) where they differenced the mean and the current image and compared it to a threshold which was proportional to the standard deviation of the background at that pixel.

In our work, a similar problem with shadows was observed. The first version of RTS^3 was implemented using monochrome cameras. An example of this problem is illustrated in Figure 10 and in Figure 11. Figure 11 shows how shadows in the image produce errors in the volumetric reconstruction.

The key assumption used for eliminating shadows by color image differencing is that the area where a shadow is cast generally shows a large change in intensity but a small change

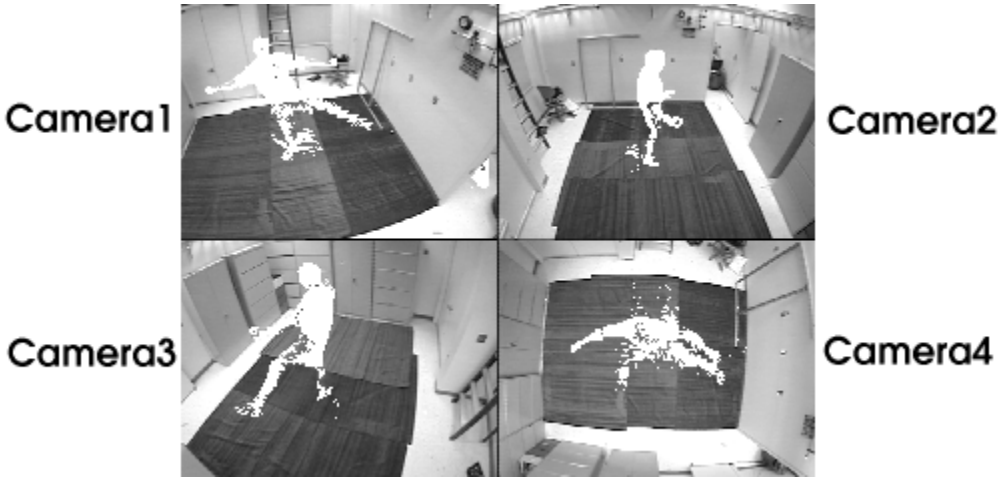

Figure 10: ***Monochrome image differencing.*** *This shows how using only monochrome images can lead to poor silhouette segmentation caused by shadows.*

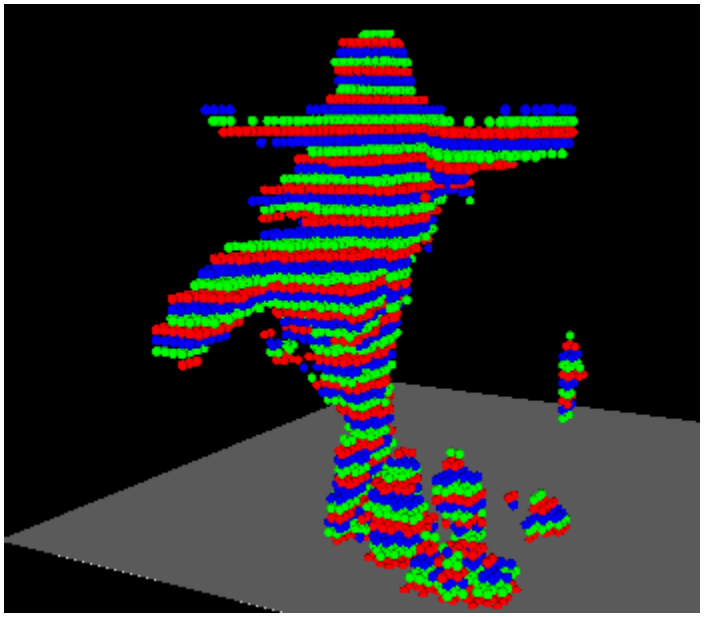

Figure 11: ***Shadows produce errors in 3D.*** *This shows how shadows can induce 3D noise in the volumetric reconstruction. The shadows on the floor of the 2D image produce errors near the legs and feet in the 3D data.*

in hue. Hue is an angular representation of color. Therefore, the angular difference between a pixel in the background and a shadowed pixel from the foreground should be relatively small. Hue is computed as follows: [6]:

$$H = \cos^{-1}\left[\frac{\frac{1}{2}((R-G)+(R-B))}{\sqrt{(R-G)^2-(R-B)(G-B)}}\right] \tag{9}$$

You can see that if $R = G = B$ the value of hue goes to infinity. Consequently, purely gray pixels have undefined color. Let $\vec{\mathbf{c_1}}$ equal the current RGB pixel. Let $\vec{\mathbf{c_0}}$ equal the

background RGB pixel. The change in hue between these two values is defined as:

$$\Delta H = \cos^{-1}\left[\frac{\vec{\mathbf{c_1}} \cdot \vec{\mathbf{c_0}}}{\|\vec{\mathbf{c_1}}\|\|\vec{\mathbf{c_0}}\|}\right] \tag{10}$$

We used hue difference as an additional check against classifying shadow pixels as silhouette. Cheung's system [4] was implemented in a distributed fashion. Specifically, the image processing for each camera was handled by a single computer. Since our goal was to use a single computer for all of the computation, this calculation was optimized by using look-up tables for the square root and arc-cosine functions. These look-up-tables are accurate to five significant digits. The addition of optimized hue differencing allowed us to raise the intensity thresholds above the level typically produced by shadows cast on the floor.

The next problem we addressed was the noise in individual cameras. Ambient noise levels can vary widely between cameras, which makes it impossible to use the same threshold values across a set of cameras that possibly have different noise properties.

One way to evaluate the noise in a digital camera is to take the standard deviation of a series of images. This is part of the method for statistically estimating a background image as discussed in Davis *et al.* [22]. The standard deviation image approximates the noise of the image sensor. An example of the standard deviation image (estimated using a set of 75 color images) is shown in Figure 12. The standard deviation is estimated across the red, green, and blue color planes individually. Brighter colors correspond to higher standard deviation values. Notice that the images from cameras one and two have a much lower standard deviation than the images from cameras three and four. This illustrates that different cameras can have very different noise characteristics.

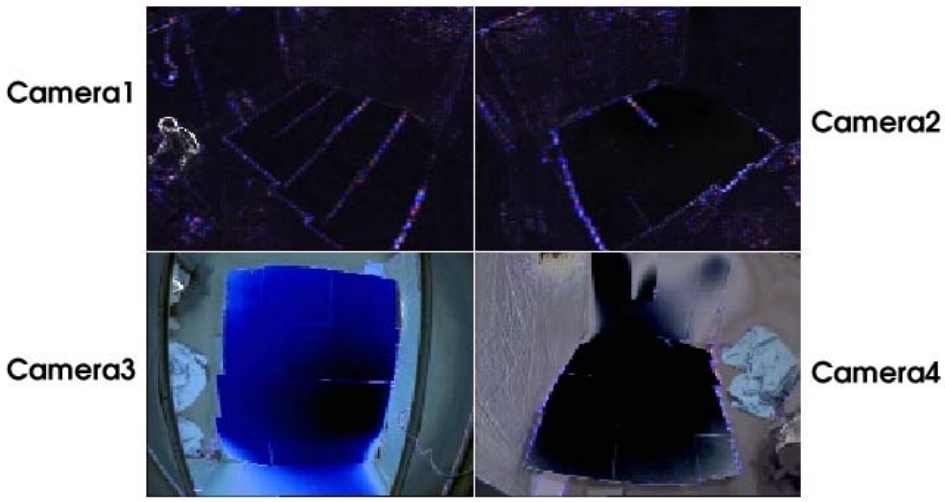

Figure 12: ***Standard deviation image.*** *This figure shows the per pixel standard deviation image estimated from 75 color images. The standard deviation image can be used for per-pixel thresholding and can compensate for differences in gain, exposure, and noise between cameras. The majority of the noise appears in the blue pixels. This is caused by the physical format of the CCD element, which gives more area to red and green pixels. This means that the blue elements on the CCD are being lit with a smaller number of photons, and must therefore use a higher gain.*

We merged these two methods (intensity differencing and hue differencing) into a single

procedure that is both fast and flexible. The pseudocode for our procedure is shown in Algorithm 5

Algorithm 5: Algorithm: Extract Silhouette from Background

input : mean,stdDev,and current RGB images

output: a binary silhouette

ExtractSilhouette(stdDev, mean, current)

begin

```
for j ← 1 to N_v do
    for i ← 1 to N_u do
        silhouette[i,j] = 0
        Δ r = current[i,j].red - mean[i,j].red
        Δ g = current[i,j].green - mean[i,j].green
        Δ b = current[i,j].blue - mean[i,j].blue
        if (not (| Δr | < I_lower || | Δg | < I_lower || | Δb | < I_lower)) then
            if (| Δr | > I_upper || | Δg | > I_upper || | Δb | > I_upper) then
                silhouette[i,j] = 1
            else
                c1 = [[current[i,j].red,[current[i,j].green,[current[i,j].blue]^T
                c0 = [mean[i,j].red,mean[i,j].green,mean[i,j].blue]^T
                ΔH = cos^-1 [ (c1 · c0) / (||c1|| ||c0||) ]
                if (ΔH > ΔH_max) then
                    silhouette[i,j] = 1
                else
                    r_σ ← stdDev[i,j].red
                    g_σ ← stdDev[i,j].green
                    b_σ ← stdDev[i,j].blue
                    if (| Δr | > r_σ || | Δg | > g_σ || | Δb | > b_σ) then
                        silhouette[i,j] = 1
                    end
                end
            end
        end
    end
end
```

The results of the background differencing algorithm are shown in Figure 13. The shadows have been successfully removed. The colored pixels have been classified as silhouette using the rules described in Algorithm 5. Pixels classified due to the intensity test are displayed as blue. Pixels classified due to the standard deviation test are displayed as red. Pixels classified due to the hue angle test are displayed in green. The result of the classification can be seen in Figure 14, where the classified pixels have been converted into binary silhouettes. In Figure 15, the volumetric data computed using color images exhibits none of the noise that was present in the volumetric data computed using monochrome images.

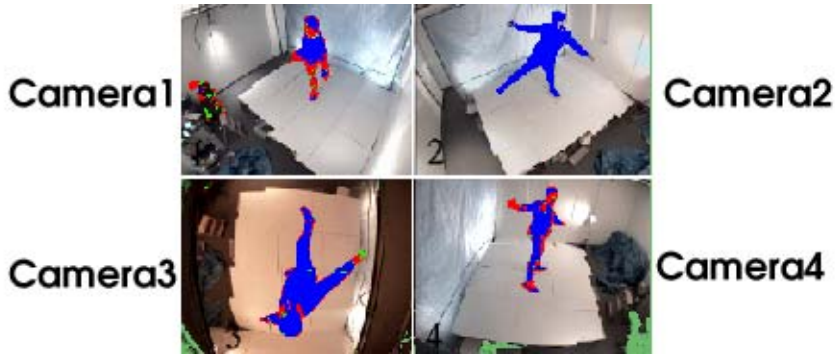

Figure 13: ***Color silhouettes with shadows removed.*** *This picture shows the improvement that can be attained in extracting the silhouettes by using color images and differencing with hue and standard deviation. Pixels classified due to the intensity test are displayed as blue. Pixels classified due to the standard deviation test are displayed as red. Pixels classified due to the hue angle test are displayed in green. The improvement in the 3D data is seen in Figure 15.*

Figure 14: ***Binary silhouettes produced from color segmentation.*** *This picture shows the binary silhouettes that can be derived from Figure 13. The improvement in the 3D data is seen in Figure 15.*

The last problem we addressed was filling the small holes in the silhouettes. Typically, for this purpose, some kind of local morphological operator is used after thresholding. Unfortunately, the use of morphological operators often introduces errors in the 2D silhouette, which, when projected, produce errors in the reconstructed 3D shape. Our approach was to search the image for contours. We used a function in the OpenCV library (see [29]) that decomposes a binary image into a two level hierarchy of contours: exterior contours and their interior holes.

