Viewport: A Distributed, Immersive Teleconferencing System with Infrared Dot Pattern

Cha Zhang, Qin Cai, Philip A. Chou, and Zhengyou Zhang
Microsoft Research

Ricardo Martin-Brualla
University of Washington

The Viewport immersive teleconferencing system reconstructs sparse 3D representations for each user and applies virtual seating to maintain the same seating geometry as face-to-face meetings.

Immersive teleconferencing has been an attractive research topic for many years, aiming to produce ''a videoconferencing experience that creates the illusion that the remote participants are in the same room'' as the user.[1] An ideal immersive teleconferencing system should be able to replicate the light and sound field that a user might experience in a face-to-face meeting, all in real-time, which is a nontrivial task.

In the past few years, several commercial group-videoconferencing systems have emerged, such as Hewlett-Packard's Halo and Cisco's Telepresence. These systems feature life-size video, spatial audio, and a virtualized environment by providing furniture, lighting, and even wall colors to maximize the conference experience. Although they provide a substantially improved conference experience over traditional Internet video chatting, they cannot faithfully preserve the mutual gaze between attendees and are thus still far from being truly immersive.[2] Research has shown that feeding videos of multiple cameras directly into multiview displays at the remote party improves the efficiency of group conferencing because it roughly preserves the gaze.[3] Nevertheless, many challenges and inflexibleness in group videoconferencing still exist, such as the low visual quality of multiview displays, the limited view position of the user, and the lack of full 3D rendering and motion parallax.

In this article, we propose a fully distributed immersive teleconferencing system that assumes there is only one user at each station during the conference. Such a system would allow users to conduct meetings in their own offices for greatest convenience. Compared with group conferencing systems, fully distributed systems let us render the light and sound fields from a single user's viewpoint at each site, which demands less of the system hardware. (See the ''Related Work in Immersive Teleconferencing'' for previous approaches.)

To create faithful eye contact in remote meetings, we present Viewport, a fully distributed immersive teleconferencing system with virtual seating. With the help of infrared (IR) dot patterns, we can reconstruct high-quality 3D models for each user in real time and embed these 3D models into a common virtual environment. Thanks to our virtual-seating algorithm and a careful calibration scheme, Viewport can faithfully maintain the seating geometry of face-to-face meetings, leading to accurate mutual gaze between meeting participants.

Immersive Teleconferencing Systems

To create an immersive experience, we need to support several aspects of teleconferencing:

- A flexible number of participants for the conference. When the number of cameras at each site is fixed, a full 3D reconstruction is necessary to ensure correct mutual gaze when the number of participants is larger than the number of cameras.
- Conference seating geometry identical to face-to-face meetings (see Figure 1). This implies that we calibrate all the equipment at each site, including cameras that capture the person, display, table height, and so on.

1070-986X/13/$31.00 © 2013 IEEE Published by the IEEE Computer Society

Related Work in Immersive Teleconferencing

The pursuit of fully distributed immersive teleconferencing has been ongoing for many years. Early representative works include the Tele-immersion Portal,[1] the Virtue project,[2] and the Coliseum.[3] Due to limitations in computation capability, the rendering quality of these systems was often limited.

More recently, a few immersive teleconferencing systems have attracted much attention. For instance, Andrew Jones and his colleagues adopted a phase-unwrapping-based structured light approach for real-time facial geometry reconstruction.[4] Ram Vasudevan and his colleagues used 48 cameras and 12 computers at each station for a multiresolution-mesh-based reconstruction, which greatly improved rendering quality.[5] Andrew Maimone and Henry Fuchs presented a teleconferencing system using six Kinect sensors at each site.[6] They merged and processed depth maps to improve rendering quality. Hybrid Kinect and regular camera systems could also be used for immersive teleconferencing,[7] thanks to their near-real-time performance.

All these systems shared the same vision: once the 3D models of the users are reconstructed, they can be placed in a common virtual environment to be rendered immersively. However, none of these works paid much attention to the seating geometry in the virtual world. Consequently, the mutual gaze between attendees has usually been poor in multiparty conferences.

References

1. A. Sadagic et al., ''Teleimmersion Portal: Towards an Ultimate Synthesis of Computer Graphics and Computer Vision Systems,'' *Proc. 4th Ann. Int'l Workshop on Presence,* 2001; www.cs.unc.edu/Research/stc/publications/Sadagic_Presence01.pdf.
2. O. Schreer et al., ''A Virtual 3D Video-Conferencing System Providing Semi-immersive Telepresence: A Real-Time Solution in Hardware and Software,'' *Proc. Int'l Conf. eWork and eBusiness,* 2001, pp. 184–190.
3. H.H. Baker et al., ''The Coliseum Immersive Teleconferencing System,'' *Proc. Int'l Workshop on Immersive Telepresence,* ACM, 2002.
4. A. Jones et al., ''Achieving Eye Contact in a One-to-Many 3D Video Teleconferencing System,'' *Proc. SIGGRAPH Emerging Technologies, ACM,* 2009, article no. 64.
5. R. Vasudevan et al., ''Real-Time Stereo-Vision System for 3D Teleimmersive Collaboration,'' *Proc. IEEE Int'l Conf. Multimedia and Expo* (ICME), IEEE, 2010, pp. 1208–1213.
6. A. Maimone and H. Fuchs, ''Encumbrance-Free Telepresence System with Real-Time 3D Capture and Display Using Commodity Depth Cameras,'' *Proc. Int'l Symp. Mixed and Augmented Reality,* IEEE CS, 2011, pp. 137–146.
7. C. Kuster et al., ''Freecam: A Hybrid Camera System for Interactive Free-Viewport Video,'' *Proc. Int'l Workshop on Vision, Modeling, and Visualization,* AK Peters, 2011.

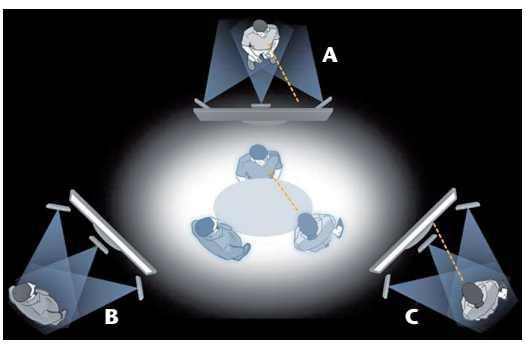

Figure 1. Distributed three-way videoconferencing that preserves the same relative seating geometry as a face-to-face meeting. Users A and C are looking at each other, and B will understand that A and C are looking at each other because B sees only their side views.

