

# X-Conference: Reinventing a Teleconferencing System

by

**Xin Wang**

M.S., Electrical Engineering  
Peking University, 1998  
B.S., Technical Physics  
Peking University, 1995

Submitted to the Program in Media Arts and Sciences,  
School of Architecture and Planning,  
in partial fulfillment of the requirements for the degree of

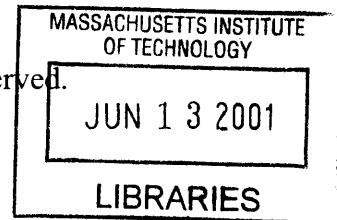
Master of Science in Media Technology


at the

Massachusetts Institute of Technology

June 2001

© 2001 Massachusetts Institute of Technology. All rights reserved.



Author  \_\_\_\_\_

Program in Media Arts and Sciences  
May 11, 2001

**ROTCH**

Certified by  \_\_\_\_\_

Dr. V. Michael Bove, Jr.  
Principal Research Scientist  
MIT Media Laboratory  
Thesis Supervisor

Accepted by  \_\_\_\_\_

Stephen A. Benton  
Chair, Departmental Committee on Graduate Students  
Program in Media Arts and Sciences



# **X-Conference: Reinventing a Teleconferencing System**

by

Xin Wang

Submitted to the Program in Media Arts and Sciences,  
School of Architecture and Planning,  
on May 11, 2001, in partial fulfillment of the  
requirements for the degree of

Master of Science in Media Technology

## **Abstract**

In looking forward to more natural telecollaboration, we can anticipate that the teleconferencing system of the future will enable participants at distant locations to share the same virtual space. The visual object of each participant can be transmitted to the other sites and be rendered from an individual perspective. This thesis presents an effort, X-Conference, to reinvent a teleconferencing system toward the concept of "3-D Virtual Teleconferencing." Several aspects are explored. A multiple-camera calibration approach is implemented and is employed to effectively blend the real view and the virtual view. An individualized 3-D head object is built semi-automatically by mapping the real texture to the globally modified generic model. Head motion parameters are extracted from tracking artificial and/or facial features. Without using the articulation model, facial animation is partially achieved by using texture displacement. UDP/IP multicast and TCP/IP unicast are both utilized to implement the networking scheme.

Thesis Supervisor: Dr. V. Michael Bove, Jr.

Title: Principal Research Scientist, MIT Media Laboratory





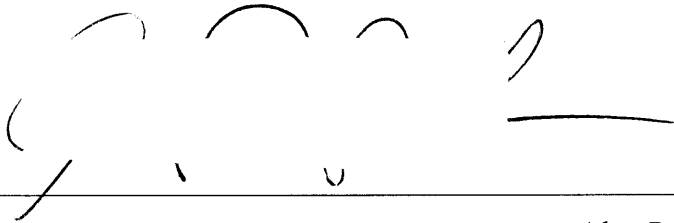
# **X-Conference: Reinventing a Teleconferencing System**

by

Xin Wang

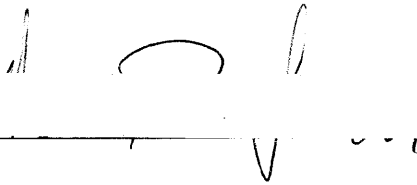
The following people served as readers for this thesis:

Thesis Reader \_\_\_\_\_



Alex Pentland  
Professor of Media Arts and Sciences, Academic Head  
Program in Media Arts and Sciences

Thesis Reader \_\_\_\_\_



Dr. Thomas R. Gardos  
Intel Research Affiliate  
Manager of Streaming Video Technology  
Intel Architecture Labs, Intel Corporation

## **Acknowledgments**

I would like to express my gratitude to all the people who have helped me.

Deep thanks to Dr. V. Michael Bove, Jr., my advisor, who offered me an opportunity to study and work at Media Lab. Thanks so much for helpful suggestions, for sincere encouragement, and for leaving me free to develop.

To Prof. Alex Pentland for being my thesis reader. His achievements and publications have always been invaluable to my research.

To Dr. Thomas R. Gardos for being my thesis reader, for his comments, and for his recommendation of OpenCV library.

To Constantine Kleomenis Christakos, Jacky Mallett, Stefan Agamanolis, Sumit Basu, and Tony Jebara for helpful discussions.

To Juan M. Rivas, Ken Kung, and Yi Li for being my office mates and for being my experiment models.

To Linda Peterson and Pat Solakoff for their patience to answer my questions and for reminding me before every deadline.

Special thanks to the staff at MIT Writing Center for polishing my English.

# Contents

## CHAPTER 1

<b>INTRODUCTION.....</b>	<b>13</b>
1.1 A NEW CONCEPT FOR FUTURE TELECONFERENCING .....	13
1.2 RELATED WORK.....	15
1.3 CHALLENGES AND APPROACHES .....	17
1.4 X-CONFERENCE.....	18
1.5 OUTLINE .....	20

## CHAPTER 2

<b>CAMERA CALIBRATION.....</b>	<b>21</b>
2.1 VIDEO CAPTURING.....	21
2.2 CAMERA CALIBRATION.....	22
2.2.1 <i>Camera parameters</i> .....	22
2.2.2 <i>Correction of camera distortion</i> .....	24
2.2.3 <i>Calibration procedures</i> .....	25
2.3 CONFIGURATION OF OpenGL VIRTUAL CAMERAS.....	26
2.3.1 <i>OpenGL camera location and orientation</i> .....	27
2.3.2 <i>OpenGL camera perspective view</i> .....	27
2.3.3 <i>OpenGL view port</i> .....	29

## CHAPTER 3

<b>BUILDING A 3-D HEAD OBJECT.....</b>	<b>31</b>
3.1 OVERVIEW .....	31
3.2 PREDEFINING FEATURE POINTS .....	32
3.3 POSING THE 3-D HEAD MODEL .....	34
3.4 TEXTURE MAPPING .....	37
3.4.1 <i>Predefining texture boundary lines</i> .....	37
3.4.2 <i>Texture image</i> .....	38
3.4.3 <i>Texture coordinates</i> .....	41
3.4.4 <i>Texture mapping</i> .....	42

## CHAPTER 4

<b>TRACKING AND ANIMATION EXPERIMENT.....</b>	<b>43</b>
4.1 OVERVIEW .....	43
4.2 TRACKING USING ARTIFICIAL FEATURES.....	44
4.3 TRACKING USING FACIAL FEATURES .....	47
4.4 ANIMATION USING TEXTURE DISPLACEMENT.....	50

## CHAPTER 5

<b>NETWORKING SCHEMES AND SYSTEM IMPLEMENTATIONS .....</b>	<b>54</b>
5.1 MANY-TO-MANY ARCHITECTURE .....	54
5.2 RATE ADAPTATION .....	56
5.3 SOFTWARE IMPLEMENTATION .....	57
5.4 USER INTERFACES.....	58

## **CHAPTER 6**

<b>FUTURE DIRECTIONS.....</b>	<b>62</b>
6.1 IMPROVEMENTS FOR X-CONFERENCE.....	62
6.2 INFRASTRUCTURES AND RESOURCES FOR FUTURE RESEARCH .....	63
6.3 FUTURE RESEARCH DIRECTION.....	64
6.4 THE FUTURE WE ENVISION.....	64
<b>REFERENCES.....</b>	<b>67</b>

# List of Figures

Figure 1-1 A conceptual sketch of the office of the future [Raskar, 1998]. .....	14
Figure 1-2 Collaborative interaction between two groups in different buildings [McLeod, 1999]. .....	14
Figure 2-1 Video capturing.....	21
Figure 2-2 The relationships between the coordinate systems involved .....	22
Figure 2-3 Correction of camera distortion .....	25
Figure 2-4 OpenGL perspective viewing frustum .....	28
Figure 2-5 OpenGL view port.....	29
Figure 2-6 Calibration results .....	30
Figure 3-1 Building a 3-D individualized head object.....	31
Figure 3-2 Predefining feature points .....	32
Figure 3-3 The 3-D point is inside a triangle.....	33
Figure 3-4 Initial state of the generic head model .....	35
Figure 3-5 Pose the 3-D head model.....	36
Figure 3-6 Predefining texture boundary lines .....	38
Figure 3-7 Three images ready for merging .....	38
Figure 3-8 Multiresolution image mosaic.....	40
Figure 3-9 Merge areas .....	41
Figure 3-10 Texture image merging.....	41
Figure 3-11 Texture coordinates.....	41
Figure 3-12 Individualized 3-D head object .....	42
Figure 4-1 A special sign for tracking .....	45

Figure 4-2 Initial state of tracking .....	46
Figure 4-3 Tracking by using artificial features .....	48
Figure 4-4 Tracking by using facial features .....	49
Figure 4-5 Predefine the animation area.....	51
Figure 4-6 Texture displacement .....	52
Figure 4-7 Animation using texture displacement.....	52
Figure 5-1 Many-to-many architecture.....	55
Figure 5-2 Software architecture of the server .....	57
Figure 5-3 Software architecture of the client .....	58
Figure 5-4 The user interface of a server.....	59
Figure 5-5 Client interface of audience .....	59
Figure 5-6 Client interface of member A.....	60
Figure 5-7 Client interface of member B.....	60
Figure 5-8 Client interface of member C.....	61
Figure 6-1 Future we envision .....	65





# Chapter 1

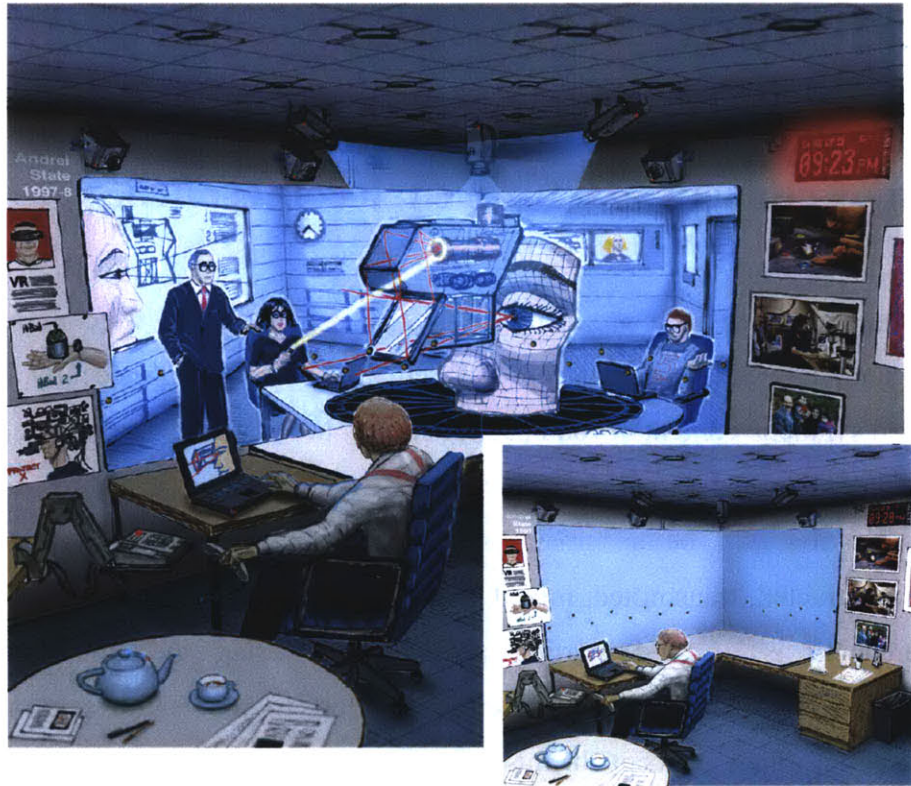
## Introduction

### 1.1 A new concept for future teleconferencing

The past decades have seen great improvements in the field of telecommunications. However, until now the basic concept has remained unchanged. Recently, delivering object-based synthetic content beyond natural content has gradually become one potential direction for multimedia telecommunications. We believe that future multimedia telecommunication will be a "smart" service, enabling visual/audio objects to be acquired, constructed, transmitted, mixed, and represented more efficiently and flexibly. Teleconferencing is a good starting point for exploring the future multimedia telecommunications. This thesis represents an effort to reinvent a teleconferencing system based on a new concept. In this effort, we focus on visual objects. The typical visual object involved in teleconferencing systems is the human upper body; the main part is the head.

The research issues relating to teleconferencing basically can be classified into two layers. One is the transmission layer, in which the main problem is how to reduce the rate and latency. The other is the representation layer, in which the main problem is how to describe the content. In the representation layer, current teleconferencing systems have two obvious problems. First, it is impossible to build gaze awareness, due to uncertainty of camera position and orientation. Second, the current systems leave in place the barrier of distance because of the use of separated display windows. Some studies even doubt the advantage of current videoconferencing over audio-only communication because of these problems [Gemmell, 2000]. To overcome them, a new concept has been introduced over

the past several years. There are several magic words to describe the new concept for future teleconferencing such as “Telepresence,” “Telecollaboration,” “Virtual Space Conference,” “Immersive Conference,” and “Media Immersion Environment.” Figure 1-1 and Figure 1-2 are the concept pictures from some researchers.



**Figure 1-1 A conceptual sketch of the office of the future [Raskar, 1998].**



**Figure 1-2 Collaborative interaction between two groups in different buildings [McLeod, 1999].**

In this thesis, we will use "3-D Virtual Teleconferencing" (3DVTC) to designate the new concept introduced above. The point is to bring the 3-D visual objects of participants from distant locations together into a virtual space so that face-to-face communication can be established naturally. As we have mentioned, conferencing is only a starting point. We can extend 3DVTC into "3-D Virtual Telepresence" (3DVTP) in which 3-D objects will not be limited to human upper bodies, and the use of virtual space will not be limited to meetings. We believe that 3DVTP might have amazing effects on human life in the future. As far as we can see today, it has commercial potential for both the telecommunication and entertainment industries.

## **1.2 Related work**

"ClearBoard" is designed to make mutual eye contact possible by using the metaphor of "talking through a big transparent glass board" [Ishii, 1994]. Instead of the normal whiteboard for a meeting, "ClearBoard" has advantages in maintaining the attention contact while participants are drawing or posting the objects on the board. This is an effective approach by concentrating participants' focus on the 2-D screen rather than rebuilding a 3-D virtual scene. This approach seems not suitable for more-than-two-participant cases.

"Magic Mirror" is a mirror in which you see not only a reflection of yourself, but also the reflections of the other participants in the virtual space as if they were standing next to you, looking at you through a real mirror, even though they are all in separate remote locations [Agamanolis, 1997]. "Magic Mirror" is an effective metaphor to combine the 2D live-video objects, but it does not deal with 3-D objects.

CU-SeeMe VR created a 3-D chat environment [Han, 1996]. CU-SeeMe provides 3-D presence of the virtual environment; however, it does not show real 3-D views of participants. Instead, CU-SeeMe uses texture-mapped flat planes with dynamic projection to indicate the navigation of the users. Besides using live video, CU-SeeMe also implements an audio spatializer.

“Virtual Conference” has been introduced in the past several years. At each sending or receiving site of the teleconferencing system, a 3-D model of each participant is constructed from a wire frame model mapped by color texture and rendered on a 3-D display [Ohya, 1995]. Researchers have not, to date, developed an effective approach to construct a realistic human model from live video. Additionally, many challenges remain on matters such as detecting human behaviors, graphic rendering, and real-time computing requirements. If humanlike avatars were utilized, some big barriers could probably be avoided, but this would hurt the sense of face-to-face communication.

BT Laboratories developed a similar virtual conferencing system for a single user [Mortolock, 1999]. The system uses some computer vision-tracking techniques to do facial feature tracking and body tracking [Machin, 1996]. A 3-D avatar with a personalized talking head is created and used to represent the participant. The synthesis of facial expression is achieved by modeling the action of the muscles within the face. The data rate to control the virtual human is less than 8 kbps. The virtual human in the BT system has limited behaviors; errors in the body- and face-tracking systems need to be eliminated.