The next step was to fill the small interior regions with the value one, and to fill the small exterior regions with the value zero. This has the combined effect of removing holes from the large silhouettes, and removing speckle from the background. Fortunately, the function in the OpenCV library is fast enough to run in real-time. Figure 16 shows the silhouette images before contour processing and Figure 17 shows the silhouette images after contour processing.

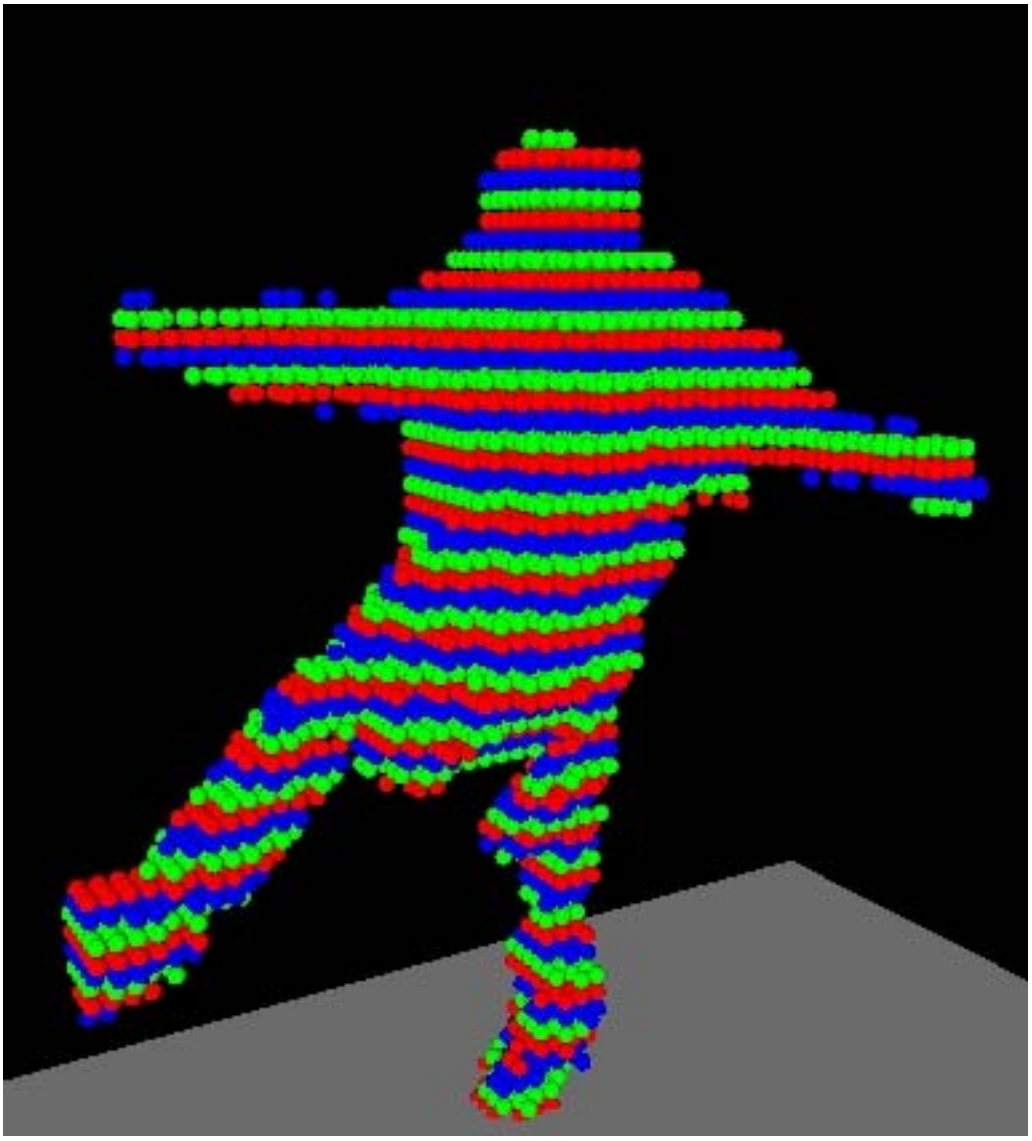

Figure 15: ***Improved 3D data.*** *This picture shows the improvement in the 3D data which results from using a color image processing technique to extract the silhouettes, as shown in Figure 13.*

3.4 Implementation of Shape-from-silhouette

We implemented two different shape-from-silhouette algorithms, which we have termed: 1) PixelVolume, and 2) HybridVolume. Both algorithms rely on table look up to speed the process of back-projection.

The PixelVolume algorithm is shown in Algorithm 6. It works by traversing every voxel in the volume, and computing the (x, y) location in each image that the voxel centroid projects to. It examines the appropriate pixels in each silhouette image and if they contained within the silhouette, that voxel is marked as occupied. If a single pixel is outside the silhouette, then that voxel is marked as empty. Because we wanted to increase the frame-rate for tracking a human figure, a key optimization was to restrict the processing to voxels in the volume around the centroid of the human figure. The centroid, (x_0, y_0, z_0), and maximum

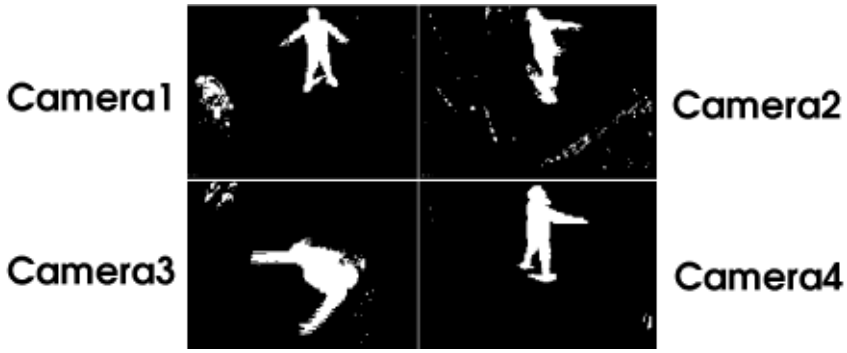

Figure 16: ***Silhouette images before contour processing.*** *Silhouette images from all four cameras before contour processing.*

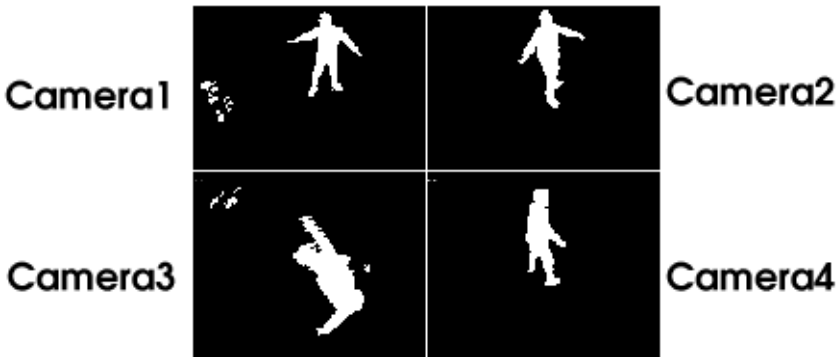

Figure 17: ***Silhouette images after contour processing.*** *Silhouette images from all four cameras after contour processing. The noise in the silhouette images has been successfully reduced without substantially changing the shape of the silhouettes.*

height in the Z axis, $maxZ$, are tracked from frame to frame. The volumetric window is a cube that is $maxZ$ wide, $maxZ$ long and $maxZ$ high. This cube is centered in the xy plane on (x_0, y_0). Depending on the full size of the monitored volume, this can result in significant, constant time savings.

The HybridVolume algorithm is a bit more complex. It is shown in Algorithm 7. A single camera is designated the *dominant* camera. For this single camera, we generate look-up table different from those used in the PixelVolume algorithm. Rather than looking up pixels from voxels, we invert the problem, and for each pixel, we store the list of voxels which that pixel projects to. For the dominant camera, we traverse all of the pixels, and we push voxels associated with pixels belonging to the silhouette onto a stack. After all pixels in the dominant camera have been examined, we begin to pop the voxels from the stack. At this point the stack contains a subset of the voxels in the volume. For each of the voxels on the stack, we use the PixelVolume algorithm to check that voxel in the images from the remaining cameras. If the voxel projects to silhouette pixels in the images from the remaining cameras, we mark it as occupied. If we encounter even a single non-silhouette pixel, then we mark that voxel as empty. The major difference between this algorithm and the PixelVolume algorithm is that HybridVolume drastically reduces the amount of time it takes to compute a volumetric frame. In practice, we have found that the camera which is

Algorithm 6: Algorithm: PixelVolume Shape from Silhouette

```
input  : silhouettes: an array of silhouette images
output: volume: a binary 3D volumetric image
PixelVolumeSFS(silhouettes)
begin
    for k ← 1 to N_x do
        for j ← 1 to N_y do
            for i ← 1 to N_z do
                for camera ← 1 to numCameras do
                    (u,v) = voxelLookUpTable[camera, i ,j, k]
                    if (silhouettes[camera, u, v] == 0) then
                        // If one pixel does not belong to a silhouette, stop
                        break
                    end
                end
                // If for-loop completed successfully, mark voxel as occupied
                if (camera == numCameras) then
                    volume[i,j,k] = 1
                else
                    volume[i,j,k] = 0
                end
            end
        end
    end
end
```

most suitable to be the dominant camera is the one mounted on the ceiling looking down. Of course, this depends on the geometry of the monitored volume and the relative positions of the cameras. In general, a downward looking camera on the ceiling observes a smaller silhouette than do the other cameras. This has the effect of sampling a smaller amount of the volume for further testing than say, a side-looking camera.

If camera noise occurs around specific features common to all camera views (e.g. shadows cast on the floor), then that noise becomes spatially correlated and produces false voxels. One nice thing about this approach is that it is reasonably tolerant of individual camera noise. This is because there is little spatial correlation between the noise of individual cameras. An example of this can be seen in Figure 18. Here, one of the cameras is producing a large amount of noise, while the others are not. Surprisingly, this results in a 3D reconstruction that does not contain large amounts of 3D noise, as shown in Figure 19. Volume intersection algorithms are tolerant to this kind of noise because a pixel must be classified as silhouette in images of *all* of the cameras in the system to create an occupied voxel.