- 3D audio and video rendering of remote participants, with motion parallax. For audio, spatial audio rendering is needed as well as multichannel echo cancelation. Video demands high-quality 3D reconstruction to support motion parallax.
- Real-time operation and low network bandwidth and latency between participants.

Figure 1 illustrates three people attending a virtual meeting from offices in three separate locations. A capture device at each location captures users in 3D with high fidelity (in both geometry and appearance). The system then places the users into a virtual room as if they were seated at the same table. The camera tracks the users' eye positions so that the virtual room is rendered appropriately at each location from the user's perspective, reproducing the motion parallax that a user would see in the real world. The mutual gaze between remote users is maintained by creating a consistent geometry and tracking the user's eye positions.

In Figure 1, users A and C are looking at each other, and B will understand that A and C are looking at each other because B sees only their side views. Furthermore, the audio is also spatialized, so the voice of each remote person comes from his or her location in the virtual room. The display at each location can be 2D

or 3D, flat or curved, single or multiple, transparent or opaque, and so forth—the possibilities are numerous. In general, the larger the display, the more immersive the user's experience.

System Overview

Our Viewport system uses a camera rig for scene capture at each site. The rig consists of three IR cameras, three color cameras, and two IR projectors (see Figure 2a). The IR projectors are identical to those in the Kinect device, although we did not build the system with multiple Kinect sensors because they cannot be reliably synchronized and their color image quality is poor. We use PointGrey Flea2 as IR and color cameras, synchronized with an external trigger and operating at 1,024 × 768 pixel resolution. For the IR cameras, we removed the IR filter in Flea2 and added a visible light filter (a cheap, black acrylic plastic) to increase contrast. Figure 2b shows some example images captured by the rig. The system also uses a Kinect camera for audio capture and head tracking, although the laser in the Kinect camera is blocked to avoid interference.

Figure 3 shows the overall workflow of our system. At the sender site, a sparse point cloud of the user is first reconstructed from the camera rig. After compression, the point cloud is sent over the network, together with the three color video streams and three binary mask videos obtained from the IR video. At the receiver site, multiple sparse point clouds and video streams are decompressed. The sparse point clouds are interpolated into dense point clouds with a regression algorithm. We also use the binary masks in this step to improve quality. The dense point clouds are then converted into triangular meshes.

Once the system has processed all remote parties' point clouds, the receiver runs a virtual-seating algorithm to arrange the meshes in their correct positions to maintain the conference geometry of a face-to-face meeting. Afterward, all the meshes are rendered to the user for an immersive videoconferencing experience.

We set up a three-site immersive teleconferencing system to test the Viewport system. Each site is equipped with two high-end workstations: one for capturing and sparse-point-cloud generation (the *capture workstation*) and the other for dense-point-cloud generation, meshing, and rendering (the *rendering workstation*). The workstations have dual Intel Xeon Six-Core X5690 CPUs (3.46 GHz), 24 Gbytes of memory, and an NVIDIA GeForce GTX 580 graphics card. At the capture workstation, the sparse-point-cloud reconstruction takes approximately 40 ms. The remaining

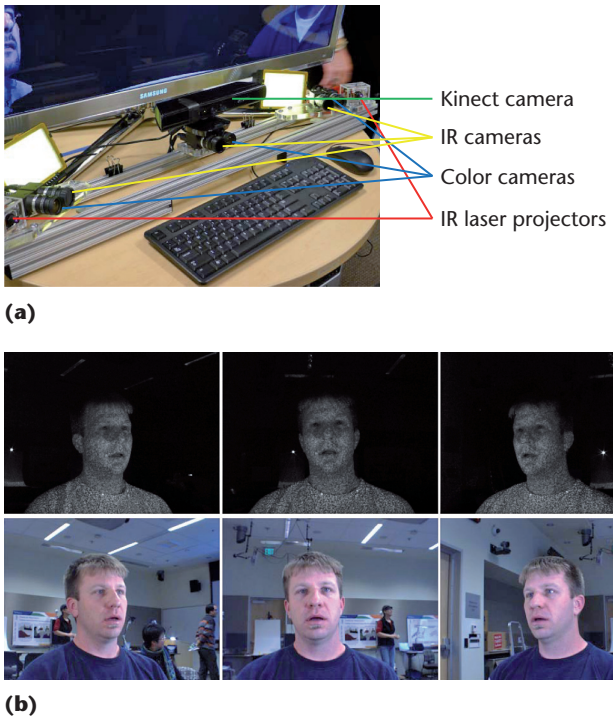

Figure 2. The Viewport camera rig. (a) The configuration of the camera rig. The Kinect camera in this setup handles audio capture and head tracking for motion parallax. The laser in the Kinect camera is disabled to avoid interference. (b) The top three example images are from the infrared (IR) cameras, and the bottom three examples are from the color cameras.

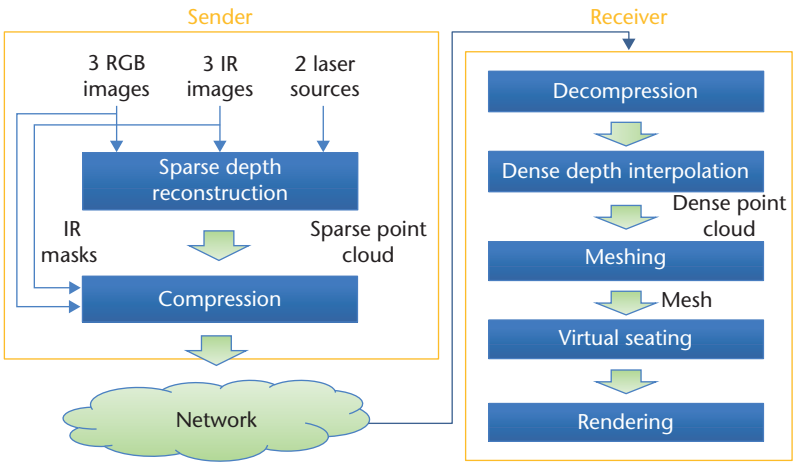

Figure 3. Overview of the Viewport system. The sender site reconstructs a sparse point cloud of the user, and then the receiver site runs a virtual-seating algorithm to arrange the meshes in their correct positions.

