Using Live Video to Represent Remote Users in Collaborative Virtual Environment

Fang Wang
Electronic Visualization Laboratories
Electrical Engineering and Computer Science Department
The University of Illinois at Chicago
Chicago, IL 60607
fwang@evl.eecs.uic.edu

April 10, 1998

Abstract

How can live video be used, and used effectively to represent remote users in a collaborative virtual environment? With today's high-end graphics workstations' video interfaces and computing power, processing and displaying live video at a real-time frame rate can be achieved. On the other hand, high-speed network which provides the bandwidth and speed necessary to transmit real-time frame rate video data is also becoming more available. This paper is a project report of a collaborative VR system which uses live video to display remote users.

1 Introduction

Each user in a collaborative virtual environment is represented by an avatar, so that users at other sites in the same virtual environment know where they are and what they are looking at. A collaborative VR application sends tracker information of the user in the VR space across the network; at the same time, it receives tracker information of other users in the same VR space and displays their avatars at the right location and orientation. In some collaborative VR applications, remote users are represented by computer generated characters which are usually designed by computer graphics artists.

As high speed network becomes more available and video compression technology advances, using live video to display remote users becomes the next natural step. Compared with avatars using computer generated characters, video avatars represent remote users as they are with real-time facial expressions.

2 Related Work

2.1 Collaborative Virtual Environment

In a collaborative virtual environment, remote users are usually represented by avatars modeled by 3D representations of a person and rendered at the location and orientation according to the tracker information of correspondent remote users sent from the network.

2.1.1 N.I.C.E.

N.I.C.E. [1] is a research here at EVL in Narrative Immersive Constructionist / Collaborative Educational Environments, where avatars were designed primarily with 3D modeling tools such as SoftImage and Alias by computer graphics artists, as shown in figure 1. These avatars are well suited for this learning environment designed for elementary school children.

However, these cartoon-character-like avatars have the major drawback of lacking realism and facial expressions which are essential for certain applications.

2.1.2 The Caterpillar Project

The Caterpillar project [2] was designed and developed at NCSA for engineers located at geographically remote factories to interact with their models in three dimensions in real-time and to communicate design information to each other. In the implemented system, tracker information communication between sites is based on IP multicast protocol. Different sites were also

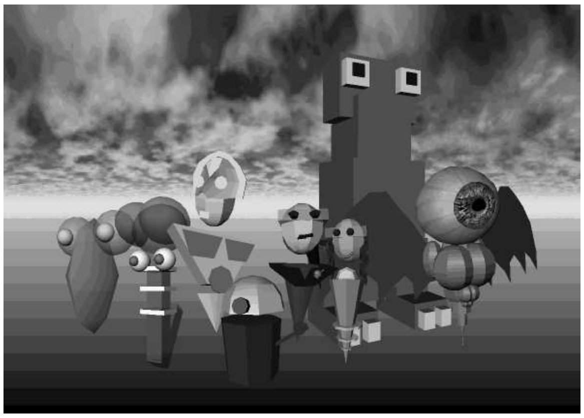

Figure 1: Avatar characters used in some collaborative VR environments

linked by an ATM-based high-speed network, through which real-time audio as well as video from a camera located in front of participants was being sent. Remote users were represented by actual video coming from remote participants placed on a polygon.

The audio and video communication software used was VAT and VIC respectively, available from Lawrence Berkeley Laboratory. The best result documented for the Caterpillar Project was 8 frames per second color video.

2.2 Image- or video-based Virtual Reality

Image- or video-based virtual reality has been the focus of research in recent years at several major research institutions. It combines technologies from both computer graphics and computer vision, and tries to create virtual environments from either photographs or video images.

2.2.1 View interpolation for image synthesis

In [3], the author presented a technique for an image-based rendering system to generate scenes for 3D walkthroughs. For a 3D scene, what they have available are images from known camera locations, and also range data at each camera location. Therefore, from one image to another, disparity maps can be computed directly from the depth information from the range data. In-between views can be generated by linear interpolating the disparities.

