Designing Perspectively Correct Multiplanar Displays

Pawan Harish and P.J. Narayanan

Abstract—Displays remain flat and passive amidst the many changes in their fundamental technologies. One natural step ahead is to create displays that merge seamlessly in shape and appearance with one's natural surroundings. In this paper, we present a system to design, render to, and build view-dependent *multiplanar displays* of arbitrary piecewise-planar shapes, built using polygonal facets. Our system provides high quality, interactive rendering of 3D environments to a head-tracked viewer on arbitrary multiplanar displays. We develop a novel rendering scheme that produces exact image and depth map at each facet, producing artifact-free images on and across facet boundaries. The system scales to a large number of display facets by rendering all facets in a single pass of rasterization. This is achieved using a parallel, perframe, view-dependent binning and prewarping of scene triangles. The display is driven using one or more target *quilt images* into which facet pixels are packed. Our method places no constraints on the scene or the display and allows for fully dynamic scenes to be rendered interactively at high resolutions. The steps of our system are implemented efficiently on commodity GPUs. We present a few prototype displays to establish the scalability of our system on different display shapes, form factors, and complexity: from a cube made out of LCD panels to spherical/cylindrical projected setups to arbitrary complex shapes in simulation. Performance of our system is demonstrated for both rendering quality and speed, for increasing scene and display facet sizes. A subjective user study is also presented to evaluate the user experience using a walk-around display compared to a flat panel for a game-like setting.

Index Terms—Nonrectangular displays, fish tank virtual reality, arbitrary shaped displays, 3D visualization, view-dependent rendering, parallel culling, user interaction

1 Introduction

Displays have transformed rapidly over the past few years and have become a common occurrence in our day to day life. They have evolved from the basic CRT to displays based on cheaper and better technologies such as LCD, Plasma, OLED, DLP, etc. Various aspects of a display system have improved including power consumption, color gamut, vertical refresh rate, and pixel resolution. The shape and planarity of a display, however, have not changed. Displays are still flat, inactive, and rectangular stand-alone windows feeding images to an observer. This has restricted displays in many ways, resulting in nonintuitive interactions, lack of focus+context, and lack of 3D viewing, to name a few. Attempts have been made to address these issues in recent times. Most notable are the introduction of touch screens and computer human interaction methods which attempt to make displays more active.

Simultaneously, much work has gone into capturing and creating 3D content. A lot of 3D content—ranging from games to CAD simulations—are viewed by users on computer screens today. New modes of real-time 3D content capture are now possible [1]. Virtual reality, haptics, and computer human interaction all deal with the central idea to seamlessly transition from the real to the virtual world. A necessary extension of this paradigm should include displays that seamlessly merge with the world and are natural to view and interact with. The goal of such a display is to blur the boundary between the real and the displayed objects. This notion has two parts: 1) the display itself being part of the viewer space, and 2) content displayed on it being an accurate depiction of the underlying world. The first condition necessitates displays to approximate arbitrary user spaces. Rendered images on such a display must also be accurate such that the depiction agrees with the natural perspective of the viewer.

In this paper, we present a framework to design, build, and render to *multiplanar displays*, built using multiple polygonal *facets*. We render perspectively correct 3D on to such a display from the viewpoint of a single head tracked viewer. Our framework effectively extends Fish Tank Virtual Reality (FTVR) displays to generic polygonal shapes. Our system scales to arbitrary polygonal shapes, which sets it apart from other FTVR displays which are limited to a few facets. Rendering to FTVR displays has a tradeoff between rendering quality and the number of facets. Projective Texture Mapping (PTM) can produce FTVR effect on any surface, but suffers from quality degradation [2]. Off-axis rendering can produce quality images at higher rendering costs, however, suffers from depth artifacts [3].

We present a novel rendering scheme that produces correct images on and across planar facet boundaries, generating a collective view of the virtual world. The rendering cost is reduced using a parallel binning of scene triangles to the display facets, implemented on commodity

• *The authors are with the Center for Visual Information Technology, International Institute of Information Technology, Gachibowli, Hyderabad, Andhra Pradesh 500032, India.*
E-mail: harishpk@research.iiit.ac.in, pjn@iiit.ac.in.

Manuscript received 10 Oct. 2011; revised 2 Apr. 2012; accepted 24 May 2012; published online 5 June 2012.
Recommended for acceptance by A. Steed.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2011-10-0248.
Digital Object Identifier no. 10.1109/TVCG.2012.135.

GPUs. We also render all facet images in a single pass of rasterization, generating one or more *quilt images*. The quilt image is unmapped to the display geometry either using multiple VGA outputs or using specialized hardware for the same. Our system is implemented on commodity GPUs and can scale to displays consisting of well over a thousand facets. Our design provides both the quality and the interactivity needed in applications such as medical visualization, computer aided design, simulations, and games. The framework contributes 1) a system to drive a generic multiplanar display through one or more rectangular quilt images which can be generated by a standard graphics system retaining pixel density, 2) a method to correctly render images to a set of arbitrarily oriented planar shapes to provide a perspectively correct image as well as depth across facets boundaries, and 3) a scalable, single-pass rendering mechanism that sorts and prewarps the scene triangles mapping to each facet, enabling rendering of all facets in a single pass of rasterization. Interactive rendering rates and correct facet rendering are the guiding concerns in designing our system. Our system is capable of rendering fully dynamic, deformable, triangulated scenes containing over 200K triangles to a display shape consisting of over 1,600 facets at 35 Hz. The setup uses a single Nvidia GTX580 GPU for rendering and also for parallel sorting.

The rest of this paper describes our framework. Section 2 presents the related work covering similar technologies to the one presented in this paper. Section 3 provides the detailed implementation of our system. The overall system design is presented followed by rendering and quality evaluation as compared to other rendering methods (Section 3.1). Scalability to arbitrary shapes is presented in Section 3.2. We validate our framework using several prototype displays, using LCD panels, projected setups, and synthetic simulated scenarios in Section 4.1. In Section 4.2, we evaluate the individual aspects of our pipeline in terms of rendering speeds. We also present a user study showcasing the utility of an arbitrary shaped walk-around display in Section 5.