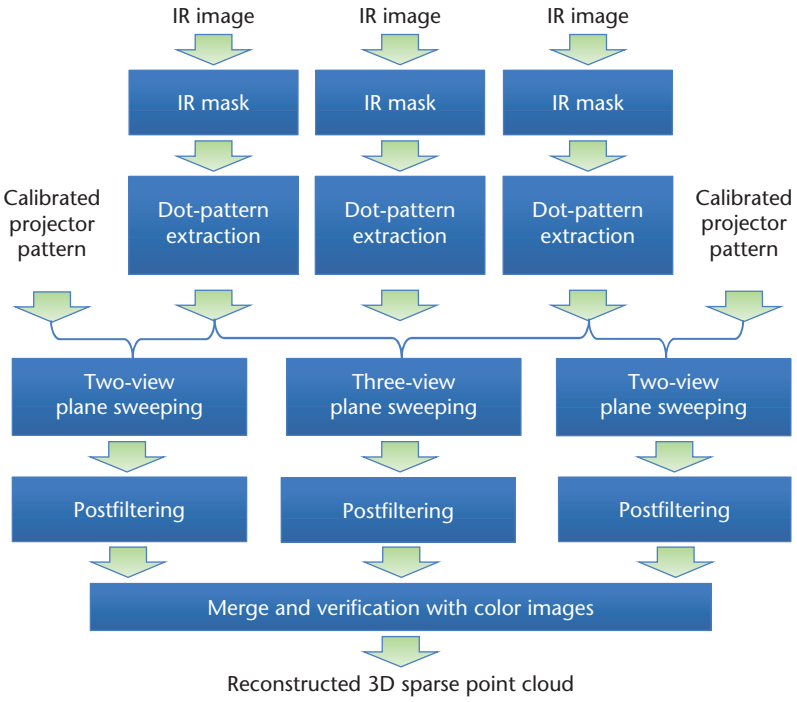

Figure 4. The workflow of the sparse 3D point cloud reconstruction algorithm. After extracting an IR mask of the foreground and the dot patterns in the IR images, multiple plane-sweeping procedures launch, followed by several rounds of post-filtering to remove outliers in the point cloud.

cycles are used for frame grabbing, Bayer pattern demosaicing, and compression of three HD color videos (1,024 × 768), three IR masks, and the sparse point cloud. At the rendering workstation, the dense-point interpolating and meshing take approximately 15 ms for *each* remote party. The remaining cycles are used for audio capture and compression, decoding all compressed streams, and audio and video rendering.

There is no dedicated audio/video synchronization across the two machines, and they both run in a best-effort mode. The overall system runs at approximately 10 to 12 frames per second (fps). All components except mesh rendering are currently implemented on the CPUs only. Implementing these algorithms on the GPU is relatively straightforward, and we expect a much faster frame rate in the near future.

3D Video Processing

After capturing and reconstructing the sparse point cloud of the user in real time, we transmit it along with the three color video streams to all remote parties. This solution is computationally efficient and generates satisfactory 3D rendering results.

Camera Rig Calibration

As Figure 2 shows, the camera rig in our system contains three color and IR camera pairs, spaced approximately 35 cm apart. Because the subject is only 50 to 120 cm from the camera rig, the baseline between cameras is relatively large. Consequently, we found it difficult to reconstruct the full geometry based solely on the cameras. To remedy the problem and minimize the number of cameras, we also calibrate the dot patterns of the laser projectors, giving additional information for reconstructing the side of the face and body.

The first step is to calibrate all the color and IR cameras. For this standard procedure, we adopted Zhang's calibration algorithm[4] to determine all the cameras' intrinsic and extrinsic parameters.

The laser projectors are calibrated using a planar white board (such as a 122 × 91 cm white form board). For each laser projector, we capture two pairs of images of the planar board from two IR cameras. The dot pattern positions in the images are then estimated with subpixel accuracy by fitting a Gaussian to the pixel intensities of a small patch around the dot. Since the two IR cameras are already calibrated, with some manual help,[5] it is possible to reconstruct the planar board's position (and thus, all 3D points) in 3D. Because we capture two pairs of images of the board and we know the homography between all four images, we effectively know a large set of laser beams in 3D. The center of the laser projector can be computed by finding the 3D point that minimizes its distance to all laser beams. Furthermore, we can compute all the light-ray directions of the laser projector.

During 3D reconstruction, we treat the IR projectors as IR cameras. Given the projectors' centers of projection and their light-ray directions, a synthesized 2D IR image is generated by spreading a small Gaussian kernel along the intersection of the light rays and the virtual imaging plane of the IR projectors. These synthesized IR images are combined with the IR and color images captured by the real cameras for sparse-point-cloud reconstruction.

Sparse 3D Point Cloud Reconstruction

Using the camera rig, we could adopt traditional dense multiview-stereo algorithms to reconstruct the user's 3D geometry, but that would be computationally expensive. In our system, we only reconstruct a sparse point cloud to represent the scene geometry. The key idea is to match only the dot patterns in the IR images. Because the dot patterns are sparse, we can perform the task quickly.

Figure 4 illustrates the proposed algorithm's workflow. For each IR image, we first extract an

IR mask that represents the foreground using a simple threshold, as Figure 5a shows. Computing the IR mask has two benefits. First, it improves the reconstruction speed because dot patterns outside the masks belong to the background and can be ignored during reconstruction. Second, the masks are used as a silhouette constraint during dense depth-map interpolation and rendering at the receiver site, which is critical for achieving high-quality rendering.

We then extract the dot patterns in the IR images and use a 15 × 15 pixel patch surrounding each dot as its descriptor. Multiple plane-sweeping procedures are then launched: a two-view plane sweeping between the left most IR camera and the virtual image of the left laser projector, a three-view plane sweeping between all three IR images, and another two-view plane sweeping between the right-most IR camera and the virtual image of the right laser projector. The three IR images are used as references in the three plane-sweeping routines, respectively. Although the IR projectors can interfere with each other near the frontal facing region of the user, the three-view plane sweeping is insensitive to such interference and can help the overall system to perform robustly.

