Distributed Interaction Processing and Visualization of 3D Scenes in Realtime

Jan-Michael Frahm, Jan-Friso Evers-Senne and Reinhard Koch
Institute for Computer Science and Applied Mathematics
Christian-Albrechts University of Kiel, Germany
{jmf, evers, rk}@mip.informatik.uni-kiel.de

Abstract

A distributed realtime system for immersive visualization is presented which uses distributed interaction for control. We will focus on a network architecture for distributed interaction processing for multiple devices and distributed visualization. Furthermore we will discuss in detail user tracking with fixed and pan-tilt-zoom cameras. One of our goals is the use of standard hardware and standard network protocols for the system. For the distributed realtime visualization we use consumer graphics hardware only.

1 Introduction

The design of novel human computer interfaces is an active research topic. The main goal is the design of interfaces which can be used in a more natural way [2, 6, 7]. This can be achieved by utilizing natural human actions like walking, pointing or gestures for interaction control. Our system captures these natural human actions with nonimmersive sensors, namely a multi camera system.

The distributed interaction and visualization system has two major parts:

1. Processing of the interaction data and fusion of information.

2. Updating of all affected data and displaying the new state of the application.

The next section will give a brief overview of the different components of the system and how they work together. Section 3 introduces a protocol to synchronize distributed visualization and distributed interaction processing. The interaction processing will be discussed in section 4. Finally, the distributed visualization is discussed in section 5.

2 System overview

In this section we will give a brief overview of our system architecture and its components. The aim of the system is to enable the user to explore a virtual scene interactively with more immersive techniques than keyboard, mouse, headtracker or any other cable connected devices.

The virtual 3D-scene is displayed simultaneously on a stereo backprojection wall in front of the user and on two displays placed beside the user (side displays). All these displays are synchronized with the presented techniques.

The user can interact with the 3D scene by simply walking throughout the scene. To get the motion of the user we track the user with three cameras. One fixed camera is located at the ceiling and two pan-tilt-zoom cameras are mounted in front of the user at the bottom of the stereo display (see fig. 1). This interaction is very natural because there is no special tracking device fixed to the user's body.

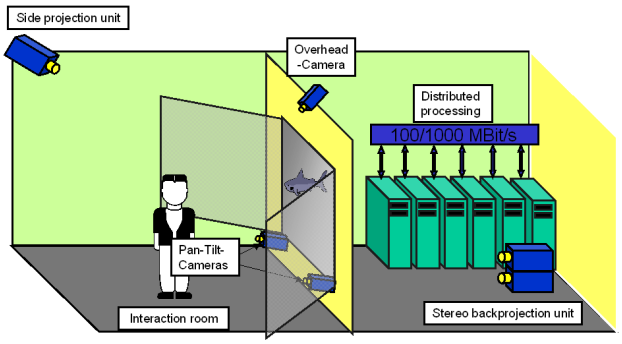

Figure 1. View of interaction area with the display and the cameras.

The system architecture is as follows. The camera at the ceiling (overhead camera) locates the position of the user's feet on the floor. This sensor delivers a 2D position that can be used to initialize and confine the search range of the two pan-tilt cameras facing the user. If the pan-tilt-zoom cameras have found the user's head by searching for skin colored blobs they will track the user's face. From the rotation angles of the pan-tilt cameras we will triangulate the user's 3D head position in space. This position is used to calculate the viewpoint for a virtual view of the scene. The viewpoint position is transmitted to four framewise synchronized visualization nodes for the stereo display and the side displays.