2 Related Work

Our work is related to several previous efforts including multiplanar, nonplanar, curved, and FTVR displays. We review the literature related to these in this section, segregated into multisurface displays, FTVR displays, and rendering to FTVR displays.

2.1 Multisurface Displays

Multiple display panels using monitors or projected surfaces have been used to increase pixel resolution for various applications [4]. Tiled display walls using LCDs or projectors scale this to extremely high resolutions [5], [6]. Display walls allow focus and context to be achieved simultaneously [7], which is also achieved using nontiled arrangements such as displaying a high-resolution focus window within a lower resolution context [8]. Nonplanar arrangements of displays have also been explored, an example being the fish eye view generated using multiple projectors projecting onto a dome using texture mapping [9], [10]. Bimber et al. [2] extended the two pass texture mapping approach to any surface using stereoscopic projection. Raskar et al. modified object appearance for any given geometry in [11], which is further enhanced using multiple projectors to create high-resolution appearance editing [12]. Such advancements have spawned much work in autocalibrating systems for multiprojector displays [13], both in terms of color and geometric corrections. Other nonplanar displays that enable users to interact with information content from all angles have also been showcased [14], [15], [16]. Multisurface displays usually show information with multiple parts of the display showing different data. Single or multiple users can interact and share the same display using touch gestures [17]. Multisurface displays provide novel interaction and appearance onto arbitrary shapes, but are limited to 2D information content. The mutiplanar display we propose aims to show 3D content accurately in addition to 2D information.

2.2 FTVR Displays

FTVR displays have extended multisurface displays to render volumetric data for one or more head-tracked viewers [18]. An early immersive implementation is the CAVE virtual environment [19] that uses four back-lit projection planes enclosing the viewer. View-dependent images are displayed on the projection screens by tracking the viewer's head. Stereoscopic projection is used to further enhance the experience. Recent FTVR displays include Cubby with three back-lit projection screens and a head tracker for a small closed FTVR environment that allows interaction with virtual objects using precision tools [20], [21]. By inverting the facets of a CAVE, with the facets of the display pointing outward, Cubee creates a walk-around cubic FTVR display [22]. PCubee enhances this to a hand-held display coupled with a motion sensor and a head tracker [23]. The display can then be moved around and observed from any angle. FTVR displays are limited to small number of facets, usually a cube, due to increase in rendering load as the number of facets increase. Iwata implemented a noncubic, rhombic dodecahedral display using projectors to cover the full solid angle [24].

2.3 Rendering 3D Content to FTVR Displays

Off-axis rendering is used to render 3D environments to FTVR displays. Deering proposed accurate head tracking along with stereo image pair generation for a single display [25]. This was extended to multiple planes in the CAVE virtual environment [19]. The method renders each facet using an off-axis, asymmetric frustum to cover the display rectangle. Rendering load increases linearly with the number of facets. The method, though extensively used, can produce geometrically incorrect images at the periphery of the frustum for an outside-to-inside display configuration as discussed in Section 3.1. Correct view can be produced using ray-casting at every pixel for any given display shape albeit at high costs. Hou et al. proposes a multiperspective rendering method for any given surface [26]. Though not intended for FTVR displays it can be extended and employed for the same. The method is implemented on GPUs using appropriate shaders and can handle dynamic scenes. It, however, interpolates barycentric coordinates per pixel and requires back projecting of

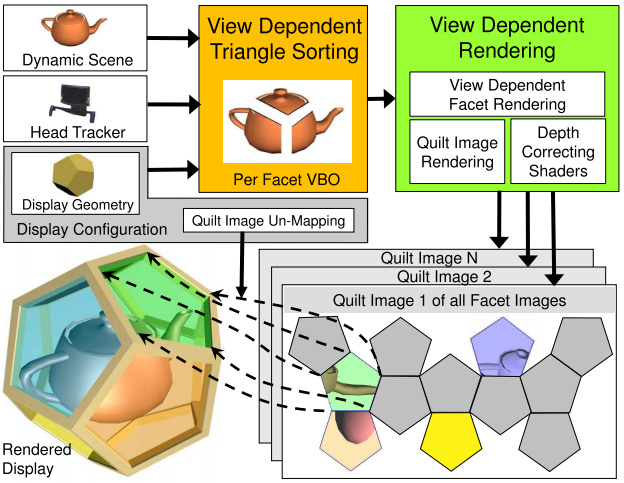


Fig. 1. The multiplanar display system. Top: rendering pipeline. Bottom: driving the display using quilt images.

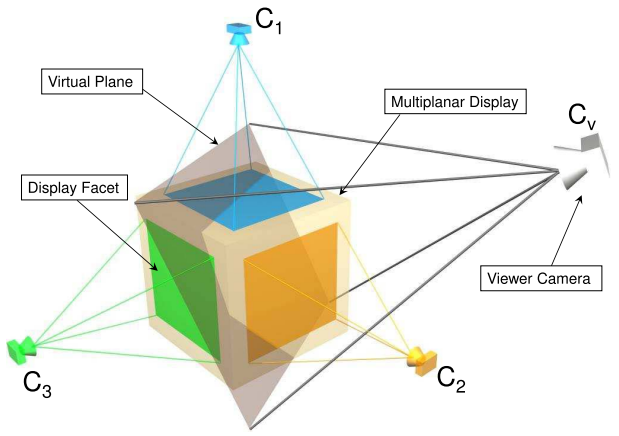


Fig. 2. Viewer camera C_v viewing the display and the *virtual* plane passing through the center of the display. C_1, C_2, C_3 are cameras corresponding to each display facet.

rays at every pixel—which translates to per pixel raycasting and thus slows down the rendering speed for large scenes. Another alternative is to use homography prewarping to render to FTVR displays [27], [28]. Our previous work uses this method to create a polyhedral walk-around display [29]. A scalable culling pipeline is required to feed FTVR rendering methods if arbitrary number of planes are to be used in a display configuration.

