Enhanced Personal Autostereoscopic Telepresence System using Commodity Depth Cameras

Andrew Maimone*, Jonathan Bidwell, Kun Peng, Henry Fuchs

Department of Computer Science, University of North Carolina at Chapel Hill, CB# 3175, Brooks Computer Science Building, 201 S Columbia St., Chapel Hill, NC 27599-3175, United States

Abstract

This paper describes an enhanced telepresence system that offers fully dynamic, real-time 3D scene capture and continuous-viewpoint, head-tracked stereo 3D display without requiring the user to wear any tracking or viewing apparatus. We present a complete software and hardware framework for implementing the system, which is based on an array of commodity Microsoft Kinect™ color-plus-depth cameras. Contributions include an algorithm for merging data between multiple depth cameras and techniques for automatic color calibration and preserving stereo quality even with low rendering rates. Also presented is a solution to the problem of interference that occurs between Kinect cameras with overlapping views. Emphasis is placed on a fully GPU-accelerated data processing and rendering pipeline that can apply hole filling, smoothing, data merger, surface generation, and color correction at rates of up to 200 million triangles/sec on a single PC and graphics board. Also presented is a Kinect-based markerless tracking system that combines 2D eye recognition with depth information to allow head-tracked stereo views to be rendered for a parallax barrier autostereoscopic display. Enhancements in calibration, filtering, and data merger were made to improve image quality over a previous version of the system.

Keywords: teleconferencing, sensor fusion, camera calibration, color calibration, filtering, tracking

1. Introduction

A long-standing goal [1, 2] of telepresence has been to unite distant workspaces through a shared virtual window, allowing remote collaborators to see into each other's environments as if these were extensions of their own.

In 2002, UNC/UPenn researchers created an early realization of this goal by combining a static 3D model of an office with near-real-time 3D acquisition of a remote user and displayed the result in head-tracked stereo at interactive rates. Since then, several improved 3D capture and display systems have been introduced. In 2004, the MERL 3DTV [3] system offered a glasses and tracker-free capture and display system using an array of 16 cameras and a lenticular autostereo display. However, framerate was low (12 Hz) and the number of viewing zones was limited and repeating. In 2008, the Fraunhofer Institute and the Heinrich-Hertz Institute introduced 3DPresence [4], an improved lenticular-display based system. The system supported multiple views for several participants seated around a table, but like in the MERL system, the number of views was limited and only horizontal parallax was available. In 2009, USC ICT researchers presented a telepresence system [5] that used structured light for 3D acquisition and a volumetric 3D display. The system provided real-time capture, nearly continuous points of view and required no tracking markers or glasses, but capture and display were limited to a head-size volume. In 2010, Holografika introduced a compelling system [6] consisting of a large array of projectors and cameras offering fully dynamic real-time 3D capture and tracker-less autostereo display. The system, however, featured only a moderate capture rate (10-15 Hz) and did not offer fully continuous points of view – interpolation was performed between a linear array of densely placed 2D cameras and only horizontal parallax was provided. Featuring 27 cameras, 3 PCs, and scores of projectors, it was also a very expensive system to build. In 2011, the FreeCam system [7] demonstrated high quality 3D acquisition using a pair of depth cameras, but capture was limited to users segmented from the background. Also noteworthy are a group of systems [8, 9, 10, 11] with the alternate goal of placing users in a shared virtual space rather than capturing and presenting users within their own physical environments.

In [12], the authors presented a telepresence system that aimed to overcome some of the limitations of previous systems and is the basis for this updated work. The system offered fully dynamic scene capture – presenting a live view of remote users as well as their environments and allowing users to enhance communication by utilizing surrounding objects. Continuous viewpoints were supported, allowing users to look around a remote scene from exactly the perspective corresponding to their head position, rather than from a single or set of fixed vantages. This granted users the ability to see around obstructions and gain more information about the remote scene. Gaze was preserved, allowing participants to make eye contact; re-

Email addresses: maimone@cs.unc.edu (Andrew Maimone), bidwej@cs.unc.edu (Jonathan Bidwell), pengkun@cs.unc.edu (Kun Peng), fuchs@cs.unc.edu (Henry Fuchs)

Figure 1: Three views of a live capture session.

Figure 2: Two users in 3D scene.

search [13] has shown the absence of correct gaze can cause a loss of nonverbal communication. Stereo views were provided, which have been shown to increase the sense of shared presence [13]. Finally, tracking and viewing apparatuses were eliminated – 3D glasses obstruct eye contact between participants and shutter glasses have been found to be "disruptive" to over 90% of users [13]. We believe this system was the first to incorporate all of these characteristics – fully dynamic 3D scene capture, continuous look-around ability with full parallax, gaze preservation, and stereo display without the use of any encumbrances – but suffered from mediocre image quality. In this paper we present a revised system which includes several enhancements that result in improved 3D reconstruction.

2. Background and Contributions

Our system is based on the Microsoft Kinect™ sensor, a widely available, inexpensive ($150) device that provides color image, infrared image, depth map, and audio capture. Depth data is acquired using imperceptible structured light techniques; a static dot pattern projected with an IR laser is captured with an IR camera and compared to a known pattern [14]. Depth images are provided at 640×480 resolution at 30 Hz; color and IR images may be captured at this resolution and rate or at 1280×1024 and approximately 10 Hz. The unit provides a $58^\circ \times 45^\circ$ field of view and a depth accuracy rated[1] as 1 cm at 1 m, with a 0.8 m to 3.5 m range[2].

Utilizing several strategically placed and calibrated Kinect sensors, an entire room-sized scene can be captured in real-time. The scene can be rendered from exactly the remote user's perspective, providing for correct gaze and continuous viewpoints. Eye position tracking is required to provide for continuous viewpoints; 2D eye detection combined with the Kinect's depth data provides a markerless tracking solution.

However, as a device not designed for general purpose 3D scene capture or tracking, the Kinect presents some challenges for our intended purposes. Since each sensor projects a fixed structured light pattern at roughly the same wavelength, inter-unit interference is a major problem. The device, as controlled with the drivers currently available, provides auto-white balance and exposure that cannot be disabled, presenting difficultly for seamlessly integrating color-matched data between cameras. The capture frame rate, 30 Hz, is suitable for scene acquisition but is inadequate for responsive tracking.

In [12], we presented solutions to these challenges and introduced an eye position tracking system based on the Kinect that was used with an autostereo display. Specific contributions were as follows:

1. A software solution to the Kinect interference problem that provides hole filling and smoothing
2. A visibility-based algorithm to merge data between cameras
3. A visibility-based method for dynamic color matching between color-plus-depth cameras
4. A system for combining 2D eye recognition with depth data to provide 3D eye position tracking
5. A technique for preserving high-quality head-tracked stereo viewing on fixed parallax barrier displays even at low rendering rates

Also presented was a GPU-accelerated software framework that implemented hole filling, smoothing, data merging, sur-

[1] `http://www.primesense.com/`

[2] We observed that our units return depth readings for surfaces as near as 0.5 m.

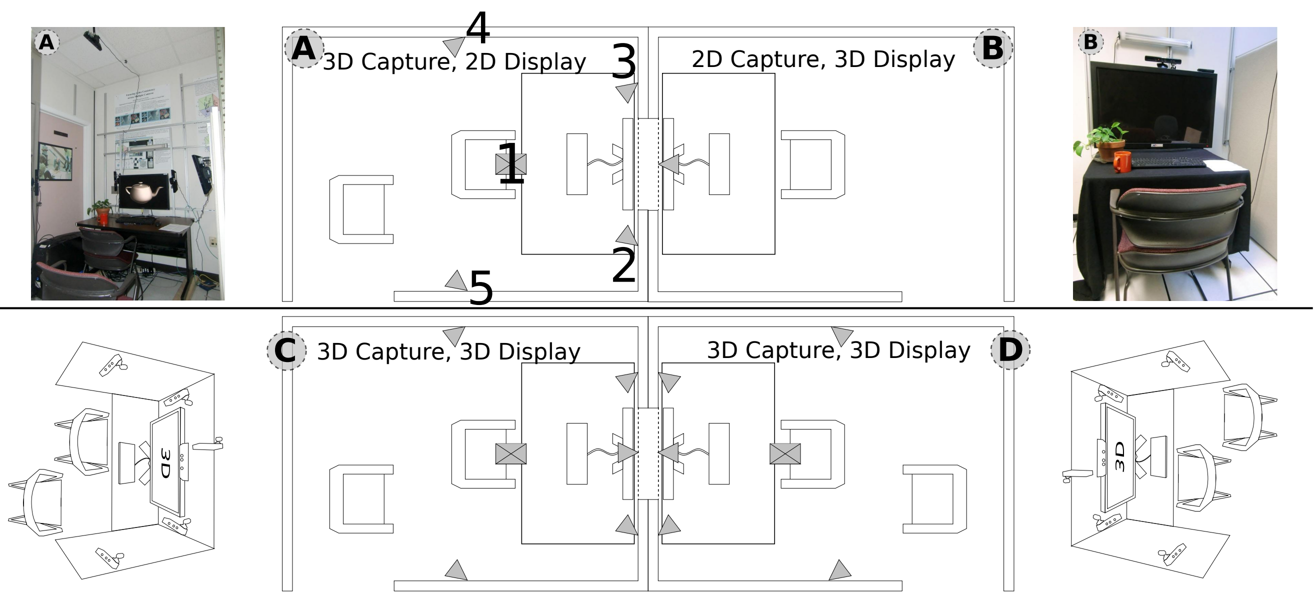

Figure 3: System Layout. Top: Demonstrated proof-of-concept system configuration. Bottom: Proposed ideal configuration.

face generation, and color correction at interactive rates for five depth cameras on a single PC and graphics board.

In this work, we have revised several aspects of the system:

1. Calibration procedures were enhanced and a new procedure was established to correct biases in the Kinect's depth measurements
2. A smoothing procedure was added to the rendering pipeline to suppress depth noise spatially and temporally
3. The data merger algorithm was updated to be more tolerant of geometric inaccuracies.

These enhancements are designed to improve the 3D reconstruction quality of the system. In the remainder of this paper, we present a complete framework for the system (based on [12]) with the improvements applied.

