Real-Time Pointing Gesture Recognition for an Immersive Environment

Roland Kehl and Luc Van Gool
Swiss Federal Institute of Technology Zürich (ETH)
Computer Vision Laboratory
CH-8092 Zürich - Switzerland
{rkehl|vangool}@vision.ee.ethz.ch

Abstract

We present an algorithm for the real-time detection and interpretation of pointing gestures, performed with one or both arms. The pointing gestures are used as an intuitive tracking interface for a user interacting with an immersive virtual environment. We have defined the pointing direction to correspond to the line of sight connecting the eyes and the pointing fingertip. If a pointing gesture is being performed, the algorithm detects and tracks the position of the user's eyes and fingertip and computes the origin and direction of that gesture with respect to a real-world coordinate system. The algorithm is based on the body silhouettes extracted from multiple views and uses point correspondences to reconstruct in 3D the points of interest. The system doesn't require initial poses, special clothing, or markers.

1. Introduction and Previous Work

Over the past few years, immersive portals have become increasingly popular. Application areas such as manufacturing, architecture, entertainment, chemistry, medicine and others have driven this research. Today, such environments are capable of simulating astonishingly realistic virtual environments. One of the goals behind these portals is to allow users at different locations to collaborate in a shared, simulated environment as if they were actually in the same room.

Examples of room-sized multiprojector portals with up to six planar display walls include the CAVE [1] and its numerous variants. They all feature a large display area at high resolution, often surrounding the user in order to convey a stronger feeling of truly being 'immersed'. Their functionality could be extended by simultaneously interpreting the actions of the users. The work reported in this paper is part of such endeavor. We present a method to detect and analyze pointing gestures in real-time. This offers a very intuitive basis for a user's interaction with the virtual world.

We propose an approach that is helped by the availability of multiple video streams. Indeed, real-time pose and gesture estimation from monocular sequences is still beyond reach as these techniques need rather sophisticated tracking, due to the high-dimensional parameter space and its many ambiguities [2, 3, 4]. Therefore, several multi-view approaches for body pose estimation have been published during the past few years. Most of them try to fit an articulated body model to 3D data [5, 6, 7] . But even then fast tracking of such articulated structures is far from trivial. Thus, we have decided to start with a reduced set of gestures and are trying to detect and analyze only pointing gestures for the moment. The emphasis is on the seamless and, hence, very fast, interaction with the user. In contrast to motion capture systems, our setup works without special clothing or markers.

Pointing gestures are an interesting test case. They are quite powerful in the information they can convey, and are simultaneously not too subtle or too prone to occlusions to be extracted in real-time. Kahn and Swain [8] introduced an early method to detect pointing gestures by using the Perseus architecture. Others such as Nickel and Stiefelhagen [9] presented pointing detection for human-machine interactions that uses Hidden Markov Models trained on different sample pointing gestures to detect the occurrence of a pointing gesture. Kolesnik and Kulessa [10] used an overhead camera where they segment the observed silhouette in two parts for the body and one for the arm. All three papers presented monocular approaches, which make it difficult to find out about the precise 3D direction of the pointing gesture. We present a *real-time*, multi-view approach that measures 3D directions of one or both arms in 3D.

2. The Image Processing Pipeline

Our pointing direction is based on the line of sight connecting the eyes and the fingertip of the pointing arm. The white lines in Fig. 1 show the projection of such a line of sight into the camera views. Therefore, we search for the 3D

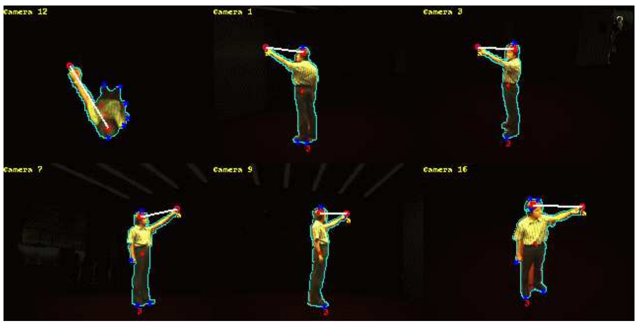

Figure 1. Pointing gesture modeled as the line of sight connecting the eyes and the fingertip of the pointing arm.

positions of the user's eyes and pointing hand in the world coordinate frame. Nickel and Stiefelhagen [9] also tried to use the orientation of the forearm to compute the pointing direction, and so have we. In our experiments the line of sight connecting the eyes and the fingertip led to a more natural experience of the pointing task.