The project of "Perceptually-driven Avatars and Interfaces" [Darrell, 1997] explores the computer vision techniques that might be useful for the virtual conferencing domain. Rather than the conventional tracking methods, this project develops an active and multi-resolution-tracking method so that the multiple scales of motion, required to control the avatar, can be potentially tracked in real time. Several applications of the perceptual interface are described in [Darrell, 1997].

TELEPORT is an experimental teleconferencing system with the goal of enabling a small group of people, although geographically separated, to meet as if face-to-face [Gibbs, 1999]. In addition to common cameras, TELEPORT uses a viewer tracker to track the viewpoint. Additionally, TELEPORT uses a wall-size screen to enhance the sense of immersion. TELEPORT is an immersive VR system like CAVE, and therefore it has to

employ high-end equipment such as SGI Onyx. The heavy load of texture mapping and rendering limits its visual quality.

3-D Tele-immersion is being developed at the University of North Carolina at Chapel Hill [Lanier, 2001]. It typically uses seven cameras and five Quad-processor PCs for one person. The system uses a "sea of cameras" and "imperceptible structure light (ISL)" to capture the 3-D objects. The frame rate is low (2 to 3 fps). For a single person, the demand for bandwidth is extremely high, 20-80 Mbps. For three-way conversation, the bandwidth requirement must be multiplied accordingly, even requiring the whole OC3 bandwidth. The UNC system uses multiple projectors, head trackers, and polarizing glasses to represent the 3-D scene. UNC Tele-immersion is very expensive, around \$100,000 for each site, and the quality is not perfect.

### **1.3 Challenges and approaches**

Although the concept of 3DVTC is interesting, many challenges remain in this field. An accurate, well-articulated 3-D human model cannot yet be constructed. Object segmentation from live video needs more improvement. Computer vision techniques cannot track a human body reliably. Large size display quality is poor. Additionally, the system has to depend on high-end equipment. Many difficulties also remain on issues such as network bandwidth and latency. Some researchers have claimed that a 3DVTC-like system will not be commercial for 10 years [Lanier, 2001].

A 3DVTC system consists of three parts: acquisition, transmission, and representation. Each part involves many research challenges. Regarding whole system solutions, we currently have three types.

#### 1) Switching 2-D objects

This solution is relatively easy technologically. At each site, we can set up multiple cameras. Each camera can be considered as the "remote eye" of each participant at the other sites. The system segments the 2-D object and transmits the object to the specific participant point-to-point. In this approach, the viewpoints actually have been fixed on

the positions of cameras, without any flexibility. Normally, because the cameras cannot be set up just at the height of the real user's eye, gaze correction cannot be avoided. Additionally, point-to-point transmission will lead to heavy traffic.

## 2) Synthetic View

This solution typically involves many cameras. For the acquisition part, a good example is the "3D room" project at Carnegie Mellon University [Kanade, 1998; Saito, 1999]. They use 51 calibrated cameras to watch the 3-D scene. The dynamic 3-D shape model can be obtained by applying the approach of multiple baseline stereo frame by frame. With the help of the 3-D model, the virtual view can be interpolated by applying the 2-D image morphing process to two selected camera view images. It can record temporally varying 3-D events with compelling quality. But it is non-real-time because of the large number of cameras and heavy computing load on 18 distributed PCs. Additionally, this solution is not suitable for networking transmission because of the high demand for bandwidth. The UNC "Tele-immersion" project uses a similar approach.

## 3) 3-D Avatar

This solution, like the BT system, typically involves a 3-D virtual human. The information about gestures and movement extracted from real cameras is used to control the virtual human that is rendered in other sites. One important advantage is that the bandwidth requirement can be reduced greatly because only some parameters of movement need to be transmitted. The other advantage is that the number of cameras involved is very small. The challenges are to attain realistic 3-D object building, reliable motion tracking, and natural animation.

# 1.4 X-Conference

During the several months of research and development, we have realized that building a 3DVTC system is complicated. We have not found effective solutions for some problems that we have already encountered, and new problems may arise. We named this thesis project "X-Conference," meaning the framework to explore and solve the problems in

various aspects of reinventing a teleconferencing system toward the 3DVTC concept, rather than the development of a concrete system itself.

For the whole system solution, X-Conference uses "3-D Avatar" as the basic approach because we intend that the system involve only a small number of cameras, relatively low-end equipment, and low bandwidth. However, multiple well-calibrated cameras will be helpful for modeling, tracking, and view synthesizing when needed. Calibrated cameras are one part of the basic infrastructure of X-Conference. We implemented the software module for multiple-camera calibration. Currently, we use three cameras. Our calibration module can easily be extended to more cameras.

As a framework, X-Conference includes a series of functions dealing with 3DVTC-related image processing problems, such as computing 3-D coordinates from corresponding 2-D coordinates, computing epipolar lines, computing the point on the 3-D model given the 2-D projection, and blending images in multiresolution pyramids. Based on these functions, X-Conference has developed software approaches to deal with the experiments for 3DVTC core problems such as 3-D head building, head tracking, and head animation. X-Conference also implements TCP/IP unicasting and UDP/IP multicasting modules to support 3DVTC-involved networking transmission. Moreover, X-Conference includes the graphic user interface for interactively guiding and controlling the process.

X-Conference is also an attempt to investigate and implement our idea of "object-based media," which concentrates on harnessing the power of digital processing and analysis to move from traditional visual and auditory media toward more semantically and physically meaningful "object-based" representation [Bove, 1995] [Agamanolis, 1997]. The impressive advance of computer graphics opens a way toward a fantastic virtual world generated by computers. However, we should also explore how to seamlessly integrate the real world and the virtual world. Although we are a very long distance from this goal, and there exist many hard problems to solve, we believe "object-based media" can combine the strengths of computer vision and computer graphics. As for software

implementation, X-Conference is an example of combining current OpenCV and OpenGL libraries for multimedia applications.

## **1.5 Outline**

The rest of the thesis is organized as follows. Chapter 2 introduces the procedures of camera calibration, and addresses the issues about configuring the OpenGL virtual cameras. Chapter 3 describes how to build the individualized 3-D head object. Chapter 4 discusses 3-D head tracking and a simple approach to animation. Chapter 5 discusses the networking transmission scheme. Finally, Chapter 6 discusses the directions for future research.



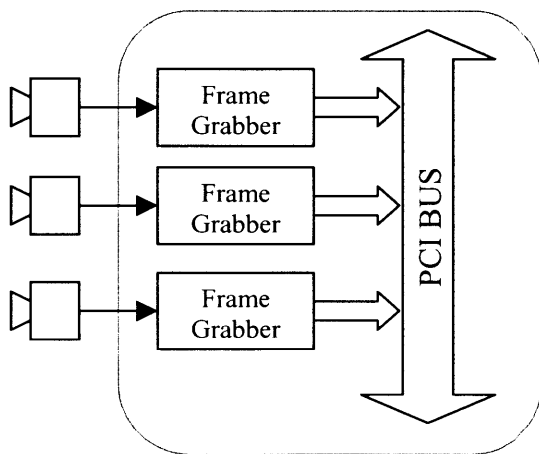
# Chapter 2

## Camera Calibration

Before building 3-D objects, we need to calibrate the cameras in our system. This chapter discusses two problems. One is how to evaluate both intrinsic and extrinsic real camera parameters. The other is how to configure OpenGL virtual cameras to view the scene the same way as the real cameras do.

### 2.1 Video capturing

In our system, we use a single PC to support three frame-grabbers to capture video



concurrently. The PC is the HP Kayak workstation XU800 (PIII 600MHz). The frame grabber is the PXC200 color frame-grabber developed by Imagination Corporation. CMU "3D Room" also used the same frame-grabber [Kanade, 1998]. Figure 2-1 shows the capture system.

Figure 2-1 Video capturing

The X-Conference system provides a testing function to test the video capturing performance. In our experiment, the system is able to capture three frames concurrently at real-time speed (30Hz) without any errors detected; the capture resolution is 320x243, and the color format is 24bits. If we use the standard NTSC format (640x480), the system can capture two frames concurrently without any error at 30Hz.

## 2.2 Camera calibration

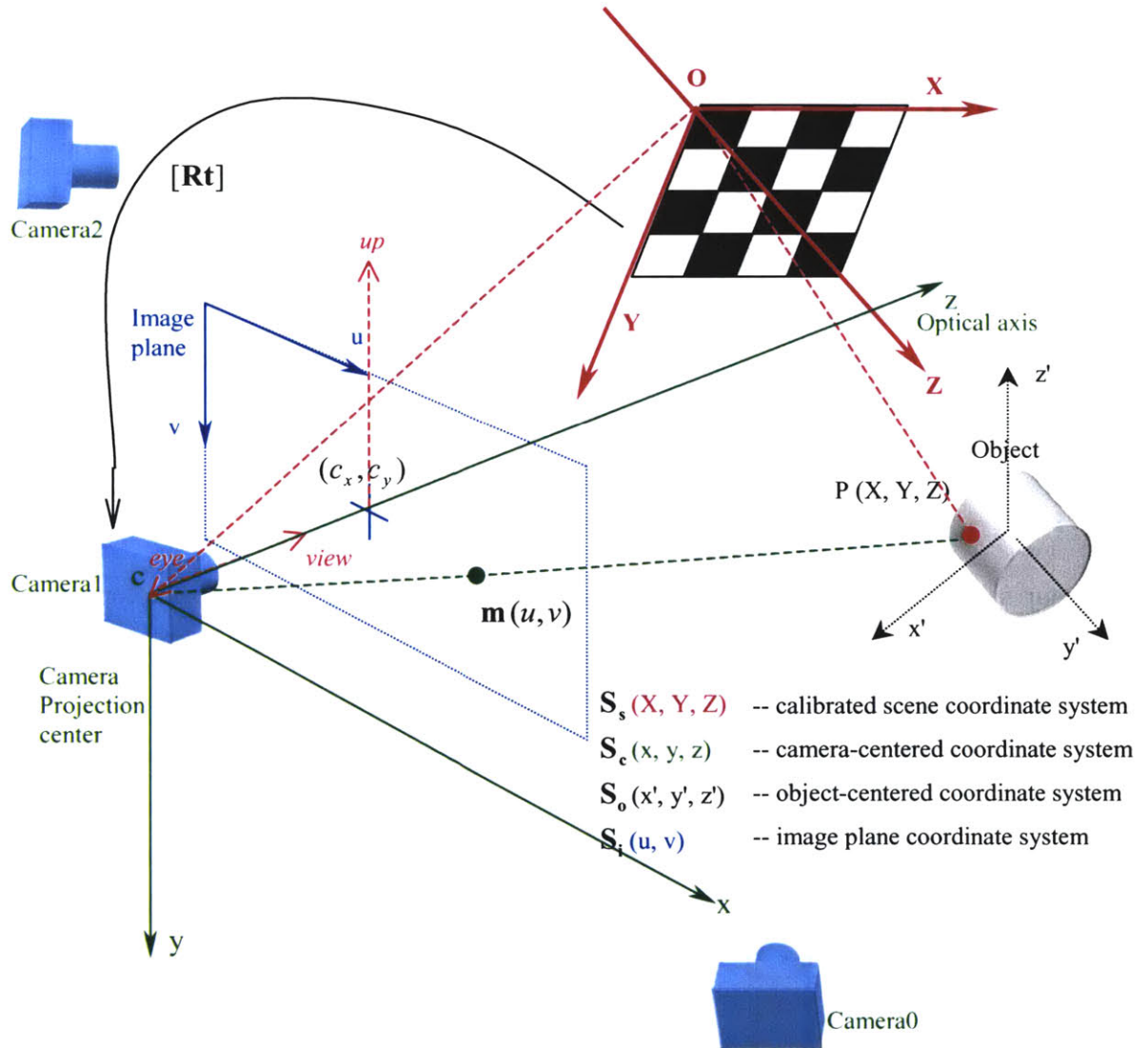


Figure 2-2 The relationships between the coordinate systems involved

### 2.2.1 Camera parameters

Camera parameters describe a particular camera configuration. The intrinsic camera parameters are used to specify the properties of the camera itself; they include the focal lengths  $f(f_x, f_y)$ , the coordinates of principal point  $c(c_x, c_y)$ , and the distortion

coefficient of the lens  $\mathbf{k} (k_1, k_2, k_3, k_4)$ . The extrinsic camera parameters are used to describe the location of the camera in the 3-D world. They include the transformation vector  $\mathbf{t}$  and the rotation matrix  $\mathbf{R}$ . The relationship between a 3-D point  $\mathbf{p}_s$  and its image projection  $\mathbf{m}$  is given by:

$$s \mathbf{m} = \mathbf{A}(\mathbf{R}\mathbf{p}_s + \mathbf{t}), \quad (2.1)$$

where

$s$  -- an arbitrary scale factor;

$\mathbf{m} (u,v)$  -- projection point in the image plane coordinate system ;

$\mathbf{p}_s (X,Y,Z)$  -- 3-D point in the calibrated scene coordinate system;

$\mathbf{A}$  -- camera intrinsic matrix;

$$\mathbf{A} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix};$$

$[\mathbf{Rt}]$  -- camera extrinsic matrix;

$$\mathbf{R} = [\mathbf{r}_1 \quad \mathbf{r}_2 \quad \mathbf{r}_3];$$

$$\mathbf{t} = [t_1, t_2, t_3]^T.$$

Usually we use the augmented vector by adding 1 as the last element:

$$\tilde{\mathbf{m}} = [u, v, 1]^T \text{ and } \tilde{\mathbf{p}}_s = [X, Y, Z, 1]^T.$$

In this manner, (2.1) is then written as:

$$s \tilde{\mathbf{m}} = \mathbf{H} \tilde{\mathbf{p}}_s \quad \text{or}$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \text{ where } \mathbf{H} = \mathbf{A}[\mathbf{Rt}] = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 & h_7 \\ h_8 & h_9 & h_{10} & h_{11} \end{bmatrix}. \quad (2.2)$$

In the camera-centered coordinate system, (2.2) can be written as:

$$s \tilde{\mathbf{m}} = \mathbf{A} \tilde{\mathbf{p}}_c, \quad (2.3)$$

$\mathbf{p}_c (x, y, z)$  -- 3-D point in the camera-centered coordinate system.

For any 3-D point  $\mathbf{p}_s$  ( $X, Y, Z$ ) in the scene, we need to compute the 2-D projection  $\mathbf{m}(u, v)$  from the perspective of the camera with intrinsic matrix  $\mathbf{A}$  and extrinsic matrix  $[\mathbf{Rt}]$ . Matrix  $[\mathbf{Rt}]$  is used to transform the coordinates  $\mathbf{p}_s$  ( $X, Y, Z$ ) in the calibrated scene coordinate system to the corresponding coordinates  $\mathbf{p}_c$  ( $x, y, z$ ) in the camera-centered coordinate system. In the camera-centered coordinate system, the zero point is camera COP (center of projection), and the  $z$  axis is parallel to the camera optical axis. Matrix  $\mathbf{A}$  is used to project the camera-centered coordinates  $\mathbf{p}_c$  ( $x, y, z$ ) onto the image plane coordinates  $\mathbf{m}(u, v)$ .

### 2.2.2 Correction of camera distortion

Usually, the pinhole model needs some corrections for the systematically distorted image coordinates.

Let  $r^2 = x^2 + y^2$ .

In [Heikkilä, 1997], the radial distortion was approximated using the following expression:

$$\begin{bmatrix} \delta x^{(r)} \\ \delta y^{(r)} \end{bmatrix} = \begin{bmatrix} x(k_1 r^2 + k_2 r^4 + \dots) \\ y(k_1 r^2 + k_2 r^4 + \dots) \end{bmatrix}. \quad (2.4)$$

Typically one or two coefficients are enough to compensate for the distortion.