3. System Overview

3.1. Physical Layout

Figure 3 shows the layout of our system. The two spaces are physically separated, but a view of the other space can be seen through the display as if the spaces were aligned with a shared hole in the wall. The bottom of the figure shows our "ideal" configuration – 3D capture and 3D display are supported on both sides (spaces C,D).

The top of Figure 3 shows the actual configuration used for our proof-of-concept system. The system utilizes two office cubicles (approximately 1.9 m × 2.4 m). Space A offers 3D capture and 2D display of space B, while space B features 2D capture and head-tracked 3D display of space A. This configuration allowed us to demonstrate 3D capture, 3D display, and eye gaze preservation (for one side) while requiring only the single autostereo display that we had available.

3.2. Hardware Configuration

Both spaces in our proof-of-concept system share a single PC with a 4-core Intel Core i7-960 CPU, 6GB of RAM and an Nvidia GeForce GTX 580 graphics board. Six Microsoft Kinect sensors are connected to the PC. The 2D display side features a 30 in LCD monitor, while the 3D display side uses a 40 in X3D Technologies autostereo display.

We avoided networking and audio in our proof-of-concept system since both spaces are run from a single PC and are in close proximity. We plan to address these omissions in a future system.

3.3. Software Overview

Operating System and APIs. Our test system runs on 64-bit Linux (Ubuntu 10.10) and uses the OpenNI[3] API along with a Kinect driver[4] to communicate with the sensors. OpenGL is used for rendering, the OpenGL shader language (GLSL) is used for programmable GPU operations, and GLUT is used for windowing and user input. The OpenCV[5] computer vision library was utilized for camera calibration and tracking.

Data Processing and Rendering Pipeline. The following rendering pipeline is used in our system (Figure 4):

1. When new data is available, read color and depth images from Kinect units and upload to GPU.
2. Smooth and fill holes in depth image.
3. For each Kinect's data, form triangle mesh using depth data.

[3] `http://www.openni.org/`
[4] `https://github.com/avin2/SensorKinect`
[5] `http://opencv.willowgarage.com/`

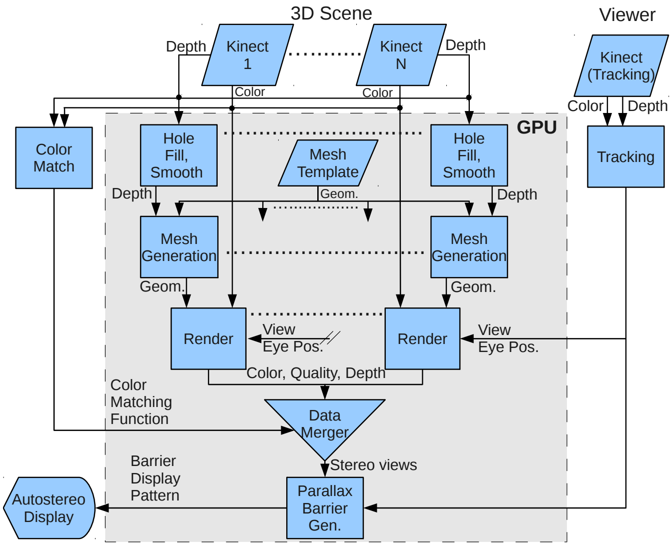

Figure 4: Data Processing and Rendering Pipeline

4. For each Kinect's data, apply color texture to triangle mesh and estimate quality at each rendered pixel; render from the tracked user's current position, saving color, quality, and depth values.
5. Merge data for all Kinect units using saved color, quality and depth information.
6. Repeat steps 3-5 for other eye's view.
7. Assemble the two stereo viewpoints into pattern required by autostereo display, draw to screen.
8. While next 3D scene is being generated, periodically redraw pattern required by autostereo display using the last rendered frame and the new estimated eye position.

GPU Acceleration Overview. To maximize performance, all real-time graphics-related data processing algorithms are performed on the GPU (using the OpenGL Shader Language) to reduce CPU/GPU memory transfer overhead and take advantage of GPU parallelism. CPU/GPU memory transfers are kept to a minimum: the five Kinects' color and depth images are uploaded to the GPU, but no other major data transfers take place.

System Component Rates. Since our entire system does not run at the desirable rate of ≥60 Hz on our current hardware with all graphical enhancements applied, we allow three rates in our system to run asynchronously to allow for interactive rendering and high stereo quality. The first is the *reconstruction rate*, the rate at which new data is incorporated into the rendered scene, which involves uploading new color and depth data to the GPU, hole filling, smoothing, and surface generation. The second rate is the *rendering rate*, the pace at which the scene is rendered from a new viewing perspective. In our system, this also includes our data merger algorithm, which is visibility-based. The final rate is the *parallax barrier pattern generation rate*, the rate at which new patterns are generated for our autostereo display from new estimated eye positions.

The independence of the reconstruction rate from the rendering rate helps to keep the system running at interactive rates as more cameras are added to the system; a study by Meehan [15] found a positive correlation between framerate and sense of presence as the former was increased from 10 to 30 fps. The independent parallax barrier pattern generation rate preserves stereo quality during head motion even if the rendering rate decreases. (See Section 5.3.)

Tracking. We combine 2D eye detection, depth data, and motion tracking to create an unencumbered 3D eye position tracker. Initial eye detection is performed on a color image and eyes are then tracked using pattern matching. Once the 2D eye position is obtained, Kinect depth data is used to transform the position into 3D. A Kalman filter is used to improve accuracy and interpolate the position of the eyes between sensor updates.

4. Implementation

4.1. Camera Placement, Calibration, and Error Measurement

Camera Placement. When placing cameras as shown in the top left of Figure 3, the following factors were considered:

1. Coverage: for our application, coverage is only necessary for surfaces that can be seen by the remote user.
2. Redundancy: Redundant coverage allows preservation of surfaces that are occluded from the viewpoint of a single depth camera, but are still visible by the remote user (e.g. an occluded chest behind a raised hand).
3. Resolution and Accuracy: the resolution available varies with angle and distance (discussion in Section 4.5).
4. Kinect Depth Range: approximately 0.5 m-3.5 m.
5. Kinect Depth Interference: discussion in Section 4.2.
6. Kinect Depth Error: perceived depth error is reduced if camera is near line of sight of user

Camera Calibration. To calibrate the Kinect sensors, we used the OpenCV camera calibration routines, which are based on Zhang's method [16]. The routines compute camera intrinsic parameters (focal length, center of projection, radial and tangential distortion coefficients) from two or more images of a detected planar pattern, taken from different orientations. Extrinsic parameters (relative positions and orientations) between two cameras were computed using one or more pairs of images of a detected pattern seen from each camera, along with the intrinsic parameters. Since the Kinect driver is able to register the depth image to the color image, only calibration of the color camera is necessary.

For our test system, camera intrinsics and extrinsics were computed using detection of a checkerboard target. For extrinsic computation, we calibrated each camera to a master ceiling-mounted camera, whose viewing frustum conveniently overlaps the frustum of each of the other cameras.

In addition to the calibration procedures of our previous system [12], we incorporated some of the enhanced calibration procedures of our larger scale 3D telepresence system [17]. To reduce calibration error introduced by the unsynchronized

Kinect cameras and motion blur, we placed the calibration target at rest before capturing each image. We also used the high resolution (1280x1024), low framerate capture mode of the Kinect's color camera during calibration and transformed the parameters to that of the low resolution (640x480) high framerate mode used during system operation. To further reduce error, radial distortion in the depth image was also corrected using the distortion coefficients measured during intrinsic calibration of the color camera, as the two images are registered by the driver.

Depth Bias Correction. After revising our calibration procedures, we continued to observe small geometric misalignments between Kinects that occurred primary in the cameras' lines of sight and hypothesized a bias in the Kinect's depth readings. To test this hypothesis, we compared the Kinect's measurements to a laser rangefinder (Leica Disto$^{\mathrm{TM}}$plus, accuracy ±1.5 mm) at several distances using the following procedure:

1. A 2 m sliding rail was placed approximately 1.8 m from a wall so that its direction of movement was approximately perpendicular to the wall.
2. The laser rangefinder was mounted on the rail facing the wall. The orientations of the rail and rangefinder were finely adjusted until the laser spot emitted by the rangefinder onto the wall remained nearly motionless as the device was moved the length of the rail.
3. A Kinect unit was mounted to the rail so that it was approximately perpendicular to the wall, and a checkerboard target was placed on the wall. The orientation of the Kinect was finely adjusted until the calculated extrinsics between the Kinect and checkerboard indicated perpendicularity.
4. At the closest position on the rail, the z-axis translation component of the extrinsics between the Kinect and the checkerboard was compared to the corresponding rangefinder distance to determine their distance offset.
5. At approximate 0.25 m intervals along the rail, the rangefinder and Kinect distances were recorded. Kinect distances were computed as the mean distance over a 100×100 sampling window near the center of the depth map.

The test setup described in steps 1-3 is pictured in Figure 5.

Using the aforementioned procedure on several units, we observed that real-world distances returned by the utilized Kinect driver were biased and tended to overestimate depth linearly with distance. We used a least-squares fitting of our measurements to build a linear depth correction function for each Kinect. Table 1 shows the original and corrected measurements for one unit.

Depth Error Measurement. Since our mesh generation and data merger techniques rely on knowledge of the relationship between the distance of a surface to a Kinect depth camera and measurement error, it is beneficial to characterize this relationship. We expect the depth resolution to fall off quadratically with distance.

