A View-Dependent, Polyhedral 3D Display

Pawan Harish
harishpk@research.iiit.ac.in

P. J. Narayanan
pjn@iiit.ac.in

Center for Visual Information Technology. International Institute of Information Technology, Hyderabad, India

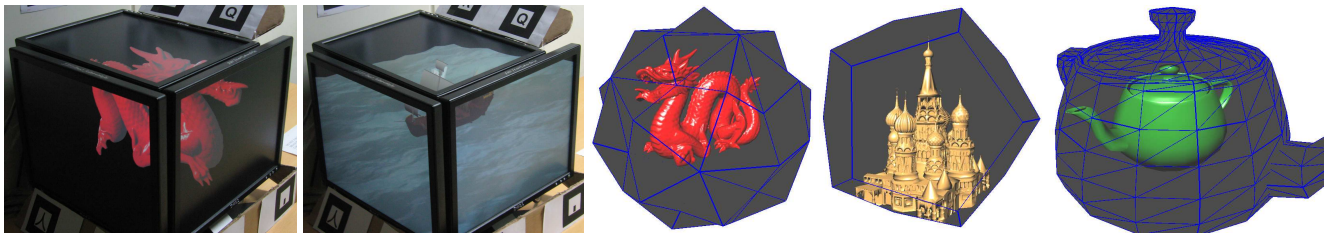


Figure 1: Left to right: Dragon and the moving boat in the 3D cube display. Synthetic polyhedral displays: Dragon in a 120-facet display. Cathedral in a Dodecahedral (12-facet) display and a Teapot model in a Teapot-shaped display with 1024 facets.

Abstract

In this paper, we present the design and construction of a simple and inexpensive 3D display made up of polygonal elements. We use a per-pixel transformation of image and depth to produce accurate picture and depth map on an arbitrary planar display facet from any viewpoint. Though the facets are rendered independently, the image and depth for rays from the eye-point through the facet pixels is produced by our method. Thus, there are no artifacts on a facet or at facet boundaries. Our method can be extended to any polygonal display surface as demonstrated using synthetic setups. We also show a real display constructed using off-the-shelf LCD panels and computers. The display uses a simple calibration procedure and can be setup in minutes. Frame-sequential and anaglyphic stereo modes can be supported for any eye-orientation and at high resolutions.

1 Introduction

We describe the design and realization of a view-dependent, polyhedral, 3D display made from multiple planar display elements in this paper. Producing the correct picture and depth map on an arbitrary planar display element for any viewpoint is at the core of our work. Any polyhedral or piecewise planar display surface can be constructed using it with no artifacts. Displays around which a viewer can walk and immersive displays that engulfs the viewer can both be produced by it. The correct picture and depth are especially critical for multiplanar displays at intersections of planes.

Three dimensional displays are of immense interest. A survey paper from Son et al. [Son et al. 2006] lists various techniques for displaying 3D images and 3D display construction. True 3D or volumetric displays place their pixels in a 3D space, which can be viewed from anywhere without any eye-wear. See the sidebar in the article by Nayar and Anand for a summary of efforts to build 3D displays, dating back to 1948 [Nayar and Anand 2007]. Most of these are severely limited by the technology used and are quite a few years away from being commonplace. Nayar et al. use a glass-etching like process and trade 2D resolution for 3D viewing [Nayar and Anand 2007]. Their display can show colors but are sparse in resolution. The 100 million voxel display simulates 3D by rotating a 2D plane at high speeds in a clear dome [Favalora et al. 2002]. Holographic and auto-stereoscopic displays using lenticular lenses hold promise but haven't been practical. GCubik [Lopez-Gulliver et al. 2008] proposes an autostreoscopic cube display with limited resolution based on the integral photography rendering mechanism. Jones et al. showed a rendering engine for 360° viewing of light fields [Jones et al. 2007]. They showed view-independent 3D for a single interocular displacement direction, but was limited in size and colors and needed a very high speed projector and rotating mirror mechanism.