Centers of curvature of lens surfaces are not always strictly collinear. The expression for the tangential distortion is:

$$\begin{bmatrix} \delta x^{(t)} \\ \delta y^{(t)} \end{bmatrix} = \begin{bmatrix} 2k_3 xy + k_4 (r^2 + 2x^2) \\ k_3 (r^2 + 2y^2) + 2k_4 xy \end{bmatrix}. \quad (2.5)$$

In the camera-centered coordinate system,  $(x, y)$  are ideal coordinates, and  $(\tilde{x}, \tilde{y})$  are distorted coordinates. We have:

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} x + \delta x^{(r)} + \delta x^{(t)} \\ y + \delta y^{(r)} + \delta y^{(t)} \end{bmatrix}. \quad (2.6)$$

Let  $(u, v)$  be ideal image projection coordinates, and  $(\tilde{u}, \tilde{v})$  be distorted image projection coordinates. Then we have:

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \end{bmatrix} = \begin{bmatrix} c_x + (f_x/s)\tilde{x} \\ c_y + (f_y/s)\tilde{y} \end{bmatrix}. \quad (2.7)$$

In the camera-centered coordinate system  $s$  is equal to  $z$ ; this equivalence can be derived from (2.3). However, we cannot find out  $z$ , the depth of the scene, only from a 2-D projection image. Therefore, when we evaluate the distortion coefficients or correct the distorted image, we have to assume  $z$  is a constant number, and  $z \neq 0$ . For convenience,  $s$  or  $z$  is assumed to be 1 when we deal with lens distortion problems. And we can use  $(\tilde{x}, \tilde{y})$  instead of  $(x, y)$  to evaluate distortion. Making all these assumptions, we can easily get ideal  $(u, v)$  based on the formulas (2.4)-(2.7).

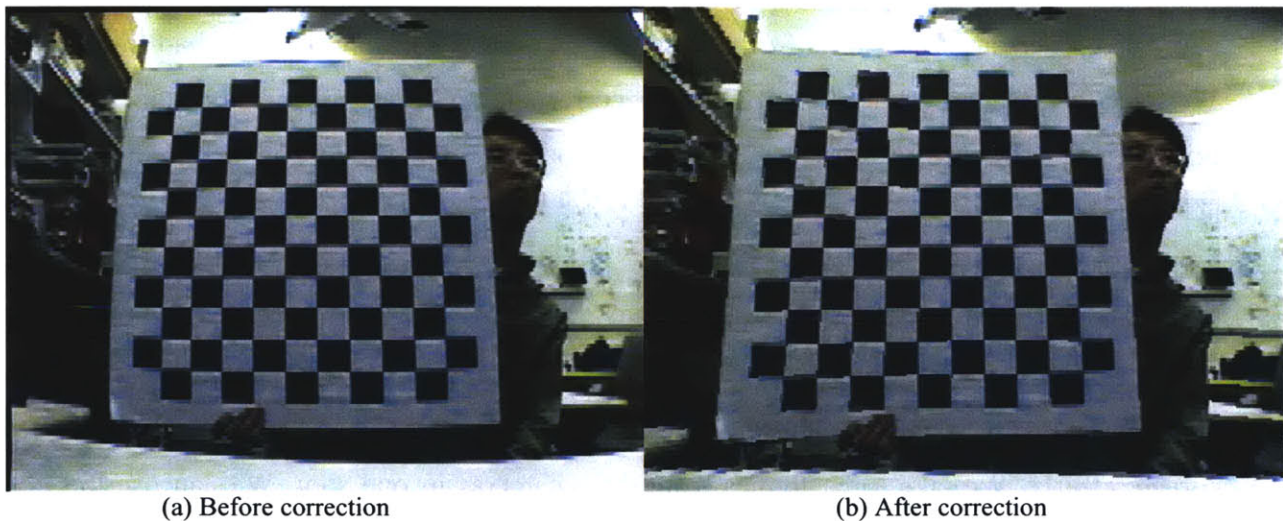


Figure 2-3 Correction of camera distortion

### 2.2.3 Calibration procedures

We use an 11x11 chessboard pattern to calibrate the three cameras. Without loss of generality, the first internal corner point on the board (shown in Figure 2-6) is assumed to be the zero point, the board is assumed to be the plane  $Z=0$ , and the  $X$  and  $Y$  axes are assumed to be parallel respectively to the chessboard row and column. We build a series of software routines with friendly graphic interface so that the calibration can be performed conveniently and almost automatically. The whole calibration session has two steps: intrinsic calibration and extrinsic calibration. We use a popular method developed

by the researchers at Microsoft [Zhang, 1999]. Our task differs in that we need to calibrate multiple cameras in a common coordinate system.

#### 1) Intrinsic calibration

In this step, the intrinsic parameters  $\{f, c, k\}$  of each camera are independently calibrated. We hold the chess board and keep moving it. The software system detects each video frame to automatically find the internal corner points. With an 11x11 pattern, we have 100 internal corners. Because of the motion and the view angle of the camera, not all the internal corners can be guaranteed to be detected by the software system. If all the internal corners are found in a frame, this frame is selected as a good frame for calibration, and all the 2-D coordinates of the corners are stored in the data file. If fewer than 100 corners are found for a frame, the software system will ignore it as a bad frame. We select many good frames (about 100) that correspond to different board locations and orientations. Then we use the closed-form solution to evaluate the intrinsic parameters [Zhang, 1999].

#### 2) Extrinsic calibration

In this step, the extrinsic parameters  $\{R, t\}$  of all the three cameras are evaluated. We pose the board in an appropriate location so that each camera can see all the internal corners clearly. We take three frames from three cameras in one shot. We can use the software routine to find out the corners, unfortunately the result is not reliable because some cameras can only see a side view of the board. In this case, we manually pick the internal corner points with the mouse. After all the internal corners are detected, and since the intrinsic parameters  $\{f, c, k\}$  have been known from the previous step, we can evaluate  $\{R, t\}$  easily.

## 2.3 Configuration of OpenGL virtual cameras

Using the real camera intrinsic matrix  $A$  and extrinsic matrix  $[Rt]$ , we can configure the virtual cameras in the virtual 3-D world so that the virtual cameras are able to perform exactly the same viewing function as the real cameras in the real 3-D world. OpenGL splits the viewing process into three separate parts: 1) to specify the location and

orientation of the camera; 2) to determine the perspective view of the camera; 3) to map the camera's image onto the display screen. OpenGL provides a series of library functions for each respective part. In this section, we will discuss how to configure the OpenGL virtual cameras by using the calibrated real camera parameters  $\mathbf{A}$  and  $[\mathbf{R}\mathbf{t}]$ .

### 2.3.1 OpenGL camera location and orientation

OpenGL uses three vectors to describe the location and orientation of the camera: vector **eye**, vector **view**, and vector **up**.

Vector **eye** is the OpenGL camera location. We have

$$\mathbf{0} = \mathbf{R} \mathbf{eye} + \mathbf{t}.$$

Therefore,

$$\mathbf{eye} = -\mathbf{R}^T \mathbf{t}. \quad (2.8)$$

Vector **view** is the OpenGL camera view direction. In the camera-centered coordinate system, **view** is  $(0,0,1)$ ; therefore in the calibrated scene coordinate system,

$$\mathbf{view} = \mathbf{R}^{-1}[0,0,1]^T. \quad (2.9)$$

Vector **up** is the "up" direction of the OpenGL camera. In the camera-centered coordinate system, **up** is parallel to the y axis; therefore, in the calibrated scene coordinate system,

$$\mathbf{up} = \mathbf{R}^{-1}[0,-1,0]^T. \quad (2.10)$$

### 2.3.2 OpenGL camera perspective view

In OpenGL the volume of space which eventually appears in the projection image is known as view frustum. Perspective projection makes the view frustum look like a pyramid. OpenGL uses *fovy* to denote the angle of the image's vertical field of view and it uses *aspect* to denote the ratio of the width and height of the frustum bottom plane.

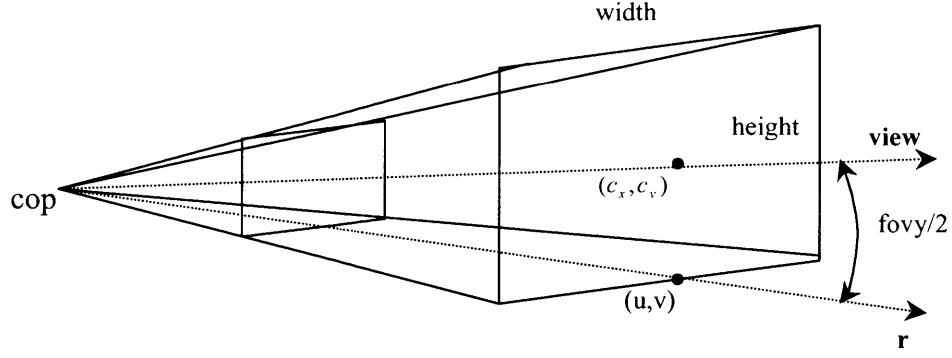


Figure 2-4 OpenGL perspective viewing frustum

For a projection pixel  $(u,v)$  on the image projection plane, the corresponding 3-D point  $(X, Y, Z)$  will be on the line linking the projection center and the pixel  $(u,v)$ .

$$\begin{bmatrix} h_8 u - h_0 & h_9 u - h_1 & h_{10} u - h_2 \\ h_8 v - h_4 & h_9 v - h_5 & h_{10} v - h_6 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} h_3 - h_{11} u \\ h_7 - h_{11} v \end{bmatrix} \quad (2.11)$$

We can consider this line to be the intersection of two planes. Let  $u=c_x$ , and  $v=c_y + height/2$ . The normal vectors of two planes are:

$$\mathbf{n}_1 = \{h_8 c_x - h_0, h_9 c_x - h_1, h_{10} c_x - h_2\};$$

$$\mathbf{n}_2 = \{h_8 (c_y + height/2) - h_4, h_9 (c_y + height/2) - h_5, h_{10} (c_y + height/2) - h_6\}.$$

Let  $\mathbf{r}$  denote the vector parallel to the line linking the projection center and the projection pixel  $(c_x, c_y + height/2)$ ,  $\mathbf{r} = \mathbf{n}_1 \times \mathbf{n}_2$ . Finally, we have:

$$fovy = 2 \cos^{-1} \left( \frac{\mathbf{r} \cdot \mathbf{view}}{\|\mathbf{r}\| \|\mathbf{view}\|} \right). \quad (2.12)$$



### 2.3.3 OpenGL view port

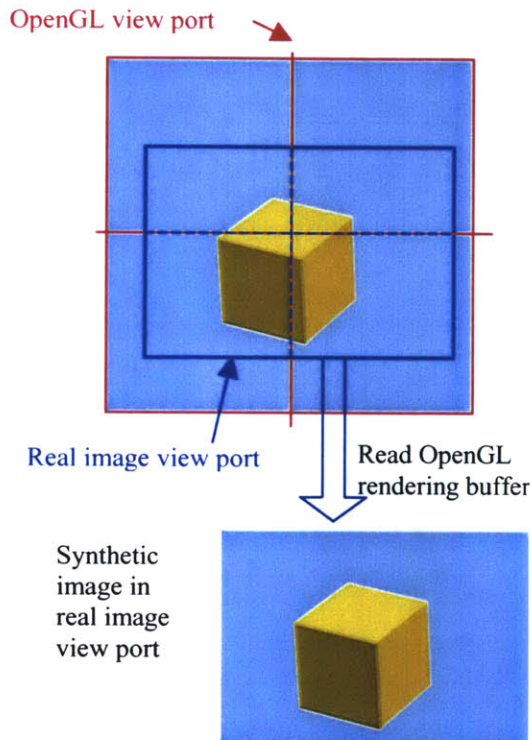
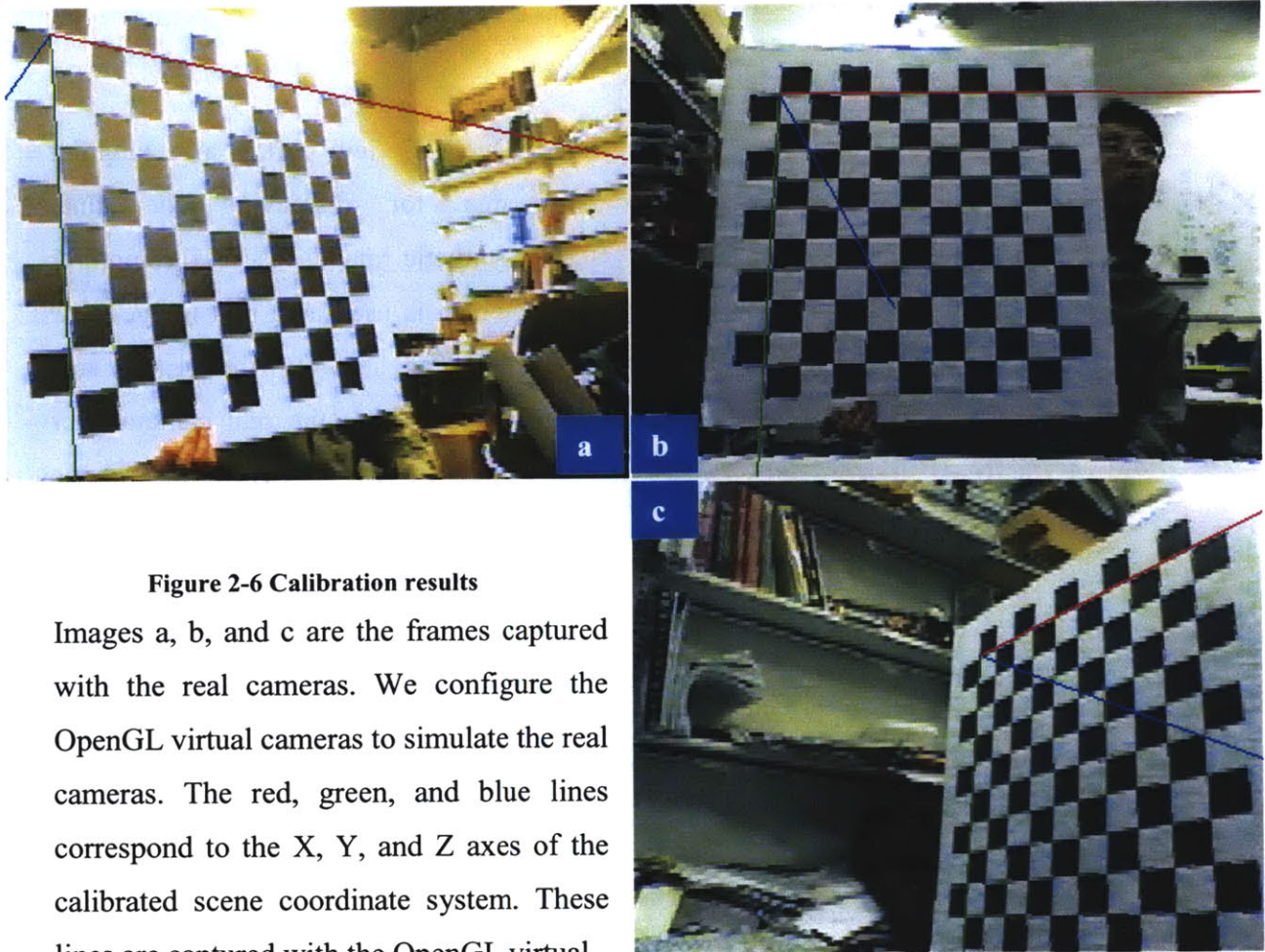


Figure 2-5 OpenGL view port

OpenGL view port is a rectangular area for displaying the final synthetic image, just as the monitor screen is used as a real image view port for showing the real image grabbed from the real camera. Our system needs to blend the synthetic view and the real view together, therefore it is necessary to address how to set the view ports and read out the OpenGL rendering buffer so that we can put the synthetic image data into the real image view port.

After setting the location, orientation, and perspective view of the OpenGL camera, the image plane coordinate system of the synthetic view overlaps the one of the real view. In order to simplify the situation, we set the size of the OpenGL view port larger than the real image size. For example, the real image size is 320x243, and we set both the width and height of the OpenGL view port at 360. Since we know the principal point  $\mathbf{c}_r(c_x, c_y)$  of the real view is identical to the principal point  $\mathbf{c}_s(c_x, c_y)$  of the synthetic view, we can determine the starting point of the synthetic image data in the rendering buffer. Note that in the OpenGL view port, coordinates  $x$  and  $y$  specify the shift from the bottom-left corner; in the real image view port, coordinates  $x$  and  $y$  specify the shift from the top-right corner. In the OpenGL view port,  $\mathbf{c}_s(c_x, c_y)$  is the central point; however, it may not be the central point in the real image view port.