3 Multiplanar Displays

We describe the design, scaling, and the rendering to multiplanar displays in this section. A general multiplanar display is piecewise planar, consisting of a number of polygonal facets forming the shape. For instance, one's personal workspace or desk could be a multiplanar display with horizontal, vertical, and slanted facets. In our design, each display facet is built using an LCD panel, micro-projector, or a suitable display mechanism and can show stereoscopic 3D based on the technology used.

A standard polygonal mesh model describes the display geometry in our scheme, with any number, shape, and size of planar facets. Each pixel of the display is addressed using a 3D coordinate (f, i, j), where f is the facet id, ranging from 1 to the number of facets, and (i, j) is the pixel id within the facet. Facets can be controlled independently using a graphics channel for each. This, however, does not scale well to a large number of facets. Multiple facets can be driven together by assigning a mapping from each facet to a rectangular quilt image, generated by a single graphics channel. By segregating facet packing into multiple quilt images quality can be improved further if needed. Unmapping of facet images given the quilt image can be achieved using multiple VGA outputs or specialized hardware. We consider the unmapping to be a part of the display hardware and is defined as part of the *display configuration* along with the facet geometry (Fig. 1). This scheme ensures quality rendering on display facets as the facet resolution dictates the size and number of quilt images needed and not vice versa, thus maintaining pixel density on each facet.

The display configuration, scene to be rendered and the user location are inputs to our system. The system generates the quilt images based on the user location and sends these to the display hardware for unmapping. The process is described in Fig. 1. As the first step, triangles of the scene are sorted in parallel to bins corresponding to each display facet (Section 3.2). This helps scale our rendering mechanism to arbitrary shapes, reducing the rendering load per facet and retaining interactivity of dynamically changing scenes. Next, the scene is rendered using facet-specific homography and per-pixel depth correction, which guarantees a consistent, artifact free view of the scene on the display (Section 3.1). A prewarping-based quilt image generation facilitates single-pass rendering (Section 3.2.2). The system is implemented using CUDA GPUs, which perform both the parallel sorting and rendering in a single pass of rasterization. The presented pipeline generates quality views from the user's perspective at interactive frame rates and can also be used for an inside-to-outside display configuration such as the CAVE.

3.1 Accurate View Dependent Rendering

In this section, we describe a novel scheme to render to multiplanar display facets. Our scheme generates quality images per facet without artifacts, eliminating inconsistencies across facet boundaries. Quality comparison with other rendering methods is also presented.

Given the viewer position and a look-direction, a symmetric frustum about the view direction preserves maximum detail. A virtual viewer camera C_v can be assigned to the viewer for generality. The position of C_v is tracked and known in a common frame of reference to that of the display. A *virtual view plane* normal to the view direction passing through the center of the display can also be assumed without loss of generality, as shown in Fig. 2. The view-dependent virtual plane intersects multiple facets of the display. Each facet f can be assigned a facet camera C_f; its image I_f can then be related to the viewer camera image I_v by a 3×3 planar homography matrix as given below [30].

$$I_f = H_{fv}\, I_v, \qquad \forall \quad f \leq \#\ facets \tag{1}$$

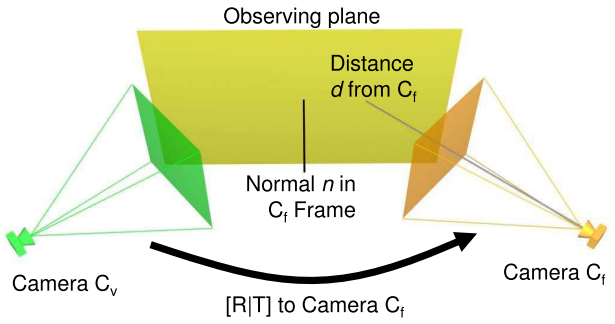


Fig. 3. Computing homography transformation between two cameras.

Each visible facet will have such a homography relating its image to the virtual plane. Rendering I_f collectively using the above equation produces the illusion of the image lying on the virtual plane when observed from C_v. To ensure exact rendering, we render I_f such that the appearance and depth corresponds to C_v at every visible pixel.

Computing facet homography. Given the rotation matrix R and the translation vector T between C_f and C_v, normal n, distance d of the virtual plane, and intrinsic parameters K_f and K_v of the cameras, H_{fv} can be computed directly as $H_{fv} = K_v[R - Tn^T/d]{K_f}^{-1}$ [30] (Fig. 3). We compute the homography for each facet independently in each frame based on the viewer position. Homographies are computed in parallel on the GPU from known information about C_v and C_f using a thread for each facet.

Facet homography in canonical space. Facet images, I_f, can be generated by applying the homography to the viewer image, I_v. This approach has its limitations. It is an expensive per pixel operation and requires I_v to be available ahead of time. It can also produce holes in the facet image as pixels in I_v may not map to any pixel in I_f, and may require additional interpolation to fill. The homography is also not affine, i.e., its last row is not of the form $[0\ 0\ 1]$. This produces an arbitrary homogeneous scale factor per pixel, removing which warrants per pixel normalization of the facet image, I_f. These problems can be solved by integrating homography into the rendering process to perform interpolation and pixel normalization using the graphics pipeline.

The homography can be equivalently computed and used in the normalized device coordinates, canonical space, since the relative positions of pixels do not change after it in the graphics pipeline (Fig. 4). This avoids pixel normalization and interpolation since the perspective division and viewport scaling stages follow the canonical space in the graphics pipeline. This idea was outlined in the context of correcting for off-axis projection by Raskar [28]. The process of transforming a scene point $\mathbf{X}$ to the facet f can now be given as

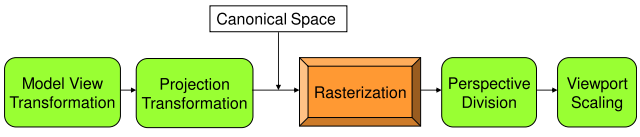
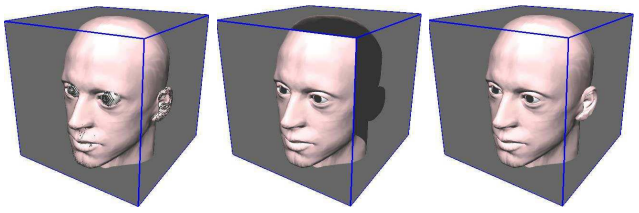


Fig. 4. Graphics pipeline with canonical space. The perspective division and viewport scaling stages only modify the scale of the point in canonical space.