View-dependent pseudo 3D displays use stereoscopy and need special eye-wear and head-tracking. The presentation could use frame-sequential stereo and shutter glasses, or two projectors with orthogonal polarizations, or red-blue display and glasses, typically on a single monitor or projection screen. The CAVE environment extends this to an immersive view in an inside-to-outside setting [Cruz-Neira et al. 1993]. Our polyhedral display has a lot in common with the CAVE. Multiplanar projection is used by both, with each plane handled independently. The main difference is on the per-pixel adjustment of the image and depth that eliminates joint inconsistencies. Stavness et al. show a cube display [Stavness et al. 2006] based on a method similar to CAVE [Deering 1992]. This approach, however, is not geometrically correct for rendering to such displays as explained in section 2. Multiplanar displays, multi-projector displays, and projector-camera systems all deal with different aspects of how a picture generated for one view can be viewed from another or projected from off-axis [Ashdown et al. 2004; Majumder and Brown 2007]. They use homographies to pre-adjust the images so that it appears correct after projection. Raskar discusses the correctness aspects of projecting to a multiplanar display from off-axis, instead of using a two-pass approach [Raskar 2000]. The two-pass method is expensive and interpolates the image twice. He integrates a planar homography with the projection and suggests an approximation to handle unwanted occlusions due to depth scaling.

We present a method to render the correct image and depth from a viewer's point of view on a multiplanar display. We integrate

the homography from the view plane to the display plane with the rendering pipeline to generate accurate views. This can produce incorrect depth values and ordering. We correct it using a per-pixel transformation implemented using appropriate shaders. Though a view is produced using multiple display elements, depths from the viewer along the view direction are stored in the frame buffer of each display. This guarantees consistent views on the facets and facilitates the construction of arbitrary planar displays that align well at the boundaries. We show results on a real cube display we constructed and on several simulated arbitrary, polyhedral displays in the paper and in the accompanying video.

2 Accurate View-Dependent Rendering

Since the viewer has a position and a look-direction, a symmetric frustum about the view direction will preserve maximum detail. Figure 2 shows the view plane corresponding to such a frustum intersecting with a cube display. The view will be distributed to multiple display elements or facets. Let C_v be the viewer camera, I_v the desired view, and I_i the image shown on facet i. The mapping from I_v to I_i is a planar homography representable by a 3×3 matrix H_{iv}. Each visible display facet will have such a homography.

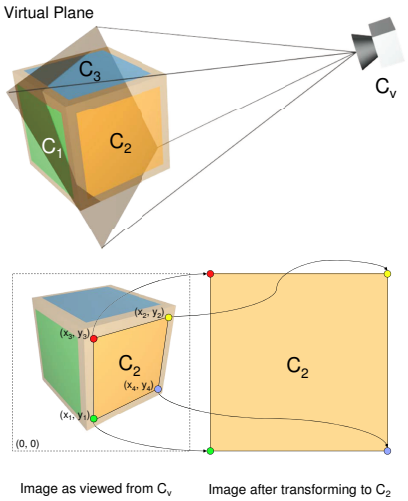


Figure 2: Top: Viewer camera C_v viewing the cube. Bottom: Correspondence between viewer image and display surface for homography computation

Computing facet homography: The 3D coordinates of each facet's corners as well as the position and orientation of the viewer's eye are known in a common frame. The polygonal display area of the facet i can be projected using C_v, giving image points corresponding to facet corners. The match between points in I_v and one facet of the cube display is shown in Figure 2. This establishes a correspondence between the screen coordinates of facet i and the viewer-image coordinates. In addition, the normal indicating the viewing side of the facet is also known. These can be used to compute H_{iv} for facet i so that the image I_i on facet i and viewer image I_v are related using the equation $I_i = H_{iv}I_v$. We use the algorithm outlined in the book [Hartley and Zisserman 2004] to compute homography for each facet.

Projecting with facet homography: Each facet image I_i is rendered independently using an appropriate camera. The image I_i can be generated by applying H_{iv} to the image I_v. This method has its problems. The image I_v must first be rendered completely before applying H_{iv} on every pixel. The homography is not affine and is defined only up to an unknown, per-pixel scale factor. The computed image has to undergo perspective division. To avoid holes in the images, the homography is usually applied in the reverse from I_i to I_v with interpolation to fill non-integer pixels of I_v. This second round of interpolation can result in more blurring. We can avoid the additional computation load and interpolation by integrating the homography with the rendering process.