The portal that we are using is constructed as a three-sided stereo back-projection system with the screens arranged in a rectangular layout as shown in Fig. 2. A special feature of this portal is that the screens actually consist of electronically switchable glass panes. Indeed, they can be switched between being translucent and transparent. The cameras are placed all around the portal, including one overhead camera. They can look into the portal during the panes' transparent state. During the translucent state, LCD projectors produce active stereo images on each of the three walls, with shutter glasses making sure the appropriate image is offered to each eye. A detailed hardware description

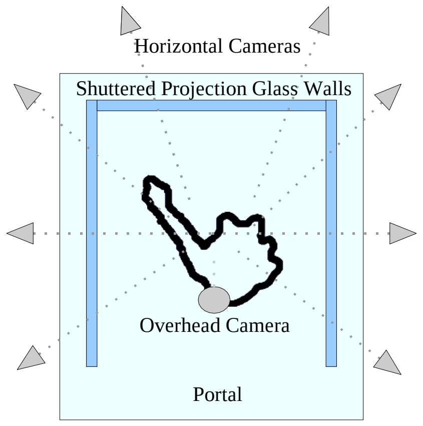

Figure 2. The Portal.

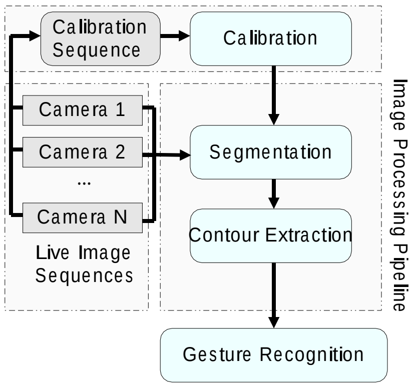

Figure 3. The image processing pipeline.

of the overall system can be found in [11].

Fig. 3 shows the image processing pipeline used for the gesture recognition. We use 4-8 live video streams from statically mounted and fully calibrated cameras. Camera calibration has been kept easy: one enters the darkened working space with a laser pointer. One simply moves the laser around in this space, covering its volume as completely as possible, while the cameras automatically track the bright spot. Then, based on the availability of point correspondences in the different views for the spot in each of its subsequent locations, the internal and external camera calibration is achieved through a self-calibration procedure. The cameras are synchronized by a common trigger signal that ensures the time-consistency of the video streams.

Whereas the camera calibration is an off-line procedure, each of the video streams undergo an on-line sequence of foreground-background segmentation to delineate the user and a contour extraction to find her silhouette.

As we are dealing with static cameras, background subtraction is a possible strategy for foreground-background segmentation, i.e. to separate the user from the surroundings of the portal as seen by the cameras during the transparent state. We use an illumination invariant approach [12] for the background subtraction that we have extended from grayscale to color images. The color values within a small neighborhood of each pixel are rearranged into a vector and compared to the vector formed by the learned mean background values. The vectors show the same hue and saturation values if they are collinear. The deviation from collinearity is measured in a statistically optimal way [12]. If it is smaller than a certain learned threshold, the pixel is considered as foreground. The benefit of the color based approach is that different colors in the foreground resp. background cause larger deviations from collinearity and therefore increase the robustness of the segmentation.

Next, the silhouettes of the segmented foreground regions are extracted. The contour extraction method is deduced from the Marching Squares Algorithm which is the 2D equivalent of the Marching Cubes Algorithm by Lorensen *et al.* [13]. The Marching Squares Algorithm first determines all corners of a cell which lie inside the object by a fixed threshold. Secondly, between all intersection points on the cell edges, straight lines are positioned. The $2^4 = 16$ different cell cases can be reduced to 6 cases using symmetry.

3. The Pointing Gesture Analysis

To detect our pointing gesture as defined in Section 2 we need to detect and track the eyes of the user and the pointing fingertip. If we assume that the user is standing upright in the portal while pointing to some location on the screens, we can expect the eyes to lie at some distance below the top of the head, along the body axis. Since both the top of the head and the pointing hand are quite salient, extremal points on the silhouettes in most views, we should be able to reconstruct their 3D positions in the world frame. Of course, this calls for solving the correspondence problem between the different views. The basic algorithm can then be outlined as follows:

1. Search for extremal points on the silhouettes, such as the head or the hands.
2. Solve correspondences between the different camera views for the previously detected points and reconstruct them in 3D with respect to the world coordinate frame.
3. Search for the head position among the 3D points.
4. Search for one or two hand positions.