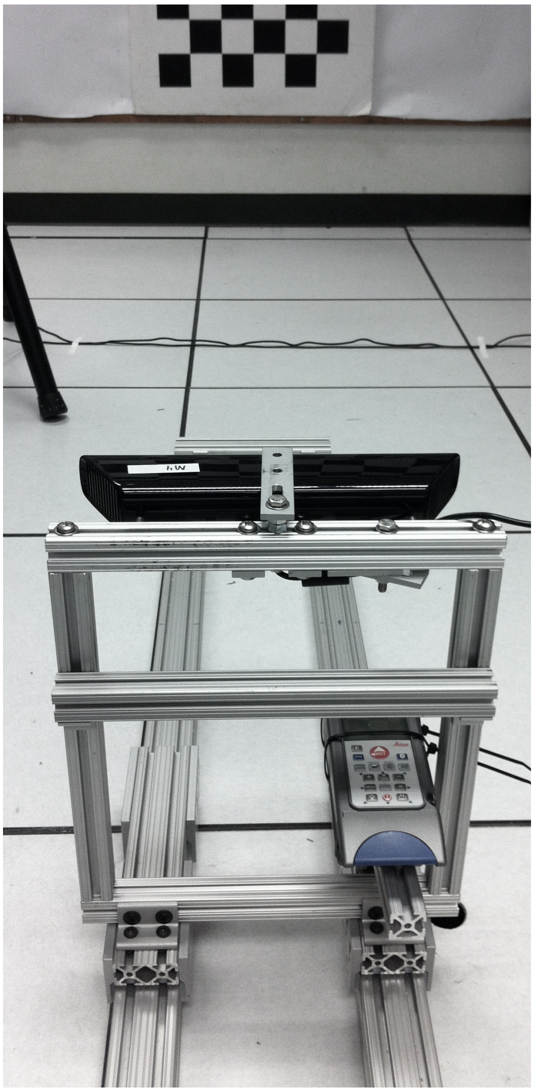

Figure 5: Test setup used to measure depth biases.

Table 1: Typical Depth Bias Correction (all values in mm)

Kinect	Rangefinder	Orig. Error	Corrected Value	Error After Correction
1820	1823	-3	1825.2	2.2
2074	2073	1	2072.6	-0.4
2327	2322	5	2319.0	-3.0
2588	2573	15	2573.3	0.3
2845	2823	22	2823.6	0.6
3103	3077	26	3074.9	-2.1
3365	3329	36	3330.1	1.1
3623	3581	42	3581.5	0.5

Fitted correction function: $f(d) = 0.9741d + 52.299$

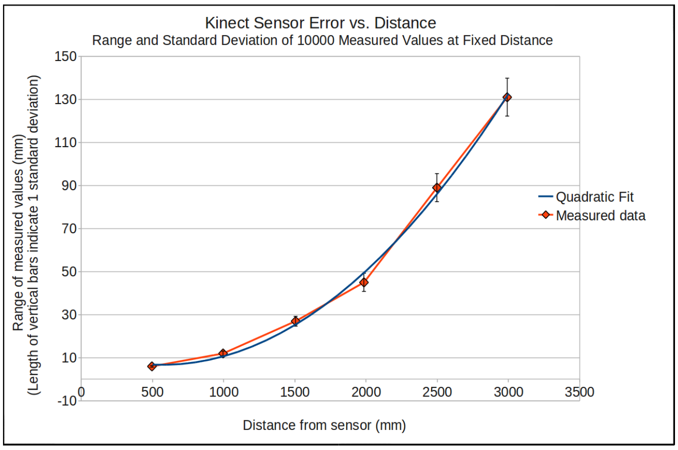

Figure 6: Kinect depth sensor precision with distance. Measured values show quadratic relationship between the distance to the depth camera and the range and standard deviation of depth values.

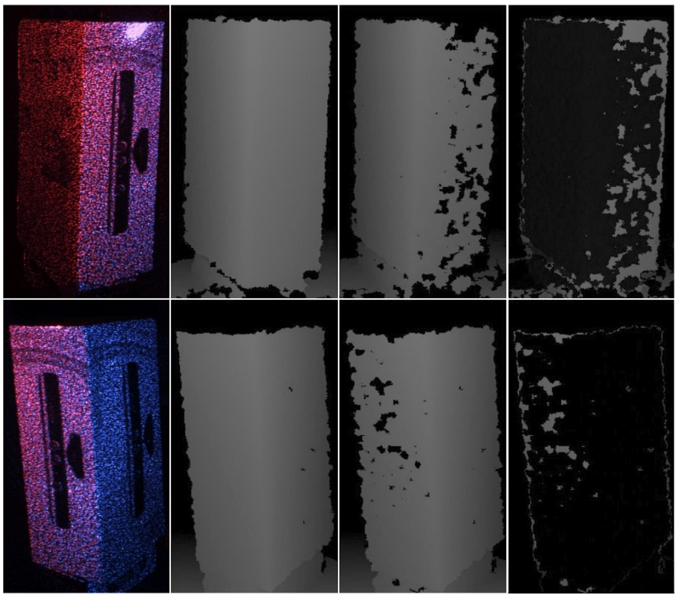

Figure 7: Kinect interference problem. First column: IR images showing combined projected dot pattern from camera 1 (red dots) and camera 2 (blue dots). Second column: depth images with no interference. Third column: depth images with interference from other camera. Fourth column: difference of second and third columns. Rows: data from each of two cameras.

To verify this, we positioned a planar target parallel to the IR camera's image plane and recorded a 100×100 grid of depth measurements at the center of the depth image. We performed this experiment at distances of 0.5 m (device minimum range) to 3.0 m (beyond the maximum range used in our system) at intervals of 0.5 m.

Figure 6 shows the min-max range and standard deviation at each test distance from 0.5 m to 3.0 m, fitting closely to a quadratic falloff.

4.2. Multi-Kinect Interference Problem and Solution

The Multi-Kinect Interference Problem. Since each Kinect unit projects the same dot pattern at the same wavelength, each Kinect unit is able to see the projected patterns of all other units and may have trouble distinguishing other units' patterns from its own.

This problem is illustrated in Figure 7. A box was placed near the minimum depth range (0.6 m) of two Kinect units; their projected patterns overlap prominently and cause interference. Although significant interference is shown in the third column of the figure (there many small areas of missing data, or "holes"), we find two promising aspects of these results. First, the difference between the depth image with and without interference corresponds mostly to the missing data, not large differences in depth values; one needs primarily to fill missing points rather than correct grossly erroneous depth values (although some additional noise is also present). Second, one can see in the third column of the figure that the missing data varies between depth cameras – to some extent, redundant coverage between units allows depth cameras to fill in each other's holes.

Hardware Solutions. We considered, but rejected several hardware solutions to the multi-Kinect interference problem. We contemplated installing a set of alternating synchronized shutters over each unit's IR projector and camera so that each unit would see only its own dot pattern, as was explored in [18]. A serious disadvantage of this approach is that it would reduce frame rate or reduce the light available to the IR camera, depending on how the shutters are used. Another technique considered, but also ruled out, was IR filtering. We measured a group of eight Kinect units with a spectrometer and found that the peak-to-peak range of wavelengths was 2.6 nm, which we found too close to filter practically.

Software Solutions. As we did not find a suitable hardware solution to the Kinect interference problem, we looked to software solutions. As mentioned, it is fortunate that the Kinect generally returns no data rather than erroneous data when interference occurs. However, there are other situations in which the sensor returns no depth data. Due to the offset between the IR projector and camera, there are typically some surfaces "in shadow" that can be seen by the camera but receive no projected pattern from the IR laser due to occlusion. Additionally, surfaces may not be seen by the depth camera if they reflect little infrared light or are highly specular. An effective software solution should be able to fill small holes (making the assumption that they are part of a continuous surface), while ignoring large missing surfaces. We hope that the missing large surfaces are captured by another camera that observes the surface from a different location. Also, Kinect interference causes a small amount of high frequency depth noise that should be smoothed.

Hole Filling. We aimed for a solution that fills small holes and provides an initial depth smoothing, but leaves alone large missing surfaces – we do not want to make large assumptions about missing data in the scene. The obvious starting point for such an approach is a simple smoothing filter (such as Gaussian, box, median, or bilateral), but our application induces additional requirements:

1. Edge preservation: we do not want to introduce new depth values across depth discontinuities – naive smoothing could result in depth values floating in space. Additionally, depth images are aligned to color textures, so color and depth edges should coincide. A small depth edge shift could cause a texture to be assigned to a physically distant surface. Depth edges at missing data boundaries must be preserved or geometry may expand or contract.
2. Scale independence: from observation, depth noise appears at a higher spatial frequency than holes. Smoothing should take place on a smaller scale than hole filling.

A standard median filter meets the first requirement (edge preservation – although not pixel-exact). We devised a fast modified median filter (Algorithm 1) that is effective at hole filling while supporting the requirements above.

To allow for scale independence, a two-pass approach is used. In the first pass, an expanded filtering window is used to fill larger holes, but no smoothing is applied (i.e. only missing values are modified). In the second pass, a smaller window is used to fill any remaining small holes and provide initial smoothing of the depth image. This method is similar to that used in [19], but we use different criteria to determine when the filter is applied.

To ensure that edges are preserved precisely and non-holes are ignored, we apply three constraints to the filtering window: a minimum amount of valid data must be present (t_c), a minimum amount of data must be present at the window edges (t_e), and the range of values in the window must be within a threshold (t_r). At each pixel, if the window constraints are not met, the pixel is left unmodified. These thresholds and heuristics were determined by applying a conventional median filter to sample depth data and inspecting cases that did not meet our requirements listed above.

To enhance our filtering implementation in the previous system [12], we incorporated the trimming operation of [17] into our hole filling and initial smoothing filter to reduce the appearance of ragged edges at depth discontinuities. This operation rejects geometry that does not meet the previously described data population and range tests, which was found empirically to occur at object edges. This operation is performed only during the second (small window) pass as to prevent over-trimming.

GPU Implementation. Our enhanced median filter implementation is based on a conventional median filter implementation by McGuire [20], which uses a branchless hardcoded selection algorithm to obtain the median for fixed radii. To provide high performance for larger radii, we find the approximate median by sampling over the filtering window. The median filter is written as a fragment shader in the OpenGL Shading Language, using textures to exchange data between passes.