**Figure 2-6 Calibration results**

Images a, b, and c are the frames captured with the real cameras. We configure the OpenGL virtual cameras to simulate the real cameras. The red, green, and blue lines correspond to the X, Y, and Z axes of the calibrated scene coordinate system. These lines are captured with the OpenGL virtual cameras.

# Chapter 3

## Building a 3-D Head Object

Now we have the calibrated cameras. In this chapter, we build a 3-D head object. The 3-D object is described in VRML (Virtual Reality Modeling Language), including a multi-triangle shape model, a 2-D texture image, and texture mapping coordinates. Our basic idea is to map an individualized texture image on a generic 3-D model. We scale the generic model to match the real head shape, but leave the detailed modification for future research.

### 3.1 Overview

Some researchers have developed a semi-automatic approach to building the 3-D head from two orthogonal views [Lee, 2000]. Our goal is to build the 3-D head object by taking advantage of the calibrated cameras. We seek to minimize the need for user guidance. Figure 3-1 shows all the steps for this task. Green boxes indicate some user guidance is needed for that process.

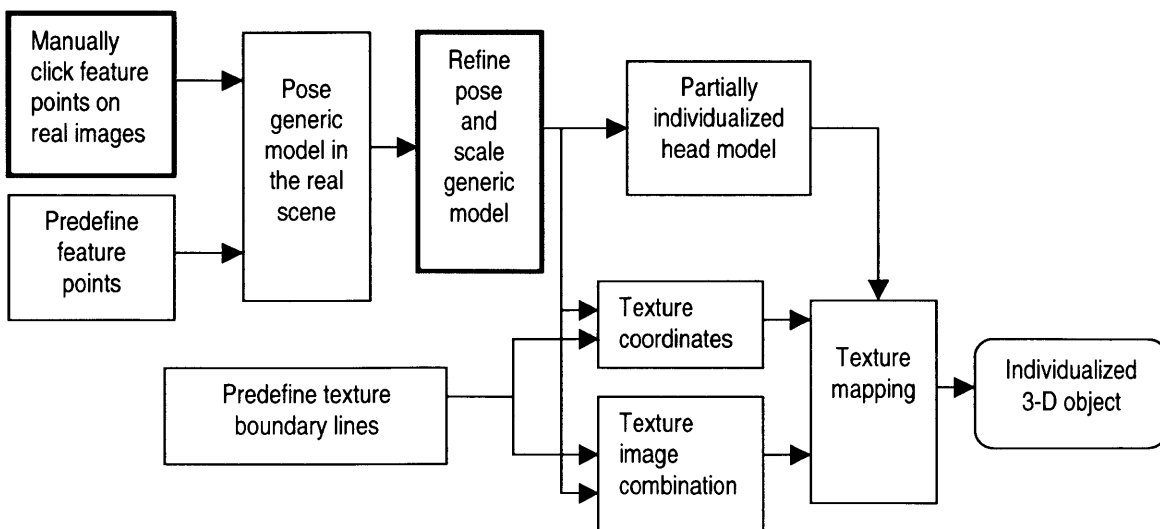
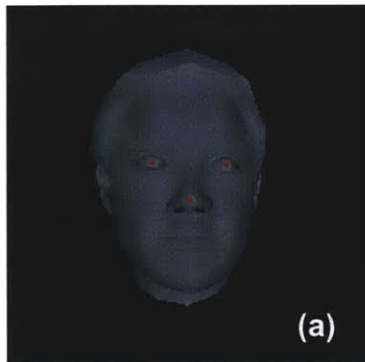


Figure 3-1 Building a 3-D individualized head object

## 3.2 Predefining feature points



In Figure 3-2, image (a) is one view from a predefined virtual camera. Three red points are picked using the mouse to indicate the locus of face feature points. Knowing the camera parameters and the shape of the 3-D model, we can calculate the 3-D coordinates of the face feature points. Images (b), (c), and (d) show the feature points on the 3-D head model.

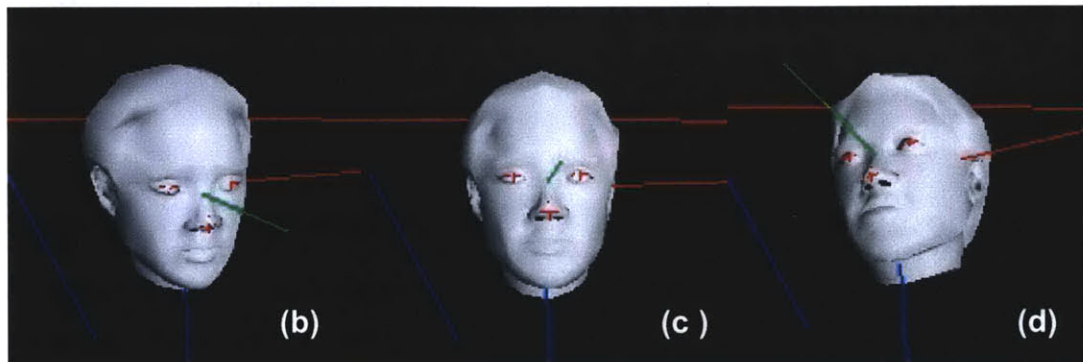


Figure 3-2 Predefining feature points

The following will address how to find out the model point  $\mathbf{p}$  in the calibrated scene coordinate system, given the corresponding pixel  $\mathbf{m}$  ( $u,v$ ) in the image plane coordinates system, the involved camera matrix  $\mathbf{H}$ , and the 3-D model composed of the triangles. This inverse problem is very common when the user wants to click some particular pixels for 3-D model analysis.

There are two steps in solving this inverse problem.

1) Determine which triangle the model point will belong to.

The points  $\mathbf{p}_1(X_1, Y_1, Z_1)$ ,  $\mathbf{p}_2(X_2, Y_2, Z_2)$ , and  $\mathbf{p}_3(X_3, Y_3, Z_3)$  are the vertices of the triangle. The pixels  $\mathbf{m}_0$ ,  $\mathbf{m}_1$ , and  $\mathbf{m}_2$  are the projections of the vertices, and  $\mathbf{m}(u,v)$  is the projection of the unknown 3-D point  $\mathbf{p}$ .

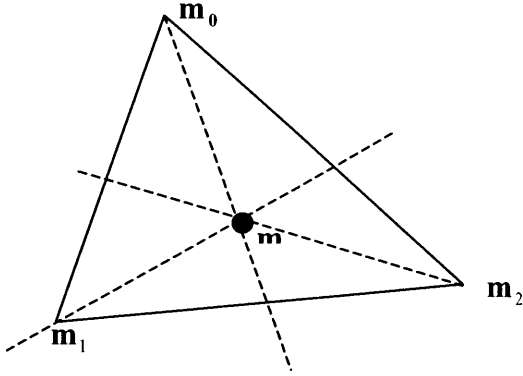


Figure 3-3 The 3-D point is inside a triangle

In Figure 3-3,

Line  $mm_0$  is:

$$f_{mm_0}(\mathbf{m}) = f_{mm_0}(\mathbf{m}_0) = 0.$$

Similarly,

Line  $mm_1$  is:

$$f_{mm_1}(\mathbf{m}) = f_{mm_1}(\mathbf{m}_1) = 0. \quad (3.1)$$

Therefore, if the following conditions are satisfied, the point is inside the triangle  $\Delta m_0 m_1 m_2$ :

$$f_{mm_0}(\mathbf{m}_1)f_{mm_0}(\mathbf{m}_2) < 0 \text{ and}$$

$$f_{mm_1}(\mathbf{m}_0)f_{mm_1}(\mathbf{m}_2) < 0. \quad (3.2)$$

Clearly, if  $\mathbf{m}$  is inside the triangle  $\Delta m_0 m_1 m_2$ , then  $\mathbf{p}$  is inside a triangle  $\Delta p_0 p_1 p_2$ .

2) Compute the coordinates of the model point.

If we have found that the model point  $\mathbf{p}$  is inside a triangle  $\Delta p_0 p_1 p_2$  which has three vertices  $\mathbf{p}_1(X_1, Y_1, Z_1)$ ,  $\mathbf{p}_2(X_2, Y_2, Z_2)$ , and  $\mathbf{p}_3(X_3, Y_3, Z_3)$ , we can calculate  $\mathbf{p}$  based on the three vertices and the camera parameters  $\mathbf{H}$ .

The plane of the triangle  $\Delta p_1 p_2 p_3$  is

$$\begin{vmatrix} X - X_1 & Y - Y_1 & Z - Z_1 \\ X_2 - X_1 & Y_2 - Y_1 & Z_2 - Z_1 \\ X_3 - X_1 & Y_3 - Y_1 & Z_3 - Z_1 \end{vmatrix} = 0. \quad (3.3)$$

In order to simplify the description, we let

$$t_0 = \begin{vmatrix} Y_2 - Y_1 & Z_2 - Z_1 \\ Y_3 - Y_1 & Z_3 - Z_1 \end{vmatrix}, \quad t_1 = -\begin{vmatrix} X_2 - X_1 & Z_2 - Z_1 \\ X_3 - X_1 & Z_3 - Z_1 \end{vmatrix}, \quad t_2 = \begin{vmatrix} X_2 - X_1 & Y_2 - Y_1 \\ X_3 - X_1 & Y_3 - Y_1 \end{vmatrix},$$

$$t_3 = \begin{vmatrix} Y_2 - Y_1 & Z_2 - Z_1 \\ Y_3 - Y_1 & Z_3 - Z_1 \end{vmatrix} X_1 - \begin{vmatrix} X_2 - X_1 & Z_2 - Z_1 \\ X_3 - X_1 & Z_3 - Z_1 \end{vmatrix} Y_1 + \begin{vmatrix} X_2 - X_1 & Y_2 - Y_1 \\ X_3 - X_1 & Y_3 - Y_1 \end{vmatrix} Z_1.$$

Then the plane equation in (3.3) is:



$$t_3 = t_0X + t_1Y + t_2Z. \quad (3.4)$$

Combining (3.4) and (2.10), we have:

$$\begin{bmatrix} h_8u - h_0 & h_9u - h_1 & h_{10}u - h_2 \\ h_8v - h_4 & h_9v - h_5 & h_{10}v - h_6 \\ t_0 & t_1 & t_2 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} h_3 - h_{11}u \\ h_7 - h_{11}v \\ t_3 \end{bmatrix}. \quad (3.5)$$

Therefore,

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} h_8u - h_0 & h_9u - h_1 & h_{10}u - h_2 \\ h_8v - h_4 & h_9v - h_5 & h_{10}v - h_6 \\ t_0 & t_1 & t_2 \end{bmatrix}^{-1} \begin{bmatrix} h_3 - h_{11}u \\ h_7 - h_{11}v \\ t_3 \end{bmatrix}. \quad (3.6)$$

### 3.3 Posing the 3-D head model

There are 4 steps in posing the 3-D generic head model properly so that the real head and the virtual head have the same location and orientation.

- (a) The input real video images and the initial pose state of the generic head model are shown as Figure 3-4. Images (a-0), (a-1), and (a-2) are the real views from the three cameras. Image (a-3) is the virtual view of the 3-D head from the second camera. We generate the 3-D head virtual view from all the perspectives of the three cameras, and post the 3-D head virtual views back to the real images in the semitransparent format so that we can observe, compare, and match the virtual and the real views. In the image (a-3), the pure red, green, and blue lines are the calibrated scene coordinate axes X, Y, and Z. Besides the scene coordinate frames, we also define the object-centered coordinate axes x', y', and z'. At the initial state, the axes x', y', and z' overlap the axes X, Y, and Z without any relative translation or rotation; and the 3-D head is on the point (0, 0, 0) with the scale vector (1.0, 1.0, 1.0).
- (b) Our software provides users with an interactive environment so that the users are able to simply pick the feature points (eyes and nose) with the mouse. In Figure 3-5, images (b-0), (b-1) and (b-2) show the video frames with the selected feature pixels (pure red, green, and blue ones) in different views. Among the three views, we can

choose any two of them to "click" the 2-D feature pixels, then calculate the 3-D feature points from the corresponding 2-D pixels and the parameters of the calibrated cameras.

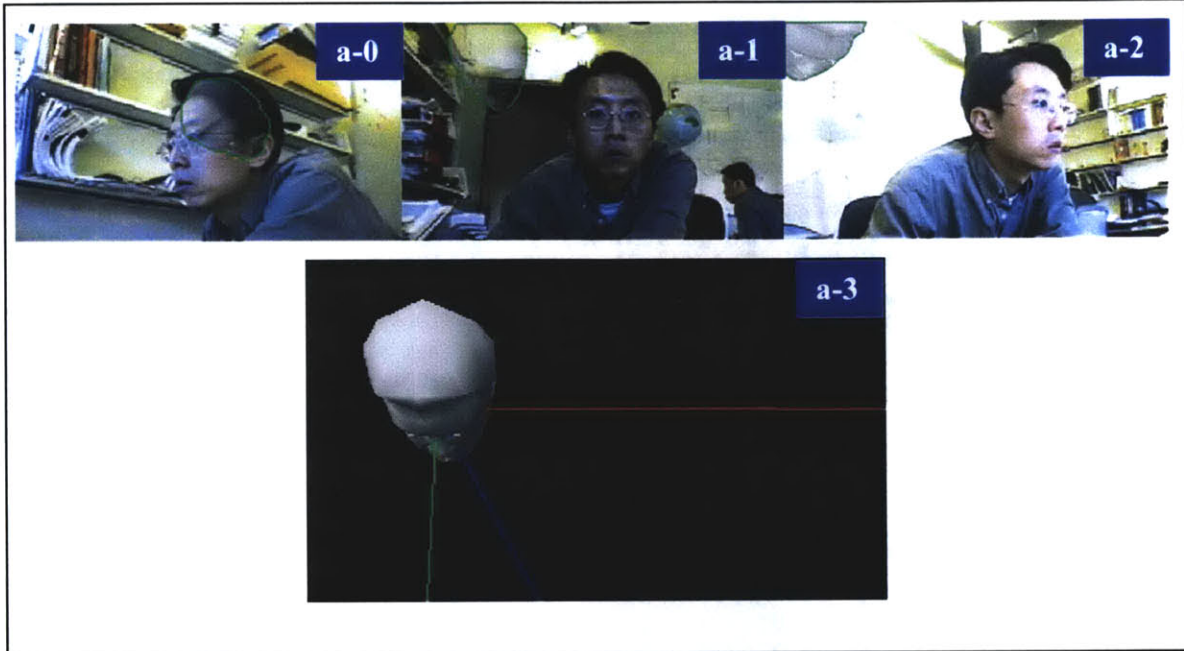


Figure 3-4 Initial state of the generic head model

(c) In step (b), three feature points have been obtained. All these feature points correspond to the feature points of the 3-D model, but some motion has happened including rotation and translation. In this step, we will address how to compute the motion  $\mathbf{R}$  and  $\mathbf{t}$  from two corresponding 3-D data sets ("3-D -- 3-D estimation").

Without loss of generality, we assume the two sets are  $\mathbf{x}_1, \dots, \mathbf{x}_N, \mathbf{y}_1, \dots, \mathbf{y}_N$ . And each  $\mathbf{y}_k$  ( $1 < k < N$ ) is obtained as a rotation and a translation of  $\mathbf{x}_k$  ( $1 < k < N$ ).

To evaluate  $\mathbf{R}$  and  $\mathbf{t}$ , we will minimize

$$\sum_{n=1}^N \|\mathbf{y}_n - (\mathbf{R}\mathbf{x}_n + \mathbf{t})\|^2 \quad (3.7)$$

subject to the constraint  $\mathbf{R}^T = \mathbf{R}^{-1}$ .