Algorithm 7: Algorithm: HybridVolume Shape from Silhouette

```
input  : silhouettes: an array of silhouette images
output: volume: a 3D binary volumetric image
HybridVolumeSFS(Silhouettes)
begin
    for j ← 1 to N_v do
        for j ← 1 to N_u do
            if silhouettes[DominantCamera, i, j] == 1 then
                for l ← 1 to VoxelList[i,j].size() do
                    stack.push[VoxelList[i,j].voxel[l]]
                end
            end
        end
    end
    for i ← 1 to stack.size() do
        voxel = stack.pop[]
        //Use the PixelVolume method for the remaining cameras
        for camera ← 1 to numCameras-1 do
            [u, v] = cameraLookUpTable[camera, voxel]
            if (silhouettes[camera, u, v] == 0) then
                break
            end
        end
        // If for-loop completed successfully, mark voxel as occupied
        if (camera == numCameras-1) then
            voxel.MarkAsFilled()
        else
            voxel.MarkAsEmpty()
        end
    end
end
```

4 Experiments and Results

4.1 Resolution vs. Frame-rate

We experimentally compared the run-time behavior of the two different volume traversal algorithms: PixelVolume and HybridVolume. We tested the frame-rate of the RTS^3 system at 24 levels of volumetric resolution. We started with a Look Up Table (LUT) resolution of 5×10^6 total voxels, and reduced the resolution of each successive set of LUTs by 2×10^5 voxels. The final LUT in the set had approximately 8×10^5 voxels. For each LUT, we presented the same set of 600 color images to the system. The first 75 images were used to compute the mean and standard deviation images used for background subtraction, and the remaining 525 images showed a subject entering the volume, performing a series of motions, and exiting. To perform these tests, the RTS^3 system was modified to work in batch mode, using "write-images" and "read-images" functions. These functions work on the raw image data. The only sense in which batch mode differs from online mode is that the images are

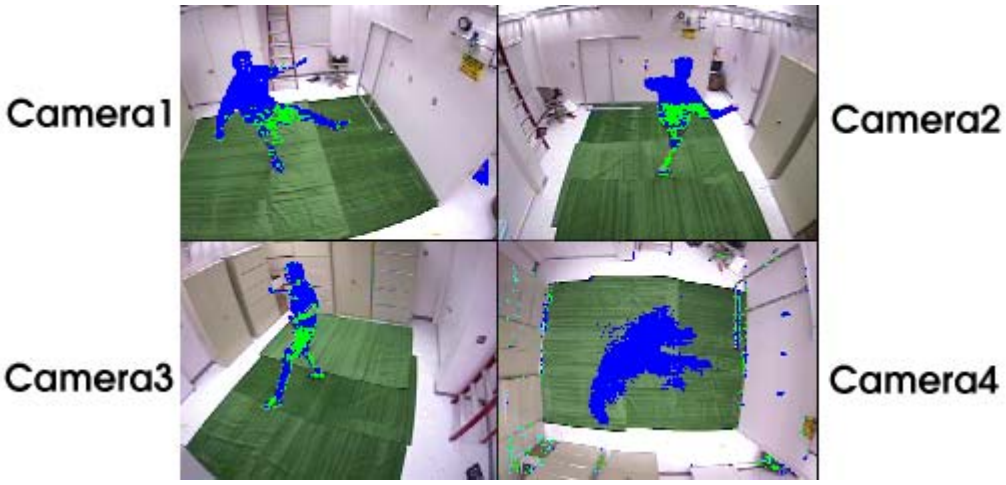

Figure 18: ***Bad data from one camera.*** *An example of how the effect of bad, noisy data from one camera is mitigated by the shape-from-silhouette algorithms. The 3D reconstruction resulting from this is shown in Figure 19*

read off of the hard disk in batch mode instead of being read directly from the framegrabbers.

The results were surprising. Figure 20 and Figure 21 are bar graphs which illustrate the performance of the two algorithms, averaged over the 600 image frames, at each LUT resolution.

In both approaches, the frame rate goes up dramatically as the number of voxels falls below 10^6. This due to cache locality. At higher volumetric resolutions, the number of context switches between cache and main memory is the main bottleneck in the algorithm. At lower resolutions, the volume is represented by substantially less memory, and thus it takes fewer and fewer main memory swaps into cache to traverse the entire volume.

The image resolution used in the tests at was 160×120 pixels. For four images of this size, we use 76 kilobytes of memory. The amount of level 2 cache on the Pentium 3 CPU is 256 kilobytes. At this image resolution, we can fit all four silhouettes into cache. The remaining cache can be used to process the volume data.

The HybridVolume approach had frame rates that were twice as fast for the high resolution volumes. This is because the number of voxels that are checked is roughly one half the number checked at the same resolutions using the PixelVolume approach. The PixelVolume approach uses a rectangular window around the subject that is as wide and long as it is high. The HybridVolume approach projects the silhouette from the dominant camera from the top of the volume to the bottom. This results in a smaller number of voxels that are checked for occupancy. As the distance between the dominant camera and the volume is increased, the percentage of voxels that is pushed on the stack for testing is decreased.

5 Conclusions on RTS^3

In this thesis, we presented work on the design, analysis and implementation of RTS^3. The system produces time-varying volumetric data at real-time rates (10–30Hz). The data is in the form of binary volumetric images. The techniques needed to implement this system

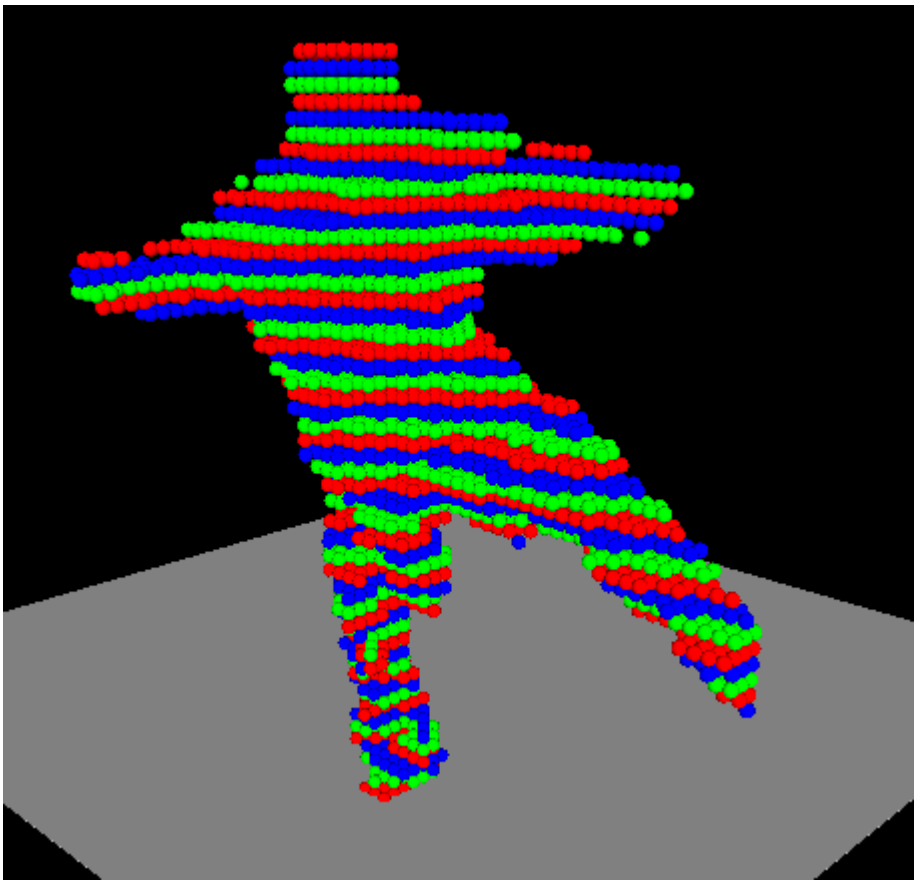

Figure 19: ***3D data with one bad silhouette image.*** *An example of how the effect of noise in one camera is mitigated by the shape-from-silhouette algorithms. One bad camera does not produce significant noise in the volumetric reconstruction. This feature makes the system very robust to small changes in any of the cameras.*

were discussed in detail: calibration, silhouette extraction, and volumetric intersection. We produced qualitative results comparable to other systems described in the literature, minimized the system's hardware requirements, and still achieved real-time performance. In this thesis we reviewed the previous work in the field, and derive the mathematics behind volumetric calibration, silhouette extraction, and shape-from-silhouette. We reviewed the topics of camera calibration and volumetric intersection. For our sensor implementation, four color cameras were installed in a lab and used with our software to track volumetric motion. An analysis of resolution vs. frame rate was performed between two different implementations of the shape-from-silhouette technique. This showed that the HybridVolume approach is superior to the simpler PixelVolume approach. An application of the system to the problem of generating synthetic 3D views was also demonstrated.

Because the RTS^3 system works without encumbering wires and photo-reflective tape, is is a natural choice as the base technology for a new generation of motion capture devices. Our system is currently being used by the Colorado School of Mines as a data acquisition system for the purposes of markerless human motion tracking. In our most recent collaborations we are tracking a nineteen degree of freedom human kinematic model.

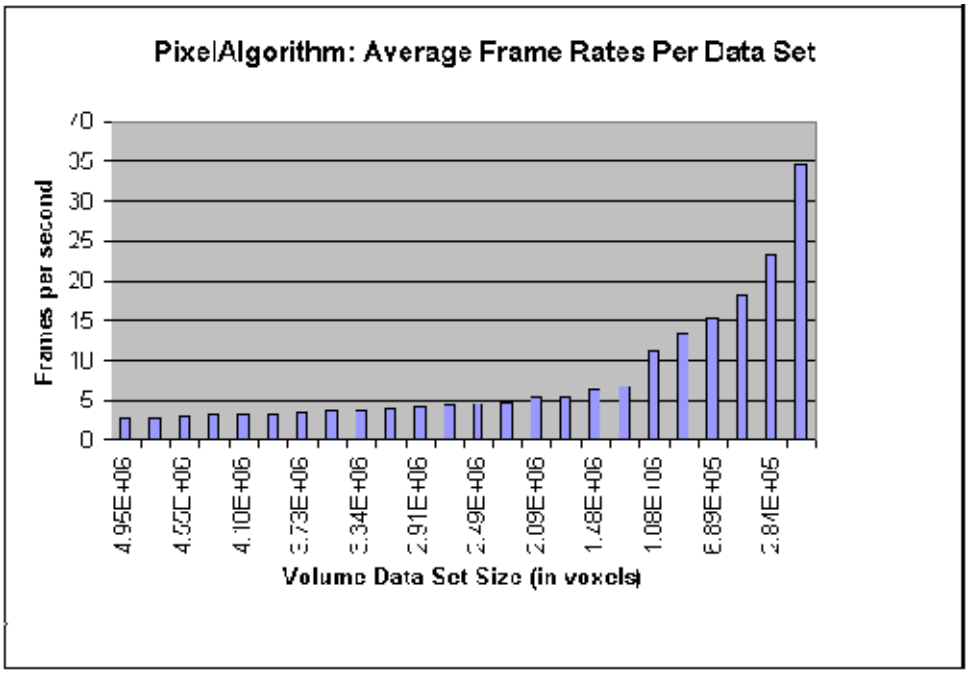

Figure 20: ***Data from run-time test 1.*** *Average frame-rate for the PixelVolume Algorithm. These timing tests for the PixelVolume Algorithm were performed using the same set of 600 images over a set of 24 linearly decreasing volumetric resolutions.*