The plane-sweeping algorithm implemented in our system is similar to the traditional scheme,[6] except that it focuses only on the dot patterns. That is, given the reference frame, only the pixels that coincide with the detected dots are used for multiple depth hypothesis testing. Moreover, because a dot in one image must match with a dot in another image, the system can quickly reject many hypothesized depth values, leading to significant speedup. When a hypothesized depth value satisfies the dot-dot-matching criteria, a score is computed based on normalized cross-correlation of the descriptors, which is commonly used in the stereo-matching literature.

After plane sweeping, for each dot in the reference image, we obtain a score vector, with each element corresponding to one hypothesized depth value. The higher the score, the more likely that the scene object resides at that depth. We further apply a filtering approach to encourage nearby dots to have similar depth values. Denoting the score vector of a dot $\mathbf{x}$ as $\mathbf{s}(\mathbf{x})$, we filter the score vector as

$$\mathbf{s}^f(\mathbf{x}) = \sum_{\mathbf{x}_i \in \mathfrak{N}(\mathbf{x})} w_i \mathbf{s}(\mathbf{x}_i) \quad (1)$$

where $\mathfrak{N}(\mathbf{x})$ is a predefined neighborhood of $\mathbf{x}$—for example, $\mathbf{x}$'s k nearest neighbor

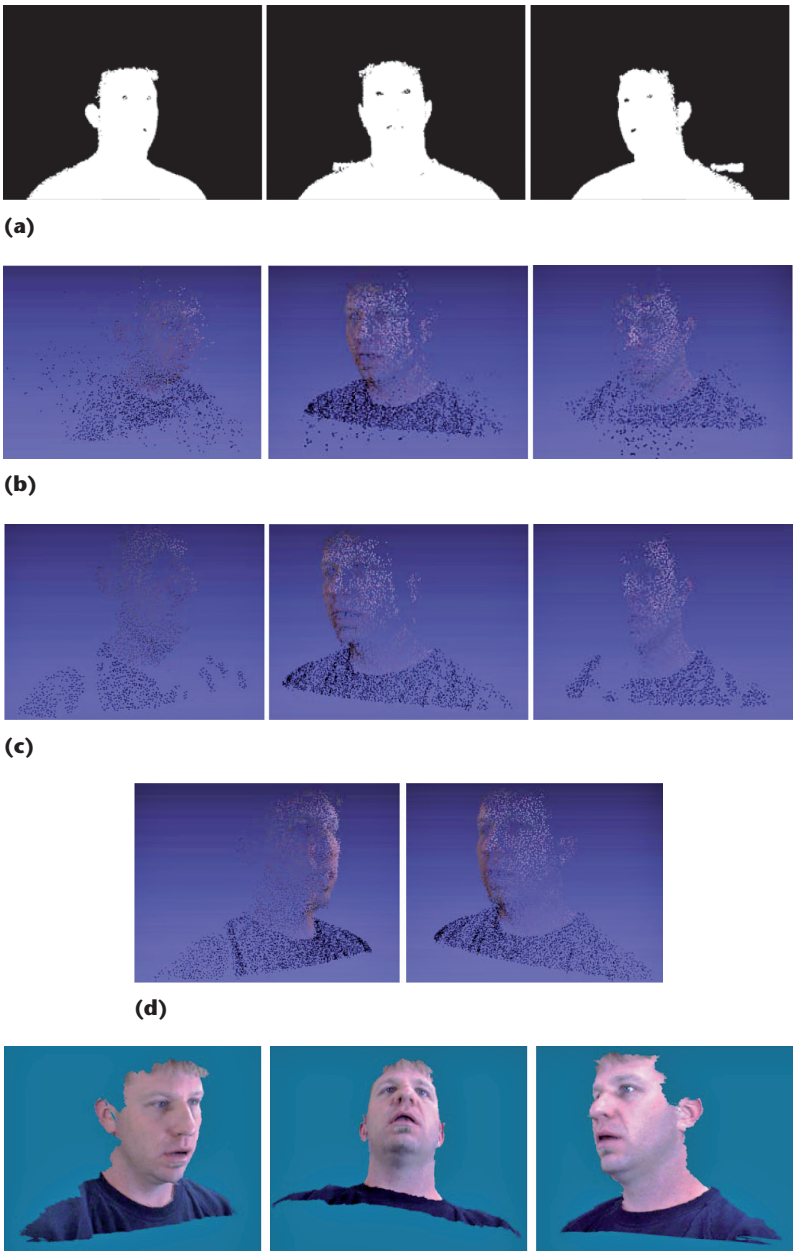

Figure 5. Results illustrating different steps during sparse-point-cloud reconstruction for the input images in Figure 2b. (a) The IR masks. (b) Raw sparse point clouds of the three plane sweeping (see Figure 4), using collaborative filtering in score volume. (c) Post-filtered sparse point clouds. (d) Two views of the merged sparse point cloud. (e) Rendering results from various viewpoints based on the sparse point cloud.

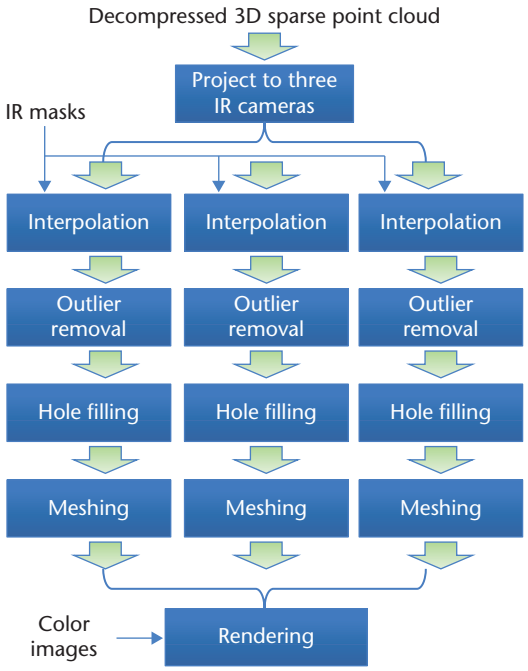

Figure 6. The workflow of the rendering algorithm. To address noises during reconstruction, we perform a regression style interpolation. We then run a gradient fill algorithm to cluster the dense depth maps into connected regions. A basic hole filling algorithm helps remove holes that are disconnected with the background.