All the modules of the system are computationally ex-

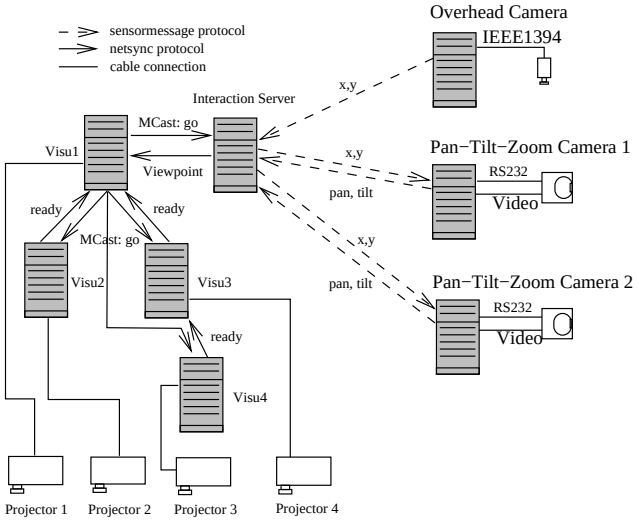

Figure 2. Components of the distributed system and their connections

pensive and demand realtime requirements of at least 10-15 frames per second for tracking and 30 frames per second for visualization. We have therefore distributed the computational load to different Linux client nodes. Currently each camera is attached to a separate node with a frame grabber, and the stereo display is splitted onto four nodes with fast OpenGL consumer graphics cards for the stereo display and the side displays. These machines need a synchronization and intercontrol protocol to meet the requirements of the realtime interaction system and to fuse and distribute the inputs. This is handled by the interaction server. All machines are connected by standard Ethernet network interfaces. Figure 2 sketches the complete system and the connectivity with the interaction server.

3 Distributed synchronized visualization and interaction

This section describes at first the synchronization of the graphics clients and in the next subsection the synchronization of the distributed interaction processing with the graphics clients.

3.1 Synchronization for distributed visualization

Distributed rendering of a dynamic scene depends on the synchronization of graphics clients to assure a coherent visualization. For framewise synchronization of the graphics clients we need a common decision that all clients have rendered their views and simultaneously display their views after decision concurrently. To guarantee this framewise synchronization of the displays we use the protocol as described in [4].

This protocol uses a spanning tree architecture for graphics nodes to distribute a *ready-to-display* message to a specific node (root node) in the network, namely one of the graphics nodes. After receiving *ready-to-display* messages from all other visualization nodes the root node distributes a *display-immediately* message via IP-multicast to all clients (see fig. 3). This protocol is optimal with respect to the number of messages sent and therefore produces few collisions on the network. In [4] it was measured that the generated time shift between different displays is not noticeable to the user.

Furthermore it provides a frame counter to synchronize interaction devices. The protocol is also able to distribute payload information on each synchronization step to the graphics nodes. In the next section we will describe the extension of the protocol used for interaction processing in detail.

3.2 Control of visualization network by interaction

In many virtual environments user interaction has to be processed. The above mentioned protocol [4] for synchronization of graphics clients can be extended to handle such interaction information. We will describe this extension of the protocol in this section.

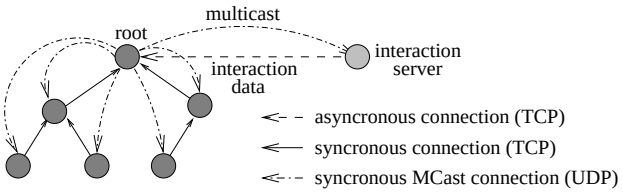

Figure 3. Architecture for visualization and interaction processing

We assume that every type of interaction processing has one process (interaction server) which is able to transmit interaction processing information to our visualization network (see fig. 3). This assumption is no constraint for interaction processing because there is only one interaction process, which is the interaction server itself. For distributed interaction processing one process has to collect the relevant interaction information, fuse the information and transmit it to the visualization network.

Tracking of user interaction does not depend on the frame rate of visualization because it can be continuous interaction, for example pointing to an object at the display, or discrete interactions like clicking, selecting, etc. Especially, interaction processing does not need to be synchronous to the frame rate of visualization. For this reason we assume interaction processing as an asynchronous task. This assumption leads to a visualization network separated from the architecture for interaction processing. In this case it is possible to use any type of interaction processing algorithm. To correlate interaction events with the displayed context it is necessary to have a globally consistent time for visualization and interaction clients. For time synchronization of

the client clocks NTP [8] could be used. For framewise synchronization the frame counter is an appropriate time stamp.

3.3 Interaction synchronization

The interaction server is a specific process not included in the visualization network. It communicates with the root of the visualization network and transmits all interaction data to the root. Furthermore, the interaction server receives the frame counters which are distributed via multicast from the root of the visualization network. These frame counters could be used to correlate the interaction events with the displayed context.

The root of the visualization tree provides interaction information to all graphic clients simultaneously through the IP-Multicast for synchronization. All graphic clients have to process this information in their context. In this way every client can modify its state depending on interaction information. If this processing produces relevant information for other graphic clients it can be submitted as payload with the *ready-to-display*-messages to the root and will be transmitted to all other clients with the next *synchronization*-message. The modified architecture for visualization and interaction is shown in figure 3.