We can obtain [Haralick, 1989]

$$\mathbf{t} = \bar{\mathbf{y}} - \mathbf{R}\bar{\mathbf{x}} \quad \text{where} \quad \bar{\mathbf{x}} = \frac{\sum_{n=1}^N \mathbf{x}_n}{N}, \quad \bar{\mathbf{y}} = \frac{\sum_{n=1}^N \mathbf{y}_n}{N}, \quad (3.8)$$



Figure 3-5 Pose the 3-D head model

$$\mathbf{R} = \mathbf{V} \begin{pmatrix} 1 & & \\ & 1 & \\ & & \det(\mathbf{V}\mathbf{U}^T) \end{pmatrix} \mathbf{U}^T, \quad (3.9)$$



Let  $\mathbf{K}$  denote the correlation matrix:

$$\mathbf{K} = \sum_{n=1}^N \mathbf{y}_n \mathbf{x}_n^T .$$

From SVD (singular value decomposition)  $\mathbf{K} = \mathbf{V} \mathbf{\Lambda} \mathbf{U}^T$ , where  $\mathbf{\Lambda}$  is diagonal.

With the  $\mathbf{R}$  and  $\mathbf{t}$  thus determined, the 3-D model can be moved to the estimated real head location. Images (c-0), (c-1), and (c-2) show the blending of the real views and the virtual views of the 3-D model with the estimated pose. We can see that the 3-D head model and the real head nearly overlap.

- (d) We should be aware that the estimation in step (c) is not perfect because of three factors. The number of feature points is very small (only three); there is noise from errors in step (b); and the 3-D head model is only a generic model, not an individualized one. In order to make the model fit the real head better, we will employ some user guidance to refine the pose of the model and also to modify the scale of the model. Our software provides users with a graphical interface so that the user can interactively translate, rotate, and scale the 3-D model with the mouse. Normally only some tiny modifications are needed. Images (d-0), (d-1), and (d-2) show the new blending of the real views and the virtual views of the 3-D model after manual refining. Image (d-3) shows the virtual view of the 3-D head from the perspective of the second camera.

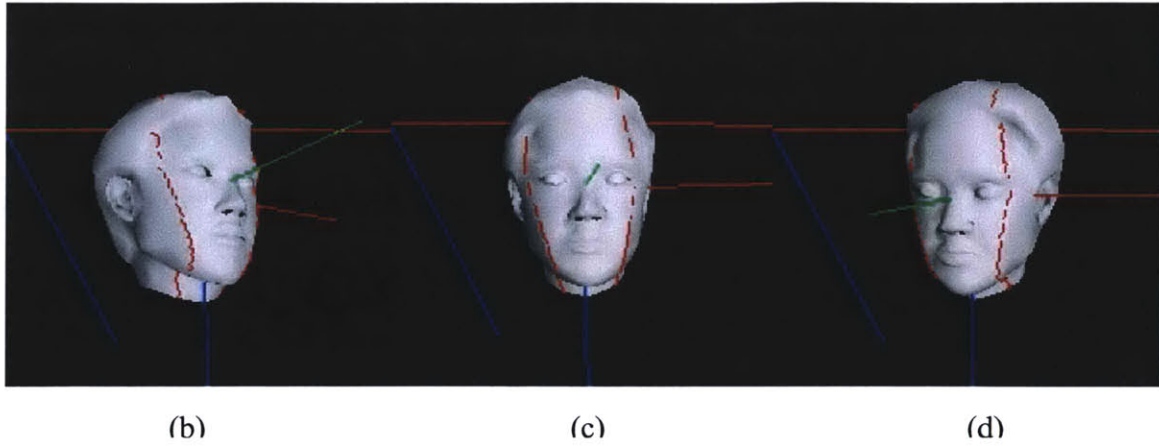
## 3.4 Texture mapping

### 3.4.1 Predefining texture boundary lines



(a)

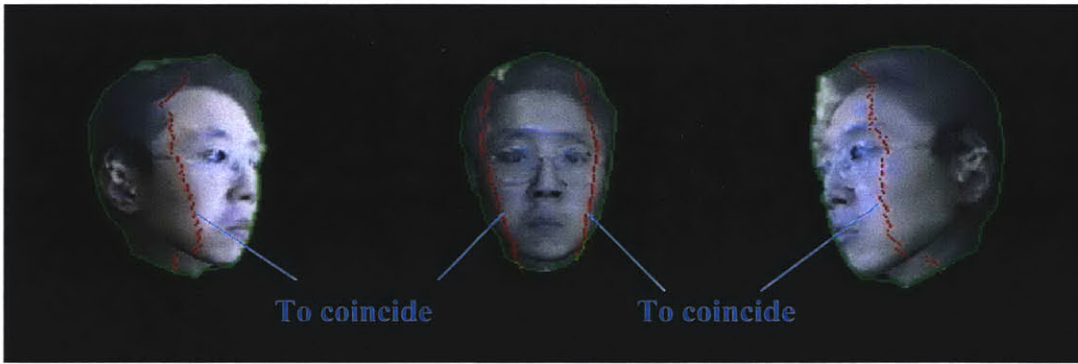
In Figure 3-6, image (a) is one view from a predefined virtual camera. The red lines illustrate the location of the predefined texture boundary lines. In the same way as the section 3.2, we can find out the 3-D coordinates of the points on the texture boundary lines.



**Figure 3-6 Predefining texture boundary lines**

Images (b), (c), and (d) show the texture boundary lines on the 3-D head model from different perspectives.

### ***3.4.2 Texture image***



**Figure 3-7 Three images ready for merging**

Figure 3-7 shows the views of the texture boundary lines from three real cameras. When combining the side views and the front view, we keep the front view as it is, and deform the side views so that the 2-D projection of the 3-D texture boundary lines on the side views and the front view can be connected seamlessly. We create a lookup table to store the deformation; the deformation is also useful for both texture image and texture coordinates generation.

We combine three input images, which correspond respectively to the left side, the front, and the right side view, to generate the whole head texture image. The boundary effects between different view images are not easily avoided. Alpha blending is a simple approach to combining textures from different camera images [Tsai, 1997]. For the better quality, we use pyramid decomposition of images to merges the different images at multiresolution levels [Burt, 1983; Lee, 2000].

Assume we need an N-level pyramid. First we need to obtain Gaussian images  $G_k$  by iteratively using REDUCE operation.

$$G_k = REDUCE[G_{k-1}] \quad (0 < k < N) \quad (3.10)$$

By REDUCE we mean

$$G_k(i, j) = \sum_{m,n=1}^5 g(m,n)G_{k-1}(2i+m, 2j+n), \quad (3.11)$$

where  $g(m, n)$  is a Gaussian filter kernel.

Then the EXPAND operation is used to obtain Laplacian images  $L_k$ .

$$G'_k = EXPAND[G_{k-1}] \quad (0 < k < N) \quad (3.12)$$

By EXPAND we mean

$$G'_k(i, j) = 4 \sum_{m,n=-2}^2 G_{k-1}\left(\frac{2i+m}{2}, \frac{2j+n}{2}\right), \quad (3.13)$$

$$L_k = \begin{cases} G_k - EXPAND[G_{k-1}] & 0 < k < N-1 \\ G_k & k = N-1 \end{cases} \quad (3.14)$$

We can see the property of the Laplacian pyramid:

$$G_0 = \sum_{k=0}^{N-1} L_k \quad (3.15)$$

Assume images A, B, and C are needed to merge into image S. The Laplacian pyramids of image S will first be obtained, then the sum of these Laplacian images will be the image S.

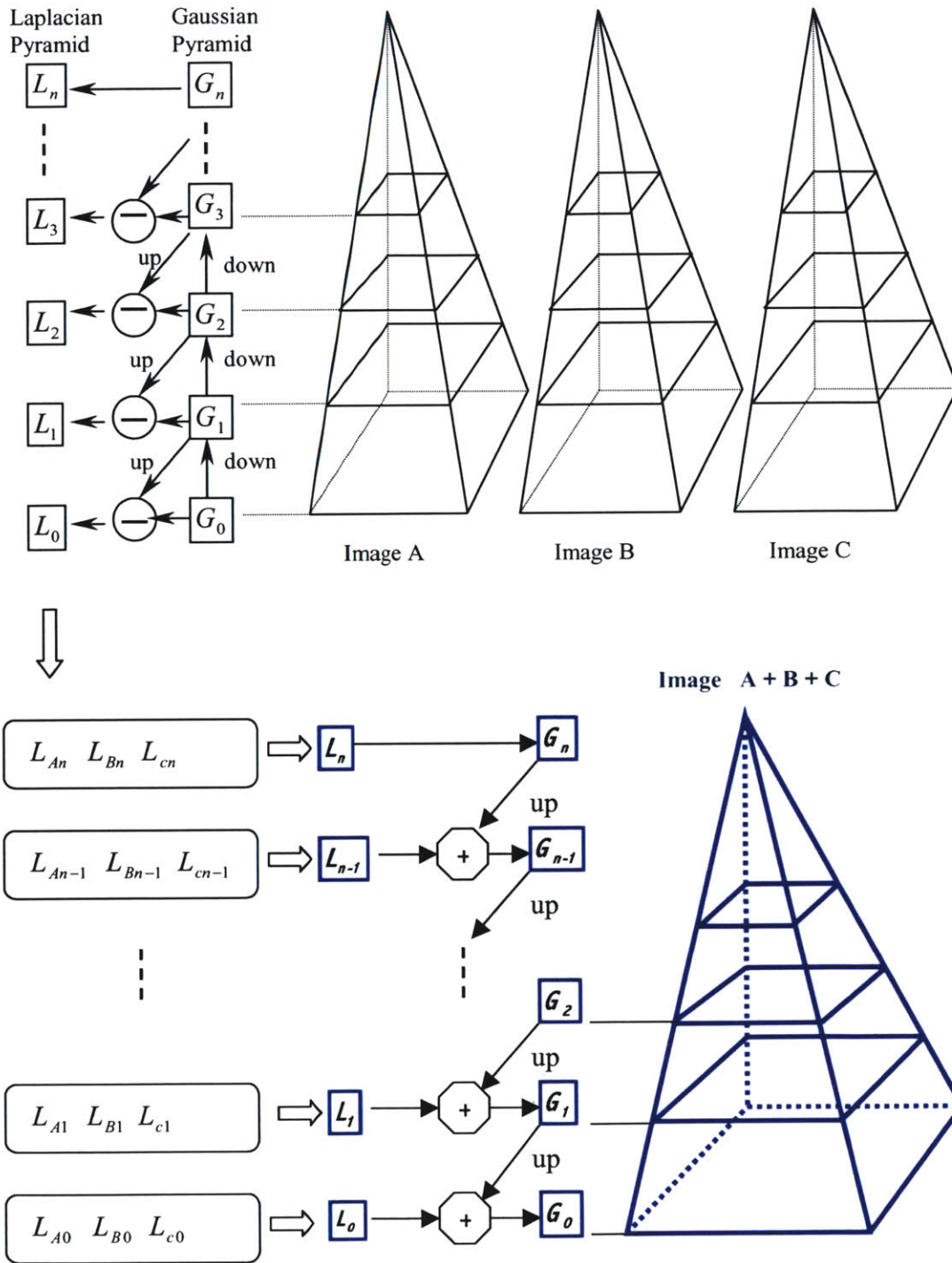


Figure 3-8 Multiresolution image mosaic

We define the level k Laplacian image of S as the following:

$$L_{S,k}(i, j) = R_k(A, i, j)L_{A,k}(i, j) + R_k(B, i, j)L_{B,k}(i, j) + R_k(C, i, j)L_{C,k}(i, j), \quad (3.16)$$

where the function R means:

$$R(P, i, j) = \begin{cases} 1 & (i, j) \in P \\ 0 & (i, j) \notin P \end{cases}, \quad (3.17)$$

where P could be A, B, or C.

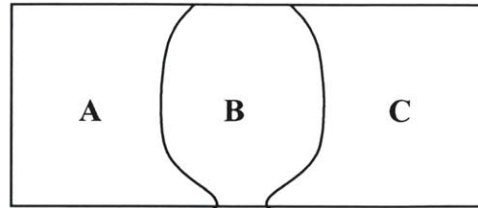


Figure 3-9 Merge areas



(a) Before multiresolution processing

(b) After multiresolution processing

Figure 3-10 Texture image merging

### 3.4.3 Texture coordinates

Basically, the 2-D texture coordinates can be obtained by projecting the 3-D triangle vertices onto the 2-D image plane. For the vertices projected onto the front view, between two texture boundary lines, the 2-D projection coordinates are identical to the texture

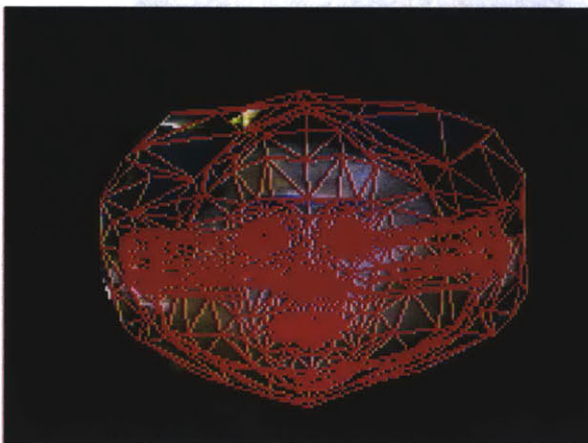


Figure 3-11 Texture coordinates

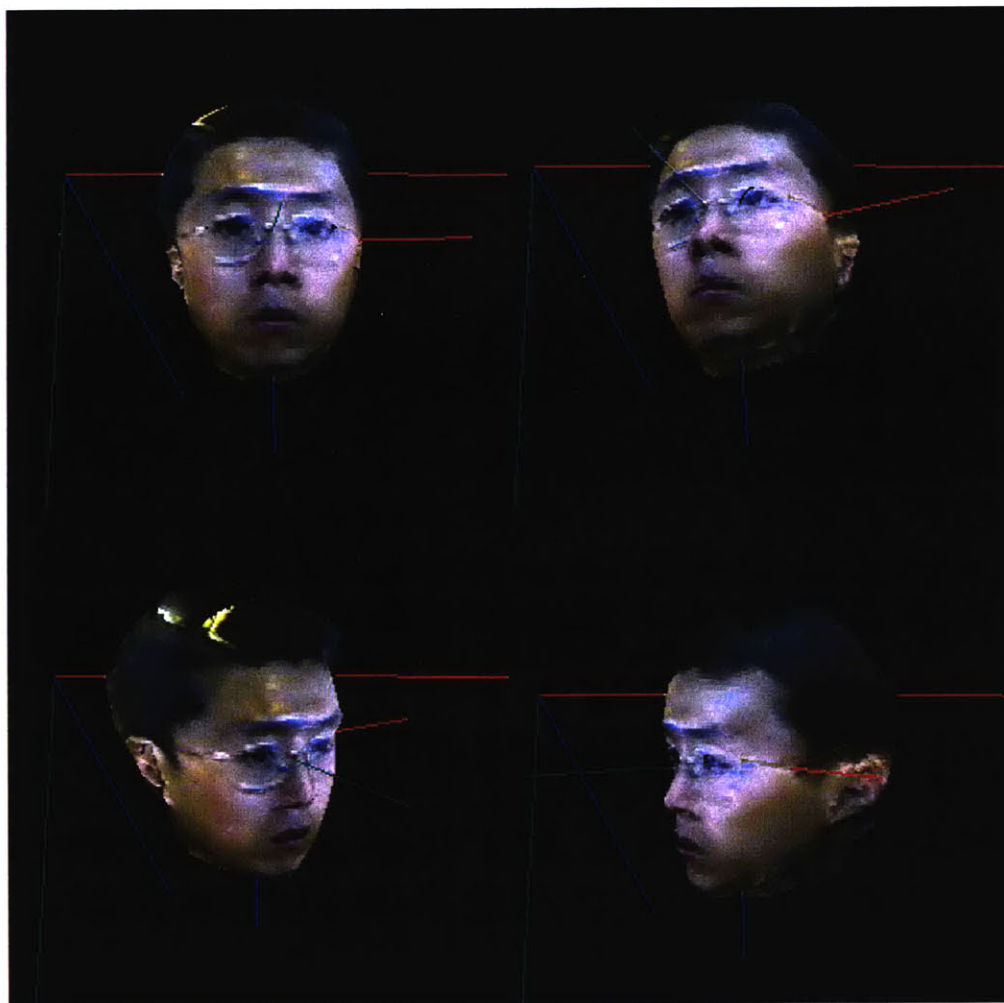
coordinates. However, for the vertices projected on the side views, outside the texture boundary lines, we should modify the 2-D coordinates based on the correction value in the deformation look-up table created in section 3.4.1. Fig 3-10 shows the texture coordinates overlaid onto a texture image.