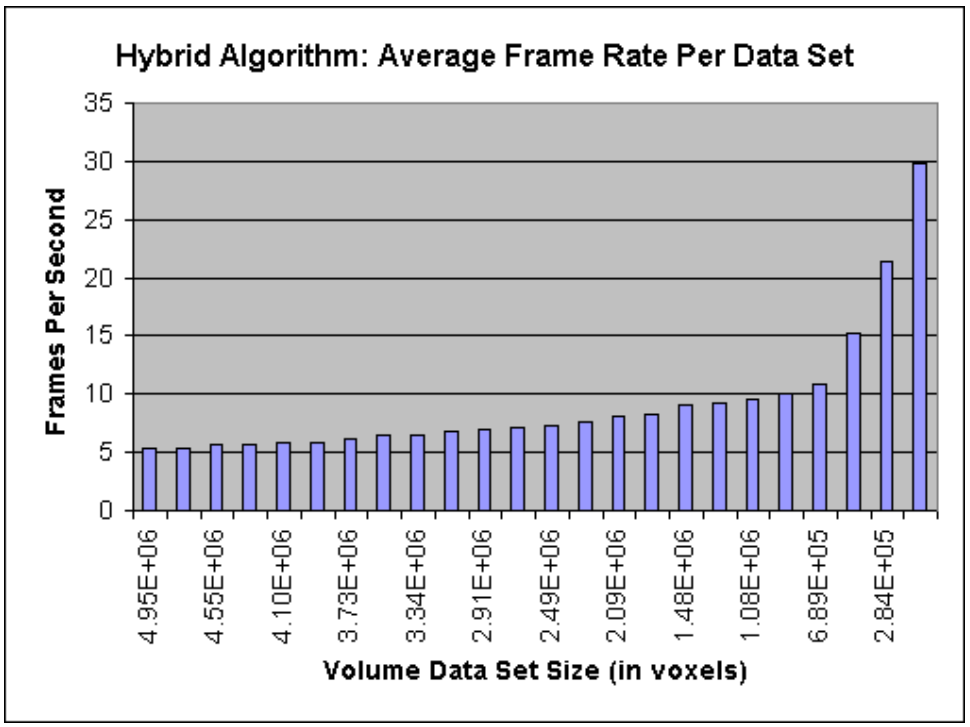

Figure 21: ***Data from run-time test 2.*** *Average frame-rate for the HybridVolume Algorithm. These timing tests for the HybridVolume Algorithm were performed using the same set of 600 images over a set of 24 linearly decreasing volumetric resolutions.*

As Moore's law continues to increase computational power and reduce the size of the electronic components, this type of 3D computer vision system will soon be seen in the home, workplace, and community.

6 Future Work on RTS^3

For our future work, we would like to examine ways of exploiting these systems for everyday user interactions. Because RTS^3 work without encumbering wires and photo-reflective tape, they are a natural choice as the base technology for a new generation of motion capture devices. Our system is currently being used by the Colorado School of Mines as a data acquisition system for the purposes of marker-less human motion tracking. We intend to stay actively involved in their efforts.

Another possible application of this technology is to build a 3D Green-Screen version of the sensor for broadcast market. Television studios that do local news could have weather forecasters flying through the 3D rendering of storms, as opposed to just blandly standing in front of them.

The application of synthetic view using these methods could be greatly enhanced by using many cameras for texture texture mapping, while only using a small subset of the cameras for the shape-from-silhouette process. This would allow one to dynamically choose the small number of cameras used for creating the 3D data, while always using the "best" camera for texture-mapping.

Yet another area of research would be to to explore the creation of an embedded version of the system that could be used as a peripheral device. As small video cameras have been reduced in cost by the advent of distributed computing, so too may we soon see a set of small web-cam sized devices that you set up without a visible computer. The possibility of distributing the computational requirements for the shape from silhouette process among several small cameras with built-in microprocessors is the next big step for this technology in hardware.

In order to have deep market penetration, the responsibility of calibration must be removed from the end user. There are two ways around this issue. The first is to build a system of pre-calibrated cameras that come in a rigid structure that is calibrated, disassembled, and reassembled by the user. This may be practical for high end systems, but for consumer grade systems, we will need to evaluate the possibility of adaptive calibration or self calibration.

7 Application To Tracking

Due to the enormous number of applications involving human-computer interaction, real-time markerless 3D human motion tracking has become a highly valued goal. Applications such as virtual reality, telepresence, smart rooms, human robot interaction, surveillance, gesture analysis, movement analysis for sports and medicine, advanced user interfaces, and many others all have a need for real-time human motion-tracking. Accordingly there has been a lot of work done in this field. However in his recent review of work done in human tracking, Gavrila states that results of markerless vision based 3D tracking are still limited.

In conclusion he lists several challenges that must be resolved before visual tracking systems can be widely deployed [1].

1. The model acquisition issue: the majority of previous work assumes the 3D model is known apriori.

2. The occlusion issue: most systems cannot handle significant occlusion and do not have mechanisms to stop and restart tracking of individual body parts.

3. The modeling issue: few body models have incorporated articulation constraints or collision constraints.

4. The ground truth issue: no systems have compared their results to ground truth.

5. The 3D data issue: few systems have used 3D data as direct input to their tracking system. Using 3D data relieves the problems associated with retrieving 3D information from a 2D-view [1]. In addition to Gavrila's challenges we believe that there are two other requirements for a tracking system to be readily deployed.

6. A system must also perform tracking in real-time to be useful for most applications.

7. Calibration of the data acquisition device must be simple and fast.

The method proposed in this paper not only expands upon the previous work but will also attempt to meet these challenges.
In this section we will cover the implementation of our tracking system in detail. Our tracking system went through two major developments. The first development only tracked the upper body dynamics of the subject. The second development tracked the full-body kinematics of the subject. Accordingly this summary is divided into two sections covering the development of each system. Within each system there exist two main components. The first component initializes our model and the second tracks the model. After discussing the development of our algorithm we will show how our system provides visual feedback of the tracking to the user. Following that a brief summary of the results for the current state of the project are provided. Lastly a synopsis of the tracking and its contributions to the field of human tracking are listed.

8 1^{st} Development - Upper Body Dynamics Only

The first system only tracks the upper body dynamics of the human subject. Accordingly the eleven degree of freedom (DOF) model includes the x, y position of the body, a torso rotation about the z axis, along with four rotations in each arm.

8.1 Initialization

To simplify tracking we require that the user performs an initialization pose upon entering the workspace. The pose is a simple cross formation with the user facing along the y axis with his arms extending parallel to the floor and straight out to the sides of the body (along the x axis). Once in this pose, the system measures the body parameters required for tracking: body radius, shoulder height, and arm length. Currently the user must always begin by initializing the system. However, the model parameters can easily be saved to a file and read in before the user enters the workspace. Our system already uses voice communication, therefore it would be easy to incorporate an option to simply tell the system your name while entering the workspace and skip the initialization phase.

8.2 Tracking of Upper Body

The original tracking procedure is broken into two phases. The algorithm first finds the (x,y) location and heading (the direction the person is pointed) of the torso. We can then predict the location of the shoulders, and assume that their locations remain constant for a particular torso orientation. These locations are then used to anchor the arms. In this way we only need to solve for the angles at which the upper arms extend from the shoulders, and then for which the lower arms extend from the elbows.

With this algorithm we only compute the x, y location of the torso and a rotation about z; accordingly we are assuming the user is standing straight up. The x, y location can easily be computed using the median value of all the voxels. The heading is computed by fitting an ellipsoid to all the data within the body radius of the x, y location. This is done by performing an eigen-decomposition on the moment matrix M. The eigenvectors of this matrix correspond to the principal axes of the ellipsoid. The principal axis closest in orientation to the last known heading is taken as the new heading of the person.

$$\begin{bmatrix} M_{xx} & M_{xy} & M_{xz} \\ M_{yx} & M_{yy} & M_{yz} \\ M_{zx} & M_{zy} & M_{zz} \end{bmatrix} where, (M_{ab} = \frac{1}{n}\sum_{i=1}^{n} a_i b_i) \tag{11}$$

To compute the angles for a particular arm segment, we simply compute the angles from all voxels that are not within the body radius to the anchor point. The result is used as a pulling force from the current orientation to a new orientation that passes through this voxel, as shown in Figure 22. In this way each voxel can have an effect on all of the arm segments, which overcomes the problem of having to decide which arm segment a voxel belongs to. However, we weight each pull by the distance, d, to each arm segment, so that a voxel exerts a stronger pull on closer arm segments than on those further away. For each voxel, let the minimum distance to any arm segment be denoted as d_{min}. The weight for each segment is given by $(d_{min}/d)^3$. Accordingly as the model is pulled into the correct orientation, the forces exerted from a particular voxel should be almost entirely on the segment closest to the voxel. This weighting strategy works extremely well for small adjustments, which is all it should have to make since our tracking rate is extremely high. In addition we employ

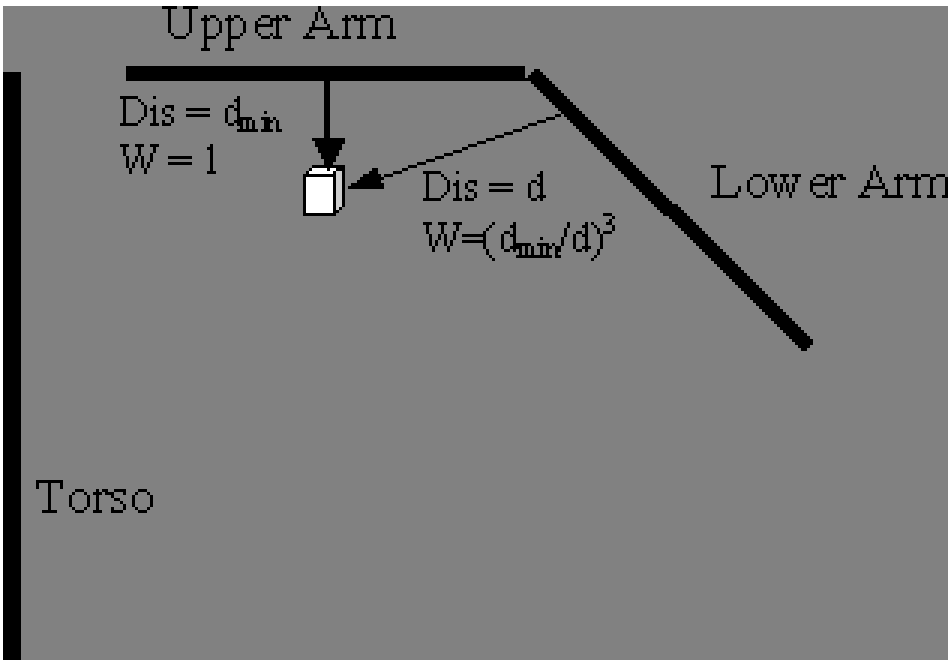

Figure 22: Example of alignment forces (the weight to the upper arm is 1 while the weight to the lower arm, which is further away, is scaled down)

zero weighting to any arm segment further than a certain distance from the voxel (d_{max}, computed from a maximum arm velocity divided by our tracking rate). Also, if a voxel is within a very small distance of an arm segment (less than d_{seg}), then we assume that it belongs to only that segment, and assign a zero weight to all other segments. The weighting assignment is shown below.

$$Weight = \left\{ \begin{array}{ll} 1 & if\ d = d_{min} \\ (d_{min}/d)^3 & if\ d_{min} < d < d_{max} \\ 0 & if\ d > d_{max} \\ 0 & if\ d > d_{min} \& d_{min} < d_{seg} \end{array} \right\} \quad (12)$$

Once the adjustment is computed for all voxels outside of the body radius, an adjustment is made to each of the arm segment angles and the loop repeats. The process stops for each arm segment when either the adjustment is too small or the move was bad (fewer points are close to new orientation of the segment).