4.3. Smoothing and Temporal Filtering

In our previous system [12], the smoothing described in Section 4.2 was the sole filtering operation applied to the depth maps. Although this filter made some improvement to the noisiness of the depth maps, several disadvantages were observed:

Algorithm 1 Modified Two-Pass Median Filter for Hole Filling

```
for pass = 1 to 2 do
  for i = 1 to numPixels do
    depth_out[i] ← depth_in[i]
    if depth_in[i] = 0 or pass = 2 then
      {Perform filtering tests}
      count ← 0, enclosed ← 0
      v ← {}, n ← neighbors(depth_in[i], radius_pass)
      min ← min(n), max ← max(n)
      for j = 1 to n.length do
        if n[j] ≠ 0 then
          count ← count + 1
          v[count] ← n[j]
          if on_edge(j) then
            enclosed ← enclosed + 1
          end if
        end if
      end for
      if max − min ≤ t_r and count ≥ t_c and enclosed ≥ t_e
      then
        {If filtering tests passed, find median}
        sort(v)
        depth_out[i] ← v[v.length/2]
      else if pass = 2 then
        {If filtering tests failed on 2nd pass, trim}
        depth_out[i] ← 0
      end if
    end if
  end for
  if pass = 1 then
    depth_in ← depth_out
  end if
end for
```

1. The median filter does not introduce new depth values while smoothing, and at greater distances, the steps in depth returned by the Kinect become large. This can result in significant depth discontinuities to appear among smoothed regions that are physically continuous.
2. No temporal noise suppression is performed, causing surfaces to have a jittering effect.

To address item 1 above, we further smoothed our hole filled depth maps with the following filter, based on the bilateral filter of [21]:

$$\frac{1}{W_{x,y}} \sum_{u=-r}^{r} \sum_{v=-r}^{r} [D(x+u, y+v) g(\|\langle u, v\rangle\|_2, \sigma_r) \\ g(|D(x+u, y+v) - D(x,y)|, \sigma_d)]$$

where (x, y) is the coordinates of the pixel to be filtered, r is the filter radius, $D(u, v)$ is the depth at location (u, v), $g(t, \sigma)$ is the Gaussian function e^{-t^2/σ^2}, σ_r and σ_d control the Gaussian falloff across the filtering window and depth respectively, and $W_{x,y}$ is a normalizing constant equal to the sum of the weights. This filter introduces new depth values while smoothing, but

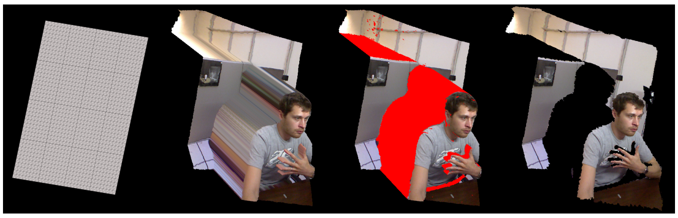

Figure 8: Fast Meshing for the Kinect. Left: Triangle mesh template stored in GPU memory. Center-Left: Vertex shader extrudes template using camera intrinsics and depth texture. Center-Right: Geometry shader rejects triangles corresponding to missing or discontinuous surfaces (shown in red). Right: Resultant textured mesh.

weighted values fall off sharply with increased depth distance from the central element to prevent smoothing over depth discontinuities.

As a first attempt to address item 2 above, we performed simple temporal noise suppression by updating values in our depth map only if they exceed a threshold from the last recorded value. The depth-dependent threshold was set conservatively based on the measured noise values plotted in Figure 6.

GPU Implementation. The bilateral filter was implemented as an OpenGL fragment shader which takes the hole-filled and pre-smoothed depth maps of Section 4.2 as input and produces a final depth map texture to use for mesh generation. The temporal filter was incorporated as an initial threshold check to the bilateral filter. The final depth map is copied within GPU memory for comparison with the next frame.

4.4. Mesh Generation

The Kinect provides per-pixel depth readings that are generally too sparse to render directly as small fixed-size points. Therefore it is useful to create a surface representation using the depth data. Our requirements for surface generation are as follows:

1. Must be continuous if physical surface is continuous
2. Must work in situations with missing data, as is common with Kinect
3. Must detect and preserve depth discontinuities at edges
4. Must be fast (5 Kinects generate >45M depth readings/sec)

Although approaches exist [22] for directly rendering points from multiple depth images, we chose a triangle mesh surface representation as it meets these requirements and is also supported natively by graphics hardware. We use a simple meshing technique which is described in Algorithm 2 and illustrated in Figure 8. The depth values from the Kinect sensor are used to extrude vertices from a template triangle mesh. Any triangle associated with a vertex that corresponds to a missing depth value is rejected, as are those with a pair of vertices that exceeds a maximum depth threshold. Since the Kinect provides depth data that varies in accuracy with depth, our depth threshold varies with depth as well.

Algorithm 2 Mesh generation algorithm

for each candidate triangle **do**

  $t \leftarrow thresh_{depth\_discontinuity} + f_{depth\_err}(min(depth_{v_i})) + f_{depth\_err}(max(depth_{v_i}))$

  {Transform vertices from normalized image coordinates to camera coordinates in physical units}

  if $depth_{v_i} \neq 0$ **and** $abs(depth_{v_i} - depth_{v_j}) \leq t$, for $j > i$ **then**

    $v_{i_x} \leftarrow \frac{(v_{i_x} - center\_proj_x)depth_{v_i}}{focal_x}$

    $v_{i_y} \leftarrow \frac{(v_{i_y} - center\_proj_y)depth_{v_i}}{focal_y}$

    $v_{i_z} \leftarrow depth_{v_i}$

  else

    reject triangle

  end if

end for

GPU Implementation. Our implementation of the simple mesh algorithm takes advantage of the connectivity of a depth image, requiring no geometry to be transferred to the GPU after program initialization. At program start we generate a triangulated plane at the Kinect's depth resolution and store it in GPU memory. For each new frame, a vertex shader shapes the template plane using camera intrinsics and Kinect depth values, which are accessed through a texture map. A geometry shader is used to reject triangles corresponding to missing depth values or discontinuous surfaces as described in Algorithm 2.

This approach is very bandwidth-efficient – it requires only 16 bits of depth information for each pair of triangles generated and uses the depth map already transferred to GPU memory for the hole filling process. The approach is also fast as all vertex positions are generated on the GPU in parallel.

4.5. Data Merger

Overview. A goal of our system is to provide coverage of all surfaces that can be seen from the perspective of a remote user. A single Kinect is not able to provide adequate coverage and therefore a means to merge data between multiple units is necessary. When generating meshes we did not discuss a means to merge overlapping surfaces geometrically. Approaches used in stereo vision, such as the visibility-based depth image fusion method of Merrell et al. [19], generally assume high levels of error and inconsistencies (outliers) between maps that must be resolved. The Kinect's structured-light based depth readings, however, are generally free of such outliers and have a low error at near range. In our application, Kinect sensors are used at close proximity and we expect lower and predictable error based on the angle and distance to the camera and measured calibration error. Therefore, we assume that the surfaces have enough fidelity that we can simply draw them on top of each other, avoiding the need for a geometric merger algorithm. This has several performance advantages: the computational expense of performing the merge is spared, runtime varies linearly with the number of cameras, surfaces for all cameras can be processed in parallel, and a fast mesh generation technique (Section 4.4) can be used.

However, even though geometry is sufficiently accurate for our purposes, texture image quality may be poor. Z-fighting between overlapping surfaces with textures that vary in resolution, color imbalances, and misalignment yields unpleasing results. Ideally, we want to utilize only the data from the camera with the highest resolution depth and color information available at a given surface, with a seamless transition to data from adjacent cameras.

Our approach addresses the problem of data merger in image space using a visibility-based approach. The data from each camera is rendered independently for the desired viewpoint, and color information is saved along with a depth and a quality estimate at each pixel. When renderings for all cameras are complete, the depth values are used to determine which cameras can see the front surface. At each pixel, the color values of cameras with a view of the front surface are weighted by the quality estimates. This process is illustrated in Figure 9.

Texture Quality and Depth Error Estimation. Since our approach relies on the notion of a "quality" measurement at each pixel, we provide an estimate based on resolution – the area on the image sensor available to determine the pixel's color or position. The area is estimated using the cosine of the angle between the surface normal of the pixel and the squared distance from the pixel to the image sensor. The relationship between area and resolution is straightforward for a color image, and we demonstrated previously that the Kinect depth error increases quadratically. We approximate quality by assuming that both color and depth error increase quadratically, yielding the quality value in Equation 1.

$$quality = \left(\frac{cos\theta_{normal\rightarrow camera}}{distance^2}\right)^2 \quad (1)$$

Note that this formulation is similar to a diffuse lighting calculation with attenuation (for a light positioned at the sensor's location) that can rapidly be performed on almost any graphics hardware.

Our approach also requires determination of which pixels represent the closest surface with respect to viewing position. We store the depth values at each pixel, but due to calibration and depth sensor error the values corresponding to the front surface do not coincide exactly. Equation 2 is used to estimate the range of each depth position, so that the range of depths corresponding to the front surface can be determined.

$$[-err_{calib} - f_{depth\_err}(depth), err_{calib} + f_{depth\_err}(depth)] \quad (2)$$

Calibration error (err_{calib}) can be estimated using the reprojection error that is returned by the camera calibration routine. Depth error (f_{depth_err}) can be estimated using the data from Figure 6.

Data Merger Algorithm. Algorithm 3 describes the process of merging the renderings for each camera. At each pixel, the front surface tolerance is determined by finding the closest depth value that represents the far end of any pixel's estimated depth range. The color values for all pixels with this depth value or nearer are weighted by quality to obtain the final pixel color.

Algorithm 3 Data merger algorithm

for each output pixel p **do**
  $depth_{far} \leftarrow \infty$
  {Determine far bound of closest surface}
  for each camera c **do**
    $d_{far} \leftarrow depth_{c_p} + err_{calib} + f_{depth\_err}(depth_{c_p})$
    if $d_{far} < depth_{far}$ **then**
      $depth_{far} \leftarrow d_{far}$
    end if
  end for
  $color_{sum} \leftarrow 0, quality_{sum} \leftarrow 0$
  for each camera c **do**
    {Only consider cameras with view of closest surface}
    if $depth_{c_p} \leq depth_{far}$ **then**
      $match_{best} \leftarrow \infty$
      {Perform photometric search for closest matching pixel to camera with best quality estimate}
      for each pixel s in search window **do**
        $match \leftarrow \|color_{q_p} - color_{c_s}\|_2$
        if $match < match_{best}$ **then**
          $match_{best} \leftarrow match$
          $color_{best} \leftarrow color_{c_s}$
          $quality_{best} \leftarrow quality_{c_s}$
        end if
      end for
      {If photometric threshold met, perform quality weighted blending}
      if $match_{best} \leq thresh_{match}$ **then**
        $color_{sum} \leftarrow color_{sum} + quality_{best} * color_{best}$
        $quality_{sum} \leftarrow quality_{sum} + quality_{best}$
      end if
    end if
  end for
  $color_{output} \leftarrow color_{sum} / quality_{sum}$
end for

Photometric Constraint. A shortcoming of our original data merger algorithm [12] was the unconditional blending of data between all cameras regardless of color consistency. If intensity varied widely between cameras (due to geometric misalignment or specularity), the resultant combined textures would appear blurry or a double image would appear at conflicting pixels. Inspired by [7], we apply a photometric search and constraint before blending each quality weighted color sample. For each camera, we perform a search within a small window for the closest matching color to the color associated with the camera with the highest quality estimate ($color_{q_p}$) and blend only if the colors match within a threshold.