2.2.2 Virtual Space teleconferencing using a sea of cameras

In [4], the authors proposed a virtual environment where a multitude of cameras were placed all around a physical room to capture the scene. Wide-baseline stereo was used to extract depths maps, which were then combined to create a virtual scene for remote display. In one implementation, 11 cameras were placed surrounding a person's head, which were part of a real-time stereo machine [5]. Dense depth maps were extracted and combined in real-time by the stereo machine. The reconstructed person's head was then displayed in a 3D virtual environment.

2.2.3 Immersive Video

In [6] and [7] the authors designed and built a system to record real-life events using multiple cameras and be able to playback the events at any chosen viewpoint by a user. Dynamic objects in the scene were reconstructed after recording. During playback, all video-images were synchronized and mapped back onto the objects using a rendering technique called "projective texture mapping".

The technique of projective texture mapping has several advantages to be used for image-based rendering. First of all, the projection frustum can be modeled from the camera geometry readily. When set up properly, back projection maps the view from camera onto the object with features automatically aligned. No additional image analysis or texture coordinate computation is needed. Secondly, it is available in hardware on most high-end graphics workstations. Therefore minimum amount of work needs to be done in software. It was also used by [8] to render architectural buildings reconstructed from photographs.

To summarize, most existing image- or video-based virtual reality systems were designed to render static scenes ([3] and [8]), or to playback dynamic events ([6] and [7]). Others relied on additional hardware ([4]) to extract 3D models in real-time, or additional networking connections ([2]).

3 Video Avatars in Collaborative VR

3.1 System Design Goals

One of the first goals for a collaborative VR system using video avatars is for it to be able to use existing networking and avatar management layer used for collaborative projects developed here at EVL, such as the N.I.C.E. project ([1]). The advantages of this are obvious: reuse of part of a working system, and ease of integrating this work into existing collaborative projects.

Another important goal is to take advantage of new SGI workstations such as O_2's hardware JPEG compression capability and unified memory architecture, so that video can be compressed and decompressed at a 30-frames-per-second-rate; and that decompressed video can be output at the same frame rate to the video interface of another workstation which generates graphics for VR applications.

3.2 System Setup

In the preliminary implementation, 2 ImmersaDesks driven by SGI Onyx's were used. The main reason to use ImmersaDesks is to have better control over lighting for video capturing. A camera is mounted on top of the screen of the each of the ImmersaDesks to capture a participant's movement and facial expressions.

The video output is fed to an O_2 placed near the ImmersaDesk. At the same time, the audio output from the microphone is also fed to the O_2. Video undergoes JPEG compression, and together with uncompressed audio, are sent across a 100Base-T fast ethernet to another O_2, which is at the other ImmersaDesk. Audio is played directly from the O_2 for the user at this ImmersaDesk. Video undergoes decompression and is sent to the sirius video interface of the Onyx driving the ImmersaDesk.

The VR application uses the video input as a texture, and maps it onto a polygon which is placed at the right translation and rotation according to the tracker information of the remote user.

The same procedure happens both ways. Figure 2 shows the data flow and communication in the system.