(a) Scale < 1 (b) Raskar (c) Our method

Fig. 5. The depth problem: comparison of approaches. (a) depth errors due to reduced z-range at eye, lip, and ear. (b) error due to near plane clipping artifacts. (c) our method without artifacts.

$$I_f = VP_dH_{fv}PM_v\mathbf{X}, \tag{2}$$

where P_d represents perspective division transformation and V the viewport transformation. P and M_v are the projection and modelview matrices of camera C_v.

Depth correction. Depth values at pixels may go out of the canonical space range of $[-1, 1]$ using the above scheme. For a point $(X_c, Y_c, Z_c) \equiv M_v\mathbf{X}$ in the camera frame when multiplied by projection and homography matrices, the effect can be understood as follows:

$$\begin{aligned}\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} &= \begin{bmatrix} h_{11} & h_{12} & 0 & h_{13} \\ h_{21} & h_{22} & 0 & h_{23} \\ 0 & 0 & 1 & 0 \\ h_{31} & h_{32} & 0 & 1 \end{bmatrix} \begin{bmatrix} I_x \\ I_y \\ I_z \\ I_w \end{bmatrix} \\ &= H \begin{bmatrix} A & 0 & B & 0 \\ 0 & C & D & 0 \\ 0 & 0 & E & F \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}.\end{aligned} \tag{3}$$

The matrix to the left of (X_c, Y_c, Z_c) is a standard OpenGL projection matrix. The final depth value z'/w' after applying the homography may not lie in the $[-1, 1]$ range as $I_z \equiv z'$ (depth without homography) belongs to $[-1, 1]$ and w' is now a function of I_x and I_y. Uniformly scaling the depth by a z-scale factor less than 1 can bring the depths within range, but will not guarantee correct ordering. This can lead to poor depth resolution and push through artifacts as seen in Fig. 5a. Raskar [28] suggests a scale factor of $(1 - |h_{31}| - |h_{32}|)$. This reduces the depth resolution and can suffer from near plane clipping. Multiplanar displays can have serious artifacts at facet junctions using this method as shown in Fig. 5b.

The problem can be solved exactly by setting the depth values at each pixel as

$$z_v = \frac{I_z}{I_w} = \frac{z'}{I_w} = \frac{z'}{-Z_c} \in [-1, 1]. \tag{4}$$

The depth buffer for each facet will have the same depths from the viewer camera C_v. The depth resolution is exactly same on all facets, ensuring uniformity at facet junctions. We change the depth of each pixel in a fragment shader, which has access to z'. The Z values of the vertices are sent by the vertex shader and are interpolated to make Z_c available at each pixel. The shader computes z_v as a ratio of these two and sends it to the framebuffer. Fig. 5c shows the correct image using our scheme. Early-z culling is avoided by setting a constant depth value of -1 at all vertices.

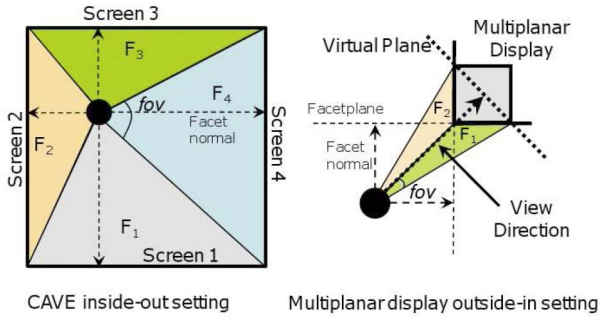


Fig. 6. Top view of CAVE and multiplanar display setups with off-axis frustas. The view angle is large for multiplanar display using this approach.

Rasterization is not affected by this and the correct depth values are computed later by the fragment shader. Algorithm 1 outlines the vertex and fragment shaders used in our rendering scheme.

Algorithm 1. Depth Correcting Shaders

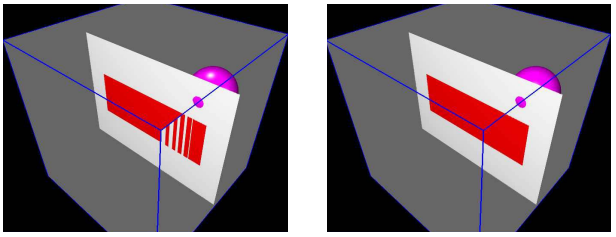
1: **Vertex_Shader(V)**
2: Perform fixed-pipeline operations
3: Compute camera space vertex coordinates V_c
4: Send V_c to pixel shader with interpolation
5: Set z coordinate of output vertex as -1

6: **Fragment_Shader(V)**
7: Perform fixed-pipeline operations for color
8: Transform V_c to canonical space as V_c'
9: Set depth as the ratio of z coordinate of V_c' and V_c.

3.2 Comparison with Other Approaches

We compare our rendering scheme with alternative methods used to render to FTVR displays. The goal of our system is to provide quality rendering at interactive frame rates. The comparison thus focuses on quality, primarily on the facet. Off-axis-rendering renders the scene directly to the facet and thus is capable of producing quality images. It is the primary alternative to our method. Projective Texture Mapping can also be used to generate a similar effect. It produces an intermediate image which is mapped to multiple facets using interpolation. This can lead to quality degradation due to image resampling. A detailed quality comparison of our method with Projective Texture Mapping can be found in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.135.

Off-axis rendering [25] has been used as the rendering mechanism for FTVR displays like the CAVE, Cubby, etc. The method renders each facet using a view plane parallel to the facet using oblique frustum boundaries (Fig. 6). The rendering is not geometrically correct due to depth errors. Depth for each facet varies with the viewer location and is not consistent across facet boundaries. Akeley and Su report that the error increases linearly with the angle of view and is particularly large if the scene resides in corners of the frustum [3]. In a display configuration like the CAVE, this angle of view is small and errors are rarely visible. In an outside-to-inside configuration, the scene always resides in one corner of the frustum and the angles could be large (Fig. 6). This results in visual artifacts that



(a) Off-Axis (b) Our method

Fig. 7. Comparing off-axis projection with our rendering scheme. Scene consists of two proximate rectangles and a push-through sphere.