GPU Implementation. Our fast GPU implementation supports calculation of depth, quality, and color values in one pass per camera and allows all cameras' renderings to be merged at once

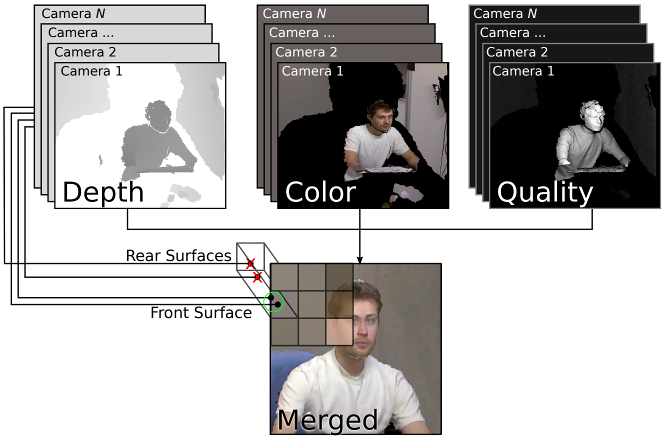

Figure 9: Data Merger Algorithm. Depth, color, and quality estimate values are determined at each pixel for each camera. The front surface is determined using the depth information, and the associated color values are weighted by the quality estimates.

in a second pass. When generating the triangle mesh in an OpenGL geometry shader, we compute the distance to the camera and the angle between the camera and surface normal and save these values as vertex attributes. During rasterization, an OpenGL fragment shader computes a color value and a quality value (using Equation 1) at each pixel, storing the quality value in the alpha channel and the depth value from the Z-buffer in a separate texture. When the renderings for all cameras are complete, all data is merged in an OpenGL fragment shader according to Algorithm 3.

4.6. Multiple Camera Color Matching

Overview. The need for color matching is common for many camera systems, as even the same model device may exhibit different color gamuts [23]. This need is exacerbated in inexpensive devices like the Kinect sensor, which allows only automatic color and exposure control (with present drivers), yielding color values that may vary dramatically between adjacent cameras. Here traditional color matching techniques, such as adjusting color to match a physical target seen by each camera, are ineffective because automatic control may alter color balances at any time. We present an automatic color matching technique that uses depth information to find color correspondences between cameras, which can be used to build a color matching function. We believe this technique may be useful when manual color adjustment is unavailable, or as a fast approximate alternative to conventional matching techniques.

Obtaining Color Correspondences. To build a set of color correspondences between cameras, we first find pairs of points from two cameras that correspond to approximately the same point in 3D space. We assume that each pair of points represents the same point on a diffuse surface in physical space, and therefore should agree in color. To find these point correspondences, we refer to our previously described visibility-based data merger algorithm. The algorithm rendered the scene individually for each Kinect camera and examined corresponding depth values to determine which represented the front surface. For color matching, if two cameras have depth values that represent the front surface at a given pixel, we add their color values to a list of correspondences.

Since this approach is visibility-based, the color correspondences obtained are sensitive to the position of the virtual camera. If the same virtual camera position is used for color matching and rendering, color matching is tailored to the colors actually seen by the user. However, if a pair of cameras have few surfaces in common from the viewpoint used for rendering, or if these surfaces have a limited range of colors, there many be too few correspondences to build a robust color matching function. In this case, point correspondences can be computed from a reference view (such as a bird's eye view), rather than from the view used for rendering. To build more robust color correspondences, additional techniques could be used. For example, the color correspondences could be built from renderings from several viewpoints, or could be collected over time.

Building a Color Matching Function. There are many advanced techniques for building color matching functions from a set of color correspondences, such as that of Ilie and Welsh [23]. To demonstrate our approach, we used a simple method – color correspondences were fit to a linear model. Since our color correspondences were noisy (small errors in surface position may result in a large difference in color), we used the RANSAC [24] method for fitting, which is robust to outliers. Figure 10 shows a plot of actual color correspondences (for one channel) and the fitted linear color matching function.

Implementation. For our test setup, we matched the colors of each camera to our ceiling-mounted master camera. We elected not to run the color matching function on every frame, as small variations in color matching functions resulted in a color cycling effect. Instead a new color matching function was built whenever the user pressed a function key. As real-time performance was not needed, we implemented color matching functionality on the CPU. We believe our implementation could be improved by performing bundle adjustment across cameras and by running the color matching function automatically when some criteria is met.

4.7. Eye Position Tracking

Overview. To allow our system to render a set of correct stereo viewpoints from the user's position, we need to obtain the position of the viewer's eyes in 3D space. Many approaches to tracking have been devised, such as measuring the magnetic field around a marker, segmenting and triangulating the position of reflective markers as seen by an array of cameras, and using computer vision techniques to recognize objects in images. The latter approach has been used to obtain the 3D positions of eyes with a conventional 2D camera, but assumptions or measurements must be made of the face. We aim to improve these techniques by incorporating depth information. One impressive recent approach [25] used depth information to build a deformable mesh that was tracked to a user's face in real-time, but required a 6.7 second initialization time and achieved only

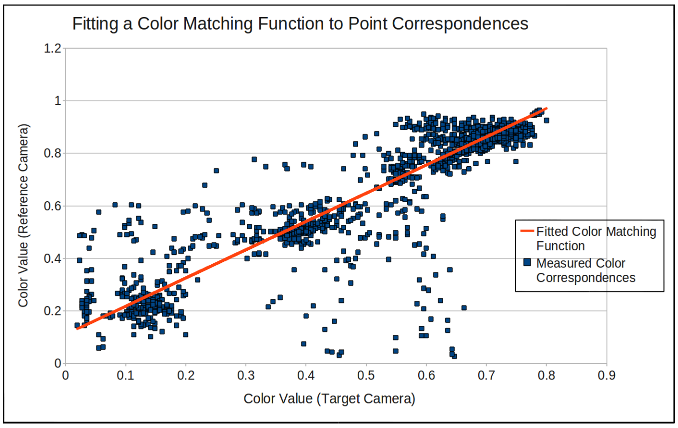

Figure 10: Color matching using 3D point correspondences. Plot shows color correspondences between a pair of cameras for one color channel and the RANSAC-fitted linear color matching function.

moderate real-time performance (10-12 Hz). Since we require higher performance and do not need tracking of the entire face, we look to an alternate approach – performing 2D eye detection and transforming the detected position into 3D using the Kinect's depth data.

Requirements. Our tracking system should meet the following requirements for use in our telepresence system:

1. Accuracy: at a 1 m distance, 15 mm of lateral movement causes the eye to sweep over one display subpixel seen through the barrier of our autostereo display; for best quality results tracking accuracy should be ±7.5mm.
2. Speed, Latency: we do not anticipate rapid head movements in our application. To support the modest movement of 25 cm/sec, framerate must be > 33.3 Hz and latency must be < 30ms to meet the accuracy requirements above.

2D Eye Tracking. To perform 2D eye detection on the color image, we use Viola [26] and Lienhart's [27] approach of boosted Haar classifiers, as implemented in OpenCV. First the face is detected (using a classifier from Leinhart), and then eyes are detected in the facial region (using classifiers from Castrillon [28]). Once the eyes are found, their pattern is saved and subsequent eye searches are performed using normalized cross correlation. An image pyramid is used to accelerate the cross correlation search. If the strongest response to cross correlation falls below a threshold, detectors are again used to locate facial features. The face is first searched for in the region surrounding the last known eye position; if not found the entire image is again searched. All detection and tracking operations were performed in the CPU, as it was not heavily utilized elsewhere in our system. A single Kinect unit, mounted above the autostereo display, was used for tracking.

Using Depth to Obtain 3D Eye Position. Once the center of both eyes have been detected, the 2D position is transformed into 3D using the Kinect's depth information and measured camera intrinsics and extrinsics. To reduce the effects of noise and missing data, depth values are averaged over a small radius around the eye position. A Kalman filter was used to improve the accuracy and stability of the 3D tracked eye positions as well as predict the locations of the eyes between sensor readings. Although our tracking system requires no prior measurements of the user's face, accuracy can be improved if the true interpupillary distance (IPD) is known. If the system is utilized by a single user over a capture session, an accurate IPD estimate can be learned over time.

Discussion. Our tracking system offers several advantages over existing systems. It uses inexpensive hardware (the Kinect sensor) and allows the same device to be used for both tracking and 3D capture at the same time. Since the eyes are tracked independently, our system allows correct calculation of 3D eye positions without measurements or assumptions of face size or IPD.

We believe our system could be improved with a more robust set of feature detectors – our current system allows for only moderate head rotations and does not work well with glasses. Depth data could also be further utilized to improve speed; for example, the face search area could be restricted to depths that are within the range of a seated user. Multiple cameras could be utilized to offer better coverage of a rotated head or to improve the accuracy of the system.

4.8. Stereo Display

Overview. As mentioned, research [13] has shown that stereo displays can increase the sense of shared presence, although systems requiring 3D glasses obstruct eye contract and have been found to be disruptive to most users. Therefore, we desire an autostereo display for our system.

Display Selection. Our display system should meet the following requirements for use in our telepresence system:

1. Preservation of full captured color and detail.
2. Large enough to allow remote scene to be observed as life-sized at proper viewing distance.
3. Support for continuous viewpoints and horizontal and vertical parallax.
4. Support for a range of movement typical of a seated user.
5. Interactive update rates that meet our tracking requirements.