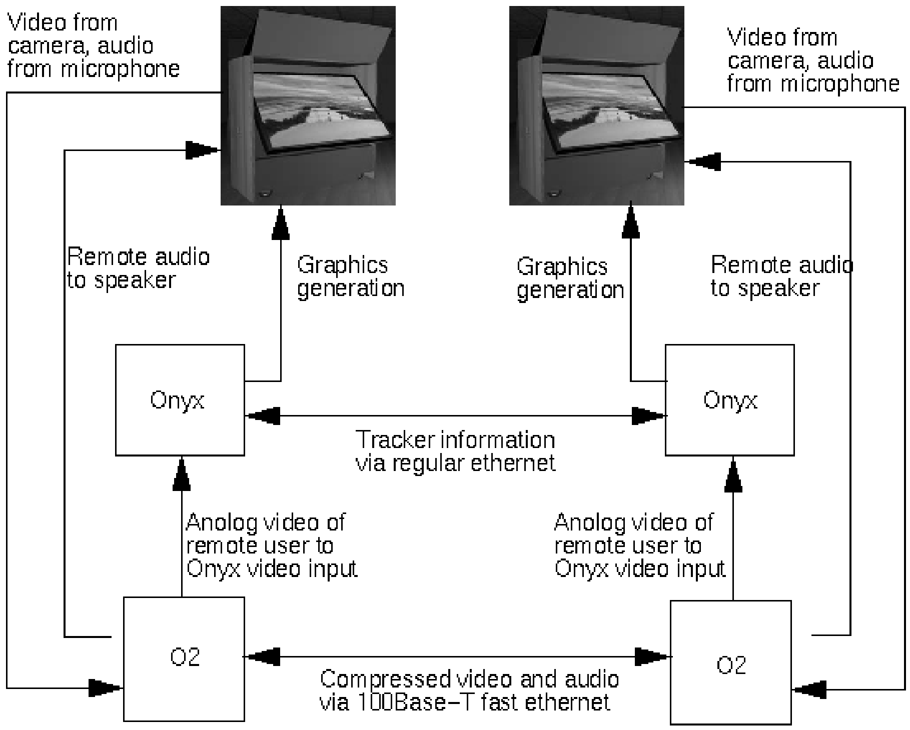

Figure 2: Hardware components and data communication in a collaborative VR system using video avatars

3.3 Implementation

Video avatar class was built as a subclass of the existing avatar base class, so that existing avatar management and networking can be readily reused. More importantly, the objected-oriented design enables applications using existing avatar technologies to easily integrate video avatars, and it provides a base class for future work to be done to create full dynamic 3D video avatars.

3.3.1 2D Video Avatars

When a remote user is represented by a 2D video avatar, the video input is being texture mapped onto a polygon which is placed at the right location and orientation according to the tracker information of the remote user.

To fully make use of the hardware chroma keying capabilities of the Onyx video interface, a blue screen is set up behind the user, so that the background of the video images becomes transparent when the chroma key values are set up correspondingly to the color of the blue screen. Although the 2D video avatar is no more than a texture on a flat polygon, the background-cutout makes the avatar appear convincingly immersed in the 3D environment as shown in figure 3.

A graphical user interface is also built so that one can adjust chroma keying parameters easily when lighting conditions change.

3.3.2 $2^1/_2$D Video Avatars

The next step of improvement in the implementation is to add more sense of 3 dimensionality to the video avatar. Instead of texturing mapping the video image onto a flat polygon, the video image is being back projected onto a static head model using a projective texture matrix. This allows the view from the camera to be mapped back onto the head model, which is placed at the head location and orientation of the remote user.

The image has to be projected in such a way so that features in the image are aligned with correspondent features of the head model. One way to achieve this is to set up a "viewing frustum" as shown in figure 4 for the texture to be projected for the head model exactly as the camera geometry relative to the remote user's head. This way the video images are being applied to the head model as if the camera is being replaced by a slide-projector which projects the images back onto the model.

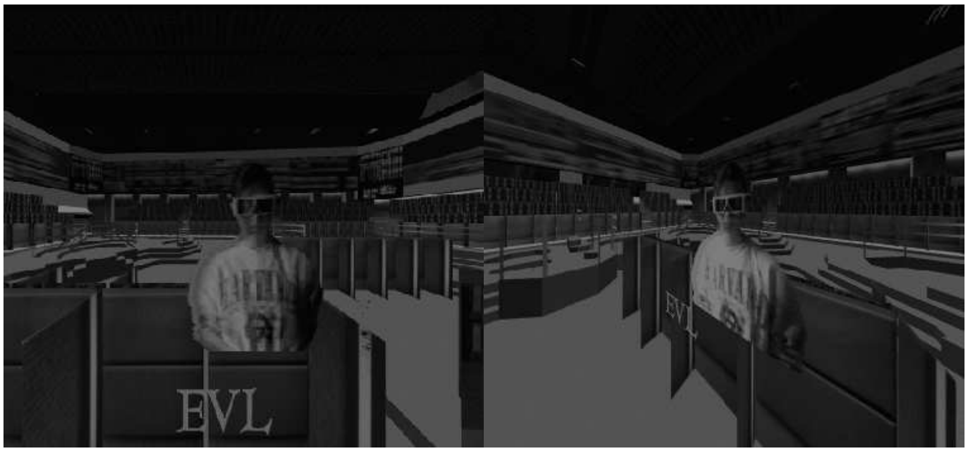

Figure 3: A 2D video avatar representing a remote user in the Virtual Chicago Mercantile Exchange