(including $\mathbf{x}$ itself) in 2D coordinate in the reference image. The weights w_i are determined by the distance between $\mathbf{x}$ and $\mathbf{x}_i$. For example,

$$w_i = \exp\left\{-\frac{\|\mathbf{x}_i - \mathbf{x}\|^2}{2\sigma^2}\right\} \quad (2)$$

where σ is a constant. The depth value corresponding to the highest element in $\mathbf{s}^f(\mathbf{x})$ is chosen as the output (see Figure 5b).

We then conduct another round of postfiltering to remove outliers in the point cloud. Given a dot $\mathbf{x}$ and its k nearest neighbors $\mathfrak{N}(\mathbf{x})$, we require that a certain percentage of the dots in the neighborhood have depth values similar to the depth of $\mathbf{x}$. Otherwise, the dot will be excluded from the final output. This post-filtering scheme is effective, as Figure 5c shows. Finally, the three point clouds are rotated and translated into the center IR camera's coordinate system for merging. During the process, the 3D points are also projected onto the three color images to make sure they are consistent in color projection.[5] Figure 5d shows the merged point cloud.

For a typical scene, we reconstruct 8,000 to 12,000 sparse points. Each point is also associated with three binary flags, indicating which images were used to obtain the highest score. These flags record visibility information about the sparse points, which is helpful for rendering. Together with the binary flags, the 3D points are quantized and transmitted to the remote sites for rendering (see Figure 5e).

3D Video Rendering

The rendering workstation receives multiple sparse point clouds and their related color videos and IR masks. The goal is to render them with high quality and be consistent in seating geometry with face-to-face meetings. For this purpose, we need to generate a 3D mesh from the input sparse point cloud. The most straightforward way of performing meshing is to use Delaunary triangulation on the sparse point cloud directly. However, due to noises during the reconstruction, such a scheme does not produce satisfactory results. To address this, we propose the rendering algorithm in Figure 6.

The sparse point cloud is first projected to the three IR cameras, resulting in a sparse 2D point set on each image. We use each point's binary flags to make sure only the visible IR cameras receive the corresponding points. For each projected image j, $\mathbf{y}_i$ denotes the sparse projections; $i = 1, \ldots, K_j$, where K_j is the number of visible IR dots in the jth image; and their depth values as $d(\mathbf{y}_i)$. For each valid pixel y inside the IR mask, we perform a regression style interpolation:

$$d(\mathbf{y}) = \sum_{\mathbf{y}_i \in \mathfrak{N}(\mathbf{y})} \alpha_i d(\mathbf{y}_i) \quad (3)$$

where $\mathfrak{N}(\mathbf{y})$ represents a 2D neighborhood of pixel $\mathbf{y}$. The weights α_i are related to the distance $\mathbf{y}_i - \mathbf{y}$:

$$\alpha_i \propto \exp\left\{-\frac{\|\mathbf{y}_i - \mathbf{y}\|^2}{2\sigma^2}\right\} \quad (4)$$

and $\Sigma_i \alpha_i = 1$. The IR masks are also used in this step such that no depth value is created outside the masks. Figure 7e shows the resulting dense depth maps. Because the sparse point cloud is noisy for this particular frame, the interpolated dense depth maps contain blobs of outliers that need to be processed

before meshing. We run a gradient fill algorithm to cluster the dense depth maps into connected regions. Each region's area is compared against a threshold. If the area is too small, the region will be removed from the dense depth map. Afterward, we apply a basic hole filling algorithm to remove holes that are disconnected with the background.

The processed dense depth map in Figure 7f shows that the quality of the dense depth maps has been significantly improved. The dense depth maps are then converted into triangular meshes via a prefixed scan pattern and rendered accordingly (see Figure 7g). Triangles with large depth gaps are removed during the process.

Virtual Seating

Although many existing systems have attempted to reconstruct the 3D geometry of the meeting participants and place them in a common virtual environment, few have been able to faithfully maintain the seating geometry, leading to poor mutual gaze during the meeting. Indeed, careful calibration of the camera rig, the monitor, and the table surface is necessary to achieve such a goal.

Figure 8a illustrates the basic idea of virtual seating for a three-party meeting. The reconstructed point clouds are in their respective camera rig coordinate systems, denoted as C^{R_0}, C^{R_1}, and C^{R_2}. To place them around a virtual table in the virtual environment, let the virtual world's coordinate be C^W, and the rotation and translation from rig k's coordinate system to the virtual world's coordinate system be $\mathbf{R}^{R_k W}$ and $\mathbf{t}^{R_k W}$. Given a 3D point $\mathbf{X}_i^k$ in C^{R_k}, where i is the index of the points in the point cloud reconstructed from the kth camera rig, we have

$$\mathbf{X}_{ki}^W = \mathbf{R}^{R_k W}\mathbf{X}_i^k + \mathbf{t}^{R_k W} \quad (5)$$

and the point will be placed in the correct location in the virtual world.

During rendering, given the viewer's eye location in the virtual world's coordinate system and the screen's display region in the same coordinate system, our system will render all point clouds or meshes correctly. Note the viewer's eye location is tracked to provide a sense of motion parallax, which is an important effect for immersive teleconferencing.[7] Because Viewport conducts capture and rendering