Homography in the canonical space: The graphics pipeline transforms primitives to the canonical or normalized device coordinates before applying perspective division and viewport transformation. All coordinates vary from $[-1, 1]$ in the canonical space, which is obtained by applying the modelling, viewing, and projection transformations. The homography between I_i and I_v can be computed equivalently in the canonical space of camera C_v. The perspective division and interpolation stages follow the homography in the rendering pipeline. This avoids additional computations as well as the second round of pixel interpolations. This procedure was outlined in the context of correcting for an off-axis projection by Raskar [Raskar 2000].

We compute the homography H_{iv} in the canonical space of I_v and I_i instead of the pixel space shown in Figure 2. The corresponding coordinates of each facet should lie in the range $[-1, 1]$. The homography transformation for each facet can be estimated from the correspondences and applied to the projection matrix. The process of transforming a scene $\mathbf{X}$ to the facet i can now be given by the equation $I_i = VP_dH_{iv}PM_v\mathbf{X}$, where P_d represents perspective division transformation and V the viewport transformation. P and M_v are the projection and modelview matrices of the camera C_v.

2.1 Depth Correction

The above scheme generates the correct image for each facet in a single pass. However, the final depth values may not lie in the range $[-1, 1]$. This effect can be understood as follows for a point (X_c, Y_c, Z_c) in the camera frame.

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ 1 \end{bmatrix} \stackrel{P_d}{=} \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & 0 & h_{13} \\ h_{21} & h_{22} & 0 & h_{23} \\ 0 & 0 & 1 & 0 \\ h_{31} & h_{32} & 0 & 1 \end{bmatrix} \begin{bmatrix} I_x \\ I_y \\ I_z \\ I_w \end{bmatrix} = H \begin{bmatrix} A & 0 & B & 0 \\ 0 & C & D & 0 \\ 0 & 0 & E & F \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (1)$$

The final depth value z'' after applying the homography is in general not in $[-1, 1]$ as $I_z \in [-1, 1]$. The relative ordering may also be affected since w' is a function of I_x and I_y. Applying a uniform z-scale factor $s < 1$ to all points can bring the depths within the range, but not guarantee correct ordering (Figure 4(a)). Raskar [Raskar 2000] suggests a scale factor of $(1-|h_{31}|-|h_{32}|)$. This doesn't solve the problem and severe artifacts can occur due to reduced depth resolution ("z-fighting") and near plane clipping. Multiplanar displays can have serious artifacts at the junctions of display planes by using this method as shown in Figure 4(b).

CAVE [Cruz-Neira et al. 1993] uses the oblique frustum projection approach which can be used in our case. The method would be to render each facet using a view plane parallel to the facet and oblique frustum boundaries (Figure 3). Such rendering will have high depth errors as the scene resides in the corners of a frustum with a wide field of view(*fov*) [Akeley and Su 2006]. In the inside-out display configuration of CAVE the *fov* is rarely large enough to see these artifacts, but not so for an outside-in configuration that we use, as seen in Figure 4. This results in poor z-resolution at facet boundaries and produce visible artifacts in the display as seen in Figure 4(c).

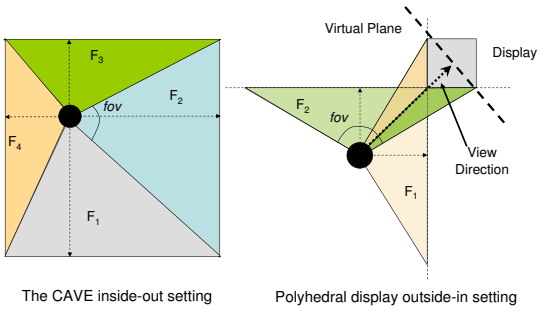
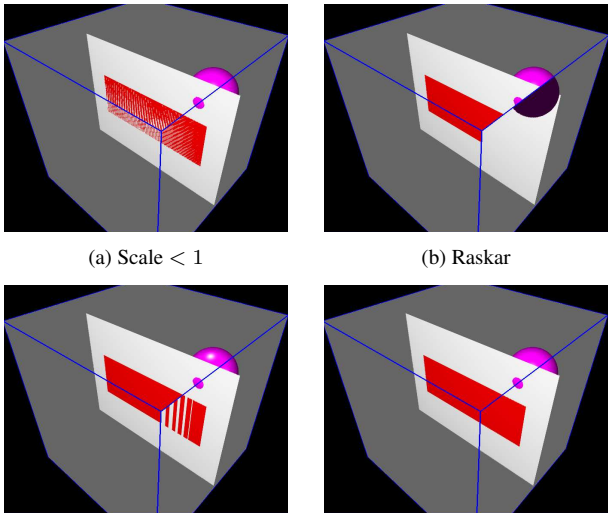


