Volume Intersection in a Plane for 2D Tracking

Adam Hoover
Electrical and Computer Engineering Department
Clemson University
Clemson, SC 29634-0915

Bent David Olsen
Department of Medical Informatics and Image Analysis
Laboratory of Image Analysis
Aalborg University
Denmark

Please direct correspondence to:

Adam Hoover
Electrical and Computer Engineering Department
Clemson University
Clemson, SC 29634-0915
email: ahoover@clemson.edu
phone: (864)-656-3377
FAX: (864)-656-5910, attn. Hoover

Abstract

A well-known limitation to volume intersection is its inability to reconstruct shape within 3D concavities. However, in this paper we show how volume intersection can be used to compute 2D occupancy in a support plane (for example, a floor or a table top) and in this case is not hindered by concavities. We prove that all visible points on the boundary of an object touching that plane are exactly reconstructible. We then describe an algorithm to compute the occupancy of space in that plane. The practical importance of this work is that using volume intersection for computing occupancy in a plane is extremely fast. We compare the speed of our methods to some other published works, and discuss results in testing our methods in a number of different environments and applications.

Key words: occupancy map, volume intersection, multi-camera system

1 Introduction

Volume intersection is a method for computing shape using multiple viewpoints. The shape of the object is inferred from its silhouette. The intersection of silhouettes from multiple viewpoints produces an approximation of the shape called the convex hull. Many works have been published demonstrating volume intersection methods [2, 4, 9, 16, 21, 23, 24, 25, 27, 31, 32]. In some of the earliest works in volume intersection, the main problems of interest were representation and algorithm, with the emphasis on computational resources. For example, a brute-force tessellation of 3D space (commonly called a voxel grid) can require a great deal of storage as the resolution of tessellation (and hence model fidelity) is increased. The oct-tree [2, 4, 23, 25] representation was used to address this problem through a hierarchical tessellation of space.

The most important limitation to volume intersection is that concavities in the object cannot be reconstructed, because the interior of a concavity does not appear in any possible silhouette (see Figure 1). Some works have tried to address this problem through additional analysis of the self-occluding contours [3, 36]. A recent promising approach is based upon projected voxel color matching (also called space sweeping) [6, 7, 10, 29, 30]. The traditional approach to volume intersection uses only the boundary of the silhouette, ignoring the colors of points in the interior of the silhouette. The voxel coloring approach matches the colors of projections of all points, which offers greater constraints.

Laurentini [22] showed that some of the points on the surface of an object are *hard points*, meaning that they are exactly reconstructible using volume intersection, while other points on the surface of an object are *soft points*, meaning that they cannot be exactly reconstructed using volume intersection. He gives proof of the conditions sufficient for hard points, and discusses their reconstruction.

In this paper we show how volume intersection can be used to exactly reconstruct the occupancy in a plane of the visible boundary of an object resting on that plane (see Figure 2). We prove that all visible points on the boundary of the object touching that plane are hard

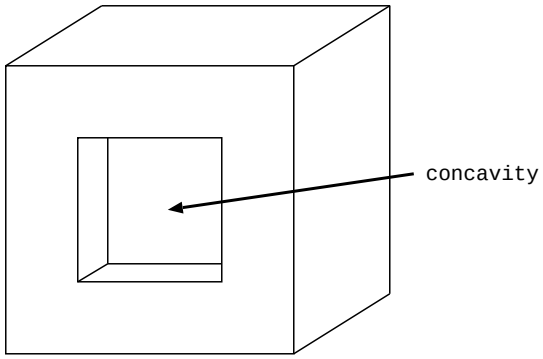


Figure 1: A concavity cannot be reconstructed using volume intersection, because the interior of the concavity does not appear in the silhouette of the object from any viewpoint.

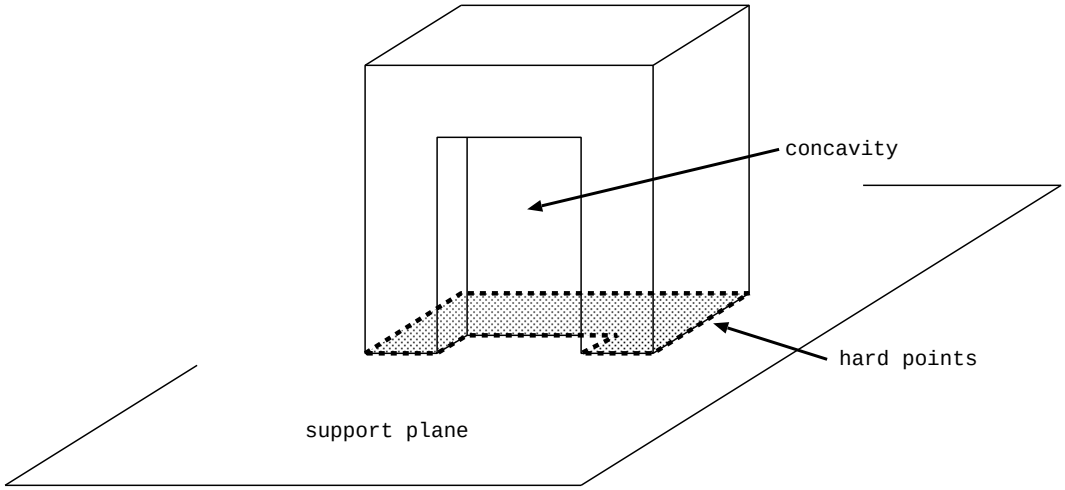


Figure 2: The 2D occupancy of space in a support plane (shaded grey in this figure) is exactly reconstructible using volume intersection, even in a 3D concavity.

points. We then describe an algorithm to compute the occupancy of space in that plane. The theoretical importance of this work is that we prove that volume intersection can be used to produce an exact shape model of occupancy in a supporting plane, and in this case is not hindered by concavities. The practical importance of this work is that this method for computing occupancy in a plane is extremely fast. We demonstrate a speedup of more than an order-of-magnitude over some previously published tracking methods that use the more common paradigm of segmentation and triangulation.

Many applications can make use of continuous monitoring of the occupancy of space in a plane. For example, a surveillance system can monitor the occupancy of ground space in

the area being monitored to detect intruders. A mobile robot can monitor the occupancy of floor space in hallways and rooms in order to avoid obstacles. An industrial robot arm can monitor the occupancy of space on a conveyor to locate objects for pick-and-plane manipulation. In Section 3 we demonstrate our methods operating in several environments and different applications.