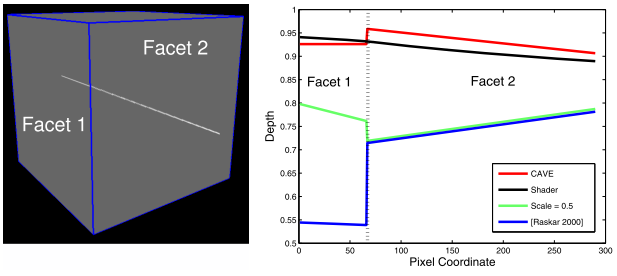
can be seen across facet boundaries. Fig. 7a shows the rendering of two proximate planes using off-axis rendering. Depth errors can be seen as opposed to our rendering scheme (Fig. 7b). Fig. 8 shows depth variation for various methods across a facet boundary for a scene consisting of a single line. We see that only our method ensures consistent views at the junction of two facets. Rendering to Multiplanar displays requires this property to avoid artifacts at facet boundaries.

3.3 Scaling to a Large Number of Display Facets

Current FTVR rendering methods render the scene to each facet independently. This leads to a linear increase in the rendering load with the number of facets and may become impractical beyond a few tens of display facets. We propose two load balancing ideas to aid our rendering scheme in this section: 1) A parallel scene sorting algorithm which sorts the triangles of the scene to the display facets quickly on the GPU per frame. 2) A rendering scheme that renders the quilt of multiple facet images in a single pass of rasterization. These achieve real-time rendering of fully dynamic scenes for displays consisting of thousands or more facets. Methods based on spatial hierarchies have been shown to cull the scene to reduce the rendering load per facet but are hard to extend to dynamic/deforming scenes due to their static design. Because of our parallel design, our method outperforms spatial hierarchies as reported in Section 4.2.

3.3.1 Visibility Determination and Triangle Sorting

We sort the triangles of the scene in parallel to respective facets of the display per frame. This provides flexibility to our system in handling dynamically changing scenes. The parallel implementation also scales well to a large number of



(a) The scene (b) Depth Buffer

Fig. 8. Comparing depth buffers for various approaches at the boundary of two facets. Note no sudden change in depth values for the shader approach, ensuring continuity at facet boundaries.

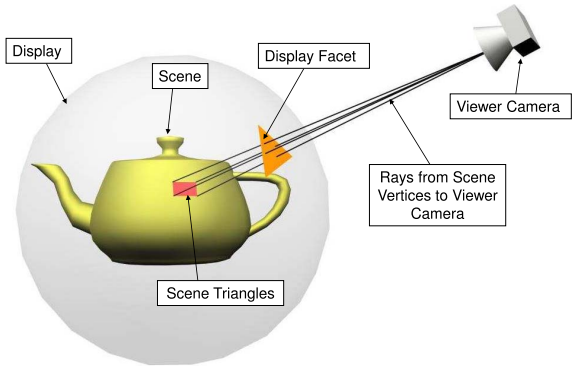


Fig. 9. Triangle sorting based on per vertex raycasting.

display facets. Per vertex ray-casting implemented on the GPU is used to achieve this in real-time. Fig. 9 shows our sorting approach for a large number of facets. A ray is shot from the viewer location to each scene vertex. The intersection of the ray with a given facet of the display determines its facet id. The information is stored on the GPU memory in a temporary array *facetid*. Three situations can arise from per vertex ray casting as shown in Fig. 10. Case 1 is the most frequent case. The second and third cases occur for fewer number of triangles, but require more expensive computations. We divide our sorting into two passes based on this observation. The first pass handles the first case while the second pass deals with the remaining triangles.

Preprocessing. To reduce the number of tests performed per ray, we preprocess the display facets into a static octree structure, with each octree leaf assigned facets it intersects spatially. We use a depth level 3 with 512 leaf nodes, stored as an array in the GPU shared memory for fast access [31]. Each ray first emulates the octree leaf nodes it intersects and performs the relative tests with facets present in these leaves only.

First pass. The first pass assigns a facet id to triangles based on facet ids of its vertices. Each triangle gets assigned a single facet id if all its vertices have the same facet id, as shown in Algorithm 2. The scene triangle id and its facet id are stored as a tuple in a temporary array TFID on the GPU, which is later used to bring all triangles of a facet together. For typical scenes, this pass assigns facet ids to 80 to 90 percent of the total scene triangles. The remaining triangles have different facet ids at their vertices and undergo the second pass. Fig. 11a depicts triangles passing the first pass for the Bunny model on a spherical display.

Algorithm 2. First Pass

1: $tid \leftarrow GetThreadID()$ **{thread id = triangle id}**
2: $count \leftarrow$ size_of(TFID)
3: **if** for all vertices vid of triangle tid, $facetid[vid]$ is same **then**
4: TFID[$count$]$\leftarrow$ (tid, $facetid[vid]$)
5: $count \leftarrow count + 1$
6: **end if**

Second pass. The second pass examines all triangles that remain (Fig. 11b). In this pass, scene triangles and display facets are projected using viewer camera, and a 2D triangle-facet intersection test is performed using Möller's procedure [32]. Overlap test shown in case 2 of Fig. 10 is

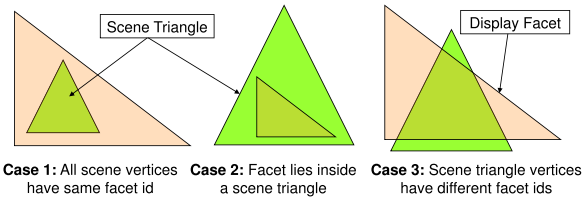


Fig. 10. Cases occurring from per vertex ray casting

also performed. Facet ids are assigned to the triangle if any of the tests are positive. Multiple facet ids may be assigned to a single triangle during this pass. Each such triangle-facet ids tuple is added to the TFID array. Algorithm 3 summarizes these steps.