Figure 3: The CAVE and polyhedral display setups with off-axis frustas. Note the large *fov* and corner viewing for polyhedral display using this approach



(a) Scale < 1 (b) Raskar

(c) CAVE (d) Our method

Figure 4: Solving the depth problem: comparison of approaches

Per-Pixel Depth Computation: We solve the problem exactly by setting the depth values for each pixel as:

$$z''' = \frac{I_z}{I_w} = \frac{z'}{I_w} = \frac{z'}{-Z_c} \in [-1, 1]. \quad (2)$$

Thus, the depth buffer for each facet will have the depths from the viewer camera C_v. The image of a facet is modified by its homography, however. The depth resolution is exactly same on all facets, ensuring uniformity at the junctions.

We change the depth of each pixel in a fragment shader. The shader has access to z'. The z-coordinates of the polygon vertices are sent by the vertex shader. The rasterizer interpolates it and makes Z_c available to each pixel. The shader computes z''' as a ratio of these two and sends it to the framebuffer. The image is rendered correctly since the projection matrix is pre-multiplied by the homography. Figure 4(d) shows the correct picture with perfect alignment at a facet boundary for the same view.

2.2 Rendering Algorithm

The multiplanar rendering algorithm first finds the visible faces and computes the homography from I_v to each. The scene is then rendered for each facet. The modelling and viewing transformations are the same for all facets; the projection matrix P_i for facet i is set as $H_{iv}P_v$. The process is outlined in Algorithm 1 along with the vertex and fragment shaders.

Algorithm 1 PolyhedralView

```
{Main algorithm}
while Rendering do
    GetViewerPosition()
    M_v ← Modelview matrix, P ← Projection matrix
    for each visible face i of the display do
        Set Modelview matrix for face i as M_v
        H_iv ← ComputeHomographyForFace(i, P, M_v)
        Set Projection matrix for face i: P_i ← H_iv P
        Render the scene using vertex and fragment shaders
    end for
end while

{Vertex_Shader(V)}
Perform fixed-pipeline operations for vertex and lighting
Compute camera space vertex coordinates V_c
Send V_c to pixel shader with interpolation
Set z coordinate of output vertex as −1 for Early-Z safe

{Fragment_Shader(V)}
Perform fixed-pipeline operations for color
Transform V_c to canonical space as V_c'
Divide z coordinate of V_c' by z coordinate of V_c, set it as depth.
```

The vertex shader sends camera-space vertex coordinates down the pipeline for interpolation. Our scheme stores viewer-camera depth values in the Z-buffer as explained above. The depth range due to the projection matrix P_i for facet i lies in another range and should not be compared with the stored values. We, therefore, disable early-Z culling by assigning a constant depth of -1 to all vertices. The rasterization is not affected by it and the correct depth values are computed later by the fragment shader. The fragment shader gets the camera coordinates of its 3D point, which is transformed to $z' = I_z$ (Equation 1) using the current projection matrix. Dividing I_z by the camera space Z gives the viewer-camera depth for the fragment, which is sent down the pipeline.

3 Display Construction

We now describe the construction of a general polyhedral display with special reference to an inexpensive cube-display. Our scheme can use any piecewise-polygonal arrangement of display elements. The display geometry is specified using a model, with vertices, polygons, and normals. The above algorithm is used to render to each facet independently. There are no artifacts at the boundaries as the rendering is geometrically correct. The corner coordinates of the display elements in a global coordinate system is all that is needed, along with the viewer location.

The displays can be made out of LCD panels or projection screens. LCDs are cheaper, more accessible, and easier to setup. The LCDs should be setup in the desired configuration and its geometry given as input to the system. Any head-tracking mechanism can be used. We try the inexpensive ones: the Wiimote and a webcam with ARToolkit. ARToolkit [ARToolkit 2002] allows greater flexibility and only needs one marker in view at any time. Large working volume and sufficient robustness is obtained with multiple markers.