3.1. Extremal Points Search

Similar to [14], we are looking for extremal points on the silhouettes. The idea is to detect the local maxima of the distance function $d_j(t)$ described by the distances d_{ji} of every point P_i lying on the silhouette S_j to the center of gravity CoG of S_j. The distance function $d_j(t)$ is not expected to be smooth as our silhouettes are usually disturbed by segmentation noise. To improve the quality of the local maxima detection a box filter is used to smooth $d_j(t)$. Fig. 4 illustrates the extremal point search.

3.2. Solving Correspondences

We have to find correspondences between silhouette extremities in the N views. In order to keep as many options open as possible, we follow a kind of divide-and-conquer approach:

1. Solve correspondences for all combinations of 3 views.
2. Combine these into N-view correspondences.

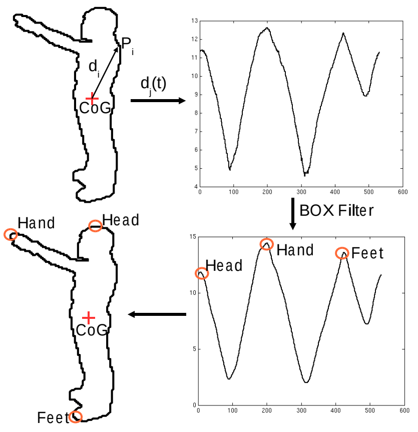

Figure 4. Computation of the extremal points.

3.2.1. 3-View Correspondence Problem Assuming that three points are in correspondence, the intersection of the epipolar lines of two points in the third view must correspond to the third point in that view. As thenumber of points in each view is restricted to the 3-7 extremities found there, we can afford to go for an exhaustive, but highly robust search. Every triplet of points from three different views is checked to whether it fulfils this constraint (which is akin to the trifocal constraint). As a matter of fact, our measured points will almost never precisely fulfill this constraint. Apart from the measurement noise, the extremities of the different silhouettes do not really correspond to identical, physical points.

Two points P_i, P_j are randomly chosen in two views, and the point P_k in the third view is searched that lies closest to the intersection $I_{i,j}$ of their epipolar lines. The triplet of these points is considered for further processing if $I_{i,j}$ and P_k have a pixel distance $d^k_{i,j}$ smaller than a tolerance t. As every point in a triplet will take on the role of the point P_k once, we will perform at most three such tests for each triplet. Hence, we get 3 distances within the tolerance for the triplets that survive the whole procedure. We sum these to obtain a total distance $d^{tot}_{i,j,k}$. We now also discard all correspondences with a $d^{tot}_{i,j,k}$ larger than a second, more restrictive threshold t^{3view}_{max}. The final set of point triplets for all possible choices of three views is then rank-ordered according to ascending values of t^{3view}_{max} to produce a table as shown in Table 3.2.1. Every column corresponds to one of the views, every row to a 3-view correspondence where the views not involved in the triplet have been replaced with an x. The numbers in the cells give the index of the corresponding extremal point in that particular view.

3.2.2. N-View Correspondence Problem The goal of this second step is to merge the 3-view correspon-

$V1$	$V2$	$V3$	$V4$	$V5$	$d_{i,j,k}^{tot}$
0	x	x	1	2	7.60
0	2	x	x	2	8.11
x	2	0	x	2	9.43
1	0	4	x	x	13.51
1	x	4	0	x	14.04
x	x	0	1	2	14.11
...	...	...	...	...	...

Table 1. Sorted 3-view correspondence table.

dences into N-view correspondences. The process starts out by marking all points in all views as "not labeled" and then assigning the label 1 to all points in the first row. To the subsequent rows the following procedure is applied:

1. if all points of the row are still unlabeled we assign the next label to them and continue with the next row;
2. else, if some points have different labels we skip the row and continue with the next;
3. else, if some are still unlabeled but the others have the same label this label is assigned to all of them.