Our system stops tracking an arm when the arm lies along the body. If we hope to track the arms when they are extremely close to the body we must move to a much more precise torso model.

The process also has a recovery algorithm in case it "loses" an arm. If too few voxels are close to one of the arm segments, the process assumes that the optimization has failed and attempts to grow the arm instead. Since we know the shoulder position, we start a growing algorithm from this point. Growing can include any neighboring point outside of the body radius, and continues until no new points are found. In this way the last point found should again be the hand and the elbow should be somewhere in the middle. Accordingly

we compute the elbow to be at the voxel that was 2/3 of the way down the arm. Angles are calculated using these locations. The growing algorithm may not be able to find the arm if we have missing data (data acquisition has failed to find the arm voxels) or if the arm lies within the body radius. In this case our algorithm will leave the arm where it was last located and wait for the next set of data.

8.3 1^{st} Development Results

The first system is able to collect data and track our model at 16 frames per second using a single computer. We are not able to indicate precisely how accurate our tracking is because we do not know the ground truth for the users movement within the workspace. However, the fit looks good when comparing the movement of the user and the avatar side by side, movies are available at `http://egweb.mines.edu/cardi/3dvmd.htm`. In addition we were able to use the joint angles to distinguish several gestures to control a crane, as will be discussed in the application section.

In general the system worked extremely well when the arms were moved at normal velocities, and were held away from the body and from each other. However, when the arms were positioned close to one another the arm segments from one arm would be pulled towards the other. A similar problem occurred when the elbow was bent well past ninety degrees. In this configuration the pull from the upper arm voxels on the lower arm was often large enough to dominate. Accordingly when the actual arm was straightened the tracking algorithm would leave the elbow bent. Lastly when an arm was moved extremely fast, the algorithm could fall behind and eventually loose the arm. In general our routine was able to detect that it had lost the arm in these situations, and would then revert to the growing algorithm to find the arm.

8.4 Application To Gesture Analysis

We employed our system to perform crane gesture analysis. A separate computer, with a voice recognition system, was utilized to read the joint locations, use them to interpret gestures, and control the crane.
The process follows this pattern. When a user enters the workspace the computer says "Hello" and asks the user to verbally identify himself (eventually this will be used to skip the initialization step). The user is then asked to stand in the initialization pose and the system verbally communicates when initialization is done and gesturing can begin. Once in this state, the system continuously inspects the joint angles to see if a gesture is occurring. Because joint angles are being used for gesture interpretation the size of the user has no impact, and the user can perform a gesture facing any direction and even while moving. Currently the system is able to reliably interpret all of the gestures shown in Figure 23. A movie can be seen at `http://egweb.mines.edu/cardi/3dvmd.htm`.

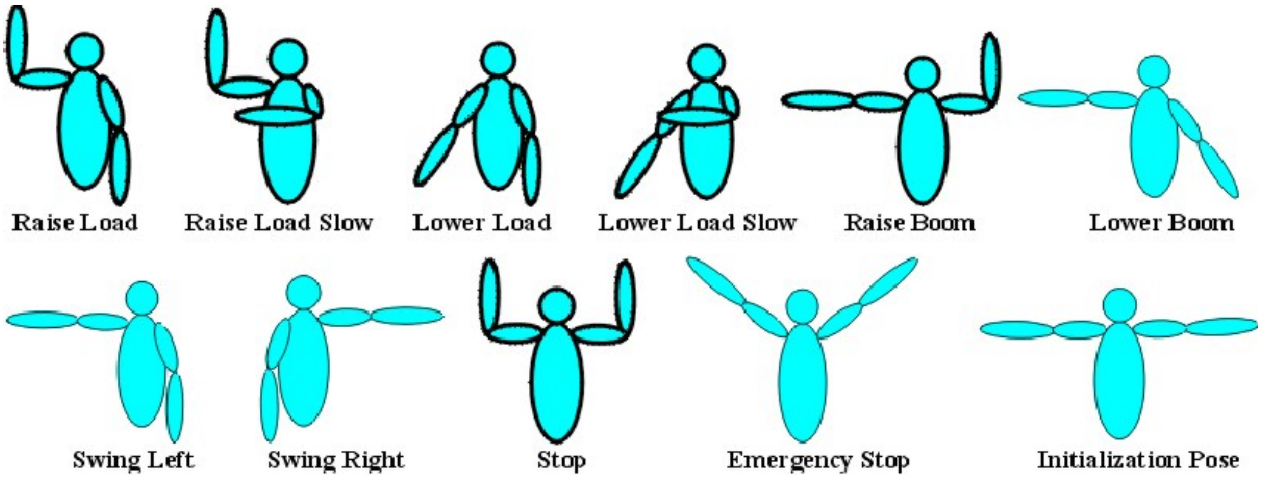

Figure 23: The 17 gestures depicted were successfully recognized by our system (gestures on the top row can be performed with the arms switched)

8.5 1^{st} Development Conclusions

In this phase of development, we were able to use the hardware and software that we had developed to track a low-dimensional human avatar at 16 Hz. The avatar we used had a total of eleven degrees of freedom: four in each arm, and x,y,θ for the position of the body. In addition we were able to employ our system to perform crane gesture interpretation and successfully interacted with a system using a robotic arm to simulate crane movement.

However, this project required many improvements. A tracking algorithm that uses a tighter torso model and allows for six DOF movement is needed. The new torso model will contain both shoulder and hip locations, hence our current tracking algorithm can be implemented to track the legs. With the improved model we would also like to implement a force based scheme in which forces transmit throughout the model. For example a point that pulls the arm outward should not only straighten the elbow but also have an effect on the shoulder. In addition, we still have not fully answered some of the challenges listed in the introduction. We plan to test the accuracy of our system by obtaining a ground truth to compare to the joint angles obtained through tracking. This will be accomplished by comparing our tracking results to values obtained with an Optotrak [3] sensor.

9 2^{nd} Development - Full Body Tracking

This development adresses the limitations of the first development and creates a usable system. The ultimate goal of this project is to autonomously track in real-time a full-body model of a human undergoing unconstrained movement in the workspace. The first stage of this process is to develop a method to acquire and track the human model. The model is a series of linked segments, which articulate at joints between the segments, as depicted in Figure 24. The model contains four degrees of freedom (DOF) in each arm and leg (rotations only), three DOF for the head (rotations), and six DOF in the back (3 translations and 3

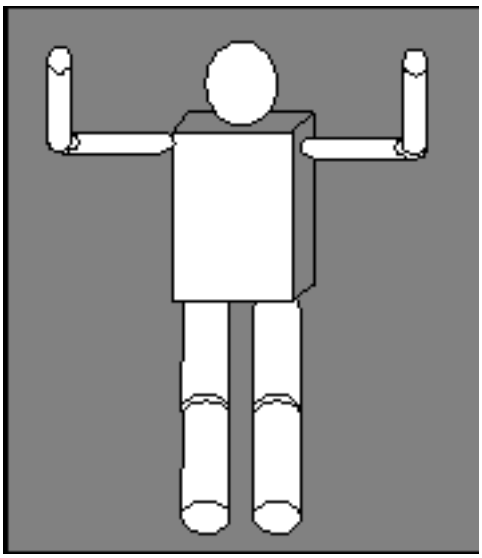

Figure 24: The reduced body model consists of a six degree of freedom torso $(x, y, z, \alpha, \beta, \gamma)$ with four degree of freedom articulated limbs (arms, legs) and a three degree of freedom head.

rotations). The segments model the body as closely as possible and do not change shape or size during tracking, which could cause errors in subsequent tracking.

9.1 Initialization

To acquire the model the user must still perform an initialization pose upon entering the workspace. The pose is simple stance in which body segments are in clear view (standing erect and facing an approximate direction with arms extended to the sides and legs separated from one another), so that it is easy to measure parameters of each body segment. Once in this pose the system segments out voxels associated with each body part using the same process that was outlined in the first development. Body parameters are then estimated for each segment by fitting an ellipsoid to all the voxels associated with that segment (as explained in Section 8.2).

9.2 Tracking a Twenty-Two Degree of Freedom Human Model

Once the model has been acquired, tracking of the model can begin. The tracking scheme is a physics based approach in which the data points, voxels, exert forces on the model. With our model one only needs to solve for the joint angles of succeeding segments, because they are anchored by the previous segment. This principal has been widely used for controlling the movement of robotic arms [2]. Along these lines each voxel exerts a force upon the model, which act like springs to pull the model into alignment with the data as shown in Figure 25. Accordingly the force exerted increases with the voxel's distance from the model, until greater than a maximum distance at which voxels are assumed to be erroneous and their pull is set to zero.

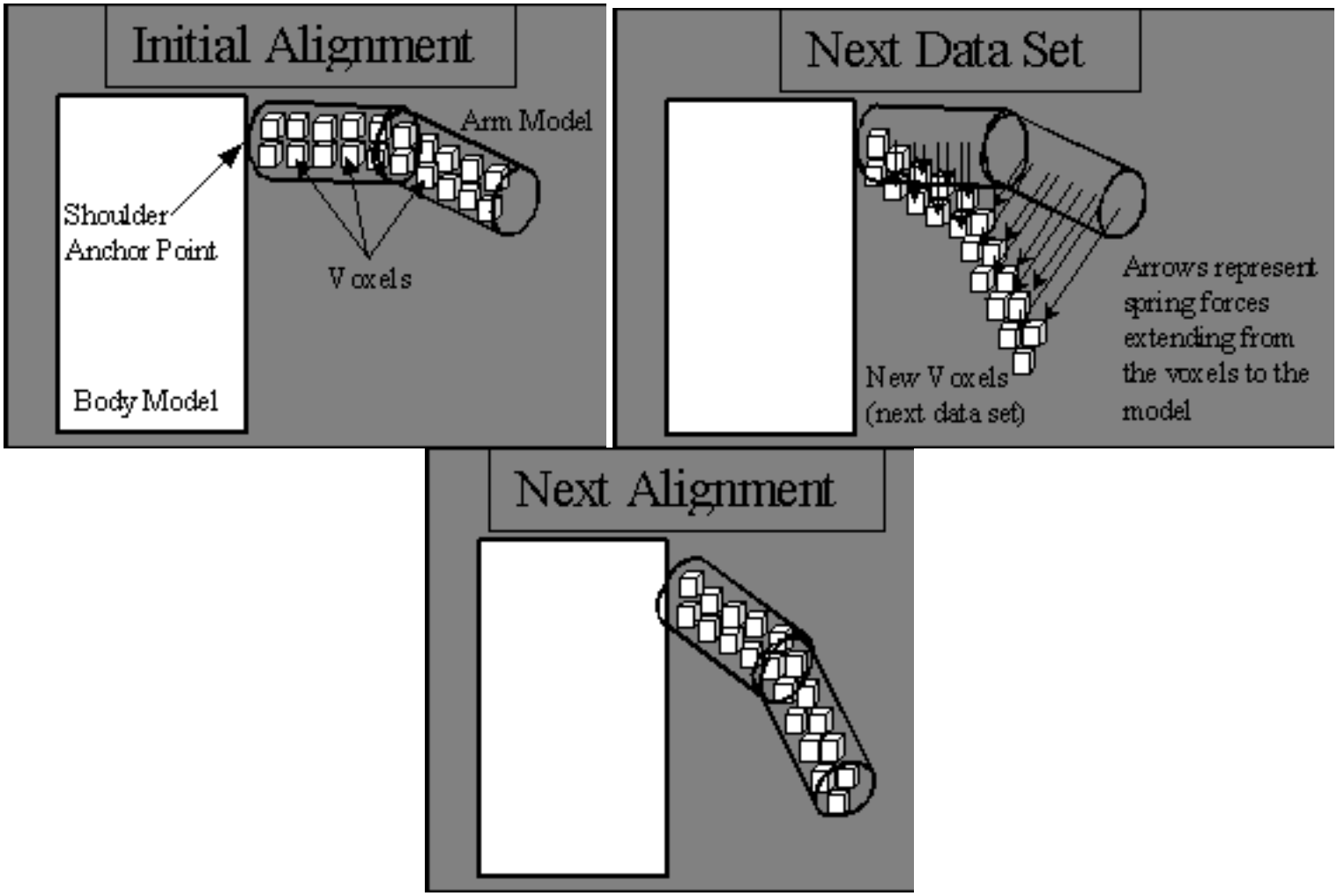

Figure 25: Voxels exert spring-like forces, which pull the model into alignment with the data.