3.1 Cube Display

We illustrate the construction of a 3D display cube. Our method can easily be scaled to any polygonal shape. The cube is made up of up to 5 LCDs. We use off the shelf LCDs to construct the display shown in Figure 5. We can take a fixed geometry file to specify the

display or infer it using a calibration step.

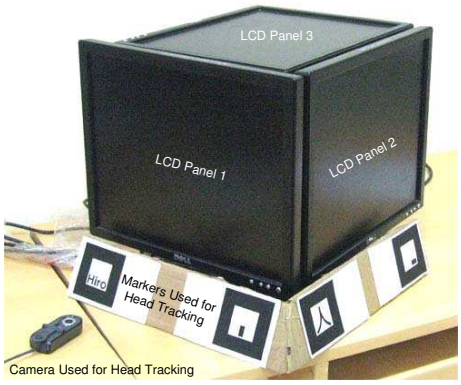


Figure 5: The 3D cube display with markers and the webcam.

Calibration: Calibration is the process of establishing a common reference frame for the display as well as the locations of the corner points of each facet in it. We calibrate the cube using a simple procedure. The dimensions of the display area of the LCD panel are known. Each panel has a pair of unique markers attached rigidly to it for the use of ARToolkit. One marker of one of the facets is designated as the origin. We first establish the transformations to each facet's markers by moving the camera slowly around the setup so that adjacent pairs of facets are visible in several frames. In the next step, the transformation between the corners of each facet's display area and its markers is calculated independently. This is done by displaying an ARToolkit marker at the very center of the screen. This helps recover the plane of the display and its center point. Combined with the given dimensions, the facet's corners are now fully known. The cube can be calibrated in less than a minute, with no special hardware or equipment. We also provide tools to interactively adjust the calibration parameters, if the automatic process is not sufficient. Planes can be adjusted interactively to correct their positions. This procedure can be extended to a general polygonal display. If each facet is considered independent – as when built from independent LCDs, – markers need not be fixed on each. One set of markers will suffice for a set of facets if they follow a known rigid configuration.

Mono and Stereo Display: Our display can be turned into a 3D display by generating left and right-eye views for each facet and using a stereoscopic image delivery mechanism. The rendering technique presented in Section 2 can do this easily. Our scheme places no restrictions on the head pose; the head can be oriented in any angle at any time. This is in contrast to the recent 3D display [Jones et al. 2007] that supports only one orientation at a time due to the basic construction issues. We use anaglyphic stereo using red/blue channels and matched glasses for the inexpensive 3D display. Frame-sequential or shutter-glass based display can be constructed using high-frequency LCDs and shutter glasses.

Rendering to Multiple Facets: The rendering load grows linearly with the number of facets as the scene needs to be rendered for each of them. Our low-cost setup uses distributed rendering, with each facet connected to a entry-level client machine controlled by a medium-capability server. The scene is replicated at the client and the server. The viewer location is available to the server, who sends it to each client. Each client also gets its homography H_{iv} and renders the scene into the back buffer as described in Section 2. The server waits for all clients to complete rendering and issues a request for all of them to swap their buffers at the same time. This establishes adequate swap synchronization with no genlock and is sufficient visually for monoscopic and anaglyphic display modes. This, however, will not suffice for shutter-glass stereo. Our medium-cost setup has two GPUs on a single machine connected to four facets. This provides genlock and framelock and is sufficient for all stereo display modes. The machine has to be of moderate capacity as it has to render the scene four times. Layered rendering introduced in Shader Model 4 GPUs can reduce the rendering load by using common vertex and geometry processing and separate fragment processing for each facet.

Details of the 3D Cube: The display we built (Figure 5) uses ordinary LCD panels and anaglyphic stereo system. The markers used to track user's head can be seen in the picture. We used a PC with two Nvidia Quadro FX 5600 cards to drive a 4-panel cube display. Images from different point of views of this setup are shown in Figure 6 and in Figure 8 on the last page. Shutter-based stereo using Nvidia 3DVision glasses can be built using high-frequency LCDs as the GPUs are genlocked. The client-server method was used for a 5-panel display. Less than half a frame delay was introduced by this arrangement, which was not visible. This display supported anaglyphic stereo display and monoscopic walk-around display. The video shows the display from the viewer's and a third person's points of view. The LCDs have visible and thick borders, which affect the quality of view. However, the display areas are modelled correctly. Thus, the borders appear like supporting bars of the box in which the object is kept.