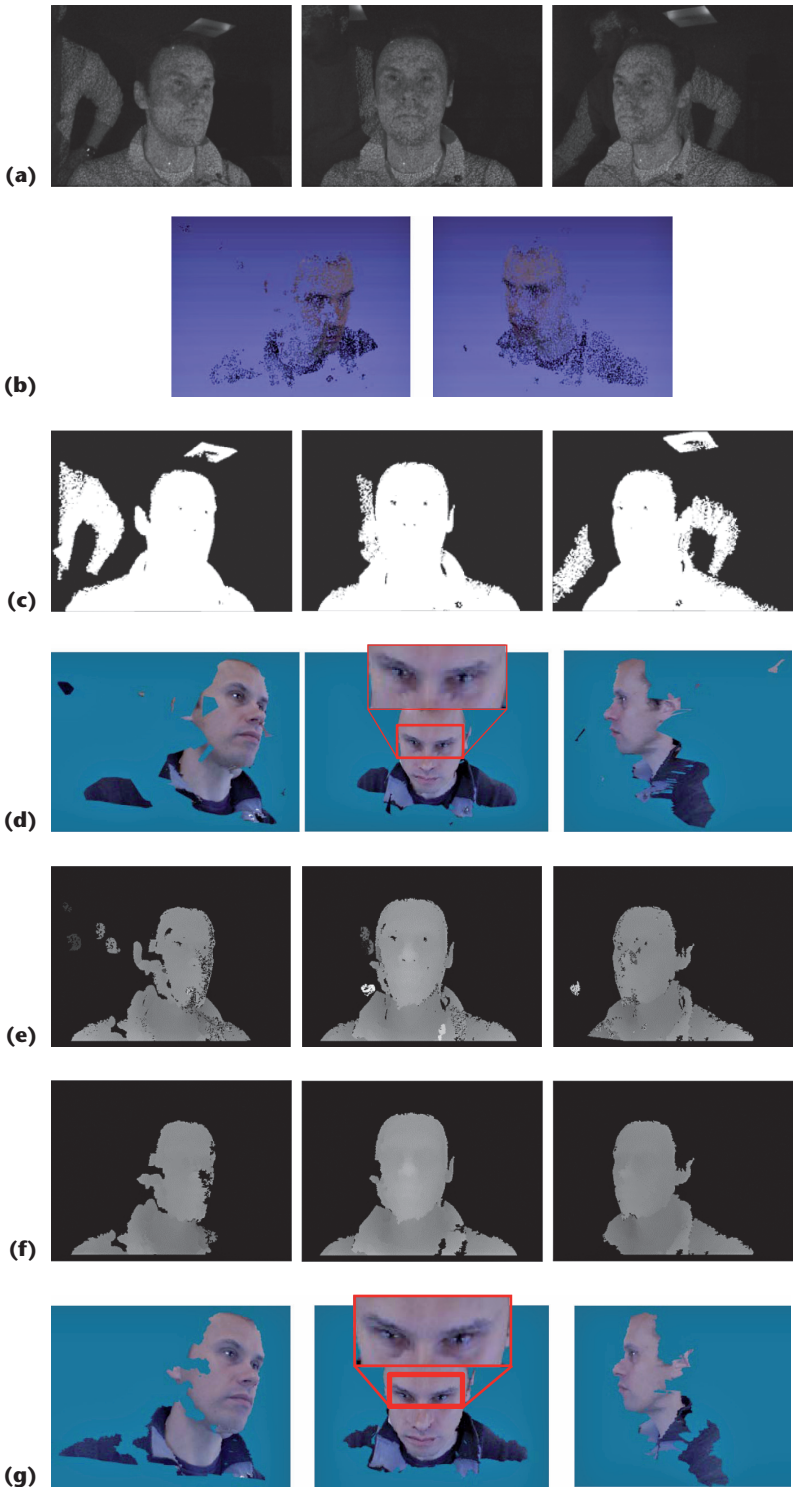

Figure 7. Results illustrating different steps during rendering. (a) Input IR images. (b) Reconstructed sparse point cloud viewed from two different viewpoints. (c) IR masks. (d) Meshing on the sparse point cloud using Delaunay triangulation. Edges longer than a threshold are discarded to avoid artifacts. (Note the resultant mesh is noisy and contains outliers.) (e) Dense depth map after interpolation. (f) Dense depth map after outlier removal and hole filling. (g) Rendering results after meshing the dense depth maps.

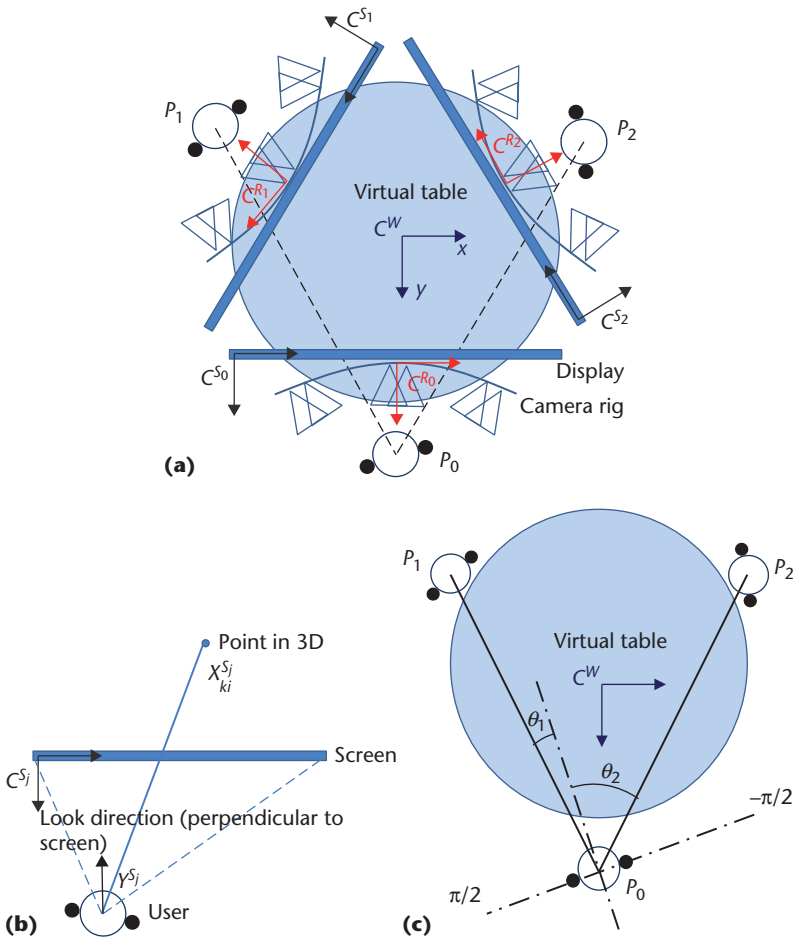

Figure 8. Illustration of virtual seating and its rendering. (a) Virtual seating. (b) Virtual view rendering in the screen coordinate. (c) 3D spatial audio rendering.

on two separate workstations, we added a Kinect camera on top of the rig (see Figure 2) to perform head tracking (using the Kinect for Windows SDK) and audio capture at the rendering workstation. Consequently, we must also calibrate the Kinect camera with respect to the virtual world.