2 Methods

We first prove that the occupancy in a support plane is exactly reconstructible. We then describe a method to compute this occupancy. Finally, we present the algorithm to implement this method and discuss its parallelism.

2.1 Proof of reconstruction

Laurentini provided the following proposition [22]:

> A necessary and sufficient condition for a point P belonging to the surface of a visual hull VH(O) to be hard is that at least one line L passes through P without intersecting VH(O) at any other point.

Laurentini proved the proposition by considering that for a point to satisfy this condition it must lie on the surface of the object (and not just the visual hull).

Figure 3 shows a diagram of the situation we are considering in this paper. The point of interest lies in a support plane (for example, a floor or a table top), and on the boundary of the object in that plane. We assume that the object does not extend past, or otherwise break, the support plane. If there exists a viewpoint above the plane that can observe the point of interest, we propose that such a viewpoint forms a line that satisfies Laurentini's proposition of a hard point.

In order to satisfy the proposition, this line must not otherwise intersect the visual hull of the object. Since the object cannot extend below the support plane, the visual hull cannot extend below the plane and so we only need consider points above the plane. For the point

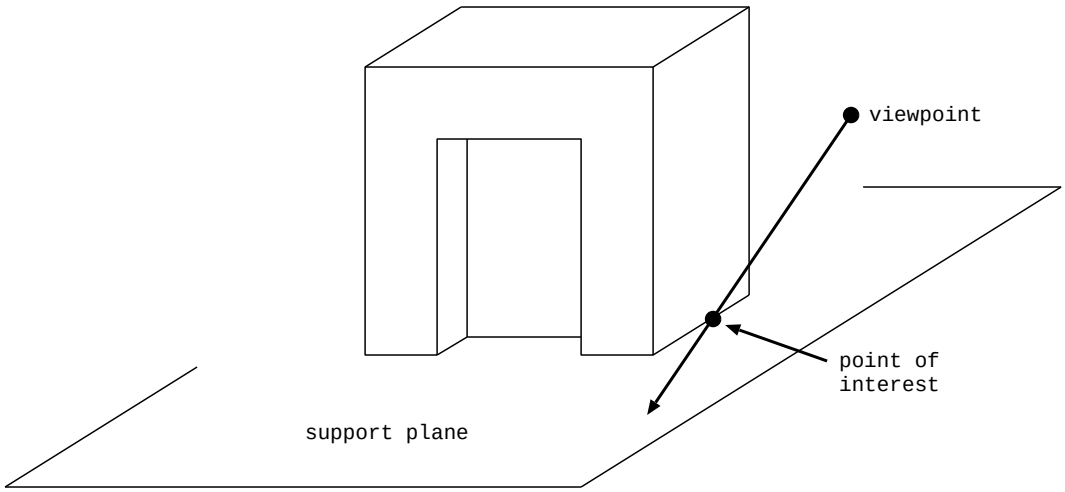


Figure 3: Any point that is visible and lies on the boundary of an object in a supporting plane satisfies Laurentini's definition of a hard point.

to be observable, there must be a line that passes through the point but does not pass through any other point in the object (else it would be occluded). Since the line does not pass through any other point on the object, it also does not pass through any other point on the visual hull of the object. (Laurentini calls this the *opaqueness property*, that "no visual ray can cross a soft surface without intersecting O (the object)".) Therefore the point under consideration is a hard point.

It is reasonable to infer that all points in the support plane enclosed by the observed boundary are occupied. (The object may be "hollow", but unless the object moves this space is not accessible.) Given that all points on the boundary are hard points, and so can be exactly reconstructed, this provides a theory to compute the occupancy of space in the support plane. As with the classic volume intersection approach, multiple viewpoints are generally necessary, in order to observe the complete boundary. We discuss the issue of number of viewpoints in more detail in Section 3.

2.2 Computing occupancy in the support plane

The idea described in Section 2.1 can be applied to the computation of occupancy in any support plane. In this section we describe a method to accomplish that construction, using

an indoor floor as an example.

Let (x, y, z) represent a right-handed three-dimensional coordinate system. The $z = 0$ plane is assumed to lie in the plane of the floor (the ground plane) of the area being monitored. Let $O[x, y]$, $0 \leq x < X$, $0 \leq y < Y$, x and y integers, define an X-columns $\times$ Y-rows discrete model of two-dimensional space in the $z = 0$ plane of the (x, y, z) world. Each $O[x, y]$ models a rectangle of space $\frac{1}{X} \times \frac{1}{Y}$ units in size. In this model, $O[x, y] = 0$ signifies a rectangle of empty space while $O[x, y] \neq 0$ signifies a rectangle of (at least partially) occupied space. We refer to $O[x, y]$ as an occupancy map [13]. Let $I[n, c, r]$, $0 \leq n < N$, $0 \leq c < C$, $0 \leq r < R$, n, c and r integers, define a set of C-columns $\times$ R-rows pixel images as captured by N cameras.

Our method is initialized in three steps. First, the cameras are calibrated to provide a transform $\mathcal{T}_n$ from each camera's image space $I[n, c, r]$ to the (x, y, z) world space. The transform $\mathcal{T}_n$ uses a set of camera model equations and calibration parameters to relate each pixel $[n, c, r]$ to a ray of space in world coordinates:

$$\mathcal{T}_n : [n, c, r] \rightarrow (x, y, z) + (i, j, k)d \qquad d > 0 \tag{1}$$

where (x, y, z) and (i, j, k) are known for each pixel $[n, c, r]$ via the transform $\mathcal{T}_n$. Our particular calibration model and methods are detailed in Section 2.3.

Second, a single background image $B[n, c, r]$ is acquired for each camera while the space to be monitored is empty. A binary mask $M[n, c, r]$ is created for each background image specifying which pixels are empty space, where $M[n, c, r] = 0$ signifies floorspace and $M[n, c, r] \neq 0$ signifies not floorspace. We use a polygon-drawing tool to delineate floorspace in the images for mask creation. The floorspace to be monitored is a matter of choice. There is no hard distinction between fixed obstacles and dynamic obstacles, but rather a continuum. For instance, a person is generally considered dynamic, a wall is generally considered fixed, and a piece of furniture may be considered either dynamic of fixed depending on its weight and function. In general, we emptied our experimental area of chairs, desks,

Figure 4: Background image and floorspace mask for one camera.

tables, etc., during setup, but ignored bookcases and heavy materials sitting along walls. An example background image and floorspace mask are shown in Figure 4.