This spring-like method is used in an iterative scheme in which several small adjustments to the model's position and orientation are made for each set of data. Since data is acquired at extremely fast rates (20 frames per second) the adjustment to the model will be small for each data set. The adjustment is calculated using the forces to calculate accelerations, which are then transformed into adjustments via the principle that $distance = at^2/2$ (time is arbitrary for this system since the forces and hence the acceleration are virtual).

This method is similar to the first development, however now forces transmit throughout the entire model. For example, now that the inertial properties of the entire arm are modeled even when voxels only project onto the lower arm the entire limb will adjust to align with the data. Hence the upper segment will still move into the correct orientation. If voxels only project onto the upper segment the angles associated with the lower segment will remain constant. This is appropriate because there is no reason to assume any change has been made if there is no data to support the change.

9.3 Torso Tracking

The torso model used has six degrees of freedom, translation in three directions and three rotations. Therefore, forces exerted on the torso must create both translational and rotational adjustments. Translation is calculated according to $\vec{a} = \vec{f}/m$, where $\vec{f}$ is a force vector of the sum of all the forces exerted on the torso, and m is the mass of the torso (mass is set to the number of voxels that project onto the torso). The rotational adjustment is calculated according to $\vec{\alpha} = \vec{t}/\vec{i}$, where $\vec{t}$ is vector of the sum of the torques created by each force about the centroid, and $\vec{i}$ is a vector of the inertia of the torso model. These equations yield accelerations that are used to calculate adjustments for the iterative alignment using $d = at^2/2$.

9.4 Limb Tracking

The legs and arms are anchored at their respective hip and shoulder locations on the torso, which are assumed to remain constant for a particular torso orientation. Therefore, the forces acting on the limbs translate to pure rotations in the form of torques. For a system of connected segments the Jacobian can be used to simplify torque calculations, $\vec{\tau} = J^T\vec{f}$ [2]. Newton-Euler dynamics relates joint torques to the velocities and accelerations of a system, $\vec{\tau} = M(\theta)\vec{\theta''} + \vec{v}(\theta, \theta') + \vec{g}(\theta)$, where M is an inertia matrix of the segments, $\vec{v}$ is a vector of the Coriolis and centrifugal terms, and $\vec{g}$ is a vector of gravity terms [2]. In this virtual environment the effects of Coriolis, centrifugal, and gravity forces can be thrown out because they do not exist. Accordingly the equations now relate the joint torques to the angular acceleration at each of the joints through the inertia matrix, $\vec{\alpha} = M^{-1}\vec{\tau}$. Again using the simple equation, $d = at^2/2$, one can compute an adjustment to the joint angles due to each force.

Since many voxels will exert pulls on each segment, it is far more efficient to combine all of the forces for each body segment into a single force before applying the Jacobian and inverted inertia matrix. It can easily be shown that if the resultant force vector is the sum of the individual force vectors applied to the segment, and if it acts on the axis of the arm a distance equal to $\vec{d} = \sum\vec{\tau}/\sum\vec{f}$ from the joint, the torques at each preceding joint will be equivalent.

Additional invariance to missing data/occlusions can also be built into this scheme by dynamically adjusting the model parameters to comply with the data. Hence the mass of each segment is dynamically set to the number of voxels that project onto the segment, and the center of mass of each segment is positioned at the center of all of the projected voxels. In this way the adjustments generated are appropriate to align the model for any data.

The largest problem encountered when using a Jacobian based method is the existence of singularities [2]. In tracking applications singularities appear because multiple degrees of freedom are modeled as a sequence of one-DOF joints with links of length zero between the joints. For example in the two-DOF joint shown in Figure 26, if θ_2 is rotated to -90 degrees, a force in the $Y2$ direction will no longer create a torque. Hence a degree of freedom is lost and a singularity has been encountered.

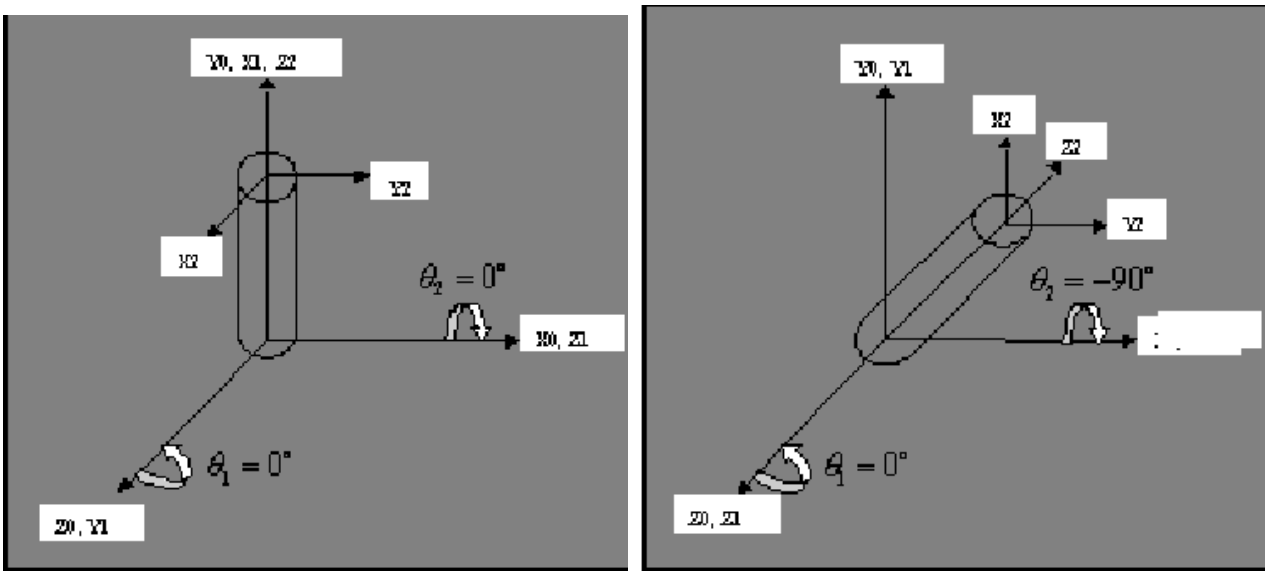

Figure 26: Modeling singularities. The left-hand picture shows the joint at the initial position where forces along $X2$ and $Y2$ directly produce torques around $Z0$ and $Z1$. The right-hand picture depicts a singularity ($\theta_2 = -90$ degrees), where a force along $Y2$ no longer produces a torque.

To achieve robust tracking, any singularities in our model must be avoided. A model with a two-DOF joint at the shoulder and a two-DOF joint at the elbow can achieve all possible configurations of the arm (since cylinders are used to model the arm). However, if either joint is modeled as two one-DOF joints the above-mentioned singularity exists. Looking at Figure 27 forces in the f_1 or f_2 direction should produce a joint torque independently of θ_1 and θ_2. This can be accomplished by first rotating into a coordinate system where θ_1 and θ_2 are 0.

To remove singularities in the second joint (elbow), forces along the f_5 and f_6 axes must also produce joint torques independently of the orientation of the lower arm. Accordingly the second joint is modeled as four consecutive one-DOF joints. In this way the first two joint angles account for the bend in the joint and torques are computed about the last two joints, whose joint angles are at zero. The inertia matrix, M, is consequently created for torques in the f_1, f_2, f_5, and f_6 directions. Performing the calculations in this manor not only removes the singularities but also decreases the complexity of the algorithm because the Jacobian and inertia matrices are now much simpler.

9.5 Head Tracking

The head model is again anchored at the neck, which is set according to the current torso orientation. Once more the rotations can easily be calculated using normal dynamics: $\vec{\alpha} = \vec{t}/\vec{i}$, where torques are calculated about the base of the head. As before adjustments are calculated according to $d = at^2/2$.

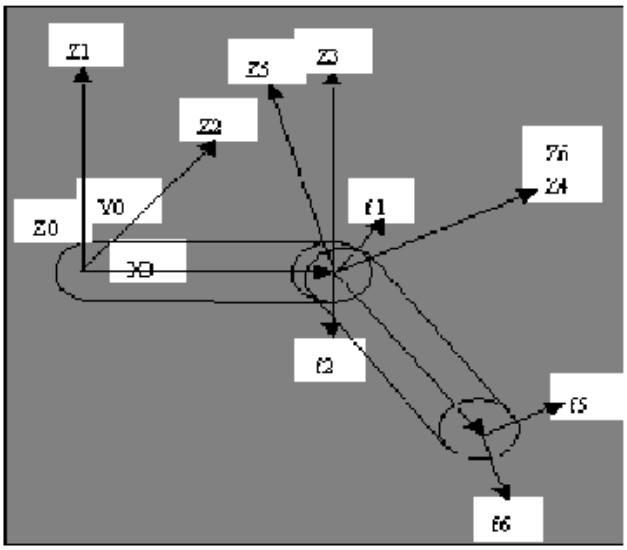

Figure 27: Removing singularities. A new arm model is shown, which removes singularities (Rotations are 1-6 are about axes $Z1-6$).

9.6 Visual Feedback

While tracking the user, real-time visual feedback is displayed on a screen in the corner of the workspace. The program can either display the voxels overlaid with colored spheres to indicate the joint positions and bars connecting the joints or display a human 3D avatar that mimics the tracking angles (upper-body only). Figure 28 shows the two display options.

9.7 Full Body Tracking Results

The current system is able to collect data, track our model, and provide visual feedback to the user at 20 frames per second on a single computer at a volumetric resolution of $2in^3$ voxels. Increasing the resolution to $1in^3$ voxels slows the system to about 9 Hz. In addition we have also made several movies of the tracked model overlaid on the volumetric data, available at `http://egweb.mines.edu/cardi/3dvmd.htm`. These movies show the system both acquiring and then aligning the model with the data in real-time while the user undergoes complex motions.