For the *j*th user, denote C^{S_j} as local coordinate system of the screen, and C^{C_j} as local coordinate system of the Kinect tracking camera. Let the user's eye position with respect to C^{C_j} be $\mathbf{Y}^{C_j}$. We can transform it to world coordinates as

$$\mathbf{Y}_j^W = \mathbf{R}^{C_jW}\mathbf{Y}^{C_j} + \mathbf{t}^{C_jW} \tag{6}$$

where $\mathbf{R}^{C_jW}$ and $\mathbf{t}^{C_jW}$ are the rotation and translation from the Kinect's local coordinate system to the virtual world's coordinate system. To perform rendering, it is more convenient to compute everything in the screen coordinate system. We can easily transform the point cloud and viewer position to the screen coordinate system C^{S_j} using

$$\mathbf{X}_{ki}^{S_j} = \left(\mathbf{R}^{S_jW}\right)^{-1}\left(\mathbf{X}_{ki}^W - \mathbf{t}^{S_jW}\right) \tag{7}$$

$$\mathbf{Y}^{S_j} = \left(\mathbf{R}^{S_jW}\right)^{-1}\left(\mathbf{Y}_j^W - \mathbf{t}^{S_jW}\right) \tag{8}$$

where $\mathbf{R}^{S_jW}$ and $\mathbf{t}^{S_jW}$ are the rotation and translation from the screen's local coordinates to the virtual world's coordinate. Once in screen coordinates, the final rendering is simple (see Figure 8b). The renderer's look direction should be the same as the normal direction of the screen.

To perform virtual seating accurately, we need the calibration information for the following:

- the transforms from the camera rig to the virtual world coordinate $\mathbf{R}^{R_kW}$ and $\mathbf{t}^{R_kW}$.
- the transforms from the tracking camera to the virtual world coordinate $\mathbf{R}^{C_jW}$ and $\mathbf{t}^{C_jW}$.
- the transforms from the screen to the virtual world coordinate $\mathbf{R}^{S_jW}$ and $\mathbf{t}^{S_jW}$.

To this end, we introduce an auxiliary planar pattern and an external camera to help register the cameras and the screen. We carved a pattern with several hollow squares out of an acrylic board, which can be seen from both sides by the camera rig, the tracking camera, and the external camera that is facing the screen.[15] We then positioned the pattern at the edge of the participants' table, roughly 80 cm from the screen. This arrangement helps properly map everyone around the table because it automatically aligns all table tops to the same height when mapping to the virtual world (more details on this later). We define a local coordinate system on the pattern, denoted as C^{P_j}. We can compute the transforms from the camera rig to the pattern coordinates as $\mathbf{R}^{R_kP_k}$ and $\mathbf{t}^{R_kP_k}$ and the transforms from the tracking camera to the pattern coordinates as $\mathbf{R}^{C_jP_j}$ and $\mathbf{t}^{C_jP_j}$. With the external camera, we also capture multiple images that include both the pattern and the screen, thus obtaining the transforms from the screen to the pattern coordinates as $\mathbf{R}^{S_jP_j}$ and $\mathbf{t}^{S_jP_j}$. If we can find the relationship between the patterns'

(a)

(b)

Figure 9. Comparison between dense multiview stereo and the proposed scheme. (a) Rendering results from dense multiview stereo. (b) Rendering results from the proposed scheme.

coordinates and the virtual world coordinate, it becomes easy to compute all the transforms we mentioned earlier.

Not surprisingly, the transforms between the patterns' coordinates and the virtual world can be defined based on the face-to-face meeting geometry. For instance, for a three-way distributed teleconference like the one in Figure 8a, the three transforms can be defined as

$$\mathbf{R}^{P_jW} = \begin{bmatrix} \cos(\theta_j) & 0 & \sin(\theta_j) \\ 0 & 1 & 0 \\ -\sin(\theta_j) & 0 & \cos(\theta_j) \end{bmatrix}, \quad \mathbf{t}^{P_jW} = \begin{bmatrix} r\sin(\theta_j) \\ 0 \\ r\cos(\theta_j) \end{bmatrix} \tag{9}$$

where $\theta_0 = 0$, $\theta_1 = -(2\pi/3)$, and $\theta_2 = (2\pi/3)$. r is the distance from the center of the virtual table to the patterns, which is determined by the virtual table's size. We assign zero translation between the patterns and the virtual world in y axis so all table tops will be aligned to the same level in the virtual world.

3D Audio Processing

An immersive teleconference would not be possible without immersive audio. All the participants' positions are computing during virtual seating (see Figure 8c). Given the local user's head orientation and location in the virtual world, we can easily compute the relative angle θ_i between his or her look direction and the remote participants. As a result, the remote participants' audio can be spatialized to the correct locations for generating faithful immersive audio.

If the user is using headphones, the spatialization task is relatively easy; we convolve the audio with an appropriate head-related transform function (HRTF). Otherwise, additional steps are necessary. For example, on the capture site, to get the true speaker signal, we need to use acoustic echo cancelation (AEC) followed by dereverberation. On the playback site, if the loudspeakers are close to the user's ears, we can simply use the spatialized audio as is done for the headphones. If the speakers are reasonably far from the listener, we might have to compensate for speaker cross-talk and the room impulse response from the loudspeakers to the listener.[8]

For the current implementation, we use the scheme we described in our earlier work,[9] where spatialization is naturally combined with multichannel AEC using constrained Kalman filtering.

Experimental Results

Figure 9 compares the proposed sparse point reconstruction and interpolation-based rendering scheme with a dense multiview-stereo algorithm we implemented earlier. The dense stereo algorithm also performs three reconstructions using the three IR images as a reference and involves all the color and IR cameras and calibrated laser patterns. We combined the scores from different modalities with the robust voting scheme presented in earlier work.[10] We also implemented the combined depth and outlier estimation approach proposed by Christoph Strecha, Rik Fransens, and Luc J. Van Gool to improve the quality of the final dense depth map.[11] The dense multiview-stereo algorithm takes approximately 40 to 45 seconds to reconstruct one frame, while our proposed method takes only 40 ms for reconstruction and an additional 15 ms to render. Thus, we achieved almost

three orders of magnitude speedup with the proposed method.