The third step to initialize our method is to determine the occupancy map cell $O[x, y]$ that each image pixel $I[n, c, r]$ views. These locations are determined using the transform $\mathcal{T}_n$ to solve the following:

$$\mathcal{T}_n : [n, c, r] \rightarrow (x, y, z) + (i, j, k)d = (x_g, y_g, 0) \tag{2}$$

where x_g, y_g and d are the three unknowns in the three independent equations. The values (x_g, y_g) are rounded to the nearest integers. If $d > 0, \quad 0 \leq x_g < X, \quad 0 \leq y_g < Y$ and $M[n, c, r] = 0$, then the mapping $\mathcal{F}$

$$\mathcal{F} : I[n, c, r] \leftrightarrow O[x_g, y_g] \tag{3}$$

is established. $\mathcal{F}$ is not isomorphic, because any number of image pixels, from any number of cameras, may view a single occupancy map cell. Our particular solution for $\mathcal{F}$ is detailed in Section 2.4.

The basic operation of our algorithm is to detect differences between the background image $B[n, c, r]$ and live image $I[n, c, r]$ for each camera. A difference image $D[n, c, r]$ is computed as

$$D[n, c, r] = \begin{cases} 1 & \text{if} |I[n, c, r] - B[n, c, r]| > T \\ 0 & \text{if} |I[n, c, r] - B[n, c, r]| \leq T \end{cases} \tag{4}$$

Figure 5: Live image and difference image.

The threshold T controls the sensitivity of the algorithm, and is discussed further in Section 3. By applying $\mathcal{F}$, the difference image $D[n, c, r]$ produces the occupancy map:

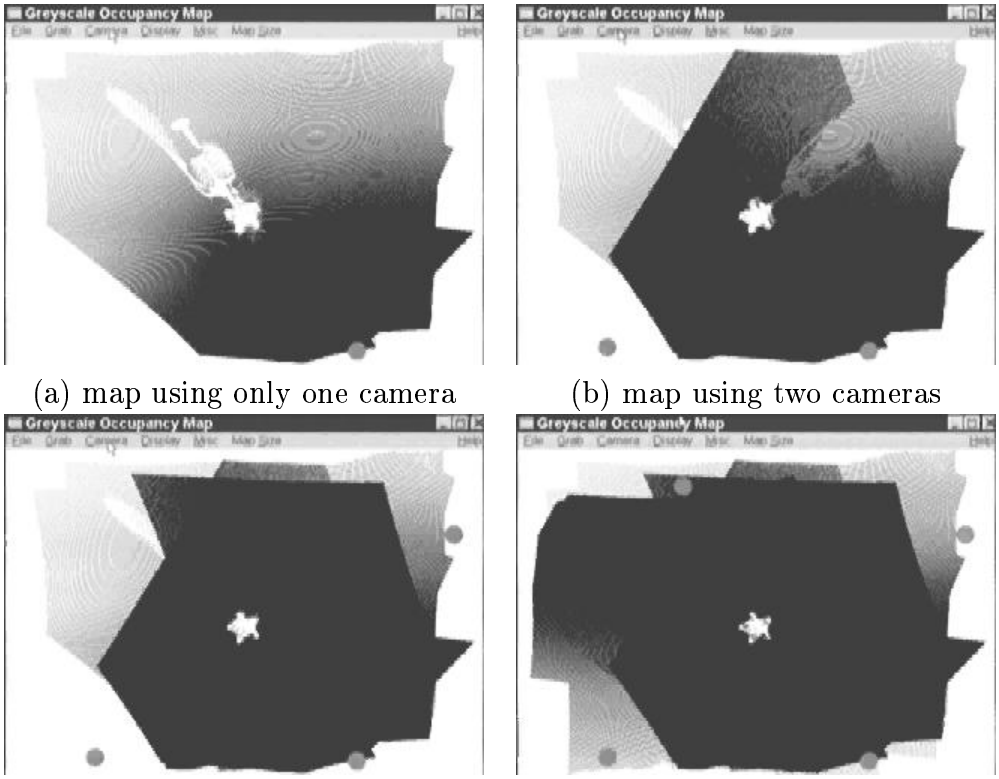
$$O[x, y] = \prod \mathcal{F}\{D[n, c, r]\} \tag{5}$$

Using Equation 5, an occupancy map cell is designated as empty if at least one image pixel that views it sees no difference ($D[n, c, r] = 0$). An occupancy map cell is only designated as occupied if every image pixel that can view it sees a difference ($D[n, c, r] = 1$). The implementation of this algorithm is detailed in Section 2.5.

Figure 5 shows an example live image and an image where differences with the background are highlighted. Pixels which are *not* different imply clear paths to the corresponding ground plane points, and therefore empty space in the corresponding occupancy map cells. From a single camera, this perception of occupancy shows the obvious effects of occlusion and perspective, as in Figure 6(a). As additional camera views are added, a reasonable depiction of the occupied floorspace emerges, as shown in Figure 6(b)-(d). Note that in this example, the concavities in the two-dimensional shape of the bottom of the chair are exactly reconstructed.

2.3 Camera Calibration Model

Any camera calibration model and technique may be used, but we recommend the camera model proposed by Tsai [34]. We have found it be robust for a variety of industrial and



(a) map using only one camera (b) map using two cameras

(c) map using three cameras (d) map using four cameras

Figure 6: Map computed using an increasing number of cameras. The camera locations are denoted with light grey circles near the edges of the occupancy map.

commercial cameras and lenses, and a dependable implementation is publicly available [35]. In related work [26] we describe steps to simplify calibrating a camera network, particularly in locating correspondences. The output from this procedure is the set of parameters for Tsai's camera model, for each camera in the network. This camera model consists of:

1. the external parameters T (translation vector) and R (rotation matrix), which locate and position the camera,

2. the internal parameters f (focal length), κ (radial lens distortion) and C_x, C_y (center of image), which describe the image projection, and

3. the internal parameters d_x, d_y (sensor element spacing), N_{cx}, N_{fx} (sampling resolution) and s_x (scale factor), which describe the process of image digitization.

These parameters are used to derive a mapping from each camera's image space to a world coordinate system.