We have also set up an Optotrak [3] system to record ground truth pose information for a user while in the workspace. The Optotrak system recorded the position of markers placed upon our user in real-time. From these markers joint angles were be calculated and compared to the joint angles obtained with our tracking algorithm (attaching markers to the user does not effect our tracking algorithm). In this way we are able to compare our tracking results to highly accurate ground truth.
The comparison is taken for the data set named GroundTruth, a movie of it can be seen at `http://egweb.mines.edu/cardi/3dvmd.htm`. The average difference between the ground truth angles and the tracked angles was between 4 and 10 degrees, with a standard deviation

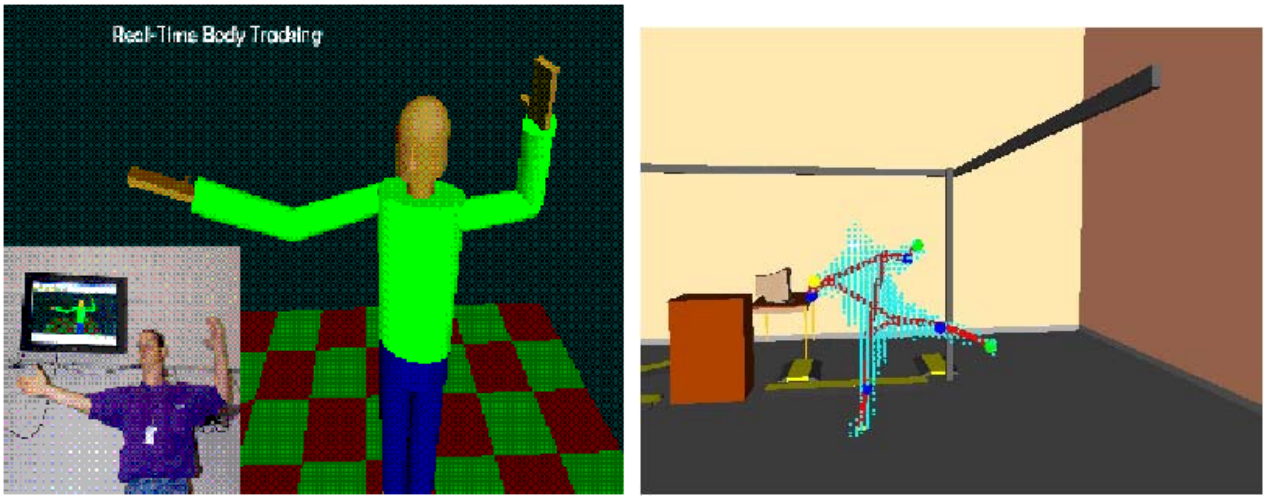

Figure 28: Real-time visual feedback. On the left is our human 3D avatar, and on the right is a display of the voxels with spheres and bars to represent the skeleton.

between 2 and 10 degrees. A plot of 2 of the joint angles is shown in Figure 29. These results are without collision, velocity, or joint limit constraints. We expect the standard deviation to lower once constraints are incorporated, however we do not expect that the average error will improve by a large amount. This is due to the both the tracking algorithm itself and because the data is a visual hull, as explained below.

One interesting aspect of the tracking results is that increasing the resolution to 1 inch voxels does not increase the tracking accuracy. In fact, the accuray decreases in many cases. This is probably because the sensor will detect more noise voxels near the boundry of limbs when smaller voxels are used (larger voxels would require noise spread over a larger area to be present before they would become active). This result is actually positive in that we do not need to exend large amounts of time in processing a large number of voxels to achieve accurate results.

An artifact of the data collection has an effect on the tracking. Since the data only approximates the visual hull and we are only using a finite number of cameras, tailing exists. In addition, any concavities in the body will be also be included in the visual hull, as seen in Figure 31. Accordingly these data artifacts can cause tracking to be inaccurate.

The system does have several limitations. Since self-collision constraints are not yet built into the system, limbs can get pulled into one another and into the body. Hence when separate body parts come into contact with one another, the tracking algorithm can fail. Currently there is no recovery algorithm to regain the pose once the tracking algorithm fails. These problems are currently being addressed.

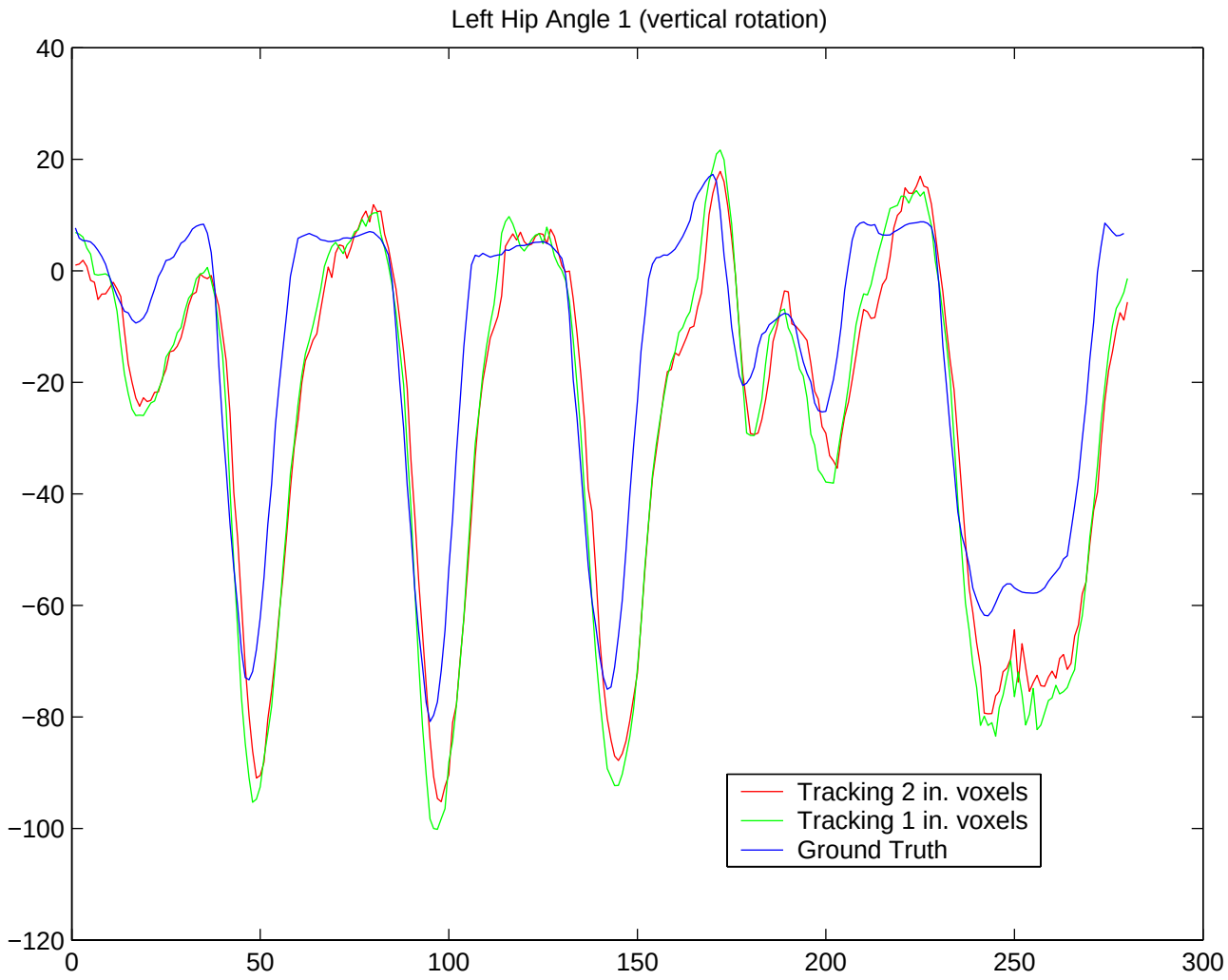

Figure 29: Ground Truth Comparison. The plots show the comparison of tracking results to ground truth data taken using an Optotrak sensor. This shows the comparisons of the left hip angles. [3].

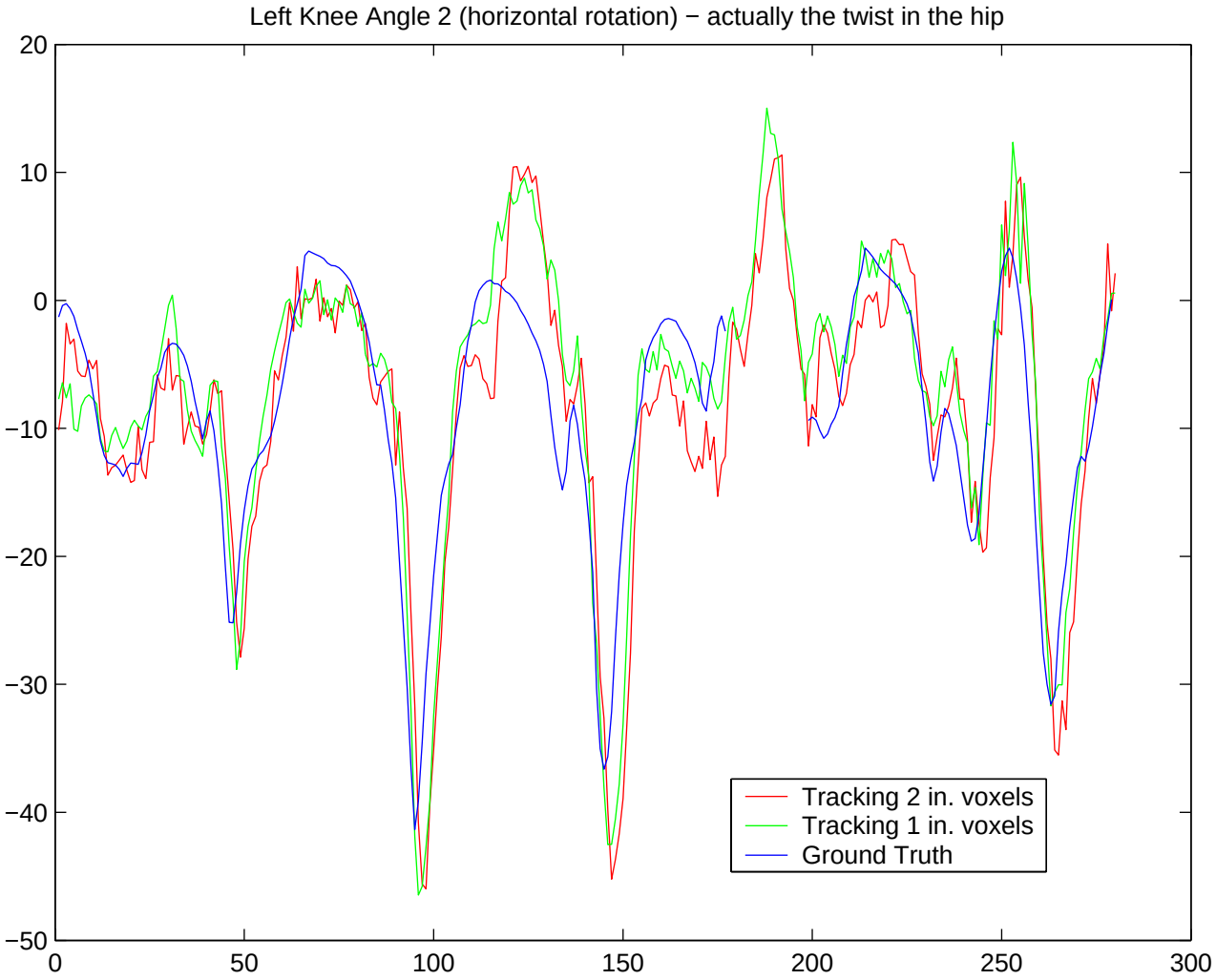

Figure 30: Ground Truth Comparison. The plots show the comparison of tracking results to ground truth data taken using an Optotrak sensor. This shows the comparisons of the left knee angles. [3].