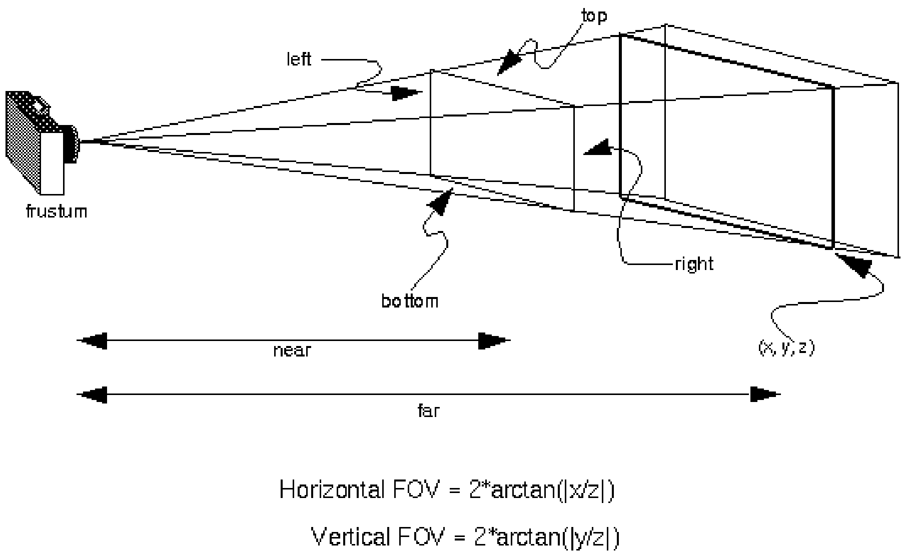

Figure 4: Texture projection frustum

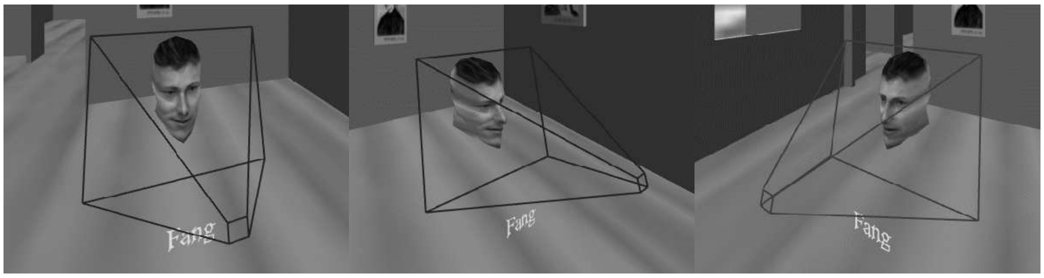

Figure 5: A $2^1/_2$D video avatar with a static 3D head model and projected texture with the projection frustum shown

In order to set up the projection frustum in the correct location and orientation relative to the head model, some camera calibration is needed. However, as found out in practice, a precise camera model is not essential. Therefore, in order to coarsely calibrate the camera mounted on top of the ImmersaDesk screen, one can simply hold the tracked glasses to the camera, and find out camera location and orientation. Camera's field of view (FOV) can also be estimated by holding the tracked glasses in front of the camera so that it appears just in one (any) corner of its video image. The (x, y, z) coordinates of the this point can then be transformed to camera's coordinate system, since its location and orientation have already been estimated. Camera's FOV can then be estimated easily. One can repeat this process to get a better estimate of camera's FOV. With this camera information, the projection frustum can be set up with its origin at camera's location relative to the user, and its viewing direction pointing at the user as the camera.