Using Tsai's camera model, Equation 1 represents a sequence of four steps. First, a point in world space $P_w = \begin{bmatrix} x_w & y_w & z_w \end{bmatrix}^T$ is transformed to a point in the camera space $P_c = \begin{bmatrix} x_c & y_c & z_c \end{bmatrix}^T$ by applying a rigid body transformation

$$P_c = RP_w + T \tag{6}$$

where T is the translation vector and R is the 3×3 rotation matrix. Second, the point in the camera space P_c is transformed through perspective projection, using a pinhole camera model, to ideal undistorted image coordinates (U_u, V_u):

$$U_u = f\frac{x_c}{z_c} \tag{7}$$

$$V_u = f\frac{y_c}{z_c} \tag{8}$$

Third, the point is transformed through radial lens distortion to distorted image coordinates (U_d, V_d):

$$U_d = \frac{U_u}{1+\kappa r^2} \tag{9}$$

$$V_d = \frac{V_u}{1+\kappa r^2} \tag{10}$$

where κ is the distortion coefficient and $r = \sqrt{U_u^2 + V_u^2}$. Fourth, the point is transformed through digitization as it is captured by a framegrabber:

$$U_f = \frac{s_x}{d_x}\frac{N_{fx}}{N_{cx}}U_d + C_x \tag{11}$$

$$V_f = \frac{1}{d_y}V_d + C_y \tag{12}$$

$$\tag{13}$$

where (U_f, V_f) are image pixel coordinates.

2.4 Transform from image space to map space

Using Tsai's camera model, the solution to Equation 2 is a series of five steps. This solution begins by taking a point backward through the steps given in Section 2.3. The first step transforms a point (U_f, V_f) from image coordinates to distorted image coordinates (U_d, V_d):

$$U_d = (U_f - C_x)\frac{d_x}{s_x}\frac{N_{cx}}{N_{fx}} \tag{14}$$

$$V_d = (V_f - C_y)d_y \tag{15}$$

where $C_x, C_y, d_x, d_y, s_x, N_{cx}$ and N_{fx} are camera parameters established during calibration. The second step transforms from distorted image coordinates to undistorted image coordinates (U_u, V_u):

$$U_u = U_d(1 + \kappa r^2) \tag{16}$$

$$V_u = V_d(1 + \kappa r^2) \tag{17}$$

where κ is a camera parameter and $r^2 = u_d^2 + v_d^2$. The third step transforms undistorted image coordinates to camera (3D) coordinates. This step relates an image pixel to a ray. This ray is described by a starting point and a directed vector. The focal point, located at $(0, 0, 0)$, is taken as the starting point. The image plane is located at f (the focal length) on the z-axis (in camera coordinates). The directed vector may therefore be described by its intersection with the image plane (U_u, V_u, f), so that

$$[U_u, V_u] \rightarrow (0, 0, 0) + (U_u, V_u, f)d \tag{18}$$

where d is an unknown. The fourth step transforms the directed vector from camera coordinates to world coordinates. Let $P_{c_1} = \begin{bmatrix} x_c & y_c & z_c \end{bmatrix}^T = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ and $P_{c_2} = \begin{bmatrix} i_c & j_c & k_c \end{bmatrix}^T = \begin{bmatrix} U_u & V_u & f \end{bmatrix}^T$ describe the ray in camera coordinates. Both P_{c_1} and P_{c_2} are transformed by

$$P_w = R^{-1}(P_c - T) \tag{19}$$

to give $P_{w_1} = \begin{bmatrix} x_w & y_w & z_w \end{bmatrix}^T$ and $P_{w_2} = \begin{bmatrix} i_w & j_w & k_w \end{bmatrix}^T$, where P_w and P_c are points in world coordinates and camera coordinates, respectively, and R and T are camera parameters established during calibration. The fifth and final step solves for the intersection of this ray with the ground plane ($z = 0$):

$$(x_w, y_w, z_w) + (i_w, j_w, k_w)d = (x_g, y_g, 0) \tag{20}$$

where (x_g, y_g) and d are the unknowns as described in Equation 2.

2.5 Algorithm

Since speed of execution is the primary practical importance of this work, it is important to discuss the actual implementation of our methods in some detail. All the calculations necessary to solve for $\mathcal{F}$ (Equation 3) are independent of image content. Therefore $\mathcal{F}$ can be computed off-line and stored as a look-up table. $\mathcal{F}$ provides a two-way mapping, so that it may be applied in two different manners. The look-up table $L_1[n, c, r]$ relates each image pixel for each camera to a unique occupancy map cell. The look-up table $L_2[x, y]$ relates each occupancy map cell to a set of image pixels, where each set may include any number of pixels (including zero) from each camera. The use of $L_1[n, c, r]$ and $L_2[x, y]$ lead to different algorithms, which we refer to as image-based and map-based.

In the following pseudo-code listings we maintain the notation established in Section 2.2: $O[x, y]$ is the occupancy map, $I[n, c, r]$ is a set of live images from N cameras, and $B[n, c, r]$ is a set of background images acquired during system initialization. The indices x and y refer to map coordinates, c and r refer to image coordinates, and n refers to camera number. $L_1[n, c, r]$ and $L_2[x, y]$ refer to look-up tables storing the mappings described by $\mathcal{F}$ (Equation 3). The threshold T controls the sensitivity of the algorithm, and is discussed further in Section 3.

The arrays $O[x, y]$, $I[n, c, r]$, $B[n, c, r]$, $L_1[n, c, r]$ and $L_2[x, y]$ are multi-dimensional, yet they can be accessed in one-dimensional order because they have discrete boundaries. For

the sake of clarity, in the following algorithm descriptions we maintain the multi-dimensional notation. However, loops on (x, y), on (c, r), and on (n, c, r), can be written using a single-index loop. This reduction in loop overhead yields faster executions.

The image-based algorithm uses the look-up table $L_1[n, c, r]$, and is described by the following pseudo-code:

```
loop ... time ...
  loop x = 0 ... map columns
    loop y = 0 ... map rows
      O[x,y] = 1
    end loop
  end loop
  loop n = 0 ... number of cameras
    loop c = 0 ... image columns
      loop r = 0 ... image rows
        if (|I[n,c,r]-B[n,c,r]| < T)
          O[L1[n,c,r]] = 0
        end if
      end loop
    end loop
  end loop
end loop
```

The arrays $I[n, c, r]$, $B[n, c, r]$, and $L_1[n, c, r]$ are accessed in sequential order, which can be exploited by a cache memory. The array $O[x, y]$ is accessed in non-sequential order, so that caching does not speed up its access. Entries in $L_1[n, c, r]$ that are unused (entries for image pixels which do not map to support plane points) are given a sentinel value that points to a harmless memory location outside the occupancy map. For instance, the occupancy map

array is allocated as $X \times Y + 1$ cells, and the address of the extra cell becomes the sentinel. An alternative is to add a second conditional statement testing the mask. In the code given above, the inner-most loop is modified as follows:

```
if (M[n,c,r] == 0)
  if (|I[n,c,r]-B[n,c,r]| < T)
    O[L1[n,c,r]] = 0
  end if
end if
```

In this case an extra conditional statement is executed for every pixel, whereas in the original code non-useful assignment statements may be executed for some pixels. In our experiments we always found the sentinel approach to be faster than the mask approach.