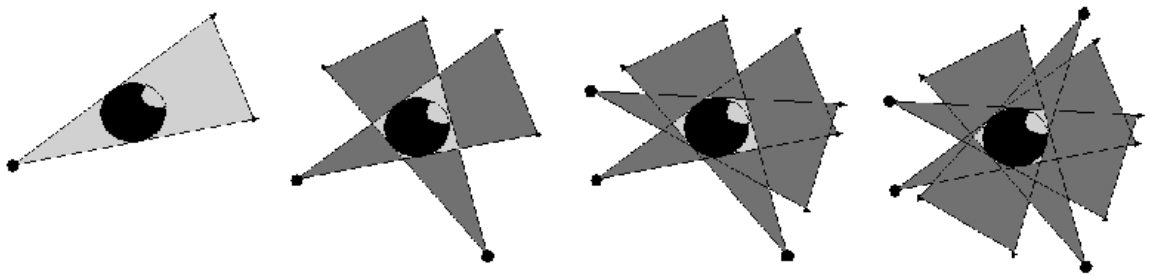

Figure 31: Visual hull - concavities and tailing problems exist in the data. This shows the progression of a 2D visual hull as more cameras are added to the system. The light gray area + the black object is the visual hull or the common intersection of the projections. The dark gray is the remainder of the union of the projections, and is eliminated from the visual hull.

9.8 Conclusions on Full-Body Tracking

We have built a shape-from-silhouette data acquisition and tracking system, which works in real-time on a single-processor commercially available computer, and designed an easy means to calibrate the system. The combination of a single computer and relatively simple calibration makes this system applicable for implementation in a large variety of applications.

The tracking system created for this sensor addresses all of the limitations of previous systems as listed by Gavrila [1] and the additional limitations presented by the authors.

1. The model is automatically acquired when the user enters the workspace by requiring the user to stand in an initialization pose.
2. The method is able to deal with occlusion because the forces exerted by voxel data transfer through the entire system. For instance if no data points project onto the upper arm, points on the lower arm will still pull both the lower and upper arm into the correct pose. Also the model parameters, mass and center of mass of each segment, adjust so the accelerations produced are correct for the current data, but do not affect the actual model shape.
3. Our process allows for easy incorporation of constraints (see future work below).
4. We are currently setting up an Optotrak [3] to obtain ground truth so that the accuracy of our tracking can be gauged.
5. Our RTS3 sensor obtains 3D data. By working directly with this data we avoid the problems of recovering 3D-pose information from 2D data.
6. We are able to work in real-time using only a single commercially available computer.
7. Lastly, we have developed a method to quickly and easily recalibrate data acquisition system, thereby making it reasonably portable for use in other areas.

The tracking algorithm is designed using sound dynamic and mathematical properties, and in doing so we have also made several valuable contributions to visual tracking. First of all, this will be the only human tracking work that we know of which compares its results to ground truth. Secondly, we have managed to eliminate the known singularities in Jacobian based tracking of our humanoid model. In doing so, we have also created a general tracking framework under which tracking can be extended to any rigid articulated model and remove known singularities. The system can also automatically acquire our model from a simple initialization pose, track the model, and provide real-time visual feedback on the same computer that acquires the data.

9.9 Future Work for Tracking

This method provides a good framework to build a robust tracking algorithm. To increase the robustness of the system we plan to make several improvements to our tracking algorithm.

- Invariance to outliers (erroneous data points) is currently built in through a distance threshold. However, a more robust algorithm could easily be incorporated, which applies weighting to the forces. Weights could either reduce or increase the force each voxel applies to the segment.
- Constraints are not included in the current algorithm. The simplest means to exercise self-collision constraints is to detect when to body segments are colliding and remove the portion of the calculated adjustment that would cause the segments to overlap. Joint limit constraints can also be incorporated through the same scheme.
- The recovery algorithm that reacquires the pose in case tracking gets lost needs to be modified for the 2^{nd} development.

Work is in progress to accomplish all of these improvements to the tracking algorithm.

References

[1] Gavrila, D.M., The Visual Analysis of Human Movement: A Survey. Computer Vision and Image Understanding, Jan. 1999. 73(1): p. 82-98.

[2] John J. Craig, Introduction to Robotics Mechanics and Control. 2nd Edition. Addison-Wesley Pub. Inc. Reading, MA, 1989.

[3] Web Site. Northern Digital Optotrak 3D Tracking Sensor, `http://www.ndigital.com/optotrak.html`, accessed August 2001.

[4] G. Cheung, T. Kanade, J. Bouguet and M. Holler. A Real-Time System for Robust 3D Voxel Reconstruction of Human Motions. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR '00)*, pp. 714-720, 2000.

[5] G. Cheung, S. Vedula and T. Kanade. (2001, May 14). Spanish Dancing (Flamenco) with Music. [WWW Document]. URL http://www.cs.cmu.edu/~virtualized-reality/dance.html

[6] K.R. Castleman. *Digital Image Processing*, Prentice Hall, Englewood Cliffs, NJ, USA, 1996.

[7] R. Tsai. An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR '86)*, Miami Beach, Florida, pp. 364-374. IEEE, June 1986.

[8] G. Wei and S. Ma. Implicit and Explicit Camera Calibration: Theory and Experiments. *IEEE Pattern Analysis and Machine Intelligence*, **16**(5), pp. 469-480. 1994.

[9] Z. Zhang. A Flexible New Technique for Camera Calibration. TechnicalReport MSR-TR98 -71, Microsoft Research, 1999.

[10] S. Bougnoux. From Projective to Euclidean Space Under Any Practical Situation, A Criticism of Self-Calibration. *Proc. 6th International Conference on Computer Vision (ICCV '98)*, pp. 790 796, 1998.

[11] D. C. Brown. Close-Range Camera Calibration. *Photogrammetric Engineering*, **37**(8), pp. 855-866, 1971.

[12] W. Faig. Calibration of Close-Range Photogrammetry Systems: Mathematical Formulation. *Photogrammetric Engineering and Remote Sensing*, **41**(12), pp. 1479-1486, 1975.

[13] S. Ganapathy. Decomposition of Transformation Matrices for Robot Vision. *Pattern Recognition Letters*, **2**(4), pp. 401-412, 1984.

[14] R. I. Hartley. An Algorithm For Self Calibration from Several Views. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR '94)*, pp. 908-912, Seattle, WA, 1994.

[15] J. Weng, P. Cohen, and M. Herniou. Camera Calibration with Distortion Models and Accuracy Evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**(10), pp. 965-980, 1992.

[16] A. Laurentini. How Many 2D Silhouettes Does It Take To Reconstruct a 3D Object?. *Computer Vision and Image Understanding*, **67**(1), pp. 81-87, 1997.

[17] A. Laurentini. The Visual Hull Concept for Silhouette-Based Image Understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(2), pp. 150-162, 1994.

[18] W.Matusik, C. Buehler, R. Raskar, S.J. Gortler and L. McMillan. Image-Based Visual Hulls. *IEEE Siggraph 2000, Computer Graphics Proceedings*, ACM Press / Addison Wesley Longman, Kurt Akeley, Editor, pp. 369-374, 2000.

[19] J.J. Carlson, S.A. Stansfield, D. Shawver, G.M. Flachs, J.B. Jordan, and Z.Bao. Real-Time 3D Visualization of Volumetric Video Motion Sensor. *Proc. SPIE Proceedings, Surveillance and Assessment Technologies for Law Enforcement*, Vol. 2935, pp. 69-79, 1997.

[20] J.J. Carson, C.Q. Little, and C.L. Nelson. Advanced 3.D Sensing and Visualization System for Unattended Monitoring. Sandia National Laboratories Tech Report SAND98-2803, 1998.

[21] J. Davis and A. Bobick. Virtual PAT: A Virtual Personal Aerobics Trainer. MIT Media Lab Perceptual Computing Group Technical Report No. 436, MIT, 1997.

[22] J. Davis and G. Bradski. Real-Time Motion Template Gradients using Intel CVLib. *Proc. IEEE ICCV Workshop on Frame-Rate Vision*, 1999.

[23] P. Rander, P.J. Narayanan, and T. Kanade. Virtualized Reality, Constructing Time Varying Virtual Worlds from Real Events. *IEEE Visualization*, Volume 552, pp. 277-283, 1997.

[24] M.Levoy and P. Hanrahan. Light Field Rendering. *Proc. SIGGRAPH* 1996.

[25] S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen. The Lumigraph. *Proc. SIGGRAPH* 1996.

[26] S.M. Seitz and C.R. Dyer. Photorealistic Scene Reconstruction by Voxel Coloring. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR '97)* pp. 1067-1073, 1997.

[27] W.N. Martin and J.K. Garwal. Volumetric Descriptions of Objects from Multiple Views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **5**(2), pp. 150-158, 1983.

[28] J.P. Luck, D.E. Small, and C.Q. Little. Real-Time Tracking of Articulated Human Models using a 3D Shape from Silhouette Method. *Proc. Robot Vision 2001*, **6**(2), pp. 78-86, February 15-17, 2001.

[29] G. Bradski, Editor. (2001, May 15) The Open Source Computer Vision Library. [WWW Document]. URL http://www.intel.com/research/mrl/research/opencv

[30] C. Bregler and J. Malik. Tracking People with Twists and Exponential Maps. *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR '98)*, 1998.

[31] D. M. Gravilla, The Visual Analysis of Human Movement: A Survey. *Computer Vision and Image Understanding*, **73**(1), pp. 82-98, 1999.

[32] Reactor Product Details. (2001, May 13), [WWW Document]. URL http://www.ascension-tech.com/products/reactor/details.html

[33] MotionStar Product Details. (2001, May 14) [WWW Document]. URL http://www.ascension-tech.com/products/motionstar

[34] Polhemus Web Site. (2001, May 14) [WWW Document]. URL http://www.polhemus.com

[35] Peak Performance Technologies. (2001, May 14) [WWW Document]. URL http://www.peakperform.com/biomech_prods.htm

[36] Northern Digital's Optotrak Sensor. (2001, May 14) [WWW Document]. URL http://www.ndigital.com/optotrak.html

[37] D.E. Small, E.J. Gottlieb, K. Edlund, and C. Slutter. A Design Patterns Analysis of the Umbra Simulation Framework. Sandia National Laboratories, Albuquerque, NM, Technical Report SAND2000-2380, 2000.

[38] J. Osterhout. Tcl/Tk Developer Exchange, (2001, May 14) [WWW Document]. URL http://www.scriptics.com

[39] OpenGL Graphics Developer Web Site. (2001, May 16) [WWW Document]. URL http://www.opengl.org

Distribution

1 Dr. Lance Williams
Computer Science Department
University of New Mexico
Albuquerque, NM 87131

1 Dr. William Hoff
Systems Engineering Department
Colorado School of Mines
Golden, CO 87131

1	MS	1010	M.E. Olson, 15222
1		1003	K.M. Jensen, 15212
1		1003	J.J. Carlson, 15212
1		1003	D.E. Small, 15212
1		1003	J.P. Luck, 15212
1		0316	S.S. Dosanjh, 15212
5		1004	RMSEL LIBRARY, 15200
1		9018	Central Technical Files, 8945-1
2		0899	Technical Library, 9616
1		0612	Review and Approval Desk, 9612
			For DOE/OSTI
1		0161	Patent and Licensing Office, 11500