The interaction processing adds a delay to the synchronization. In the worst case we will have a delay of two frames. The aspects of quality-of-service criterion for interaction are discussed in detail by Holloway [5]. If the visualization reaches frame rates of 30 or more frames per second, this interaction delay meets the quality-of-service criterion for most applications.

4 Interaction processing

In this section we will discuss the distributed interaction processing for the multi camera system. Furthermore the fusion of the cameras interaction data to compute the current viewpoint for visualization will be introduced.

4.1 2D Foot tracking with overhead camera

The overhead camera mounted at the ceiling is equipped with a wide-angular lens. It is tilted to view the whole floor of the interaction area in front of the display. Furthermore the radial distortion is compensated during computation. Since the camera views the planar floor we can use four points on the floor to compute a homography H_{floor} that relates ground floor scene coordinates and image coordinates.

We use a background image to substract background information from the current image [3] (see fig. 4(a) and (b)). After this background substraction the difference image only contains noise, the user, and the shadows caused by the user. The noise caused by the camera CCD is canceled out by adaptive thresholding. To determine the threshold we compute the camera noise and mean of this noise in the previous segmented image. Then we are able to compute

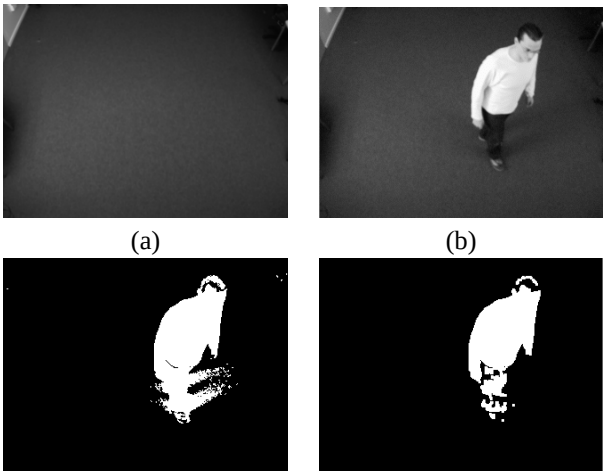

Figure 4. (a) background image of the overhead camera, (b) current image, (c) segmented difference image, (d) segmented image after shadow removal

a confidence intervall for intensity differences caused fom noise. Normally we use an $\alpha = 99.9\%$ confidence intervall.

After noise rejection the difference image contains the user and the shadows (see fig 4 (c)). We assume for shadows the following properties:

- a shadow pixel is darker than the corresponding pixel in the background image,
- the texture of the shadow is correlated with the corresponding texture of the background image.

In the next step we will remove the segmented shadows.

At first we erode single segmented pixels because these segments are normally caused by noise but were not rejected by the previous test. Then we compute a 9x9 normalized cross correlation for each remaining segment which is darker than in the background image.

For shadow pixels this cross correlation is near to one except for camera contrast changes. These contrast changes are caused by the nonlinear system transfer function of the CCD sensor. The range of illumination changes caused from the displays leaves the contrast of the camera approximately constant in our interaction room. Each pixel with a normalized cross correlation greater than a given threshold θ_{NCC} is segmented as background. Normally $\theta_{NCC} = 0.7$ is a good choice. After this segmentation update we use a dilatation to close the segmented structure (see fig 4 (d)).

Now the current image is segmented into foreground which contains only the user and background which is the interaction area. At last we have to locate the user's feet on the floor. We exploit the fact that, due to the tilted viewing frustum of the camera, the feet are always visible in the camera even if the head of the user in the interaction area may not be visible. With respect to the viewing geometry

of the cameras, the user's feet are always located on the bottom most part of the foreground.

We identify the user position with the bottom most foot position in the segmented camera image. The foot position is found by scanning the segmented image from the bottom right to the top left and searching for the first occurrence of a block of the size $u(x, y)$, where $u(x, y)$ models the expected feet size depending on the position (x, y) of the feet on the interaction area.

The reliability of this pose estimation depends on the noise in the difference image. To avoid noise in the estimated position we only update the estimated position if the new position has a distance from the last estimated position in pixel greater than a given threshold ω. Normally $\omega = 2$ pixel is a good choice.

It can be assumed that the feet move on a plane, namely the floor, so the above mentioned homography H_{floor} from the camera coordinates to the floor coordinates is applied to get the position of the user's feet on the floor. These 2D coordinates are submitted to the interaction server for further processing.