The map-based algorithm uses the look-up table $L_2[x, y]$. Entries in $L_2[x, y]$ are sets of image pixel identities. The size of each set varies depending on how many image pixels view the occupancy map cell. This detail can be simplified by placing a maximum on set size, so that $L_2[x, y]$ may be implemented as a three-dimensional array. The constant set size S is selected so that the majority of the mappings in Equation 3 may be found in $L_2[x, y, s]$.

The map-based algorithm is described by the following pseudo-code:

```
loop ... time ...
  loop x = 0 ... map columns
    loop y = 0 ... map rows
      O[x,y] = 1
      loop s = 0 ... S
        if (|I[L2[x,y,s]]-B[L2[x,y,s]]| < T)
          O[x,y] = 0
          exit loop s
        end if
```

```
        end loop
      end loop
    end loop
  end loop
```

In the map-based algorithm, the arrays $L_2[x, y, s]$ and $O[x, y]$ are accessed in sequential order, while the arrays $I[n, c, r]$ and $B[n, c, r]$ are accessed in non-sequential order. As with the image-based algorithm, unused entries in $L_2[x, y, s]$ may be handled using sentinel addressing or masking. The sentinel version of the code is shown above. In this case entries in $L_2[x, y, s]$ which do not map to image pixels are given a sentinel value that points to memory locations outside the image and background image spaces that cause the conditional statement to fail.

In our experience, the image-based algorithm is almost always faster than the map-based algorithm. However, the map-based algorithm implicitly solves the correspondence problem (all pixels which see the same support plane point are grouped), so it offers potential for future work to exploit a more complex fusion approach than the simple AND operator used in Equation 5. Both the image-based and map-based algorithms have great potential for parallelism on a multiprocessor architecture. In [12] we explored that potential and discovered near linear speedup for both algorithms on a multiprocessor architecture. We expect this to be increasingly important as common desktop computers move from uniprocessor to dual-processor architectures (common today) to N-processor architectures (based on the current trend it seems as though this is likely to become common in the future).

3 Experiments

We have tested our methods in five different locations, using three different computing platforms, and applied to four different applications. The following locations were used in experiments:

1. UCSD lab. A university lab, carpeted floor, monitored space 9×6.5 m^2 in area. Bookshelves lined most of the walls, on which were mounted four Sony XC-999 CCD cameras.

2. Clemson lab. A university lab, carpeted and tiled floor, monitored space 5×4 m^2 in area. Six Sony XC-75 CCD cameras were mounted on a drop ceiling near the corners and centers of two walls.

3. Industrial lab. An industrial lab, linoleum floor, monitored space 3.5×3.5 m^2 in area. Six Sony XC-75 CCD cameras were mounted on industrial framing surrounding an industrial robot arm and conveyor centered in the monitored space.

4. Living room. A residential room, ceramic tiled floor, monitored space 6×4 m^2 in area. Four Sony XC-ST50 CCD cameras were mounted on tripods raised to ceiling height in the corners of the room.

5. CVPR 2000. A hotel ballroom, carpeted floor, monitored space 5×5 m^2 in area. Four Sony XC-75 CCD cameras were mounted near the top of poles placed at the corners of the area.

All locations were indoor, and sunlight was minimal. In each location the cameras were deployed in positions similar to those used for video security (approximately 2.5 to 3 meters above the ground plane, evenly distributed on the perimeter of the area being monitored). The cameras were equipped with lenses ranging from 3.5 to 6.0 mm.

In every location, all the cameras were synced and hardwired to a single computer, equipped with two framegrabbers (using the red, green and blue channels to grab synced greyscale images from three separate cameras). The following computers were used in experiments:

1. 1997 PC. A custom PC equipped with a 233 MHz Pentium 2 processor and two Matrox Meteor framegrabbers. Used in the UCSD lab.

2. 1999 PC. A Dell PC equipped with dual 450 MHz Pentium 2 processors and two Imaging Technology IC-RGB framegrabbers. Used in the Clemson lab, the living room, and CVPR 2000.

3. 2001 SMP. A Gateway rack workstation equipped with eight 550 MHz Pentium Xeon processors and two Imaging Technology PC-RGB framegrabbers. Used in the industrial lab.

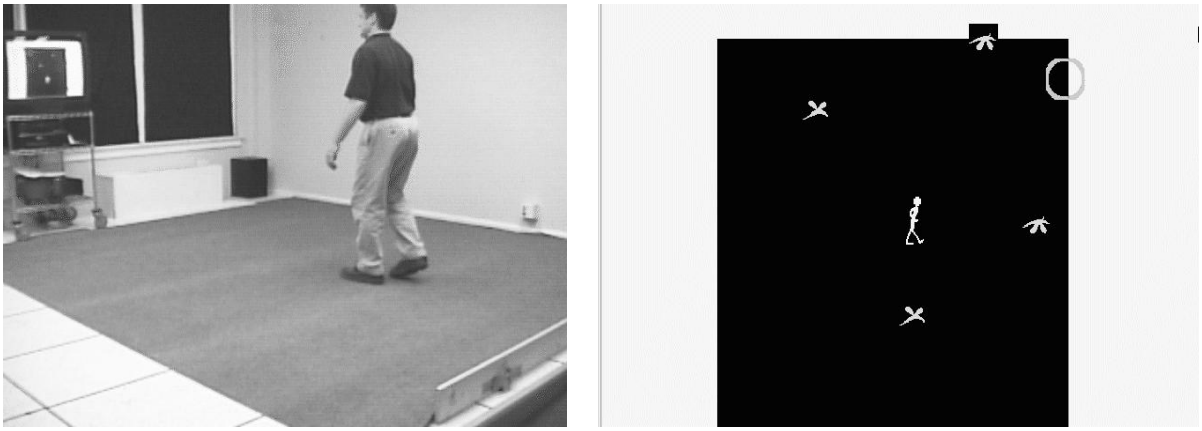
For every platform, the code was written in C, compiled using Microsoft's Visual C++, and executed under the Windows operating system (NT 4.0 or Advanced Server). The images were always processed at full resolution (640×480) and the occupancy map size was always 640×480. Not including camera installation and wiring, the total time to initialize each system, including calibration, mask creation, and look-up table creation, took under ten minutes. In the UCSD, Clemson and industrial labs, the system has been run continuously for up to three months using a single initialization.