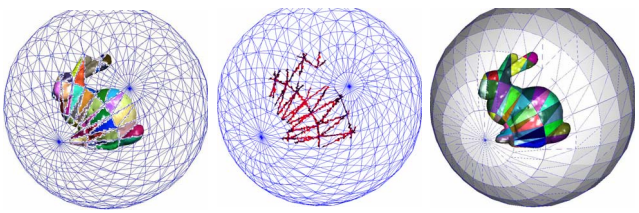
Algorithm 3. Second Pass

1: $tid \leftarrow GetThreadID()$ **{thread id = triangle id}**
2: $count \leftarrow$ size_of(TFID)
3: Project scene triangle using Viewer Camera C_v
4: **for** all facets $fid \in$ Emulated Octree Nodes **do**
5: Project facet fid using viewer camera C_v
6: **if** triangle-facet intersection [32] OR projected facet lies inside projected scene triangle **then**
7: TFID[$count$]$\leftarrow$ (tid, fid)
8: $count \leftarrow count + 1$
9: **end if**
10: **end for**

We also reduce the work done by the GPU for the second pass of the algorithm using thread compaction, ensuring only as many threads are executed on the GPU as the number of residual triangles.

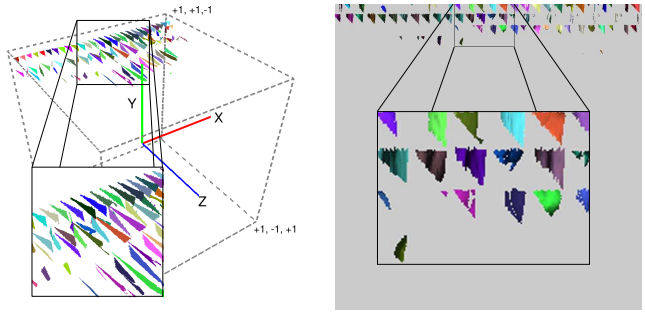
The TFID array holds the triangle-facet id tuples for all possible triangle-facet intersection cases after the second pass. We bring all triangles belonging to a facet together by performing a split [33] operation on TFID using the facet id as the key. A kernel runs over the length of the split TFID array and copies the scene triangle data to the independent VBOs per facet in parallel. VBOs thus created undergo the rendering process reported in Section 3.1 to produce accurate views for each facet as shown in Fig. 11c, with colors indicating different facet ids.

The scene sorting procedure. The overall sorting procedure is described in Algorithm 4. Our algorithm increases the number of triangles to be rendered, replicating triangle data to facets in which they appear during the second pass, but this increase is small in practice and empirically the rendering is faster than both spatial hierarchies and independent rendering for each facet as



(a) First Pass (b) Second Pass (c) Rendering

Fig. 11. The stages of our triangle sorting algorithm.



(a) Scene in Canonical Space (b) Rendered Quilt Image

Fig. 12. The canonical space triangle separation and the corresponding rendered quilt image

reported in Section 4.2. An average increase of 4-5 percent is seen for the number of triangles.

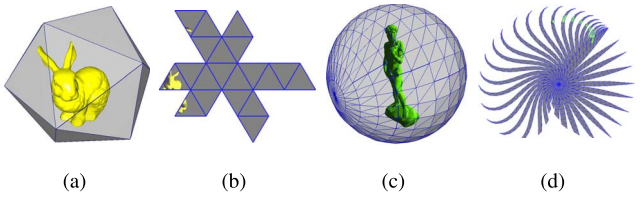
Algorithm 4. The Scene Sorting Algorithm

```
1: for all scene triangles in parallel do
2:     Perform the first pass
3:     Compact triangle ids that failed the first pass
4:     Perform the second pass
5:     Store scene triangle-display facet pair in TFID
6: end for
7: Split TFID using display facet as the key
8: for all entries of TFID in parallel do
9:     Copy scene triangle data to display facet id VBO
10: end for
```

3.3.2 Rendering the Quilt Image for all Display Facets

In this section, we describe a method for reducing the facet image rendering load for a large number of facets. Rendering to each facet independently requires a setup phase of the graphics pipeline. This overhead can increase significantly as the number of facets increase. For speed, we render all facet images in a single pass of rasterization. A tradeoff exists between the required resolution and number of rasterization passes, as high facet resolution can dictate facet packing into multiple quilt images—each requiring a separate rasterization pass. An optimal packing is thus needed to utilize the quilt image space efficiently.

From (2), we observe that each display facet has a different projection matrix ($H_{fv}P$). The difference in projection matrix warrants independent rasterization, setting independent viewports and projection matrices for each facet. This can slow the rendering as the number of facets increase. We can avoid this by prewarping each point $\mathbf{X}$ for each facet. The mapping for a facet image f in the quilt image is a fixed 2D transformation (Fig. 1). Let Y_f denote this in the canonical space. The point $\mathbf{X}$ per facet is modified by the facet-specific quilt image mapping (Y_f), homography (H_{fv}), and the projection and modelview matrices (PM_v) to $\{Y_f H_{fv} P M_v \mathbf{X}\}$. This modified point can be thought of as a point in another canonical space with its modelview and projection matrices set as identity. The point in this space depends on facet id f, but the pipeline parameters are same for all facets. This enables the same rendering pipeline to render all facets to the quilt image in a single pass of rasterization using



(a) (b) (c) (d)

Fig. 13. Figures (a) and (c) show the display as seen from user's point of view with their respective BFS expanded quilt image given in Figures (b) and (d).

$$Quilt = V P_d I_{M_v} I_P \sum_{f=1}^{\#facets} Y_f H_{fv} P M_v \{VBO_f\}, \qquad (5)$$

where V is the viewport for the quilt image, P_d the perspective division stage and I_{M_v} and I_P the modelview and projection matrices respectively set to identity.

A vertex $\mathbf{X}$ in the list of facet f can be prewarped to $Y_f H_{fv} P M_v \mathbf{X}$ after the scene sorting stage. The facet VBOs created at the end of the sorting phase are thus modified while copying the data from the TIFD array. Fig. 12a shows such a transformed scene in the canonical space, with colors indicating facet ids. It can be seen that VBOs deform under these transformations. However, projecting them to the quilt image generates the correct image per facet as shown in Fig. 12b. For a display with 1,600 facets, a 15-20 percent decrease in rendering time is seen using this method compared to setting up the graphics pipeline for each VBO at the same facet resolution.