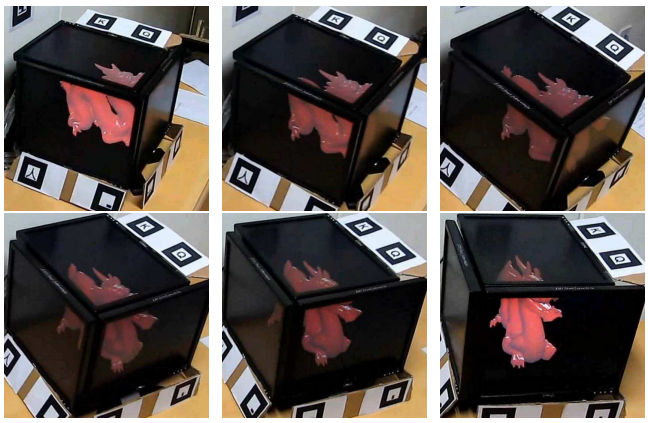


Figure 6: 3D cube display displaying the Stanford Dragon Model

3.2 Synthetic Displays

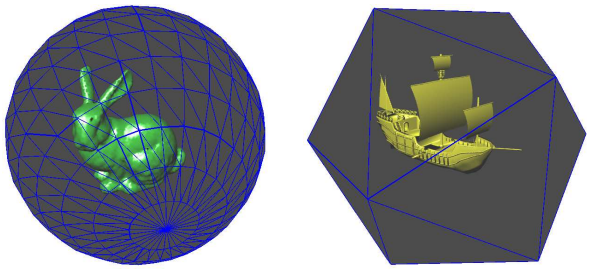


Figure 7: Bunny on a Spherical and ship on an Icosahedral display

Our method can render to any polygonal display geometry correctly. Figures 1 and 7 show displays shapes like a Dodecahedron, Icosahedron, a Sphere with 840 facets and a triangulated Teapot model with 1024 facets. The accompanying video shows more examples. The video shows the view rendered directly to the viewer

camera on the upper right. The image at the center is generated using our method on a simulated display. The image on each facet is generated independently using $H_{iv}P$ as the projection matrix and depth correction as described in Section 2. These are drawn on a wire mesh with the facet geometry. The top-left image is a flattened or opened-out view of the display. The facet-images and arrangement can be seen in it. The per-pixel correction of the image and depth guarantees correct rendering in each case.

3.3 Limitations of the Display

The main drawback of our scheme as a 3D display is its view-dependence. Correct 3D is seen only from one viewpoint. Other viewers will see a distorted scene due to the viewer specific homography being applied. This is a limitation common to most view-dependent 3D displays. The inexpensive cube display we made has several limitations, though it is low-cost and can be put together quickly. The borders of the LCDs create fixed obstructions or "bars" that are distracting. The viewer tracking needs to be more robust. Also, correction of colors across facets may improve the appearance. Lack of genlock mechanism prevents the use of frame-sequential stereo beyond the 4 panels driven by synchronized Quadro GPUs. The basic method we presented, however, can be used to construct arbitrary polyhedral, 3D displays using LCDs with no or low borders, good head-tracking, and proper synchronization of the images. Since, each facet is rendered independently, a proportionate number of rendering stations are also necessary, increasing the cost.

4 Conclusions and Future Work

We presented a design of an arbitrary polyhedral display with correct image and depth and the construction of an inexpensive cube display in this paper. Our design is especially attractive to the low-cost segments. Commodity LCDs and graphics hardware as well as inexpensive head-tracking are sufficient to construct such a display. The simple calibration technique enables the setting up of a cube display in a matter of a few minutes. The 3D display our scheme enables has high resolution and full color. It can be built to any size and can support any orientation of eyes for stereoscopy. The scheme we presented can be used to drive any polygonal display surfaces. Single displays with a multiplanar surface can easily be constructed as a single unit today. We would like to explore rendering images correctly to such a display without being view dependent.

References

AKELEY, K., AND SU, J. 2006. Minimum triangle separation for correct z-buffer occlusion. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, 27–30.