These systems supported experimental work in several application domains. The following applications have been tested:

1. Mobile robot navigation. Tracking and controlling the motion of one or two mobile robots that are otherwise sensor-less [19, 20]. Static obstacles such as furniture and dynamic obstacles such as people were tested. A video of some experiments in the UCSD lab was published at ICRA in 1999 [18]. This application is under continuing development in the Clemson lab, and was demonstrated in a day-long session at CVPR 2000 [5], with as many as five people at a time in the experimental space.

2. Learning object dynamics. Tracking the motion of one or more unknown objects after a controlled collision with a mobile robot, in order to learn the object dynamics (e.g. mass, coefficient of friction, etc.). Tested in the Clemson lab. Video clips may be seen at the project website[1].

[1] www.ces.clemson.edu/~ahoover/snr



person playing game virtual game display

Figure 7: Snapshot of our method applied to augmented reality game.

3. Dynamic object manipulation. Tracking the unknown semi-random motion of an object in order to effect manipulations. Tested in the industrial lab. Video clips may be seen at the project website[2].

4. Augmented reality gaming. Tracking a person as he or she tries to cross a room, watching a screen of the tracking result superimposed with virtual enemies. Tested in the Clemson lab and the residential living room. Figure 7 shows a snapshot of a person playing the game.

Tracking in the occupancy map was accomplished by finding the centroid of occupied space in a window surrounding the previous location of the object(s). For each application, the window size was set based upon the maximum expected velocity of the object(s) being tracked.

Figure 8 shows some snapshots from a sequence of chairs being pushed across the UCSD lab. Figure 9 shows some snapshots from a sequence of two people walking around the lab. Figure 10 shows some snapshots of a robot navigating a path in the lab. In each figure the observation viewpoint in the accompanying camera views is from the lower-left corner of the occupancy map, looking rightward across the map space.

[2]www.ces.clemson.edu/~ahoover/workcell

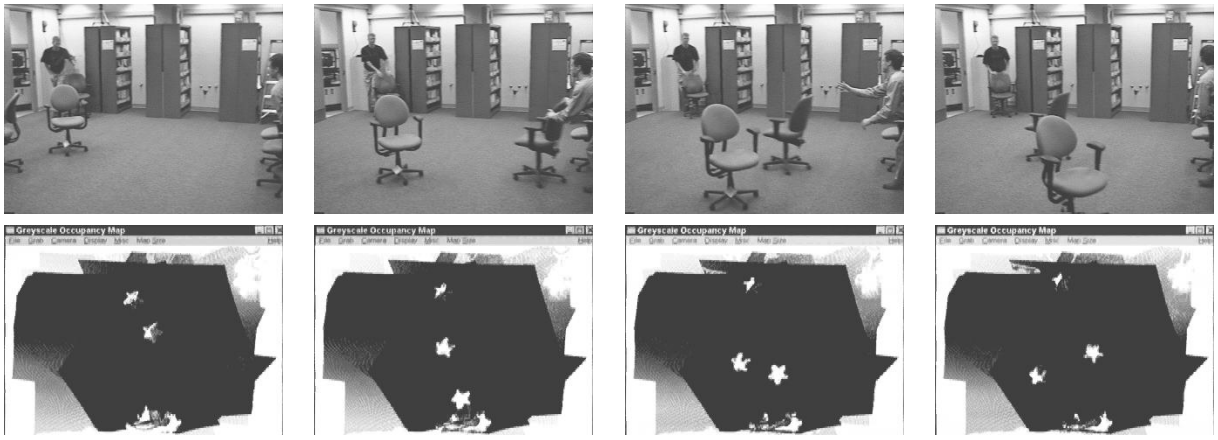


Figure 8: Example result for chairs being pushed across the room.

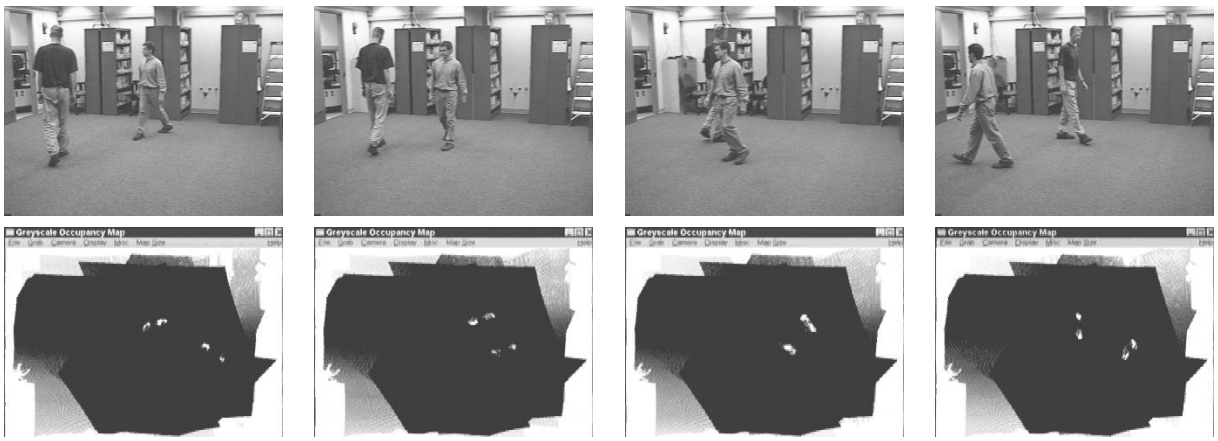


Figure 9: Example result for people walking around the room.

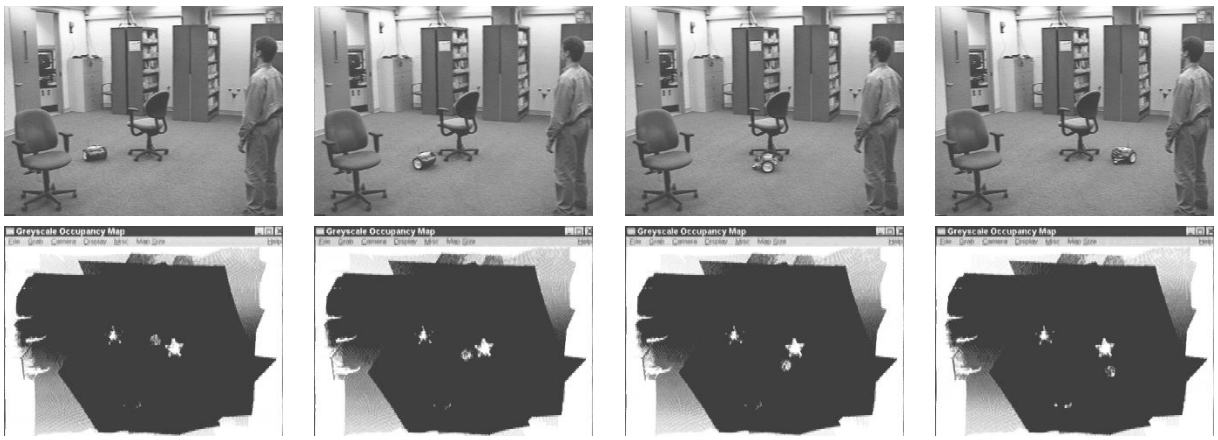


Figure 10: Example result for a robot moving around a chair.