Figure 5 is a static head model with a static face image projected from the frustum shown. It shows that, when the projection frustum is set up accordingly as to the camera's viewing frustum by which the image was taken, the result is quite convincingly real and 3 dimensional even though the head model is not that of the person whose image is being projected.

The projective texturing mapping is implemented in OpenGL by manipulating the texture matrix. The projection frustum is, at all times, at the exact same location and orientation as the camera is relative to the remote user the avatar is representing. Therefore, when the remote user virtually navigates in the VR space, and the head model of the avatar moves and turns in the virtual environment according to the remote user's tracker information, the frustum should move virtually together with the user, as shown in figure 6. To achieve this effect, texture coordinate generation has to be done for the camera in "OBJECT_LINEAR" space of the head model it is being applied to, as shown in the following code segment:

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGeni(GL_Q, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
GLfloat sPlane[] = {1.0, 0.0, 0.0, 0.0};
GLfloat tPlane[] = {0.0, 0.0, 1.0, 0.0};
GLfloat rPlane[] = {0.0, -1.0, 0.0, 0.0};
GLfloat qPlane[] = {0.0, 0.0, 0.0, 1.0};
glTexGenfv(GL_S, GL_OBJECT_PLANE, sPlane);
glTexGenfv(GL_T, GL_OBJECT_PLANE, tPlane);
glTexGenfv(GL_R, GL_OBJECT_PLANE, rPlane);
glTexGenfv(GL_Q, GL_OBJECT_PLANE, qPlane);
```

The goal here is to generate texture coordinates from the camera with r along the camera viewing vector and s and t along the x and y axis in camera space. An easy way to accomplish this is once s, t and r coordinates along x, y and z OBJECT_LINEAR axes are being generated, use texture matrix transformations to translate and rotate these coordinates to the camera location and orientation.

Once this is done, camera's FOV is used to compute the parameters needed to construct the perspective texture coordinates between [-1, 1] within the viewing frustum of the camera. The next step is to change the [-1, 1] coordinate mapping to the [0, 1] coordinate mapping of the image stored as an OpenGL texture.

So once the texture coordinate generation is applied, the texture matrix manipulations outlined above can manipulate the generated coordinates as described above. The transformations appear in reverse order from the described sequence due to the multiplication order required for texture matrix manipulations in OpenGL:

```
glMatrixMode(GL_TEXTURE);
glLoadIdentity();
```

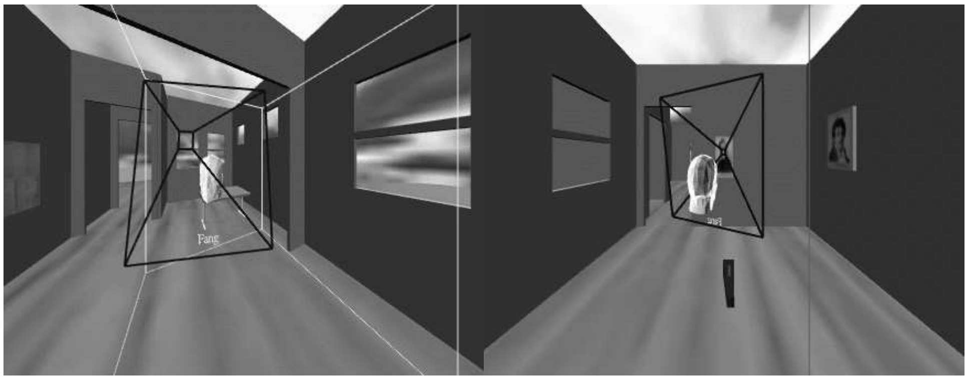

Figure 6: Projection frustum moves with the head model when remote user virtually navigates in the environment