ARTOOLKIT, 2002. http://www.hitl.washington.edu/artoolkit/.

ASHDOWN, M., FLAGG, M., SUKTHANKAR, R., AND REHG, J. M. 2004. A flexible projector-camera system for multi-planar displays. In *Proc of IEEE Computer Vision and Pattern Recognition*, 165–172.

CRUZ-NEIRA, C., SANDIN, D. J., AND DEFANTI, T. A. 1993. Surround-screen projection-based virtual reality: the design and implementation of the cave. In *SIGGRAPH*, 135–142.

DEERING, M. 1992. High resolution virtual reality. *SIGGRAPH Comput. Graph. 26*, 2, 195–202.

FAVALORA, G. E., NAPOLI, J., HALL, D. M., DORVAL, R. K., GIOVINCO, M., RICHMOND, M. J., AND CHUN, W. S. 2002. 100 million voxel volumetric display. In *Proceedings of the SPIE*, vol. 4712, 300–312.

HARTLEY, R. I., AND ZISSERMAN, A. 2004. *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press.

JONES, A., MCDOWALL, I., YAMADA, H., BOLAS, M. T., AND DEBEVEC, P. E. 2007. Rendering for an interactive 360degree light field display. *ACM Trans. Graph. 26*, 3.

LOPEZ-GULLIVER, R., YOSHIDA, S., YANO, S., AND INOUE, N. 2008. gCubik: a cubic autostereoscopic display for multiuser interaction: grasp and group-share virtual images. In *SIGGRAPH Posters*, ACM, 133.

MAJUMDER, A., AND BROWN, M. S. 2007. *Practical Multi-Projector Display Design*. A K Peters Ltd.

NAYAR, S. K., AND ANAND, V. N. 2007. 3d display using passive optical scatterers. *IEEE Computer 40*, 7.

RASKAR, R. 2000. Immersive planar display using roughly aligned projectors. In *VR '00: Proceedings of the IEEE Virtual Reality 2000 Conference*.

SON, J., JAVIDI, B., AND KWACK, K. 2006. Methods for displaying three-dimensional images. In *Proceedings of the IEEE*, vol. 94, 502–523.

STAVNESS, I., VOGT, F., AND FELS, S. 2006. Cubee: a cubic 3D display for physics-based interaction. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, 165.

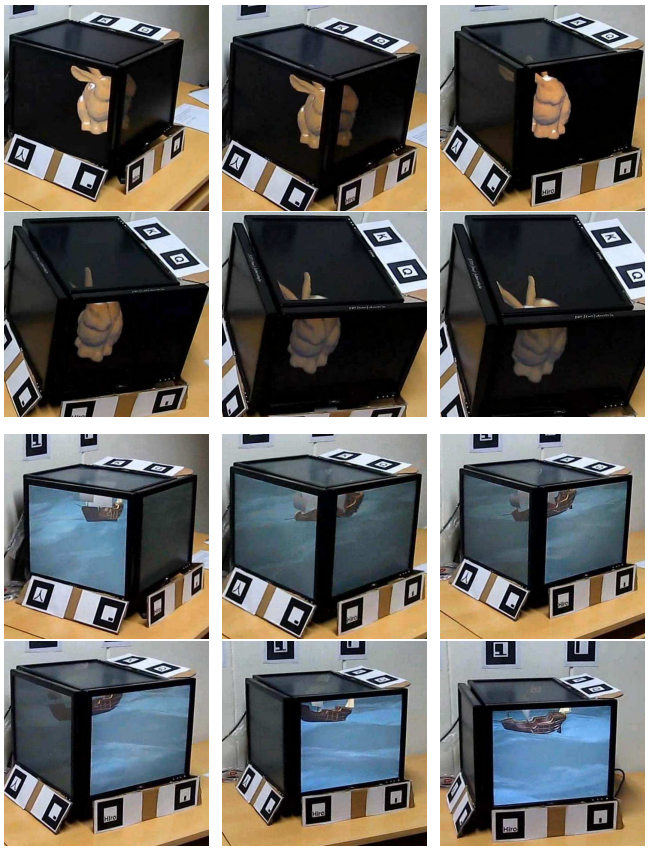


Figure 8: 3D cube display displaying the Bunny and the Dynamic Ship and the Sea models