Note that neither the texture coordinates nor the texture image can cover all the triangles of the model because of the view angle limitations of the three cameras.

### ***3.4.4 Texture mapping***

After getting both the texture image and texture coordinates, we can use OpenGL texture mapping library functions to do texture mapping and to get the individualized 3-D head object. Figure 3-11 shows the 3-D head object rendered from different perspectives.



**Figure 3-12 Individualized 3-D head object**

# Chapter 4

## Tracking and Animation Experiment

Now, we have the individualized 3-D head object. In this chapter, we use the 3-D head object to mimic the motion of the real head. There are two kinds of motion with a head: rigid and non-rigid. This chapter focuses on rigid motion.

### 4.1 Overview

Non-rigid motion typically refers to facial animation, which usually has to involve numerous well-designed animation parameters such as MPEG-4's 68 facial animation parameters (FAP). Rigid motion is relatively simpler; we can use the rotation matrix  $\mathbf{R}$  and the translation vector  $\mathbf{t}$  to describe rigid motion. Estimation of  $\mathbf{R}$  and  $\mathbf{t}$  from video sequences has been an interesting research topic for years. Typically, a 3-D model is involved in the tracking process to constrain the 2-D individual tracking features in a global 3-D structure. Three approaches need to be mentioned.

In [Basu, 1996], a 3-D ellipsoidal model is employed for tracking. In this way, both the actual image optical flow and the model optical flow are computed. Then we can compare the model flow with the actual flow to find the optimal parameters  $\mathbf{R}$  and  $\mathbf{t}$ . This method is robust because computing optical flow in a large area is able to compensate for individual errors. However, this method is non-real-time.

In [Jebara, 1997; Jebara, 1999], a real-time system for 3-D adaptive feedback tracking is introduced. The system uses eight 2-D squares for 2-D template matching; each square represents two points. The resulting sixteen 2-D points are fed into the SfM (structure from motion) algorithm to recover sixteen 3-D points and other camera and motion parameters, including  $\mathbf{R}$  and  $\mathbf{t}$ . In [Strom, 1999], a simple texture-mapped 3-D face model was added to this method for better selection of feature points and more robust 2-D

template matching. This method actually deals with the problem of "2-D projection -- 3-D pose estimation," which is a non-linear problem requiring an iterative least-square solution. Additionally, this method is only for small displacement, normally one pixel per frame.

In [Valente, 2000], a near-real-time approach for visual analysis/synthesis feedback tracking is introduced. This method uses the extended Kalman filter to estimate the 3-D pose from 2-D coordinates of the tracking features found in the 2-D image. This approach employs a realistic head model. Its strength is that the difference between the synthetic face image and the real video frame is used for more robust 2-D feature tracking. This approach makes possible improved solutions to the problems of lighting, scale, large rotation, and geometry deformations.

All these algorithms mentioned above are relatively complex mathematically and are typically used for the non-calibrated single camera case. They are not suitable for our system because we have multiple calibrated cameras that should be taken advantage of.

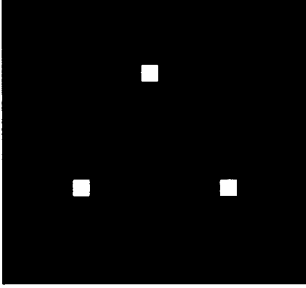
Animation is not our main focus because we only employ a rigid head model for the current thesis project. Here we use texture displacement to implement animation.

## **4.2 Tracking using artificial features**

The existing head tracking systems using computer vision technology have several limitations in common. Working time is relatively short, from several seconds to several minutes; fast motion and large rotation out of plane usually lead to the loss of the features; a real-time requirement is hard to meet when complex algorithms (especially recursive computing sessions) are involved. Besides vision approaches, several other kinds of products exist for detection of human head poses, such as magnetic sensors; however, this kind of equipment is expensive and is not comfortable for users. Sometimes marks are attached on the face for tracking [Ohya, 1995], which may annoy the users. Since we are just trying to track the rigid motion of the head, and we know it is hard to estimate the motion by directly detecting the facial features, why not identify the



motion of other objects attached to the head? This idea is straightforward. We attach a specially designed sign to a cap. When the user wears the cap, the system will estimate the pose of the head by viewing the sign. Since our system has calibrated cameras, we can use two of them to recover the 3-D pose from the 2-D correspondences.



We use the sign as shown in Figure 4-1. In order to avoid light reflection, the sign is made of cloth. Even in an uncontrolled environment, it is quite easy to detect the white points on the black background. One problem we encountered is that the camera sometimes cannot view the white point clearly because of the rotation out of plane. A better sign needs to be designed for more robust detection.

**Figure 4-1 A special sign for tracking**

We use stereo vision to detect the 3-D positions of the three white points on the sign. After detecting the points in one camera view, the epipolar constraint is used for faster detection in the other camera's view.

On the image plane of the camera 1,  $(u,v)$  is known. We have:

$$\begin{bmatrix} h_8 u - h_0 & h_9 u - h_1 & h_{10} u - h_2 \\ h_8 v - h_4 & h_9 v - h_5 & h_{10} v - h_6 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} h_3 - h_{11} u \\ h_7 - h_{11} v \end{bmatrix}. \quad (4.1)$$

Then, on the image plane of the camera 2,  $(u', v')$  is the point corresponding to  $(u,v)$ , and we have:

$$\begin{bmatrix} h'_8 u' - h'_0 & h'_9 u' - h'_1 & h'_{10} u' - h'_2 \\ h'_8 v' - h'_4 & h'_9 v' - h'_5 & h'_{10} v' - h'_6 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} h'_3 - h'_{11} u' \\ h'_7 - h'_{11} v' \end{bmatrix}. \quad (4.2)$$

Selecting one of (4.2) to combine with (4.1), we obtain

$$\begin{bmatrix} h_8 u - h_0 & h_9 u - h_1 & h_{10} u - h_2 \\ h_8 v - h_4 & h_9 v - h_5 & h_{10} v - h_6 \\ h'_8 u' - h'_0 & h'_9 u' - h'_1 & h'_{10} u' - h'_2 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} h_3 - h_{11} u \\ h_7 - h_{11} v \\ h'_3 - h'_{11} u' \end{bmatrix}.$$

Then we know

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} h_8 u - h_0 & h_9 u - h_1 & h_{10} u - h_2 \\ h_8 v - h_4 & h_9 v - h_5 & h_{10} v - h_6 \\ h'_8 u' - h'_0 & h'_9 u' - h'_1 & h'_{10} u' - h'_2 \end{bmatrix}^{-1} \begin{bmatrix} h_3 - h_{11} u \\ h_7 - h_{11} v \\ h'_3 - h'_{11} u' \end{bmatrix}. \quad (4.3)$$

When (4.3) is fed into (4.2), we can get the equation of the epipolar line in the image plane of camera 2

$$\begin{bmatrix} h'_8 u' - h'_0 & h'_9 u' - h'_1 & h'_{10} u' - h'_2 \\ h'_8 v' - h'_4 & h'_9 v' - h'_5 & h'_{10} v' - h'_6 \end{bmatrix} \begin{bmatrix} h_8 u - h_0 & h_9 u - h_1 & h_{10} u - h_2 \\ h_8 v - h_4 & h_9 v - h_5 & h_{10} v - h_6 \\ h'_8 u' - h'_0 & h'_9 u' - h'_1 & h'_{10} u' - h'_2 \end{bmatrix}^{-1} \begin{bmatrix} h_3 - h_{11} u \\ h_7 - h_{11} v \\ h'_3 - h'_{11} u' \end{bmatrix} \\ = \begin{bmatrix} h'_3 - h'_{11} u' \\ h'_7 - h'_{11} v' \end{bmatrix} \quad (4.4)$$



**Figure 4-2 Initial state of tracking**

Before the real-time tracking session, we should first link the 3-D pose of the sign and the 3-D pose of the head. Two video frames are taken by the stereo vision system. We select three facial feature points in each image by hand. The three white points on the sign can be detected automatically. All the 3-D coordinates can be estimated from the

corresponding 2-D projections on the image views of the stereo vision system. In this way we can get the initial coordinates of the feature points both on the head and on the sign. The motion applied to the sign is exactly that applied to the head.

During the real-time tracking session, the white points on the sign are automatically detected frame by frame. We can estimate the 3-D pose by using the method of "3-D -- 3-D estimation" described in the chapter 2. The 3-D pose is determined by the rotation matrix  $\mathbf{R}$  and the translation vector  $\mathbf{t}$ . Here we can use a 4x4 motion matrix  $\mathbf{M}$  to describe the 3-D pose.

$$\mathbf{M} = \begin{bmatrix} [\mathbf{R}] & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \quad (4.5)$$

OpenGL uses the similar 4x4 matrix. The difference is that the motion matrix stored in OpenGL is  $\mathbf{M}^T$ .

Figure 4-3 shows the tracking result. The tracking speed is 30Hz. The colored crosses denote the artificial feature points on the sign. The little colored squares denote the facial features. The colored lines in images (1-b) and (2-b) are the epipolar lines. The 3-D head object can mimic the real head; images (1-c) and (2-c) show the virtual view of the 3-D head object from the perspective of the left camera.

### 4.3 Tracking using facial features

The artificial sign is helpful for more reliable tracking, but we still try to use only facial features for tracking. We can use the sign to set the three feature templates and initialize the pose; the sign can be got rid of during the real-time tracking session. The feature templates are 11x11 patches. When a new frame comes, the searching area is determined by the feature locations in the previous frame. The templates are matched with the patches in a 5x5 search window by using normalized correlation. Let vector  $\mathbf{b}$  denote the candidate patches and vector  $\mathbf{t}$  denote the template. The normalized correlation is given by:



Figure 4-3 Tracking by using artificial features





Figure 4-4 Tracking by using facial features

$$\cos\theta = \frac{\mathbf{b} \cdot \mathbf{t}}{\|\mathbf{b}\| \|\mathbf{t}\|}. \quad (4.6)$$

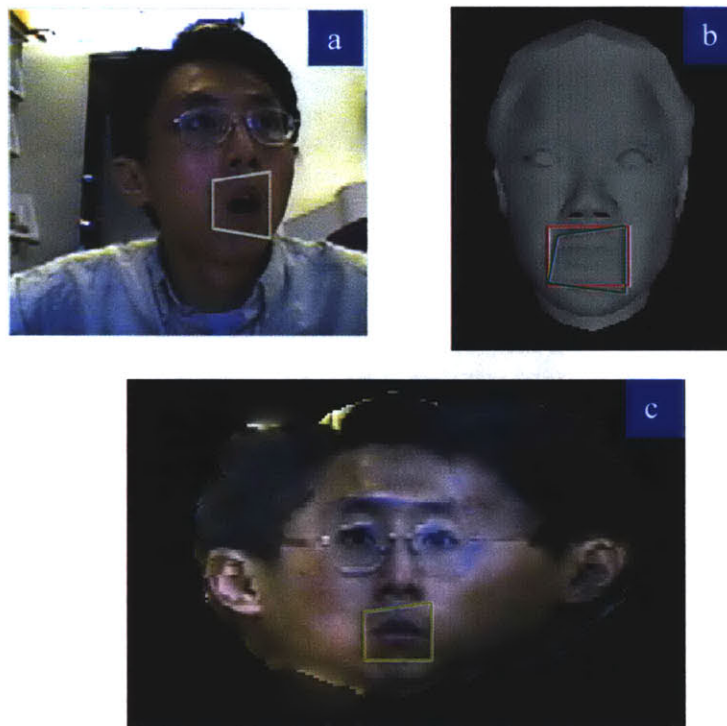
The  $\mathbf{b}$  with the maximum  $\cos\theta$  is selected as the candidate of the 2-D feature location in the current frame. The 3-D coordinates of each feature can be computed from 2-D correspondences in the stereo views. Then "3-D -- 3-D estimation" is used to obtain the rigid motion. Finally, the 3-D coordinates of the features are adjusted by applying the rigid motion to the 3-D model, and they are projected back onto the 2-D image plane to update the new locations in the current frame. Figure 4-4 shows the tracking result. The speed is 30Hz. Images (1-c) and (2-c) show the virtual view of the 3-D head object from the perspective of the left camera. In the images (1-a), (1-b), (2-a), and (2-b), the crosses indicate the 2-D feature matching results, and the squares indicate the projections of the 3-D features. This tracking approach is not robust; a more reliable solution needs to be developed.

## 4.4 Animation using texture displacement

When the image resolution is relatively low (the videoconferencing case), face animation can probably be implemented by altering the texture patches in the areas of the mouth and eyes. In this section, we will apply this animation approach to the mouth area.

The texture patch that can be displaced should be predefined. The red rectangle in Figure 4-5 (b) shows the predefined patch. This patch consists of multiple triangles. All these triangles should be deformed to implement facial animation. To simplify the problem, we can use a quadrilateral on a plane to approximate the shape of the mouth since the accurate shape is unknown. For the convenience of further computing, we use the closest vertices on the VRML model to replace the four vertices of the rectangle region indicated by Figure 4-5 (b). The texture coordinates of the four VRML model vertices having been determined, the corresponding quadrilateral on the 2-D texture image is thereby defined. We use  $Q_T$  to denote this quadrilateral on the texture image and  $Q_M$  to denote the quadrilateral on the 3-D model. We select a camera that can maximize the cosine of the angle between the normal of  $Q_M$  and the camera viewing direction.  $Q_M$  is projected onto

the view plane of the selected camera. The projection quadrilateral is denoted by  $Q_{PM}$ . Figure 4-5 (a) shows the camera view; the green quadrilateral in (a) is  $Q_{PM}$ . Figure 4-5 (b) shows the 3-D model; the green quadrilateral in (b) is  $Q_M$ ; the red rectangle is the predefined animation area drawn by hand. Figure 4-5 (c) shows the texture image; the green quadrilateral in (c) is  $Q_T$ . Note that the quadrilaterals in Figure 4-5 are used only to illustrate the relationships between  $Q_{PM}$ ,  $Q_M$ , and  $Q_T$ . They are not accurately generated by the system.



**Figure 4-5 Predefine the animation area**

In the process of texture displacement,  $Q_{PM}$  is mapped onto  $Q_T$  to replace the texture patch with the current image data. Multiresolution pyramids are used for blending the new patch with the original texture image. The mathematical details can be found in (3.10-17). To improve the processing speed, five-level pyramids of the original texture image can be obtained ahead of time, and the results can be loaded directly when needed. Figure 4-6 shows the result of texture displacement. Figure 4-6 (a) is a video frame; the

green contour is the position of the 3-D head model; the green quadrilateral is the projection of the animation area. Figure 4-6 (b) is the updated texture image.



Figure 4-6 Texture displacement



Figure 4-7 Animation using texture displacement



Figure 4-7 shows the results of texture displacement. The advantage of using texture displacement is that the articulation model and non-rigid motion parameters can be omitted. The drawback is that the quality is not as good; and when the tracking result is not reliable, non-natural faces cannot be avoided.

# Chapter 5

## Networking Schemes and System Implementations

X-Conference is implemented on the Win32 platform. In this chapter, we discuss networking schemes in the X-Conference system. Additionally, we introduce the software implementations of X-Conference.

### 5.1 Many-to-many Architecture

On each conferencing site, two PCs are involved. One PC performs as a server that captures the real video, builds the 3-D object, extracts the motion parameters, and transmits the objects and parameters to the other sites. The other PC performs as a client that receives the visual objects and motion parameters, composes the visual objects into the scene, and renders the scene based on the perspective of the local user. In addition to such conferencing sites, there can be more audience sites that require only client PCs to receive the objects and render the scene. Each server sends the object to multiple clients, and each client receives the objects from the multiple servers. This is a typical many-to-many application. The sending server is a one-to-many application, and the receiving client is a many-to-one application.