3.3.3 Mapping and Unmapping of Facets In the Quilt Image

Facet mapping transformation Y_f is a 2D transformation matrix per facet that maps the facet pixels (f, i, j) to the pixels in the quilt image (u, v) as shown in Fig. 1. This can be done without any interpolation using equal number of pixels in each. Y_f differs from texture mapping as a result. Optimal mapping of polygonal shapes into a 2D plane is an NP-hard problem [34]. We can use a heuristic mapping arrangement that attempts to utilize the quilt image space optimally. This is a preprocessing step done while designing the display configuration. A simple mapping algorithm would be to assign a bounding rectangle to each facet polygon and pack them in a linear order across the dimensions of the quilt image, as used in Fig. 12. This scheme may not utilize the image space efficiently. Better packing using a BFS tree can be used to generate a tighter quilt image as shown in Fig. 13.

Unmapping the quilt image to display facets is inverting Y_f, i.e., mapping (u, v) pixel in the quilt image to a facet pixel (f, i, j). Given Y_f this can be achieved either using VGA outputs or specialized hardware. For example, for our LCD prototype (Fig. 15), we render a large image comprising of 4 rectangles with same resolution as each LCD facet, placed in each corner. The display driver is configured to map these regions to the LCD displays using VGA outputs as used in multiheaded displays. Our system, however, treats the entire region as the quilt image. For generic shapes, the mapping is more involved. Electronics needed to send pixels to facets using the display configuration is easy to build. This can best be thought of as part of the

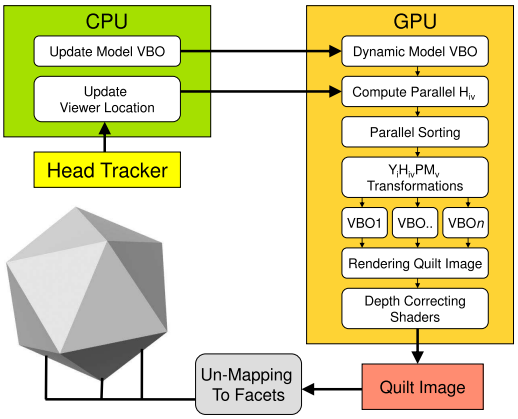


Fig. 14. The overall rendering process of our multiplanar system.

display hardware, such that a rectangular quilt image can drive any multiplanar display.

3.4 The Rendering Pipeline

Fig. 14 describes the overall rendering process for our multiplanar display system. The scene being displayed can be completely dynamic since triangle sorting and VBO creation takes place every frame. We can use one or more commodity GPUs for the processing and rendering, keeping the cost low. Our setup allows us to render scenes with up to 200K triangles in real time to a display made up of up to 1,600 facets on current generation GPUs. For larger scenes the framerate drops to below interactive rates as shown in Fig. 21. Multi-GPU solutions can be used to enhance the speed in such cases, as the problem is parallel friendly. Distribution of the quilt image to physical facets can be done using appropriate electronics in the display system [23] or using a system with multiple display channels and coordinated rendering. We use VGA outputs and texture mapping for our prototype displays.

4 Displays and Performance

In this section, we describe prototype displays that we built and simulated to validate our multiplanar display framework. We also give performance results for different form factors comprising of varying number of facets and orientations along with scene complexity to study the scalability of the framework across different factors.

4.1 Display Prototypes

Building a multiplanar display of a general shape requires engineering at the display device level that only a display company can undertake. The objective of considering different prototypes is to establish 1) the generality of the multiplanar display framework, 2) the correctness of displaying 3D information using our method on challenging display configurations, and 3) the scalability of the approach to arbitrarily large displays. We use three types of display prototypes for this. Examples showing dynamic 3D scenes and information on our prototypes can be found in the accompanying video, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.135.

Fig. 15. LCD-based display showing various static and dynamic scenes.

4.1.1 LCD-Based Setup

We prototype a cube display using off the shelf LCD panels with up to 5 display facets located around the cube. The prototype is similar to the Cubee [22], but uses our rendering scheme as opposed to off-axis rendering. The display follows our rendering process with facet mapping achieved using VGA outputs. The cube is setup by loosely placing LCD panels in the desired configuration. Off the shelf LCDs are used to construct the display shown in Fig. 15. We can take a fixed geometry file to specify the display or infer it using a calibration step. Viewer location is tracked using an infrared based head tracker, TrackIR5 [35], intended for use in gaming applications. With a refresh rate of 120 Hz the latency is minimized and the location of the viewer is given within an error of 5-10 percent.

Calibration. We calibrate the cube using a simple procedure based on ARToolkit [36] markers. We establish transformations from an origin marker to facet markers using a camera. This helps recover the plane of the display and its center point. Combined with the display 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. The procedure can be easily extended to a general polygonal display.

Brightness correction. Brightness/color correction is an important issue for tiled display setups [37]. For LCDs, intensity and colors fadeout with increasing viewing angle. We use a simple method to compensate for this; we change intensity of pixels based on the dot product of the facet normal and the view vector with maximum intensity at oblique viewing angles. The scheme produces a satisfactory visual experience.

Hardware details of the 3D cube. We used a PC with two Nvidia Quadro FX 5600 cards to drive a four-panel cube display. The display supports anaglyphic stereo display and monoscopic walk-around display. Shutter-based stereo using Nvidia 3DVision glasses can be built using high-frequency LCDs as the GPUs are genlocked. The LCDs have visible and thick borders, which affect the quality of view. However, the display areas are modeled correctly. Thus, the borders appear like supporting bars of the box in which the object is kept (Fig. 15).