This procedure follows a strategy of better being safe than sorry, in that the same 3D point may end up as multiple, but low-cost correspondences in non-overlapping subsets of the views, rather than the latter being combined via higher-cost triples deeper down in the table. For each set of corresponding points, a 3D point is reconstructed. As a consequence, the process may reconstruct multiple points in 3D for what in reality is a single point on the body. We have used 6 views, in which case this happens in the fewest of cases.

3.3. Head Position Search

The top of the head is quite a salient point on the silhouettes of most views and thus will be included in the set of reconstructed points 3.2.2. In searching it, we assume that the user is in upright position:

1. The top of the head will have a Z coordinate above a certain value (threshold t_{head}).
2. If an overhead camera is available, then in that view the projection of the top of the head must lie inside the silhouette.
3. The top of the head will be close to the main body axis, i.e. close to the axis parallel to the Z axis through the *reconstructed* center of gravity CoG.

The third constraint has to protect the system against cases where the user raises the hand above t_{head}. Fig. 5 illustrates these three constraints. In case multiple points would still

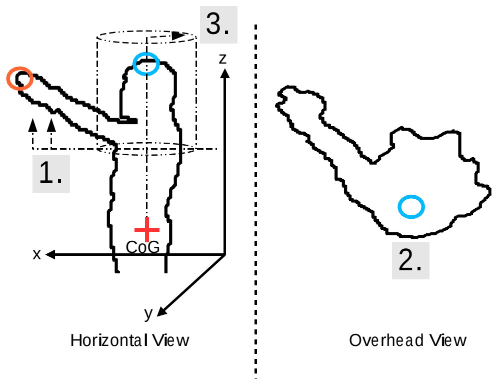

Figure 5. Constraints used for head detection.

survive, the system chooses the point closest to the body axis. Finally, as we are not really interested in the position of the top of the head, but rather in that of the eyes, we estimate the latter as that of the former lowered by a fixed amount of 15 cm. No horizontal correction is foreseen.

3.4. Hand Position Search

Just as was the case for the detection of the top of the head, an overhead camera can be very helpful for finding the fingertip, in case such a camera is available. As we want to keep our algorithms as generic as possible, we will discuss either case. The main difference between these two algorithms is the way how they compute the 3D coordinates of the hand(s), as described in Sections 3.4.1 resp. 3.4.2.

3.4.1. Algorithm 1 - Using an Overhead Camera The idea behind this algorithm is to detect the hand(s) solely on the basis of the overhead silhouette. There, the tip of the pointing arm is a very salient point. The algorithm goes as follows:

1. Compute the center of gravity CoG from the overhead silhouette S_o.
2. Search for extremal points on the overhead silhouette as described in Section 3.1.
3. Among these points p_i^o all those with a distance to CoG larger than some threshold $t_{min}{}_{CoG}^{HandPoint}$ will be treated as potential hand positions p_i^o.
4. Find correspondences for all p_i^o in one horizontal view at a time, as points closest to the epipolar line and within a threshold $d_{max}^{point2view}$.
5. From the N correspondences, reconstruct the potential fingertip in 3D.

Fig. 6 illustrates this procedure which results in a 3D reconstruction for each p_i^o. The final fingertip position is determined via the validation step described in Section 3.4.3.

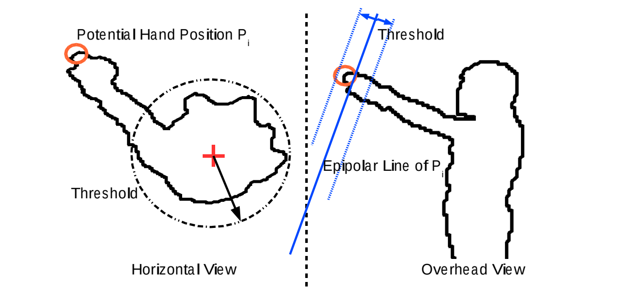

Figure 6. Constraints used for hand detection.

3.4.2. Algorithm 2 - Without an Overhead Camera In case no overhead view is available, we directly pass on the reconstructed points from Section 3.2, excluding the previously identified head position, to the validation step described in Section 3.4.3.

3.4.3. Validation Several causes may lead to multiple candidates, only one or two of which are correct. In the case with an overhead view, the most frequent cause is a foot sticking out of the overhead silhouette due to walking. In the case without an overhead view, several points, including the feet will enter the validation step. In either case, false candidates get pruned by discarding all points that lie outside a cylindrical volume described by two simple constraints around the main body axis:

1. The Z coordinates of the true fingertips must be within a certain range $[z_{min}^{hand}..z_{max}^{hand}]$.
2. The distance of the true fingertips to the main body axis must be within a certain range $[d_{min\,axis}^{\;\;hand}..d_{max\,axis}^{\;\;hand}]$.