As we know, multicast is suitable for one-to-many application because it can reduce server load and network congestion by sending a single data packet to multiple receivers simultaneously. We use the multicasting scheme for real-time conferencing sessions. However, there are still two problems that need to be solved.

- UDP/IP multicast is unreliable. The packets may get lost or arrive out of order. This is not a critical problem for the real-time session. However, before the real-time

session, the geometry model, texture image, texture coordinates, and probably texture pyramids are required to be transmitted to the receiving site reliably. Otherwise, the errors in the 3-D object will exist for the whole conferencing session.

- UDP/IP multicast sends the packet to the whole group, without knowledge of the receiving sites. If a new receiving client intends to join the conferencing session after the beginning, the objects that only the new client needs should not be sent to the whole group.

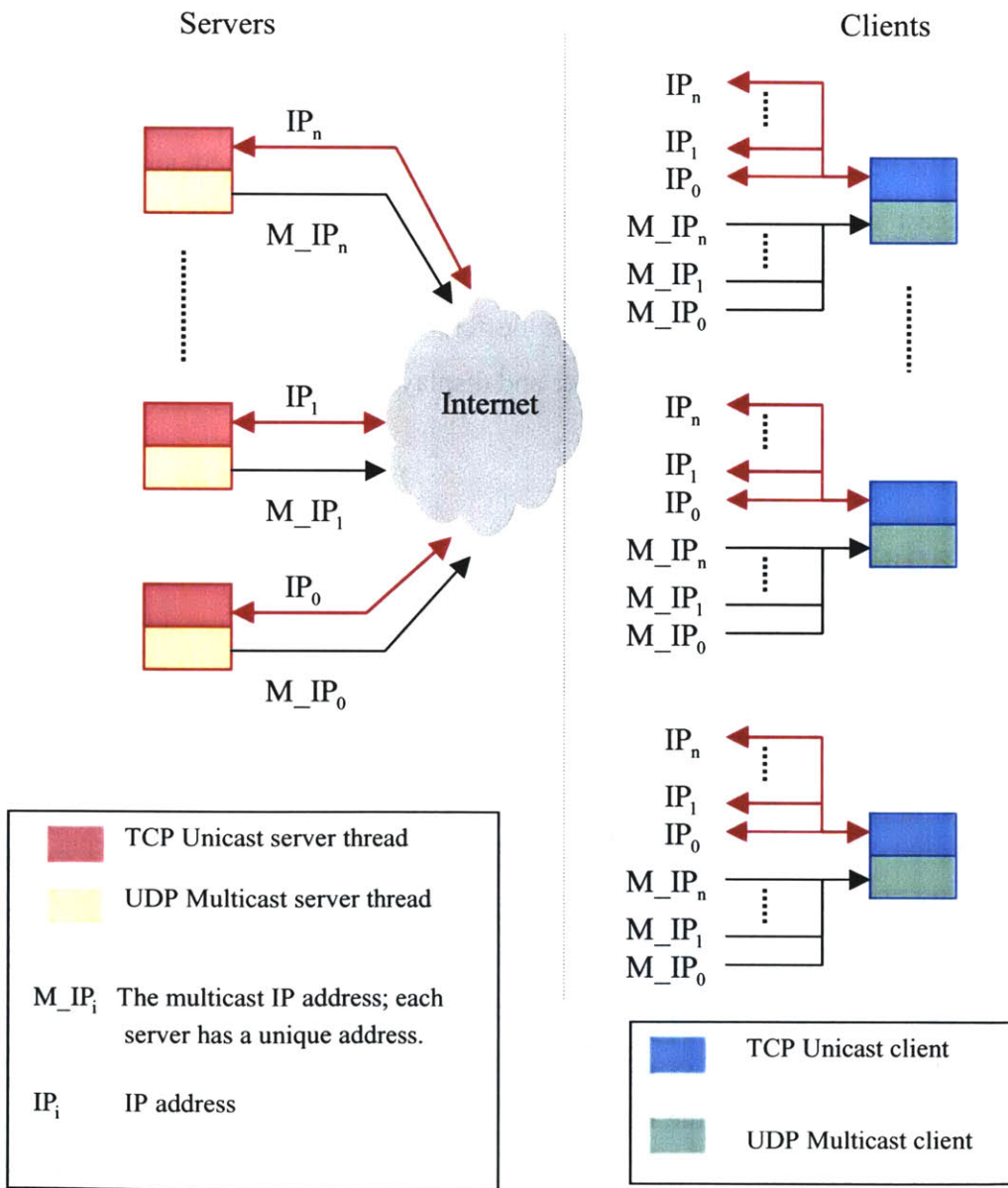


Figure 5-1 Many-to-many architecture

To overcome these two problems, two networking threads are involved in one server. One thread performs as a TCP unicast server; the other thread performs as a UDP multicast server. The TCP thread handles the transmission of the objects. The UDP thread deals with the real-time conferencing session. The TCP server thread is always listening to any TCP client socket. The object will be sent to the client once the request from the client is accepted. The UDP multicast server thread sends the motion parameters out at nearly 30Hz (the speed is dependent on the receiving sites). Each server has a unique IP address and a unique multicast group IP address. The multicast group IP address range is from 224.0.0.0 to 239.255.255.255.

Before joining the multicast group, the client connects to the TCP server sockets one by one to receive the objects. Then the client creates multiple multicast client sockets to join multiple multicast groups. After that, the real-time thread is generated to receive the motion parameters of multiple objects from multiple multicast group servers. When all the motion parameters are brought into the buffer, the real-time thread will send a message to notify the other modules to render and display the scene. Figure 5-1 illustrates the many-to-many architecture.

## **5.2 Rate adaptation**

The additional TCP connections between servers and clients have advantages in improving multicast efficiency. As an example, this section will introduce the rate adaptation strategy in our system.

For current video multicast systems, rate adaptation is used to match the available network capacity. The situation in our system is somewhat different. Our system uses the 3-D avatar approach, which requires low transmission bandwidth. At the current stage, only the rigid-motion parameters are transmitted for each frame. All these parameters consist of 12 float numbers -- 3 for translation and 9 for rotation. Therefore, the output rate of a server is 512bits/frame; the input rate of the client is  $n$  times 512bits/frame ( $n$  is the number of the objects). In this case, network capacity is not a problem. However, 3-D rendering speed varies significantly between different client PCs. Rate adaptation in our

system is intended to efficiently serve all these clients with variability in graphics processing capabilities.

The rendering time for each client may vary dynamically. New clients may join in and old clients may quit at any time. Every five seconds (this clock is adjustable), each client detects the rendering time for a single frame and sends this number to the server through the TCP connection. The server receives the reports from all the clients and adjusts its multicast sending rate based on the fastest rendering speed reported so that the high-graphics-capability client is not being underutilized. Other clients also receive all the frames transmitted from the server, but only render some of them, based on their own graphics processing capabilities.

### 5.3 Software implementation

The software system is implemented on the Win32 platform, utilizing OpenGL, OpenCV, and WinSock functionality libraries. Figure 5-2 illustrates the software architecture of the server. Figure 5-3 illustrates the software architecture of the client.

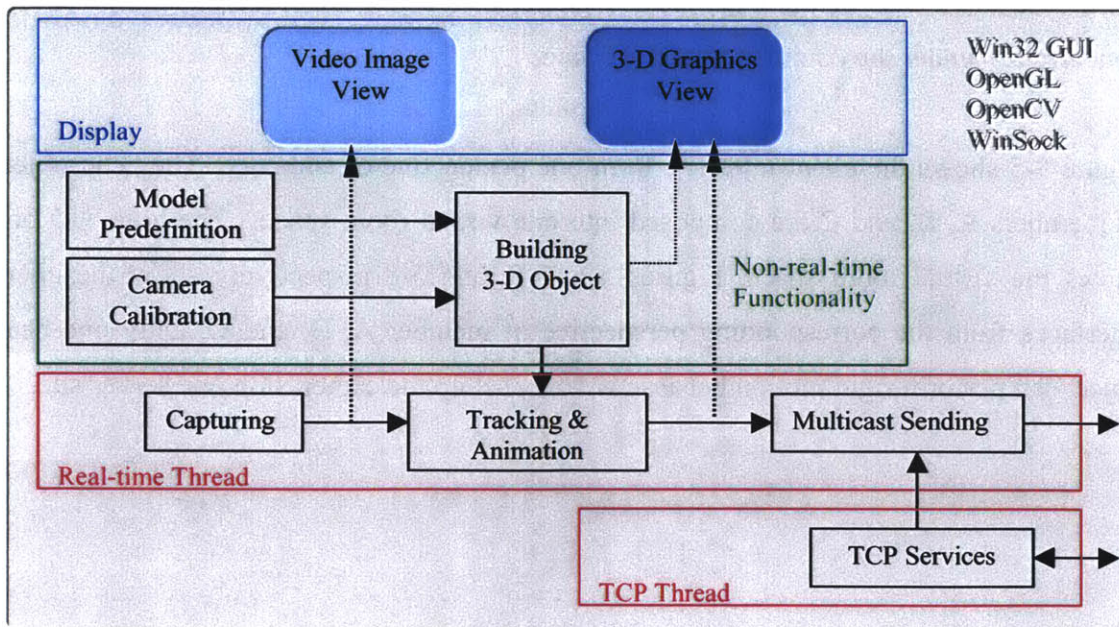


Figure 5-2 Software architecture of the server



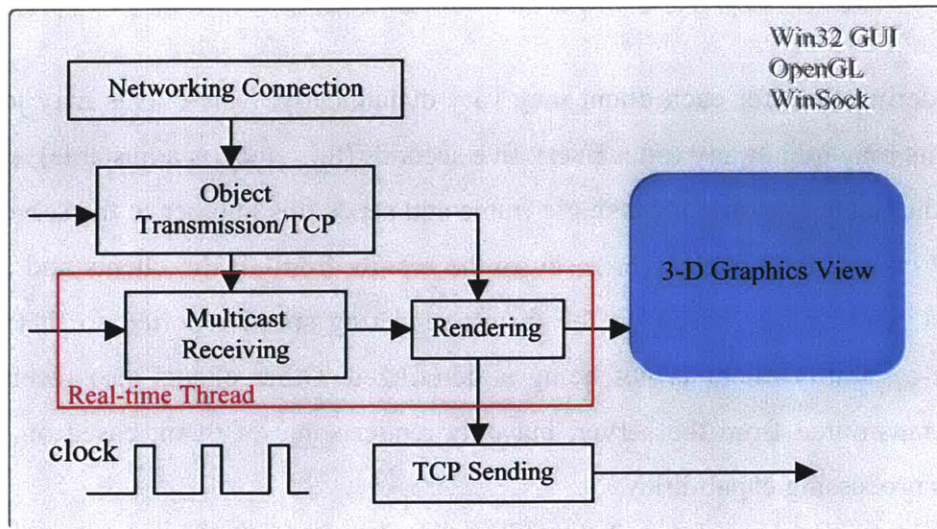


Figure 5-3 Software architecture of the client

## 5.4 User interfaces

Figure 5-4 shows the user interface of a server. The user can supervise the process with the mouse and the keyboard. Both the 3-D graphics view and the video images are shown concurrently under the common window frame.

Figure 5-5 shows the client interface from one perspective of audience. The head object of members A, B, and C are composed into one virtual room space. The blue 3-D box shows the virtual room space. Figures 5-6, 5-7, and 5-8 respectively show the client interfaces from the corresponding perspective of member A, B, and C. Only one head object can perform rigid motion because we only set up the cameras on one server site.

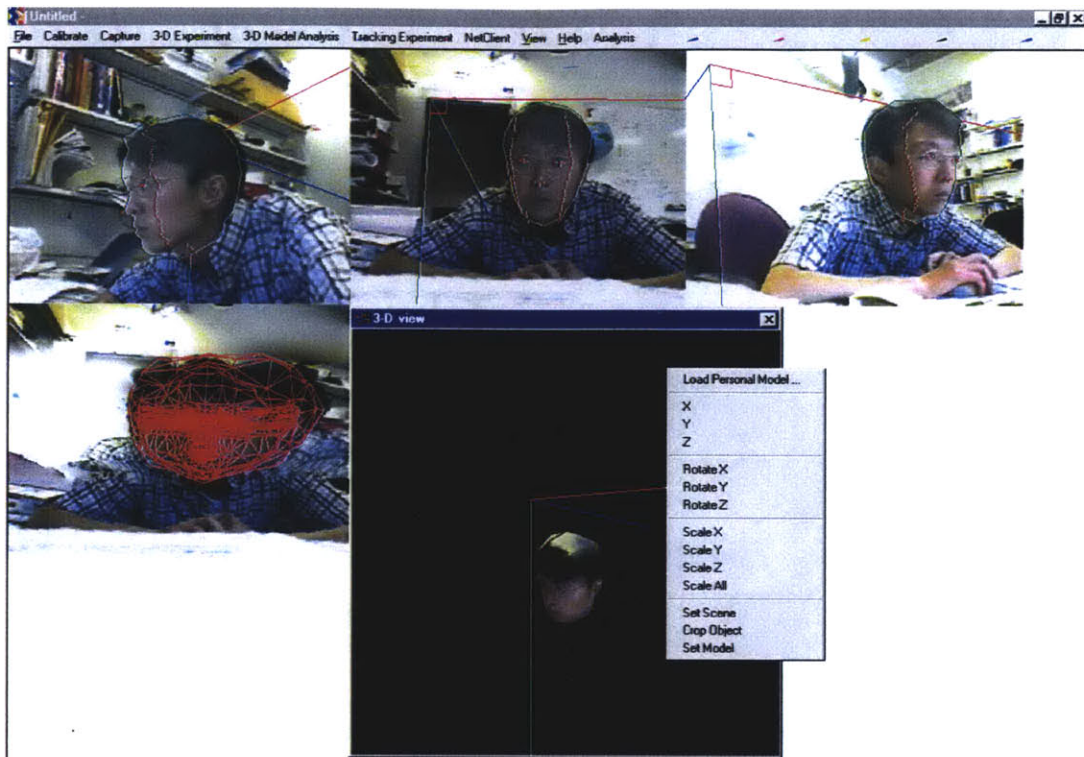


Figure 5-4 The user interface of a server

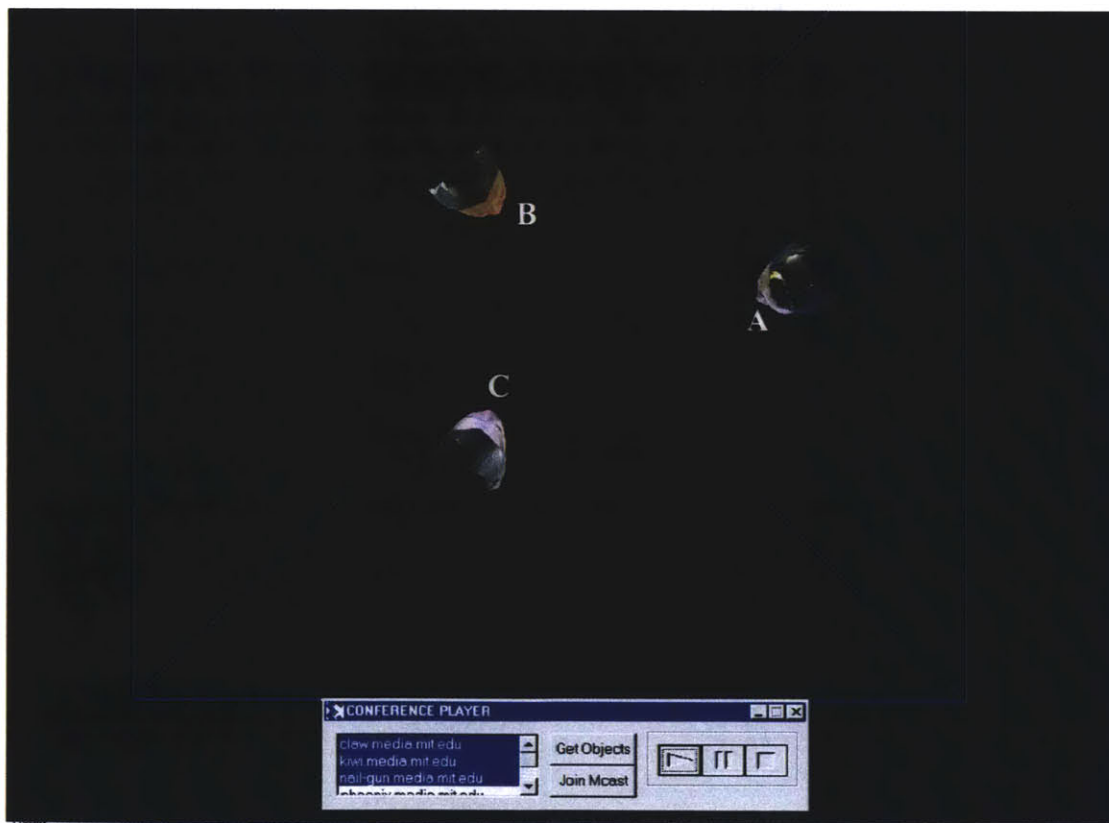


Figure 5-5 Client interface of audience



Figure 5-6 Client interface of member A



Figure 5-7 Client interface of member B





**Figure 5-8 Client interface of member C**

# Chapter 6

## Future Directions

The previous chapters have introduced several aspects of our system. This chapter discusses the limitations of, and proposes improvements for, our system. Moreover, this chapter explores the future directions of teleconferencing reinvention.