We were in possession of a fixed parallax barrier display that met these requirements – an X3D-40 display by X3D technologies (circa 2004). The display measures 40 in diagonally and has a 1280×768 pixel resolution and a 60 Hz update rate. Since the display supports only a limited number of views, tracking was employed. In a future system, we intend to utilize a display that supports multiple users, such as the Random Hole display of Ye et al [29].

Rendering for the Display. Since our system uses head tracking, we rendered views for the display using off-axis frustra between the eyes and the display. The position of the eyes was determined using the tracking system, and the position of the monitor was measured using our 3D capture system. An OpenGL fragment shader was used to generate the diagonally interleaved pattern needed by our parallax barrier display for each pair of stereo views.

Tracking Problem and Intra-Frame Rendering. While using our fixed parallax barrier display, a user may see an incorrect view or significant artifacts (dark black bands or fuzziness) if out of the expected viewing position. If the rendering update rate is lower than the rate required by our tracking system, a user may experience these effects if moving, resulting in poor stereo perception.

This problem has been addressed previously for dynamic barrier displays [30] by generating the parallax barrier *stripes* asynchronously at higher rates than rendering takes place. For fixed barrier displays, we developed a new technique to address this problem – rendering barrier *display patterns* at a higher rate while new frames are rendered more slowly offscreen.

Since the time it takes to generate a parallax barrier pattern for a new eye position is very short and fixed with our GPU implementation, we can draw one or more new barrier patterns while in the process of rendering a frame for the next viewing perspective. These intra-frame barrier patterns use the new estimated eye position and the last rendered viewing position, saved in textures. Using OpenGL, we are able to draw to the screen mid-frame by switching between multiple frame buffers. To keep our parallax barrier generation rate and rendering rate independent, we stop to draw a new barrier pattern whenever a fixed amount of time has elapsed during rendering, periodically flushing the pipeline to allow for better time granularity between asynchronous GL calls. The result is a high fixed barrier display rate, independent of rendering rate, at the expense of a small decrease in rendering rate. Specific rates are listed in Section 5.4

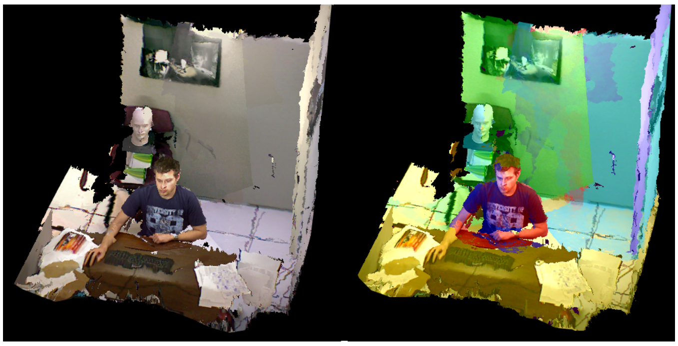

Figure 11: Camera Coverage. Left: Camera coverage in our cubicle area using five depth cameras. Right: Color coded contributions of individual cameras.

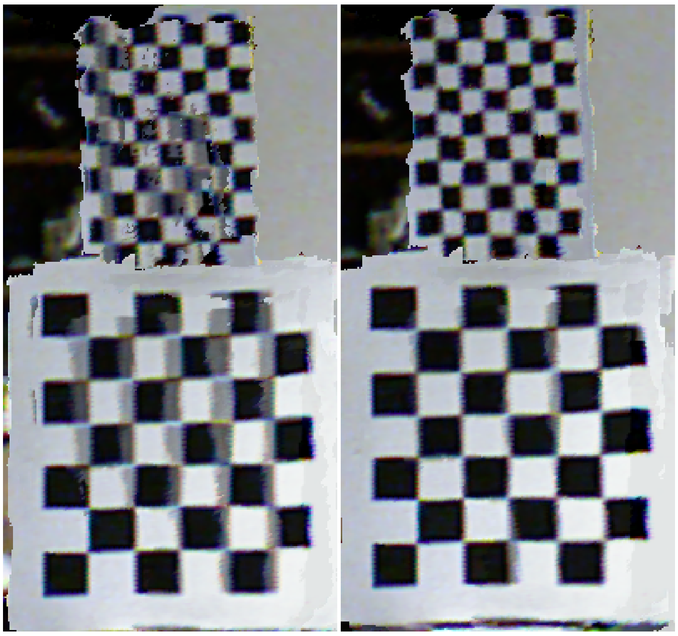

Figure 12: Depth bias correction results. Left: No depth bias correction applied. Right: Depth bias correction applied.

5. Results

5.1. Camera Coverage and Calibration Results

Camera Coverage. Our camera arrangement (shown in upper left of Figure 3), includes most of the surfaces seen by a seated remote user, as shown in Figure 11. Overlapping camera coverage preserves large surfaces on the rear wall, which would otherwise be occluded by the seated user. Redundant coverage also helps prevent self shadowing. For example, in Figure 1 the coffee cup occludes part of the user's chest with respect to one of the front cameras, but the missing surfaces are filled with data from the other front camera.

Depth Bias Correction. To test the depth bias correction functions determined in Section 4.1, two checkerboard targets were placed centered in front of two Kinect units, so that nearer and farther targets were approximately 3.0 m and 3.5 m from the units respectively. The Kinects were aimed at the targets and positioned with a wide baseline of 2.5 m to accentuate errors in depth. The data from the two cameras was rendered using the techniques described in Section 4, except that the photometric constraint of Section 4.5 was disabled during blending so that the misalignments of the high contrast targets could be observed.

The results are shown in Figure 12. One can see a significant improvement in the alignment of both checkerboards between cameras, although a small amount of misalignment remains.

5.2. Data Processing and Rendering Results

Mesh Generation. All images in Figure 13 show the result of mesh generation. Our requirements are met: the mesh offers a continuous surface, discontinuous surfaces (such as from the body to the rear wall) are properly separated, and missing data (small area under chin) is tolerated.

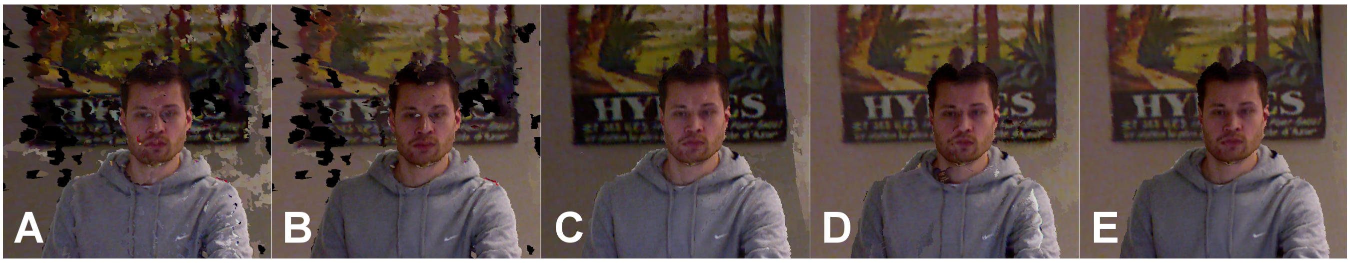

Figure 13: Data processing results. A: No enhancements applied. B: All enhancements except hole filling and smoothing. C: All enhancements except color matching. D: All enhancements except quality weighted data merger. E: All enhancements applied. (“All enhancements” includes to hole filling, smoothing, data merger, and color matching)

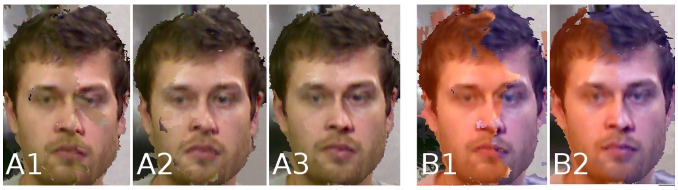

Figure 15: Data merger results. A1: No merger applied, meshes drawn on top of each other. A2: Merger with simple average. A3: Merger with quality weighting. B1: Color coded camera contributions with no merger. B2: Camera contributions with quality-weighted merger.

Hole Filling and Smoothing. Image E of Figure 13 (as compared to image B of Figure 13) shows the result of the hole filling and smoothing filters applied on a scene with four overlapping cameras. In this example, 100% of holes caused by interference were filled while textures remained aligned to the mesh (rates of > 90% are typical).

The effect of the additional bilateral smoothing filter is illustrated in Figure 14. Note that the more distant surfaces (ex: wall) appear flatter and smoother with the addition of the bilateral filter (right image) than with the median-only smoothing of our previous system [12] (center image).

Data Merger. Image E of Figure 13 (as compared to image D of Figure 13) and all of Figure 15 show the result of the data merger algorithm on four cameras, which is cleaner and smoother than meshes simply drawn over each other or averaged. In image B1 of Figure 15, one can see in the unmerged example that the mesh of the right-front camera (tinted blue) is drawn entirely over the data from the left-front camera (tinted red). These surfaces should coincide exactly, but a small calibration error places the surface from right-front camera closer to the viewer. In image B2 of Figure 15, one can see that the quality-weighted merger algorithm smoothly transitions between camera data across the face.

Figure 16 shows the improvement gained when the photometric constraint is incorporated into the data merger algorithm. Without the photometric constraint, some areas of the poster (for example, around the walking figures and the large letter È) have ghost images due to slightly misaligned surfaces between cameras. With the constraint enabled, the ghost images have mostly disappeared, although a faint outline is still visible.

Figure 16: Photometric constraint. Top: No photometric constraint in data merger algorithm. Bottom: Photometric constraint applied.