This segmentation and searching for the users feet is computed with an unoptimized implementation in 30ms on a computer with an Athlon 1.2GHz processor and 512MB RAM.

4.2 3D Head tracking with pan-tilt-zoom cameras

For the correct estimation of the users viewpoint it is not sufficient to know the 2D position of the user as given by the overhead camera. Instead we need to know the 3D position of the user's head. The missing unknown parameter is the height of the head above the floor. In the most simple approach this height could be assumed to be fixed. The drawback would be, that movements in vertical direction could not be recognized and therefore would not effect the virtual camera position, which is used for visualization.

As an extension we use two pan-tilt-zoom (ptz) cameras positioned on either side of the screen to determine the user's head position. The mapping between the camera coordinates and the interaction floor coordinates can also be described by a homography H_{ptz}.

In order to obtain information for 3D triangulation two tasks have to be solved:

- The position of the user's head within the image has to be determined depending on the current pan angle, tilt angle, and zoom.
- If the user's head is not within the image, initialization angles have to be computed from the users location on the interaction floor.

For the first task, we use a color based algorithm instead of face recognition which utilises a self-adapting bounding box in the HSV color space. This approach is described in detail in [3].

The second task requires a technique to compute the pan angle and the tilt angle given a position on the floor of the interaction area. For this calculation we use the concept of a virtual camera from [3]. With this virtual camera and a fixed mean body height as initial assumption we are able to compute the pan and tilt angle for each ptz-camera from the floor position given from the overhead cameras.

Figure 5. Left: Picture of the user as seen by the pan-tilt-zoom camera. Right: The according binary image after color segmentation.

It is not necessary to find the user's head exactly (in the image center) at first guess. It suffices if the user's head is visible somewhere in the image. Therefore we use a wide angular view at first. Then the exact pan and tilt angles are computed from the deviation of the user's head position to the image center. As an additional advantage the pan and tilt angles can be used for triangulation if the user's head is exactly centered in both pan-tilt-zoom camera images. If one camera is not able to recognize the face, it sets an invalid flag. These data are sent to the interaction server that computes the final 3D head position based on the available estimates from all three cameras.

The described user tracking with a Sony (DVI-30 video conferencing pan-tilt-zoom camera) is computed in less than 40ms on a computer with an 1.2GHz Athlon processor with 512MB RAM. Normally the computation is much faster than 40ms but the processing time depends on the area in the image of skin colored blobs.

4.3 Sensor fusion with interaction server

The interaction server acts as a central communication platform between one or more sensors and the visualization network.

The interaction server is connected to the root node of the visualization network which uses the netsync protocol to control visualization. The interaction server receives multicast messages containing the frame counter of the current render cycle. In addition, the interaction server sends interaction data like viewpoint updates asynchronously to the root node.

The interaction server accepts TCP connections from arbitrary sensor clients, each implementing a specialized interaction device. These clients send their data, marked with the frame counter of the time it was acquired, asynchronously to the interaction server. The frame counter of the sensor data is used to maintain data consistency. The fusion of the data from multiple sensors is done in the interaction server to assure a coherent status.

If new data from any sensor client arrives, the current

state (in this case position of the user's head) is recalculated based on the known data of other sensor clients and the newly received data.

The sensor clients are also allowed to request data from other sensors. These requests are handled by the interaction server to avoid network traffic between individual sensor clients and again to assure coherence. For example the sensor client controlling the pan-tilt-zoom cameras for face tracking uses this mechanism to initially query the 2D user position from the overhead camera.

The concept of sensor fusion with the interaction server is generic. As an example we will describe 3D head position estimation from our multi camera setup. Three cameras are used to observe the interaction area: one overhead camera and two pan-tilt-zoom cameras. Therefore we achieve a robust pose estimation in the case of one user and we are also able to select the controling user in case of multiple users.

For a 3D tracking of the user the overhead camera and one pan-tilt-zoom camera is sufficient. Alternatively we can also use only the two pan-tilt-zoom cameras to estimate the 3D position of the users head. Knowing the position of the two pan-tilt-zoom cameras, the interaction server receives the orientation from each pan-tilt-zoom camera and calculates the intersection of the two rays originating from the cameras. In the general case these two rays do not intersect due to the problems of wide base line stereo position estimation. Therefore the point which has the minimal distance to each ray is used as user's head position.