### 6.1 Improvements for X-Conference

Our current system has several limitations. The future improvements may be on the following issues:

- Build a more realistic 3-D head by taking advantage of image sequences. A generic head model is used in the current system. In many cases only scaling in three directions cannot cover all the shape differences between the generic model and the real human head. We need to explore the approaches to building a 3-D head from image sequences by tracking and comparing the real images and the model views so that more individualized details can be reconstructed.
- Find a more reliable solution for tracking. The animation quality is highly dependent on the tracking results. To get more reliable tracking results, we need to track more features. We also need to develop some approaches to facial features finding and confidence measurement so that the tracking errors can be recovered when the tracking process fails. Moreover, some methods other than the pure computer vision approach also need to be considered.
- Design a generic model more suitable for video conferencing. Our current system uses a generic VRML model that has 2313 triangles and 1257 vertices. If the expected resolution is similar to that of current videoconference systems, such a sophisticated 3-D model is not necessary. Additionally, more details in the shape of eyes, mouth, and nose usually worsen the quality of texture mapping, especially when

texture displacement is employed for animation. A simpler but more effective 3-D model needs to be designed for our purposes, because current 3-D models available on the Web or scanned by the specialized equipment are usually not qualified for our applications.

- Explore immersive display approaches. In our system, each site has an independently calibrated 3-D scene coordinate system. To smoothly compose the objects of different sites into a single virtual scene, we need to convert the separate 3-D scene coordinates to the common 3-D scene coordinates based on the predefined spatial relationships. Additionally, we hope that a flexible approach can be developed so that precise configuration work is not necessary.

## **6.2 Infrastructures and resources for future research**

Reinventing a teleconferencing system goes much beyond designing a specific system. To support the research effort, some facility and software infrastructures and resources need to be built. Some of them are being developed in other research groups around the world.

- Build multiple calibrated camera systems. Although there exist some solutions for evaluating 3-D structure and the camera parameters from the non-calibrated camera, a multiple calibrated camera system will definitely provide more flexible opportunities to explore a variety of approaches including synthetic view, visual hull, and so on. Additionally, a special space is needed to contain the camera system. CMU's "3D room" [Kanade, 1998] is an example. This effort not only includes facility configuration but also requires the development of the relevant software.
- Build the 3-D head Database. One of the most promising techniques for modeling textured 3-D faces is the learning strategy introduced in [Blanz, 1999]. This technique uses a linear combination of a large number of 3-D face models to describe an arbitrary human face. The process of modeling is to obtain the optimized combination parameter sets. The result is compelling. Building or having access to the 3-D head Database will bring us further improvement through use of some learning algorithms.

- Access MPEG-4 animation resources. MPEG-4 has specified three types of parameters related to the 3-D human face object: Facial Animation Parameters (FAP), Facial Definition Parameters (FDP), and FAP Interpolation Table (FIT). Some research activities are being conducted in this area. Having access to the relevant resources such as a well-designed articulation facial model containing MPEG-4 parameters will be helpful for our future work.

### **6.3 Future research direction**

One of the future research directions is to capture, model, and deliver the motion of the human upper body including gestures. We can also use an avatar to represent the motion. However, we find that visual hull may be a more effective approach to solving this problem. In [Buehler, 1999;Matusik, 2000], a real-time approach to computing visual hulls from silhouette image data is introduced. Only four cameras are used. Visual hull is not suitable for head modeling because it will lose too many details. However, it may be good for real-time human body modeling. Although body motion is important for visual communication, the resolution requirement can be lower, compared to the required head resolution. Therefore, we plan to combine both avatar and visual hull approaches for future development. With this approach, merging the head and the body smoothly is one challenge; compression of visual hull's shape and texture for transmission is another problem to be considered.

### **6.4 The future we envision**

Figure 6-1 shows the future teleconferencing system we can envision today. On each site, the server acquires the 3-D visual object of the local participant, and transmits it to the Internet; the client receives the 3-D visual objects of the other participants and composes the objects into the scene, being rendered from the individual perspective of the local participant. Additionally, the visual objects of all the participants can be composed together for broadcasting news for delivery to a larger population other than the meeting group. In Figure 6-1, the research problems are indicated for each aspect. Although there have been many years of efforts, the 3-D virtual teleconferencing system is still a

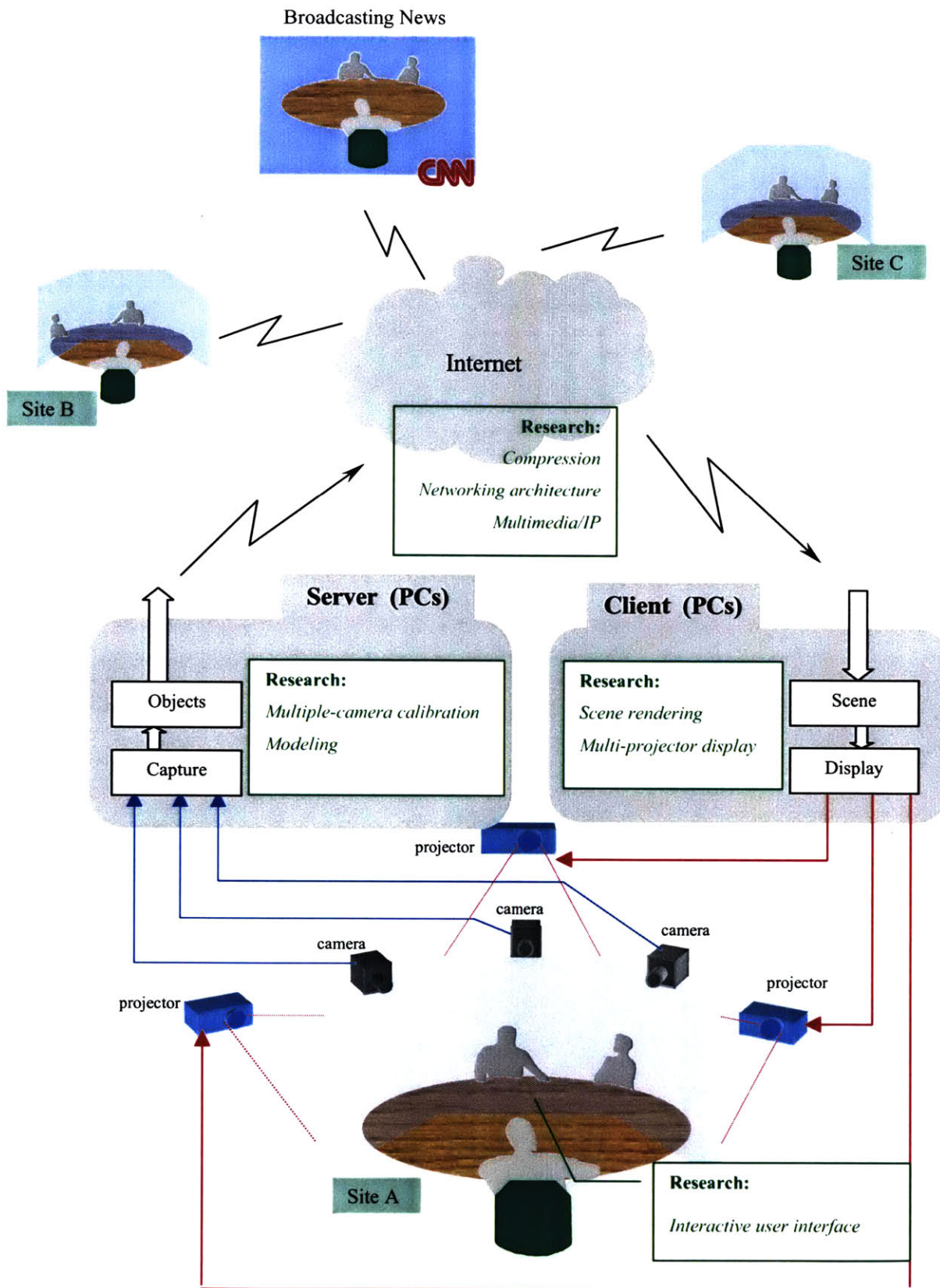


Figure 6-1 Future we envision

challenging research topic. We hope this thesis will make available our investigations and insights for some parts of the topic.

# References

- [Agamanolis, 1997] Stefan Agamanolis, Alex Westner, and V. Michael Bove, Jr., "Reflection of Presence: Toward More Natural and Responsive Telecollaboration," Proc. SPIE Multimedia Networks, 3228A, 1997.
- [Basu, 1996] Sumit Basu, Irfan Essa, Alex Pentland, "Motion Regularization for Model-based Head Tracking," Proceedings of 13<sup>th</sup> Int'l. Conf. on Pattern Recognition (ICPR'96).
- [Blanz, 1999] Volker Blanz, Thomas Vetter, "A Morphable Model For The Synthesis of 3D Faces," SIGGRAPH 99, Los Angeles.
- [Bove, 1995] V. M. Bove, Jr., "Object-Oriented Television," SMPTE Journal, 104, Dec. 1995, pp. 803-807.
- [Buehler, 1999] Chris Buehler, Wojciech Matusik, Steven Gortler, and Leonard McMillan, "Creating and Rendering Image-based Visual Hulls," MIT Laboratory for Computer Science Technical Report MIT-LCS-TR-780, June 14, 1999.
- [Burt, 1983] Peter J. Burt, Edward H. Adelson, "A Multiresolution Spline with Application to Image Mosaics," ACM Transactions on Graphics, vol. 2, no. 4, pp. 217-236 (1983).

- [Darrell, 1997] Trevor Darrell, Sumit Basu, Christopher Wren, and Alex Pentland, "Perceptually-Driven Avatars and Interfaces: Active Methods for Direct Control," SIGGRAPH'97.
- [Gemmell, 2000] Jim Gemmell, Kentaro Toyama, C. Lawrence Zitnick, Thomas Kang, Steven Seitz, "Gaze Awareness for Videoconferencing: software Approach." IEEE MultiMedia, October-December 2000.
- [Gibbs, 1999] Simon J. Gibbs, Constantin Arapis, Christian J. Breiteneder, "TELEPORT - Towards immersive copresence," Multimedia Systems 7(3): 214-221 (1999).
- [Han, 1996] Jefferson Han, Brian Smith, "CU-SeeMe VR Immersive Desktop Teleconferencing," ACM Multimedia'96, ACM New York, 1996, pp.199-207.
- [Haralick, 1989] Robert M. Haralick, Hyonam Joo, Chung-nan Lee, Xinhua Zhuang, Vaidya G. Vaidya, Man Bae Kim, "Pose estimation from corresponding point data." Systems, Man and Cybernetics, IEEE Transactions on , Volume: 19 Issue: 6 , Nov.-Dec. 1989.
- [Heikkilä, 1997] Janne Heikkilä, Olli Silvén, "A Four-step Camera Calibration Procedure with Implicit Image Correction." IEEE Conference on Computer Vision and Pattern Recognition (CVPR'97).
- [Ishii, 1994] Hiroshi Ishii, Minoru Kobayashi, Kazuho Arita., "Iterative Design of Seamless Collaboration Media," Communications of the ACM (CACM), Special Issue on Internet Technology, ACM, Vol. 37, No. 8, August 1994, pp. 83-97.



- [Jebara, 1997] Tony Jebara, Alex Pentland, "Parameterized Structure from Motion for 3D Adaptive Feedback Tracking of Faces." IEEE Conference on Computer Vision and Pattern Recognition (CVPR'97).
- [Jebara, 1999] Tony Jebara, Ali Azarbayejani, Alex Pentland, "3D structure from 2D motion," IEEE Signal Processing Magazine, Volume: 16 Issue: 3 , May 1999.
- [Kanade, 1998] Takeo Kanade, Hideo Saito, Sundar Vedula, "The 3D Room: Digitizing Time-Varying 3D Events by Synchronized Multiple Video Streams", CMU-RI-TR-98-34.
- [Lanier, 2001] Jaron Lanier, "Virtually There," Scientific American, April 2001.
- [Lee, 2000] Won-Sook Lee, Nadia Magnenat-Thalmann, "Fast Head Modeling for Animation," Journal Image and Vision Computing, Volume 18, Number 4, pp.355-364, Elsevier, March 2000.
- [Machin, 1996] David Machin, "Real-time facial motion analysis for virtual teleconferencing," Proceedings of the Second International Conference on Automatic Face and Gesture Recognition, 1996.
- [Matusik, 2000] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven Gortler, and Leonard McMillan, "Image-based Visual Hulls," SIGGRAPH 2000.
- [McLeod, 1999] Dennis McLeod, Ulrich Neumann, Chrysostomos L. Nikias, And Alexander A. Sawchuk, "Integrated Media Systems: The Move Toward Media Immersion," IEEE Signal Processing Magazine, January 1999 VOL. 16, No.1

- [Mortlock, 1999] A N Mortlock, D Machin, S McConnell and P J Sheppard, "Virtual Conferencing," Telepresence, Kluwer Academic Publishers, Boston, 1999, pp. 209-225.
- [Ohya, 1995] Jun Ohya, Yasuichi Kitamura, Fumio Kishino, Nobuyoshi Terashima, Haruo Takemura, Hirofumi Ishii, "Virtual Space Teleconferencing: Real-time Reproduction of 3D Human Images," Journal of Visual Communication and Image Representation Vol.6, No.1, March, pp. 1-25, 1995.
- [Raskar, 1998] Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin, Henry Fuchs, "The Office of the Future: A unified Approach to Image-Based Modeling and Spatially Immersive Displays," SIGGRAPH 98, Orlando, Florida, 1998.
- [Saito, 1999] Hideo Saito, Shigeyuki Baba, Makoto Kimura, Sundar Vedula, Takeo Kanade, "Appearance-Based Virtual View Generation of Temporally-Varying Events from Multi-Camera Images in the 3D room," CMU-CS-99-127.
- [Strom, 1999] Jacob Strom, Tony Jebara, Sumit Basu, Alex Pentland, "Real Time Tracking and Modeling of Faces: An EKF-based Analysis by Synthesis Approach." Proceedings of the Modeling People Workshop at ICCV'99.
- [Tsai, 1997] Chun-Jen Tsai, Peter Eisert, Bernd Girod, and Aggelos K. Katsaggelos, "Model-based Synthetic View Generation from a Monocular Video Sequence," IEEE International Conference on Image Processing, p. I-444, Santa Barbara, Oct. 1997.

[Valente, 2000] Stephane Valente, Jean-Luc Dugelay, "Face Tracking and Realistic Animations for Telecommunicant Clones," IEEE Multimedia, January-March 2000.

[Zhang, 1999] Z. Zhang. "Flexible Camera Calibration By Viewing a Plane From Unknown Orientations." International Conference on Computer Vision (*ICCV'99*), Corfu, Greece, pages 666-673, September 1999.