If no point matches this validation step the whole algorithm will stop because no pointing gesture has been detected. If more than two points have passed the validation step we use a Kalman prediction based on previous fingertip positions, and keep the points closest to these. This Kalman filter simultaneously has the beneficial effect of smoothing out the jitter from the fingertip trajectory(ies). As a matter of fact, the eye position gets smoothed in the same way.

3.5. The Camera Alignment

In order to guarantee a robust result from our pointing gesture algorithm, experiments have shown that the choice camera positions is quite important. For every direction the user is pointing in, the pointing arm should be seen from at least three cameras from three angles as different as possible. Assuming that we use an overhead camera, at least two horizontally placed cameras should see the pointing arm. It is obvious that if an overhead camera is available it should also be included in the camera set since it has a very different viewing direction than the horizontally placed cameras and therefore it improves the correspondence search significantly. For our setup we used the camera configuration shown in Fig. 2.

4. Experiments

Several experiments have been carried out for our pointing gesture recognition. Fig. 1 shows one frame of a sequence where the user points with one hand. Fig. 7 shows one frame of a sequence where the user performs a pointing gesture with both hands. The white line shows the projection of the reconstructed line(s) of sight into the camera views. Note that in Fig. 7 both hands have been detected correctly even if at least one of them is occluded in most views and thus not detectable in the corresponding silhouettes. The extremal points were computed on one dedicated client machine for each camera and took $0.56\ ms$ on average on a PIV 3.0GHz. Correspondences and eye/fingertip(s) positions have been computed also on a dedicated and similar machine and took $21.8\ ms$. We have used 6 cameras for these experiments having 5 extremal points on average in each view.

As a demo of the pointing gesture recognition, we here show a flashlight application. The user enters a virtual, dark room while holding a virtual flashlight in his hand. The user directionally lightens the virtual room in real-time by pointing at some location on the projection walls as shown in Fig. 8. The whole pipeline runs at 15Hz. Note that in Fig. 8 the system doesn't loose track of the gesture even that the user changes the pointing arm.

The qualitative accuracy of the method is good. The visual feedback of the light cone seems to be accurate to the

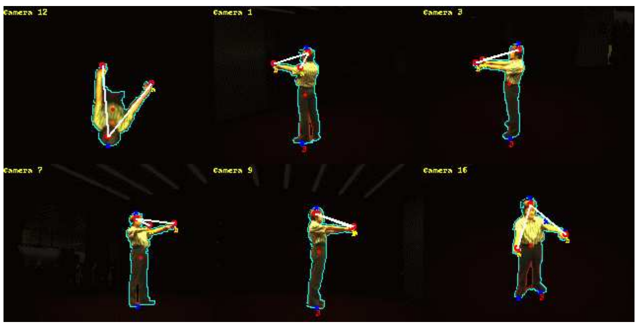

Figure 7. Pointing gesture performed with both hands.

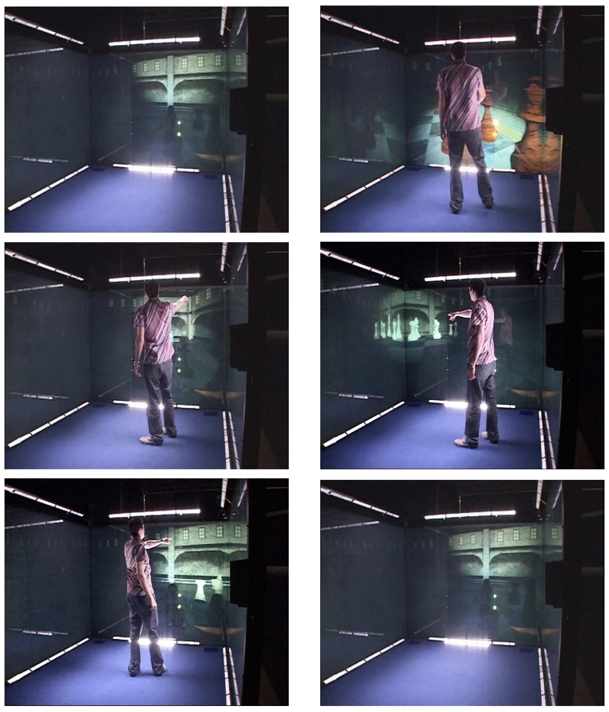

Figure 8. Flashlight Demo.

user. An exact measurement of the accuracy is not applicable since the user is pointing on a virtual object in a virtual scene, thus a qualitative measurement is preferable. Other factors such as the quality of the segmentation or the pointing direction also have a strong influence on the accuracy of the method.