Now we can use every combination of two cameras to compute additional pose estimates. It is then possible to compute a mean value of the positions as a reliable estimate of the users head position.

If only the overhead camera is available it is possible to track the users position in 2D on the floor. In addition we can extend the feet position received from the overhead camera to a 3D position by assuming a fixed height of the head above the floor.

If both pan-tilt-zoom cameras are tracking and the overhead camera does not work properly, this does not affect the user tracking, as long as there is only one single user in the interaction area.

If there is more than one face visible for the pan-tilt-zoom cameras, it has to be decided which one should be tracked. In this case pan and tilt angles for all faces seen from both pan-tilt-zoom cameras can be used to compute a 3D position for each visible face. The one which is the nearest to the display is chosen to be the *control user*. Using this control user's position, the matching blob in the overhead cameras image is selected and the two redundant positions from overhead camera and one pan-tilt-zoom camera are calculated. Finally, the mean value of all three positions is taken as the user position.

The calculated 3D user position in world coordinates is transformed to a coordinate system appropriate to the visualization subsystem and transmitted to the root node of the visualization subsystem.

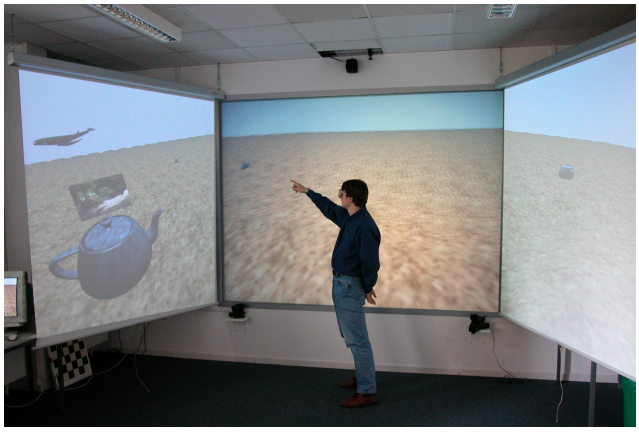

Figure 6. A view of the interaction area with a user wearing polarized glasses. The pan-tilt-zoom cameras can be seen on the left and right side of the display.

5 Synchronized 3D scene rendering

The users head position which has been computed by the interaction server is now sent to the visualization subsystem. Three different screens are used to create an immersive visualization. The center screen is $3m \times 2m$ stereoscopic backprojection system driven by two projectors equipped with polarization filters. Two additional screens are mounted besides the center screen and the two corresponding projectors are mount at the ceiling. These two side views are monoscopic only because the user focuses on the center display and the side views are only seen by one eye.

Each projector is driven by one standard Linux PC equipped with modern consumer 3D accelerator graphics hardware. Figure 6 shows the view of the interaction area with display and cameras.

The user can explore the scene by walking around the interaction area. These movements are captured by the previously described multi camera system.

The four graphics nodes are synchronized as described in section 3. The root node listens for a connection from the interaction server. Once connected, the interaction server sends updates of the data describing the virtual camera (position and orientation of the user) to the root node, which distributes this update with the next synchronization cycle. Each of the nodes receives this data and calculates the parameters of its virtual camera accordingly.

5.1 Multiple Virtual Views

Each projector displays one view of the virtual scene. To ensure consistency for the user, care has to be taken to calculate the parameters for each view. Using standard OpenGL these parameters are position, orientation and viewing frustum of the virtual camera. For an easier handling of movements, two nested coordinate systems are

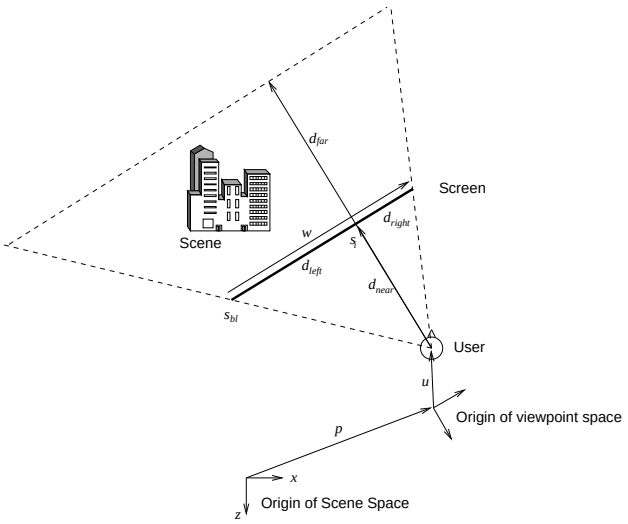

Figure 7. To display a virtual scene on an arbitrarily plaec screen two coordinate systems are nested. This models a user inside a vessel viewing through a window.