It is interesting to compare our system against other published tracking systems. Table 1 presents statistics for some systems built under the VSAM (Video Surveillance and Monitoring) project [8]. The first three rows list previously published systems, the last two rows list our UCSD and Clemson lab systems. Before discussing the statistics, it is important to note the limitations of this comparison. All the cited methods use the more common paradigm of tracking in individual camera views, followed by fusing the tracking results across multiple cameras. Our method performs fusion first (in the creation of the occupancy map), followed by tracking in the fused space. Thus, our approach is limited to 2D tracking while the cited methods can potentially provide 2D or 3D tracking results. The columns in Table 1 list the number of cameras, pixel resolution of images, and frames per second (fps) for data processed by a single computer in each system. SPEC is a non-profit organization that benchmarks computer architectures for comparison. Roughly speaking, a SPEC ratio of 2:1 indicates the computer with the higher number runs programs an average of twice as fast as the computer with the lower number. The complete architecture details were not available in the cited publications so the SPEC numbers reported here are best approximations based upon available data. The summary statistic (final column in Table 1) is the number of pixels processed per second per SPEC, which can be thought of as the algorithmic throughput of pixels per computing-power. As seen in the summary statistic, our method shows more than an order of magnitude speedup over these previously published methods. It is important to note that this comparison should not be given undue weight, as these tracking algorithms were all developed with somewhat different goals. None-the-less, in applications where 2D tracking in a plane is sufficient, our method shows a clearly favorable speed.

The gaming application in the Clemson lab is used as a demonstration and recruiting tool for visiting high school students, general engineering freshmen students, and other tour groups. During 2001-2003 it has been tested on over two thousand people, with no precautions taken regarding the footwear or clothing worn. During all these demos the system rarely failed to detect and track a person, perhaps one per hundred participants. In the few

	cameras	resolution	fps	computer architecture	SPEC[1]	$\frac{\text{pixels}}{\text{sec} \cdot \text{SPEC}}$
[33]	1	160 × 120	12	SGI O2 R10000	10	23k
[11]	1	640 × 240	15	dual 550 MHz Pentium 3	45	51k
[15]	1	320 × 240	25	dual 400 MHz Pentium 2	34	56k
UCSD	4	640 × 480	5	266 MHz Pentium 2	10	614k
Clemson	6	640 × 480	20	dual 450 MHz Pentium 2	37	996k
[1] SPEC '95 int base (approximated)						

Table 1: Comparison of speed of our method to some previously published works.

cases where it has failed, this was usually due to a person wearing footwear of a color too close to the flooring to be distinguished. At our CVPR 2000 demonstration [5], the usual interaction involved one or more people blocking the path of the robot to force a change in its planned path. Some participants hovered over the robot, almost sitting on it, in an attempt to force the tracking of the robot to fail. Over a six hour period, there were only five tracking failures.

We have witnessed three problems that can affect our method. First, the system is based upon image differencing, which is implemented in the algorithm threshold T. It is of course possible to introduce an object of similar color to the background, and set T to a value such that the system misses the object. In practice, setting $1 \leq T \leq 10$ forces the system to see things that people generally cannot see, while setting $T > 60$ forces the system to miss things that people usually can see. In our experiments, setting $15 < T < 50$ produced results that generally matched human perceptual expectations. Second, the system depends upon having multiple views of each mapped cell of the monitored plane. These views are optimally from cameras with opposing views. Depending on camera placement, as an object moves towards the boundary of the monitored space this principle can be violated, causing occlusion distortion. This problem is reduced if the cameras can be placed outside the perimeter of the monitored area, instead of right on its boundary. The third problem we have witnessed with our method is the deleterious effect of shadows. Since each pixel is processed independently, other than setting T appropriately it is impossible to differentiate between a shadow and

an actual object. In general we have tried to control the lighting and environment in our experiments to reduce this problem, but it remains an open issue.

4 Discussion

The common approach to tracking is to model the world as being empty, then fill that model with observed objects (for example see [28]). Using a network of cameras, the image captured by each camera is segmented. Segmented areas of interest are then matched across cameras, and triangulation is used to compute object locations. Abidi and Gonzalez [1] call this *symbolic-level data fusion*. In contrast, the volume intersection approach to tracking assumes the world is completely filled, then empties parts of the model based on observations of free space. This allows for what Abidi and Gonzalez call *pixel-level data fusion*. The advantages of this approach may be enumerated as follows:

1. The segmentation problem is avoided, because each pixel from each camera is processed independently.

2. The correspondence and camera-handoff problems are avoided, because data fusion is performed at the pixel level (before tracking).

3. All the necessary triangulations can be computed off-line (for example, in our approach they are stored in lookup tables).

4. During tracking, an object maintains a constant size in a fused occupancy space, whereas size diminishes in a camera space as the object moves away from the camera.

One drawback is that the strength of avoiding segmentation can also be a weakness. In the case of a shadow, a single pixel does not generally provide sufficient information to determine its identity. Our current work is focused on the possibilities of extending Equation 5. We are working on methods to analyze the set of image differences at a single location in space, as observed from different viewpoints, in order to reason about the identity of that

point. Another potential drawback to our approach is its reliance upon background images for differencing. Several works (for example [14]) have explored methods to periodically or continuously update background images to overcome slow changes in lighting or in background content. It should be possible to implement one of these methods to automatically update background images for our system, without sacrificing our speed advantage. Currently, we manually perform this update as needed.

References

[1] M. Abidi and R. Gonzalez (ed.), *Data Fusion in Robotics and Machine Intelligence*, Academic Press, 1992.

[2] N. Ahuja and J. Veenstra, "Generating Octrees from object silhouettes in orthographic views", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 11, 1989, pp. 137-149.