The rendering quality of the two schemes is comparable. The dense multiview-stereo scheme does better in some aspects, such as eyeglasses, and has fewer holes. On the other hand, due to the low-resolution of the sparse point cloud, the proposed method generates a smoother mesh and has fewer outliers. The outlier-filtering scheme we described earlier is aggressive, leading to more holes in the final rendering results. We hope some of these holes will be filled once we include the color images for matching score computation in our future implementation.

Regarding the network bandwidth required for our system, we currently transmit three HD color videos at 1,024 × 768 pixel resolution, three IR binary masks at 512 × 384 pixels resolution, and 8,000 to 12,000 sparse points. At 10 fps, with rudimentary compression of the sparse point clouds using quantization and run-length coding for the binary masks, the total bandwidth is approximately 6 Mbps for the videos and 4 Mbps for the rest. We are currently exploring better schemes for compressing the sparse point clouds.

We demonstrated the three-way Viewport system at a Microsoft internal event with hundreds of visitors to our booth, and the general feedback was positive. (A demo video of the system is available at http://research.microsoft.com/apps/video/default.aspx?id=169617). People found it had perfect gaze awareness and were able to tell easily whether a remote party was paying attention to them. At one of the stations, we used a 3D display to show that the rendering algorithm can also output stereoscopic video. Visitors enjoyed the 3D conferencing experience and commented that although the rendering still has artifacts the artifacts were not as disturbing on a 3D display as on a 2D display.

Future Work

There is still much room for improvement in Viewport. In particular, the sparse-point-cloud-based reconstruction algorithm does not perform well on hair due to poor IR reflection or on thin objects due to smoothing. In addition, when the requested virtual viewpoint is far from the camera's capturing viewpoint, the system still cannot guarantee the rendering quality.

Another interesting research issue is whether we can apply the presented virtual-seating algorithm on small screens, where faithful seating geometry could cause the remote participants to be rendered outside the display. We are exploring schemes to distort the seating geometry and still maintain the mutual gaze by tracking the users' head orientations. **MM**

References

1. A.W. Davis and I.R. Weinstein, ''Telepresence 2007: Taking Videoconferencing to the Next Frontier,'' revision 2.0, Wainhouse Research, 2007; www.wrplatinum.com/content.aspx?CID=6749.
2. M. Argyle and M. Cook, *Gaze and Mutual Gaze,* Cambridge Univ. Press, 1976.
3. D. Nguyen and J. Canny, ''Multiview: Spatial Faithful Group Video Conferencing,'' *Proc. ACM SIGCHI Conf. Human Factors in Computing Systems* (CHI), ACM, 2005, pp. 799–808.
4. Z. Zhang, ''A Flexible New Technique for Camera Calibration,'' *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 11, 2000, pp. 1330–1334.
5. C. Zhang et al., ''Viewport: A Fully Distributed Immersive Teleconferencing System with Infrared Dot Pattern,'' tech. report MSR-TR-2012-60, Microsoft Research, 2012.
6. R.T. Collins, ''A Space-Sweep Approach to True Multi-image Matching,'' *Proc. IEEE Computer Vision and Pattern Recognition* (CVPR), IEEE CS, 1996, p. 358.
7. C. Zhang, D. Florencio, and Z. Zhang, ''Improving Immersive Experiences in Telecommunication with Motion Parallax,'' *IEEE Signal Processing Magazine,* vol. 28, no. 1, 2011, pp. 139–144.
8. M.-S. Song et al., ''An Interactive 3D Audio System with Loudspeakers,'' *IEEE Trans. Multimedia,* vol. 13, no. 5, 2011, pp. 844–855.
9. Z. Zhang, Q. Cai, and J. Stokes, ''Multichannel Acoustic Echo Cancelation in Multiparty Spatial Audio Conferencing with Constrained Kalman Filtering,'' *Proc. Int'l Workshop on Acoustic Echo and Noise Control* (IWAENC), 2008; www.iwaenc.org/proceedings/2008/contents/papers/9059.pdf.
10. G. Vogiatzis et al., ''Multi-view Stereo via Volumetric Graph-Cuts and Occlusion Robust Photo-Consistency,'' *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 29, no. 12, 2007, pp. 2241–2246.
11. C. Strecha, R. Fransens, and L.V. Gool, ''Combined Depth and Outlier Estimation in Multi-view

Stereo,'' *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (CVPR), IEEE CS, 2006, pp. 2394–2401.

Cha Zhang is a researcher in the Multimedia, Interaction and Communication Group at Microsoft Research. His research interests include multimedia, computer vision, and machine learning. Zhang has a PhD in electrical and computer engineering from Carnegie Mellon University. Contact him at chazhang@microsoft.com.

Qin Cai is a senior research engineer in the Multimedia, Interaction and Communication group at Microsoft Research. Her research interests focus on creating immersive human-to-computer and human-to-human interaction using multimedia. Cai has a PhD in computer engineering from the University of Texas at Austin. Contact her at qincai@microsoft.com.

Philip A. Chou is a principal researcher at Microsoft Research. His research interests include telepresence and immersive communication. Chou has a PhD in electrical engineering and computer science from Stanford University. He is a member of the IEEE Signal Processing Society Board of Governors and a fellow of IEEE. Contact him at pachou@microsoft.com.

Zhengyou Zhang is a research manager and principal researcher with Microsoft Research. His research interests include computer vision, audio processing, multimedia, and biologically inspired machine learning. Zhang has a PhD in computer science from the University of Paris XI and a doctor of science (habilitation) from the University of Paris XI. He is a fellow of IEEE. Contact him at zhang@microsoft.com.

Ricardo Martin-Brualla is a graduate student at the University of Washington. His research interests include 3D reconstruction and visualization. Ricardo has a BS in computer science and a B. Math from the Technical University of Catalonia. Contact him at rmartin@cs.washington.edu.