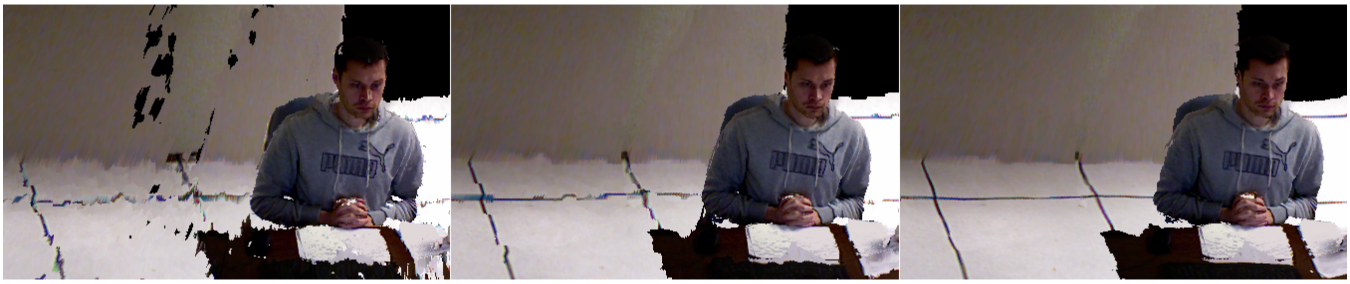

Figure 14: Smoothing results. Left: No smoothing (or hole filling) applied. Center: Median smoothing. Right: Bilateral smoothing.

Figure 17: Cumulative enhancements. Left: Rendering using our previous system [12]. Right: Rendering using the improved calibration, data processing, and rendering techniques described in this paper.

Table 2: Tracking performance over 1600 frames.

Case	#	%	Avg Time(ms)
Eyes found (full frame face/eye detect)	3	0.19	140.5
Eyes found (partial frame face/eye detect)	9	0.56	19.8
Eyes found (pattern search)	1585	99.06	2.3
Eyes not found	3	0.19	13.6

Color Matching. Image E of Figure 13 (as compared to image C of Figure 13) shows the result of color matching in a scene with four cameras. The unmodified image shows small color inconsistencies between the front cameras (on the face and clothing) and significant inconsistencies between the four cameras that overlap in the background – most notably on the far right side of the image. The automatic color matching algorithm mostly resolved these deviations, although some faint color seams are still visible.

Cumulative Result. Figure 17 shows the cumulative effect of the enhanced data processing and rendering (as well as calibration) techniques over those from our previous system [12]. Note the improved straightness of lines on the poster and checkerboard, clearer textures on the face, t-shirt, and poster, and the absence of ghosting on the checkerboard.

5.3. Eye Tracking and Stereo Display Results

Eye Detection Rate and Speed. Table 2 shows the tracking performance typical of a seated user over a 1600 frame sequence. For the sequence, the user was seated centered 1 m from the display and tracking camera and moved his head left, right, forward and backward over a range of ±0.5 m. The average head movement speed was 48 cm/s, measured using the detected 3D eye positions. Positive eye detection occurred on >99% of the frames at an average rate of 2.7 ms. In the worst case, when the face was lost and the entire frame had to be searched, a noticeable delay of 140.5 ms on average occurred.

Tracking Performance. Figure 18 provides a measure of the performance of the eye tracking by comparing the 3D distance between a pair of tracked eyes and the true measured interpupillary distance (IPD). IPD was used as the ground truth for accuracy as we were not in possession of equipment that would allow us to measure our positional accuracy directly. This metric was measured over a sequence of 1761 frames, in which a user seated 1 m from the tracking camera moved his head to the left, right, forward and backward over a range of ±0.5 m. 85.6% of measurements were within ±5 mm of the true IPD, and 96.4% were within ±10 mm.

Tracking Accuracy and Stereo Quality. Since our tracking system is designed to support a stereo display, it is useful to test the two systems together. To demonstrate that our tracking system is fast and accurate enough to support our parallax barrier autostereo display with good quality, we shot video of our system through a tracking target (shown in Figure 20). Our tracking system is able to detect the target as if it were a real face and thus one of the stereo views will be generated from the correct perspective of the camera placed behind an eye. Using this setup, the target and camera were positioned 1.25 m from the tracking camera and display and were moved at a rate of approximately 24 cm/sec.

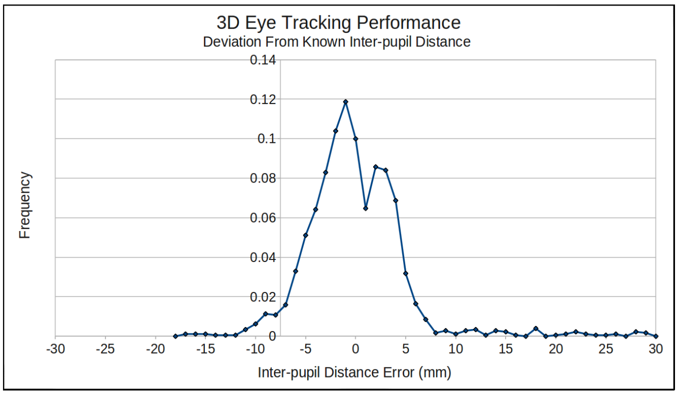

Figure 18: 3D Eye tracking performance. Plot shows measured deviations from a known inter-pupil distance.

Without tracking prediction and intra-frame rendering enabled, the rendering and parallax barrier pattern generation rate was 21 Hz in our four camera test setup. As seen in the left of Figure 19, results were very poor; the tracking and rendering could not keep up with the target as it moved into the viewing zone intended for the other eye and thus both views could be seen prominently and simultaneously. With tracking prediction and intra-frame rendering enabled (right of Figure 19), the rendering rate dropped slightly to 18 Hz but the barrier generation rate more than doubled to 48 Hz. Results were much improved – the view seen by the camera is crisp and only very faint ghosting can be seen to the right of the mannequin head and box. [*Please note* that the performance numbers quoted in this paragraph and the images of Figure 19 were measured using our previous system [12] and performance has since increased due to a graphics card upgrade (see Table 3). Although the higher display rates of our upgraded hardware lessen the need for intra-frame rendering in our current setup, we believe the technique remains applicable to those with more modest hardware. It will also allow us to maintain stereo quality as more advanced (and computationally complex) reconstruction algorithms are utilized and more Kinects are added to the system in the future, and will allow increased head movement speeds.]

5.4. System Performance

Table 3 lists the performance achieved with our test system in various configurations. When rendering for a single view, the system was able to maintain average frame rates of 48 Hz for five depth cameras with all enhancements (meshing, hole filling, quality-weighted data merger) enabled. For tracked stereo configurations, rendering rates fell to 34 Hz, but a parallax barrier pattern rate of 74 Hz preserves smooth head tracking and stereo quality.

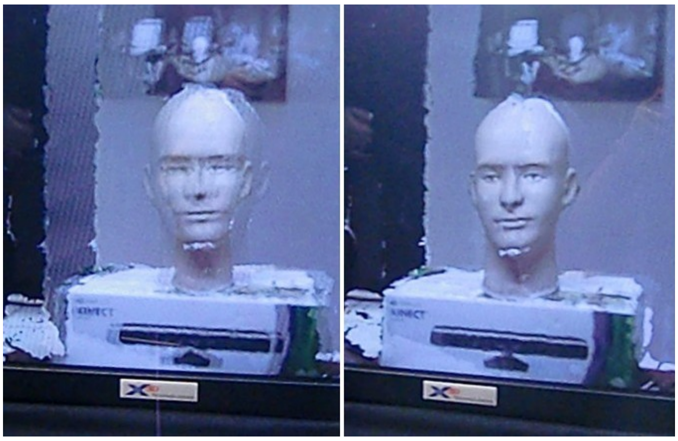

Figure 19: Head-tracked stereo in motion. Left: Tracking prediction, intra-frame rendering disabled. Right: Prediction, intra-frame rendering enabled. (Note: faint image on right side is reflection of photographer).

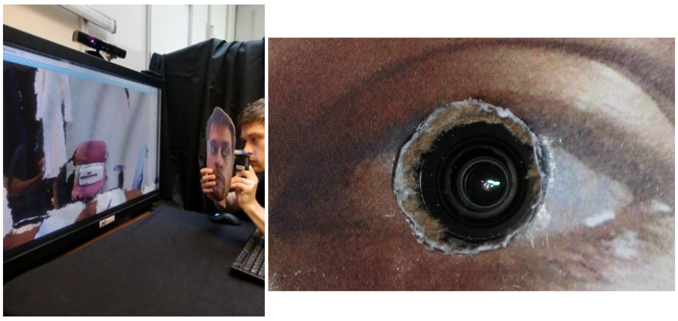

Figure 20: Tracking target. Left: Head cutout used to test eye tracking, with camera behind eye. Right: close-up of camera through eye.

Table 3: Average display rates (frames per second) for 5 Kinects

Single View w/ Selected Enhancements	
Meshing	160
Meshing, Hole Filling/Smoothing	117
Meshing, Hole Filling/Smoothing, Data Merger	48
Stereo Views w/ All Enhancements	
Head-Tracked	36
Head-Tracked w/Prediction (Render Rate)	34
Head-Tracked w/Prediction (Barrier Rate)	74

6. Conclusions and Future Work

We have presented solutions to several issues related to building a 3D capture system using multiple depth cameras: resolving interference, data merging, and color matching between units. We have also introduced an eye position tracking system using depth sensors and demonstrated effective stereo display using rendering rates that would not usually support significant head motion. We have also incorporated improvements in calibration, data filtering, and data merger that served to improve image quality over a previous version of the system [12].

Using these solutions, we have demonstrated a telepresence system that is able to capture a fully dynamic 3D scene the size of a cubicle while allowing a remote user to look around the scene from any viewpoint. The system preserves eye gaze and does not require the user to wear any encumbrances. Using a single PC and graphics card, our system was able to render head-tracked stereo views at interactive rates and maintained stereo percept even with moderate head movement speeds.

Although our test system is functional, there are areas that we would like to improve, notably image quality. Although we incorporated a simple temporal noise suppression function in our system, some temporal noise artifacts are still present at the edges of objects where the depth camera alternates between providing a value and reporting no data at a given pixel. These depth pixels could be modified to keep a steady state or object contours could be smoothed and gaps could be filled in. Color calibration could be enhanced by combining our color correspondence-building algorithm with more robust color matching functions.

We also intend to expand our test setup into the “ideal” system shown in the bottom of Figure 3 by supporting 3D capture and 3D display for multiple users in both spaces. As seen in Figure 2, we already support 3D capture of multiple users. In this future system, we intend to add support for multiple tracked users on both sides.

Finally, we would like to expand the communication ability of our system by adding support for virtual objects that can be manipulated naturally by persons in the scene. Figure 21 shows an early experiment.