[3] J. Carr, W. Fright, A¿ Gee, R. Prager and K. Dalton, "3D shape reconstruction using volume intersection techniques", in proc. *IEEE Int'l Conf. Computer Vision*, Bombay, India, 1998, pp. 1095-1100.

[4] C. Chien and J. Aggarwal, "Model reconstruction and shape recognition from occluding contours", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 11, 1989, pp. 372-389.

[5] T. Christensen, M. Noergaard, C. Madsen and A. Hoover, "Sensor Networked Mobile Robotics", demonstration in *IEEE Computer Vision & Pattern Recognition*, Hilton Head, SC, 2000, pp. 782-783.

[6] R. Collins, "A Space-Sweep Approach to True Multi-Image Matching", in proc. *IEEE Comp. Vision & Pattern Recognition*, 1996, pp. 358-363.

[7] R. Collins, "Multi-Image Focus of Attention for Rapid Site Model Construction", in proc. *IEEE Comp. Vision & Pattern Recognition*, 1997, pp. 575-581.

[8] R. Collins, A. Lipton and T. Kanade, "Special Section on Video Surveillance", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 22 no. 8, August 2000, pp. 745-746.

[9] C. Connolly, "Cumulative Generation of Octree Models from Range Data", in proc. *IEEE Int'l Conf. on Robotics*, Atlanta, GA, 1984, pp. 25-32.

[10] S. Coorg and S. Teller, "Extracting Textured Vertical Facades from Controlled Close-Range Imagery", in proc. *IEEE Comp. Vision & Pattern Recognition*, 1999, pp. 625-632.

[11] R. Cutler and L. Davis, "Robust Real-Time Periodic Motion Detection, Analysis, and Applications", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 22 no. 8, August 2000, pp. 781-796.

[12] N. DeBardeleben, A. Hoover, W. Jones and W. Ligon, "Parallelization Techniques for Spatial-Temporal Occupancy Maps from Multiple Video Streams", in proc. *IPDPS 2000 Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia*, Cancun, Mexico, 2000, pp. 202-207.

[13] A. Elfes, "Using occupancy grids for mobile robot perception and navigation", in *IEEE Computer*, vol. 22 no. 6, June 1989, pp. 46-58.

[14] D. Gibbins, G. N. Newsam and M. J. Brooks, "Detecting Suspicious Background Changes in Video Surveillance of Busy Scenes", if proc. of *Third IEEE Workshop on Applications of Computer Vision*, Sarasota, FL, 1996, pp. 22-26.

[15] I. Haritaoglu, D. Harwood and L. Davis, "W^4: Real-Time Surveillance of People and their Activities", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 22 no. 8, August 2000, pp. 809-830.

[16] T. Hong and M. Schneier, "Describing a robot's workspace using a sequence of views from a moving camera", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 7, 1985, pp. 721-726.

[17] A. Hoover and B. Olsen, "A Real-Time Occupancy Map from Multiple Video Streams", in proc. of *IEEE Conf. on Robotics & Automation*, 1999, pp. 2261-2266.

[18] A. Hoover and B. Olsen, "Sensor Networked Robotics", in video proc. of *IEEE Conf. on Robotics & Automation*, 1999.

[19] A. Hoover and B. Olsen, "Path Planning for Mobile Robots that Sense Using a Video Camera Network", in proc. of *IEEE/ASME Int'l Conf. on Advanced Intelligent Mechatronics*, 1999.

[20] A. Hoover and B. Olsen, "Sensor Network Perception for Mobile Robotics", in proc. of *IEEE Conf. on Robotics & Automation*, 2000, pp. 342-347.

[21] Y. Kim and J. Aggarwal, "Rectangular parallelepiped coding for solid modeling", in *Int'l J. Robotics Automation*, vol. 1, 1986, pp. 77-85.

[22] A. Laurentini, "How far 3D shapes can be understood from 2D silhouettes", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 17 no. 2, Feb 1995, pp. 188-195.

[23] A. Li and G. Crebbin, "Octree Encoding of Objects from Range Images", in *Pattern Recognition*, vol. 27, 1994, pp. 727-739.

[24] W. Martin and J. Aggarwal, "Volumetric description of objects from multiple views", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 5, 1983, pp. 150-158.

[25] H. Noborio, S. Fukuda and S. Arimoto, "Construction of the Octree Approximating Three-Dimensional Objects by Using Multiple Views", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 10 no. 6, Nov. 1998, pp. 769-781.

[26] B. Olsen and A. Hoover, "Calibrating a Camera Network Using a Domino Grid", in *Pattern Recognition*, vol. 34 no. 5, May 2001, pp. 1105-1117.

[27] M. Potemesil, "Generating octree models of 3D objects from their silhouettes in a sequence of images", in *Computer Vision, Graphics, and Image Processing*, vol. 40, 1987, pp. 1-29.

[28] B. S. Y. Rao, H. F. Durrant-Whyte and J. A. Sheen, "A Fully Decentralized Multi-Sensor System for Tracking and Surveillance", in *Int. J. of Robotics Research*, vol. 12 no. 1, Feb. 1993, pp. 20-44.

[29] H. Saito and T. Kanade, "Shape Reconstruction in Projective Grid Space from Large Number of Images", in proc. *IEEE Comp. Vision & Pattern Recognition*, 1999, pp. 49-54.

[30] S. Seitz and C. Dyer, "Photorealistic scene reconstruction by voxel coloring", in proc. *Comp. Vision & Pattern Recognition*, 1997, pp. 1067-1073.

[31] K. Shanmukh and A. Pujari, "Volume intersection with optimal set of directions", in *Pattern Recognition Letters*, vol. 12, 1991, pp. 165-170.

[32] P. Srinivasan, S. Fukusa and S. Azimoto, "Computational geometric methods in volumetric intersection for 3D reconstruction", in *Pattern Recognition*, vol. 23, 1990, pp. 843-857.

[33] C. Stauffer and W. Grimson, "Learning Patterns of Activity Using Real-Time Tracking", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 22 no. 8, August 2000, pp. 747-757.

[34] R. Y. Tsai, "A versatile camera calibration technique for high accuracy 3d vision metrology using off-the-shelf tv cameras and lenses", in *IEEE Trans. on Robotics & Automation*, vol. 3 no. 4, 1987, pp. 323-344.

[35] R. Willson (maintainer), "Tsai Camera Calibration Software", http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/rgw/www/TsaiCode.html.

[36] J. Zheng, "Acquiring 3-D models from sequences of contours", in *IEEE Trans. on Pattern Analysis & Machine Intelligence*, vol. 16 no. 2, Feb. 1994, pp. 163-178.