used. The outer one is called *scene space* and all virtual objects are located in this space. The origin of the second coordinate system is also located in scene space at position p and called *viewpoint space*, $\{v_1, v_2, v_3, p\}$ is an orthonormal base of the affine viewpoint space. Viewpoint space coordinates can be transformed into scene space with

$$T_{4\times4} = \begin{bmatrix} v_1 & v_2 & v_3 & p \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The user is located inside the viewpoint and sees the virtual scene on the screens similar as if looking through windows into the scene. The relative position u of the user and of all screens are given in viewpoint space. Each screen is specified by the position of the bottom-left corner s_{bl} and two vectors w, h spanning it, the lengths $\|w\|_2$ and $\|h\|_2$ correspond to the width and height. All positions are given in homogeneous coordinates. We also need the normal $N = w \times h$ of the screen pointing inside the interaction area. The screen should be the near clipping plane for the virtual camera so that virtual objects in front of the screen get clipped. This is specified by the distance to the user:

$$d_{\text{near}} = u \cdot N - s_{\text{bl}} \cdot N$$

The distance of the far clipping plane d_{far} can be chosen as needed. To calculate the right, left, bottom and top values according to the OpenGL viewing frustum the point-of-intersection s_i of the normal through u is used:

$$s_i = u - d_{\text{near}} N$$

Now d_{left}, d_{right}, d_{bottom} and d_{top} can be calculated as follows:

$$d_{\text{left}} = \frac{s_i \cdot w - s_{\text{bl}} \cdot w}{\|w\|_2}, \quad d_{\text{right}} = d_{\text{left}} + \|w\|_2$$

$$d_{\text{bottom}} = \frac{s_i \cdot h - s_{\text{bl}} \cdot h}{\|h\|_2}, \quad d_{\text{top}} = d_{\text{bottom}} + \|h\|_2$$

Positioning the virtual camera requires to specify two points in scene space: c_{pos} and c_{target} and one vector for "up": c_{up}. With the transformation from (1) these are:

$$c_{\text{pos}} = T \cdot u, \quad c_{\text{target}} = T \cdot s_i, \quad c_{\text{up}} = T \cdot h \quad (2)$$

Stereo visualization can be achieved with the described setup, too. Assuming the distance between the viewers eyes is approximatly 7 cm, the virtual camera for the left eye is moved 3.5 cm to the left, and the virtual camera for the right eye is moved 3.5 cm to the right. This leads to a method called "parallel axis asymmetric frustum perspective projection" as described in [1].

6 Conclusions

We presented a technology for an interaction room which only requires standard consumer graphics hardware, standard network hardware, and standard network protocols. It allows intutive user interaction with non immersive sensors, namely a multi camera tracking system. Furthermore we discussed in detail the distributed interaction processing and the distributed realtime visualization with standard consumer hardware.

References

[1] Paul Bourke. 3D Stereo Rendering Using OpenGL (and GLUT).

[2] T. Darrel, P. Maes, B. Blumenberg, and A. Pentland. A novel enviroment for situated vision and behavior. In *Proc. CVPR-94 Workshop for Visual Behaviors*.

[3] J.-F. Evers-Senne, J.-M. Frahm, J.F. Woelk, J. Woetzel, and R. Koch. Distributed realtime interaction and visualization system. In *VMV-Workshop*, 2002.

[4] J.-M. Frahm, J.-F. Evers-Senne, and R. Koch. Network protocol for interaction and scalable distributed visualization. In *IEEE Proc. of 3D Data Processing Visualization Transmission*, 2002.

[5] Richard Lee Holloway. Registration errors in augmented reality systems. Technical Report TR95-016, 1, 1995.

[6] R. E. Kahn and M.J Swain. Understanding people pointing: The perseus system. In *Proc. IEEE Int. Symp. on Computer Vision 95*.

[7] M. Kolesnik and T. Kuleßa. Detecting, tracking and interpretation of a pointing gesture by an overhead view camera. In B.Radig, editor, *LNCS: Pattern Recognition*, 2001.

[8] Network Time Synchronization Project. NTP. www.ntp.org.