Acknowledgments

The authors would like to thank Herman Towles for proposing this system, Andrei State for helping to produce the supplemental video and both for making helpful suggestions for improving this paper. We also thank Kurtis Keller for advice on reducing Kinect interference, Adrian Ilie for recommendations regarding color calibration, Ava Pope for helping to measure the wavelength of the Kinect’s IR lasers. This work was supported in part by the National Science Foundation (award CNS-0751187) and by the BeingThere Centre, a collaboration of UNC Chapel Hill, ETH Zurich, NTU Singapore, and the Media Development Authority of Singapore.

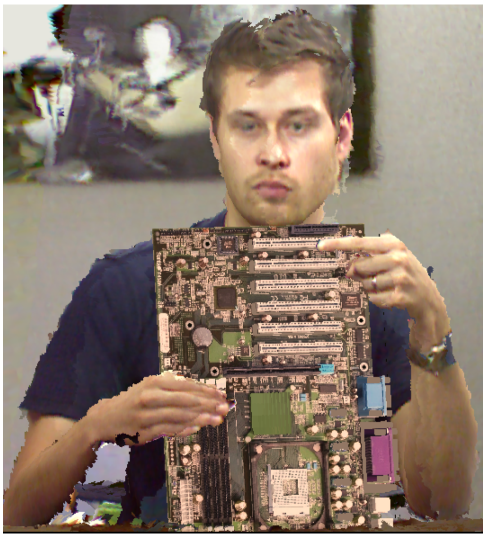

Figure 21: Mixed Reality Application. A 3D virtual object (circuit board) is incorporated into the scene during real-time 3D capture and naturally occludes and is occluded by real objects.

References

[1] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, H. Fuchs, The office of the future: a unified approach to image-based modeling and spatially immersive displays, in: Proceedings of the 25th annual conference on Computer graphics and interactive techniques, SIGGRAPH '98, ACM, New York, NY, USA, 1998, pp. 179–188. doi:http://doi.acm.org/10.1145/280814.280861.
URL http://doi.acm.org/10.1145/280814.280861

[2] S. J. Gibbs, C. Arapis, C. J. Breiteneder, Teleport towards immersive copresence, Multimedia Systems 7 (1999) 214–221, 10.1007/s005300050123.
URL http://dx.doi.org/10.1007/s005300050123

[3] W. Matusik, H. Pfister, 3d tv: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes, ACM Trans. Graph. 23 (2004) 814–824. doi:http://doi.acm.org/10.1145/1015706.1015805.
URL http://doi.acm.org/10.1145/1015706.1015805

[4] O. Schreer, I. Feldmann, N. Atzpadin, P. Eisert, P. Kauff, H. Belt, 3dpresence -a system concept for multi-user and multi-party immersive 3d videoconferencing, in: Visual Media Production (CVMP 2008), 5th European Conference on, 2008, pp. 1 –8.

[5] A. Jones, M. Lang, G. Fyffe, X. Yu, J. Busch, I. McDowall, M. Bolas, P. Debevec, Achieving eye contact in a one-to-many 3d video teleconferencing system, ACM Trans. Graph. 28 (2009) 64:1–64:8. doi:http://doi.acm.org/10.1145/1531326.1531370.
URL http://doi.acm.org/10.1145/1531326.1531370

[6] T. Balogh, P. T. Kovács, Real-time 3d light field transmission, Vol. 7724, SPIE, 2010, p. 772406. doi:10.1117/12.854571.
URL http://link.aip.org/link/?PSI/7724/772406/1

[7] C. Kuster, T. Popa, C. Zach, C. Gotsman, M. Gross, Freecam: A hybrid camera system for interactive free-viewpoint video, in: Proceedings of Vision, Modeling, and Visualization (VMV), 2011.

[8] M. Gross, S. Würmlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. Van Gool, S. Lang, K. Strehlke, A. V. Moere, O. Staadt, blue-c: a spatially immersive display and 3d video portal for telepresence, ACM Trans. Graph. 22 (2003) 819–827. doi:http://doi.acm.org/10.1145/882262.882350.
URL http://doi.acm.org/10.1145/882262.882350

[9] G. Kurillo, R. Bajcsy, K. Nahrsted, O. Kreylos, Immersive 3d environment for remote collaboration and training of physical activities, in: Virtual Reality Conference, 2008. VR '08. IEEE, 2008, pp. 269 –270. doi:10.1109/VR.2008.4480795.

[10] H. H. Baker, N. Bhatti, D. Tanguay, I. Sobel, D. Gelb, M. E. Goss, W. B. Culbertson, T. Malzbender, Understanding performance in coliseum, an immersive videoconferencing system, ACM Trans. Multimedia Comput. Commun. Appl. 1 (2005) 190–210. doi:http://doi.acm.org/10.1145/1062253.1062258.
URL http://doi.acm.org/10.1145/1062253.1062258

[11] B. Petit, J.-D. Lesage, C. Menier, J. Allard, J.-S. Franco, B. Raffin, E. Boyer, F. Faure, Multicamera real-time 3d modeling for telepresence and remote collaboration, International Journal of Digital Multimedia Broadcasting 2010 (2009) 247108–12.

[12] A. Maimone, H. Fuchs, Encumbrance-free telepresence system with real-time 3d capture and display using commodity depth cameras, in: Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on, 2011, pp. 137 –146. doi:10.1109/ISMAR.2011.6092379.

[13] L. Muhlbach, M. Bocker, A. Prussog, Telepresence in videocommunications: a study on stereoscopy and individual eye contact, Human Factors 37 (1995) 16.

[14] B. Freedman, A. Shpunt, M. Machline, Y. Arieli, Depth mapping using projected patterns, Patent Application, uS 2010/0118123 A1 (05 2010).

[15] M. Meehan, Physiological reaction as an objective measure of presence in virtual environments, Ph.D. thesis, University of North Carolina at Chapel Hill (2001).

[16] Z. Zhang, Flexible camera calibration by viewing a plane from unknown orientations, in: Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, Vol. 1, 1999, pp. 666 –673 vol.1. doi:10.1109/ICCV.1999.791289.

[17] A. Maimone, H. Fuchs, A first look at a telepresence system with room-sized real-time 3d capture and large tracked display, in: Artificial Reality and Telexistence (ICAT), The 21st International Conference on, 2011.

[18] K. Berger, K. Ruhl, C. Brümmer, Y. Schröder, A. Scholz, M. Magnor, Markerless motion capture using multiple color-depth sensors, in: Proc. Vision, Modeling and Visualization (VMV), 2011.

[19] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J.-M. Frahm, R. Yang, D. Nister, M. Pollefeys, Real-time visibility-based fusion of depth maps, in: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on, 2007, pp. 1 –8. doi:10.1109/ICCV.2007.4408984.

[20] M. McGuire, A fast, small-radius gpu median filter, in: ShaderX6, 2008.
URL http://graphics.cs.williams.edu/papers/MedianShaderX6/

[21] C. Tomasi, R. Manduchi, Bilateral filtering for gray and color images, in: Computer Vision, 1998. Sixth International Conference on, 1998, pp. 839 –846. doi:10.1109/ICCV.1998.710815.

[22] H. Kawata, T. Kanai, Image-based point rendering for multiple range images, in: Proceedings of the 2nd International Conference on Information Technology for Application (ICITA 2004), Macquarie Scientific Publishing, Sydney, 2004, pp. 478 –483.

[23] A. Ilie, G. Welch, Ensuring color consistency across multiple cameras, in: Proceedings of the Tenth IEEE International Conference on Computer Vision - Volume 2, ICCV '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 1268–1275. doi:http://dx.doi.org/10.1109/ICCV.2005.88.
URL http://dx.doi.org/10.1109/ICCV.2005.88

[24] M. A. Fischler, R. C. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, Commun. ACM 24 (1981) 381–395. doi:http://doi.acm.org/10.1145/358669.358692.
URL http://doi.acm.org/10.1145/358669.358692

[25] Q. Cai, D. Gallup, C. Zhang, Z. Zhang, 3d deformable face tracking with a commodity depth camera, in: Proceedings of the 11th European conference on computer vision conference on Computer vision: Part III, ECCV'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 229–242.
URL http://portal.acm.org/citation.cfm?id=1927006.1927026

[26] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, in: Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, Vol. 1, 2001, pp. I–511 – I–518 vol.1. doi:10.1109/CVPR.2001.990517.

[27] R. Lienhart, J. Maydt, An extended set of haar-like features for rapid object detection, in: Image Processing. 2002. Proceedings. 2002 International Conference on, Vol. 1, 2002, pp. I–900 – I–903 vol.1. doi:10.1109/ICIP.2002.1038171.

[28] M. Castrillón Santana, O. Déniz Suárez, M. Hernández Tejera, C. Guerra Artal, Encara2: Real-time detection of multiple faces at differ-

ent resolutions in video streams, Journal of Visual Communication and Image Representation (2007) 130–140.

[29] G. Ye, A. State, H. Fuchs, A practical multi-viewer tabletop autostereoscopic display, in: Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on, 2010, pp. 147 –156. doi:10.1109/ISMAR.2010.5643563.

[30] T. Peterka, R. Kooima, J. Girado, J. Ge, D. Sandin, A. Johnson, J. Leigh, J. Schulze, T. DeFanti, Dynallax: Solid state dynamic parallax barrier autostereoscopic vr display, in: Virtual Reality Conference, 2007. VR '07. IEEE, 2007, pp. 155 –162. doi:10.1109/VR.2007.352476.

[31] J. Kopf, M. F. Cohen, D. Lischinski, M. Uyttendaele, Joint bilateral upsampling, ACM Trans. Graph. 26. doi:http://doi.acm.org/10.1145/1276377.1276497.
URL http://doi.acm.org/10.1145/1276377.1276497

[32] H. Towles, W.-C. Chen, R. Yang, S.-U. Kum, H. F. N. Kelshikar, J. Mulligan, K. Daniilidis, H. Fuchs, C. C. Hill, N. K. J. Mulligan, L. Holden, B. Zeleznik, A. Sadagic, J. Lanier, 3d tele-collaboration over internet2, in: International Workshop on Immersive Telepresence, Juan Les Pins, 2002.