```
//Transform to texture coordinate system [0, 1]
glTranslatef(0.5, 0.5, 0);
glScalef(0.5, 0.5, 1.0);
// perspective projection
float hfov = pfTan(*cameraHFOV/2.0f);
float vfov = pfTan(*cameraVFOV/2.0f);
*xmin = -hfov* *znear;
*xmax = hfov* *znear;
*ymin = -vfov* *znear;
*ymax = vfov* *znear;
glFrustum(*xmin, *xmax, *ymin, *ymax, *znear, *zfar);
// Transform the projection frustum to camera position and rotation
glTranslatef(*cameraX, -*cameraZ, *cameraY);
glRotatef( *cameraRotX, -1.0, 0.0, 0.0);
glRotatef( *cameraRotY, 0.0, 0.0, 1.0);
glRotatef( *cameraRotZ, 0.0, -1.0, 0.0);
```

However, the head model of the avatar moves and rotates in the virtual environment not only as the remote user virtually navigates, but also as he/she physically moves or turns in front of the ImmersaDesk. In the latter case, however, the projection frustum should not move with the head model. Instead, it should always stay at the location and orientation of the camera as relative to the remote user, as shown in figure 7. This is equivalent to have the projection frustum stay fixed in the ImmersaDesk coordinate system of

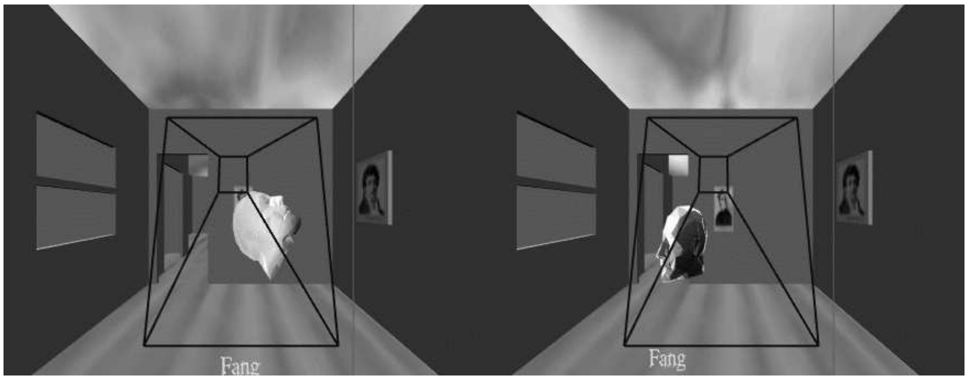

Figure 7: Projection frustum does not move with the head model when remote user physically moves or turns

the remote user as where the camera is. This can be done by adding a few more matrix manipulations at the end the texture matrix operations, to first transform the coordinate system from the origin of the OBJECT_LINEAR space of the head model to that of the remote user's ImmersaDesk coordinate system:

```
//(x, y, z) is the head position of the remote user
//in ImmersaDesk coordinate system as returned by tracker
glTranslatef(-x, z, -y);
//(rotX, rotY, rotZ) is the head orientation of the remote user
//in ImmersaDesk coordinate system as returned by tracker
glRotatef(-rotX, 0.0, -1.0, 0.0);
glRotatef(-rotY, -1.0, 0.0, 0.0);
glRotatef(-rotZ, 0.0, 0.0, 1.0);
```

When the projection frustum is set up as described above, and the texture being back projected is the live video of the remote user, the resultant video avatar is a static head model with real-time video texture, thus is termed as a "$2^1/_2$D" video avatar. As shown in figure 8 and figure 9, live video images are being back projected onto a static 3D head model. Here again, the head model is not that of the person whose images are being projected, the avatar looks surprisingly real as well as 3 dimensional.

Figure 8: A $2^1/_2$D video avatar with a static 3D head model and projected real-time video texture viewed from the front

Figure 9: A $2^1/_2$D video avatar with a static 3D head model and dynamic facial expressions viewed at an angle

4 Conclusions

4.1 Visual Effect

One surprising result from the implementation of the $2^1/_2$D video avatar is that a common head model has worked reasonably well with different people's facial textures. When the lighting for the head model is disabled, the shading in the projected texture becomes a strong visual cue of the 3 dimensionality of the head model, which seems to explain this observation.

Another surprise is its visual effect as a fully dynamic 3D video avatar. Since the head model is not animated at all, this visual effect seems to come from the dynamic facial expressions being projected onto the model. Does this imply that a large part of the effect of facial animation can be achieved by simply animating the facial texture, without distorting the underlying facial structure?

4.2 System Performance

The system is designed and implemented to make maximum use of hardware capabilities of graphics workstations, so that no more than necessary work is to be done in software which will hinder the frame rate of an interactive virtual environment. When running with 2D video avatars, VR applications usually do not experience a significant slowing down of the frame rate. Graphics are updated at 20 frames per second, which is necessary for interactive VR.

Video and audio are exchanged via a 100Base-T fast ethernet connection between the O_2's by a teleconferencing software developed by Javier Girado at EVL. It is also developed to make maximum use of O_2's JPEG capability and its unified memory architecture. It handles all the video compression / decompression, video / audio synchronization, and communication via the fast ethernet. Audio sampling rate is 8 kHz. Video is compressed, and sent to the remote site together with an audio sample at 30 frames per second. Decompressed video from the remote site is output to a workstation running VR applications at 30 frames per second.

The system has successfully been run with 2D video avatars locally at EVL between 2 ImmersaDesks each with an Onyx a desktop O_2. Since the graphics update rate is around 20 frames per second, that becomes the actual video rate experienced by the user. Delay in video and audio is typically experienced by ten's of a second, or around 3-5 frames in video.

The system has been also integrated with various virtual environments such as the Virtual Chicago Mercantile Exchange shown in figure 3 , and has proven to be an effective way to present remote users in a collaborative virtual environment.

It is to be further tested by running simultaneously at Washington, DC and here at EVL.

5 Future Work

$2^1/_2$D video avatars are still under further development. Extensive tests are necessary to determine the robustness and efficiency of the back projection of video texture. Since the projection result is largely dependent on how accurately the projection frustum models the camera geometry, a user interface to enable easier and more accurate and precise camera calibration is desirable.

Future work is also desirable to developed full 3D video avatars. This can be done by using a dynamically animated head model of the remote user with live video texture projection. One should also explore the possibility and validity of view interpolation using two or more video streams.

6 Acknowledgment

The author is grateful for every EVL'er who helped with the design and implementation, as well as those who helped with system setup for every test run. Specially, the author would like to thank the following individuals: Dan Dandin, for his continuous support that pushed the project forward; Javier Girado, who single handedly developed the teleconferencing software for the O_2's; N.I.C.E developers, who provided a robust object-oriented framework for this work to take place.

References

[1] Johnson, A., Roussos, M., Leigh, J., Barnes, C., Vasilakis, C., and Moher, T.: The NICE project: Learning together in a virtual world. Proceedings of VRAIS '98, March 1998.

[2] Lehner, V. and DeFanti, T.: Distributed virtual reality: Supporting remote collaboration in vehicle design. IEEE Computer Graphics and Applications, March 1997.

[3] Williams, L. and Chen, S. E.: View interpolation for image synthesis. SIGGRAPH '93, 1993.

[4] Fuchs, H., Bishop, C., Arthur, K., and McMillan, L.: Virtual space teleconferencing using a sea of cameras. Proceedings of the 1st International Symposium on Medical Robotics and Computer Assisted Surgery, September 1994.

[5] Kanade, T., Yoshida, A., Oda, K., Kano, H., and Tanaka, M.: A stereo machine for video-rate dense depth mapping and its new applications. Proceedings of 15th CVPR, June 1996.

[6] Moezzi, S., Katkere, A., Kuramura, D. Y., and Jain, R.: Immersive video. VRAIS '96, 1996.

[7] Moezzi, S., Katkere, A., Kuramura, D. Y., and Jain, R.: Reality modeling and visualization from multiple video sequences. IEEE Computer Graphics and Applications, 16(6):58–63, November 1996.

[8] Debevec, P. E., Taylor, C. J., and Malik, J.: Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. SIGGRAPH '96, 1996.