Human-machine interfacing is an obvious application of pointing gesture analysis. In an e-shopping environment customers may pick out goods, data analysts may interact with information, domestic appliances can be commanded, games can be played, etc.

5. Summary and Conclusions

We have presented a fast and robust algorithm to detect a pointing gesture in real-time from multiple views. With the flashlight demo we have proven that our pointing gesture detection is ready to use out of the box in any calibrated multi camera system. We don't need an initialization sequence or a special initial pose to start tracking the gesture. Any user can just enter the virtual world and start performing actions by pointing at any direction without being forced to wear special clothes or markers. An overhead camera can increase the robustness of the algorithm but isn't an essential requirement. Furthermore, the multi-camera approach allows pointing in almost any direction, even towards the cameras but not towards the ground. The experiments have also shown that our algorithm is capable of recognizing pointing gestures performed with both hands. Further research will go into the direction of estimating the full body pose in order to detect a complete set of gestures.

References

[1] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, "Souround-Screen Projection-Based Virtual Reality: The Design and Implementation of the Cave," *In ACM SIGGRAPH*, pp. 135-142, 1993.

[2] J. Deutscher, A. Blake, I. Reid, "Articulated Body Motion Capture by Annealed Particle Filtering," *In CVPR*, pp. 126-133, 2000.

[3] C. Sminchisescu, B. Triggs, "Covariance Scaled Sampling for Monocular 3D Body Tracking," *In CVPR*, pp. 447-454, 2001.

[4] H. Sidenbladh, M. J. Black, D. J. Fleet, "Stochastic Tracking of 3D Human figures using 2D Image Motion," *In ECCV*, pp. 702-718, 2000.

[5] G. K. M. Cheung, S. Baker, T. Kanade, "Shape-From-Silhouette of Articulated Objects and its Use for Human Body Kinematics Estimation and Motion Capture," *In CVPR"*, 2003.

[6] I. Mikic, M. Trivedi, E. Hunter, P. Cosman, "Human Body Model Acquisition and Tracking Using Voxel Data," *Intl. Journal of Computer Vision*, Vol. 53(3), pp. 199-223, 2003.

[7] K. Tollmar, D. Demirdijan, T. Darrell, "Gesture and Play - Exploring Full-Body navigation for Virtual Environments," *In CVPR*, 2003

[8] R. E. Kahn, M. J. Swain, "Understanding People Pointing: The Perseus System," *Intl. Symposium of Computer Vision ISCV*, pp. 11A Motion III, 1995.

[9] K. Nickel, R. Stiefelhagen, "Real-time Recognition of 3D-Pointing Gestures for Human-Machine-Interaction," *In DAGM*, 2003.

[10] M. Kolesnik, T. Kulessa, "Detecting, Tracking and Interpretation of a Pointing Gesture by an Overhead View Camera," *In DAGM*, pp. 429-436, 2001.

[11] M. Gross, S. Würmlin, N. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. Van Gool, S. Lang, K. Strehlke, A. Vande Moere, O. Staadt, "blue-c: A Spatially Immersive Display and 3D Video Portal for Telepresence," *In ACM SIGGRAPH*, pp. 819-827, 2003.

[12] R. Mester, T. Aach, L. Dümbgen, "Illumination-Invariant Change Detection Using a Statistical Collinearity," *In DAGM*, pp. 170-177, 2001.

[13] W. E. Lorensen, H. E. Cline, "Marching Cubes: A high Resulotion 3D Surface Construction Algorithm," *In ACM SIGGRAPH*, Vol. 21, pp. 163-196, 1987.

[14] H. Fujiyoshi, A. J. Lipton, "Real-time Human Motion Analysis by Image Skeletonization," *In Proc. of the IEEE Workshop on Applications of Computer Vison*, 1998.