4.1.2 Projection-Based Setups

Since LCD-based curved surfaces consisting of thousands of facets are hard to physically implement, we show the scalability of our approach using projection-based setups. These setups use our pipeline to generate the quilt image by following the stages of parallel sorting, facet rendering, quilt image rendering, and depth correcting shaders. The inverse mapping stage is absent as there is no hardware

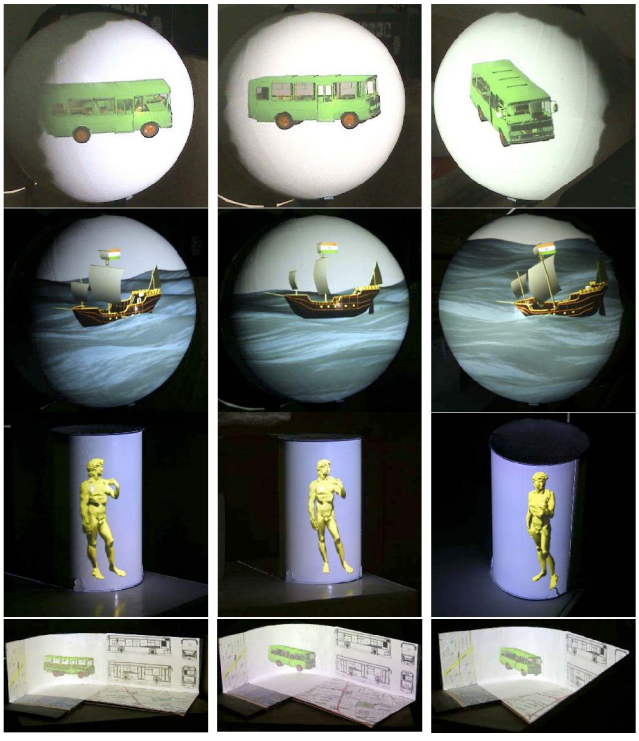


Fig. 16. Projection-based setups showing various scenes on sphere, cylinder, and desktop form factors

unmapping for these setups. We replace this step with texture mapping to generate the final output. It should be noted that though we use texture mapping in these setups, a physical display will not require this, as images will be directly mapped to facets. The overall display resolution is also affected because of this, since the final image is shown using an off-the-shelf projector, of which only about 40 percent pixels lie on the given shape.

To maintain the resolution quality, as if built using LCD panels, we render all projection-based setups to a constant quilt image resolution of 64 mega pixels (maximum texture size supported on current GPUs). This can be further improved by changing facet mapping and using more than one rasterization pass. These prototypes are intended to demonstrate the scalability of our system to various form factors with large number of display facets and to provide an estimate of system performance if implemented using proper electronics. Fig. 16 shows projection-based setups displaying static and dynamic scenes on spherical (840 facets), cylindrical (216 facets), and desktop (816 facets) form factors.

4.1.3 Simulated Display Setups

We also demonstrate rendering to display surfaces that may be concave or self-intersecting using simulated setups. For these setups we render to facets using our pipeline, map the rendered images to the display geometry and observe from the user's perspective. Thus, system performance in terms of rendering time and scalability of the simulated display is a conservative estimate of a real display of the same shape, if built. A 64 mega pixel quilt image is rendered for each of these setups since the final image is viewed on a low resolution display. Fig. 17 shows

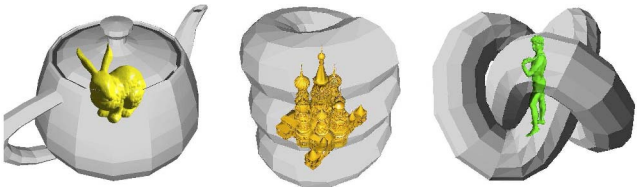


Fig. 17. Simulated setups showing various scenes on teapot, spring, and knot form factors

form factors shaped like a teapot (1,024 facets), spring (902 facets), and a knot (1,200 facets). The system is capable of rendering to these shapes in real time. In the triangle sorting stage, ray-casting finds the facet nearest to the viewer location. This ensures correct depth ordering for facets and allots triangles to the correct facet even when the surface is self-intersecting, as is the case in knot and spring.

4.2 Performance Evaluation

A single Nvidia GTX 580 with 1.5 GB of RAM on an Intel Core i7 930 processor with 4 GB RAM is used as a testbed for the following experiments. All experiments are reported at 64 mega pixel quilt image resolution with times averaged over a 1,000 frame walk-through.

Comparison with spatial hierarchies. To compare our method with spatial hierarchies, we implemented a dynamic octree structure. The octree is built every frame over the scene triangles on the GPU. Each triangle is assigned to a thread. Each thread finds the octree leaves its triangle intersects spatially and sets it as a part of the VBO of the respective octree leaf nodes. The scene hierarchy is culled to facet frustums in parallel and VBOs of the intersected leaf nodes are rendered to the corresponding facets. We use an octree depth of three, with 512 leaf nodes. Increasing the depth reduces triangles rendered per facet but increases the culling time and hence overall performance is reduced. Dynamic scenes require building the entire structure every frame, which slows down this approach for larger scenes. We found spatial hierarchies to be slower than our method due to the increase in computation as the number of facets increase, as shown in Fig. 18. The number of triangles

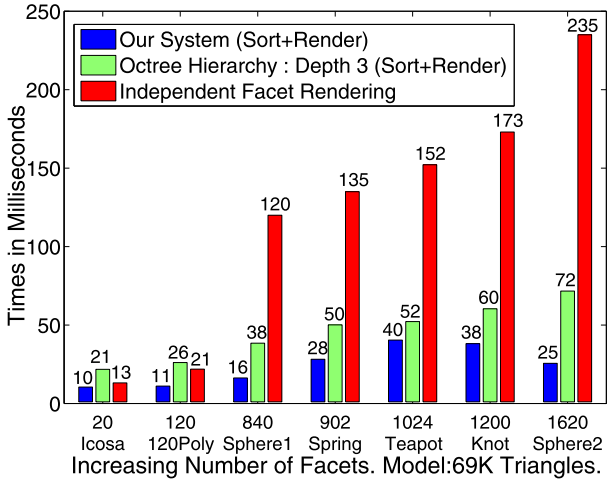


Fig. 18. Comparing performance of our system with spatial hierarchy and independent rendering for increasing number of facets.

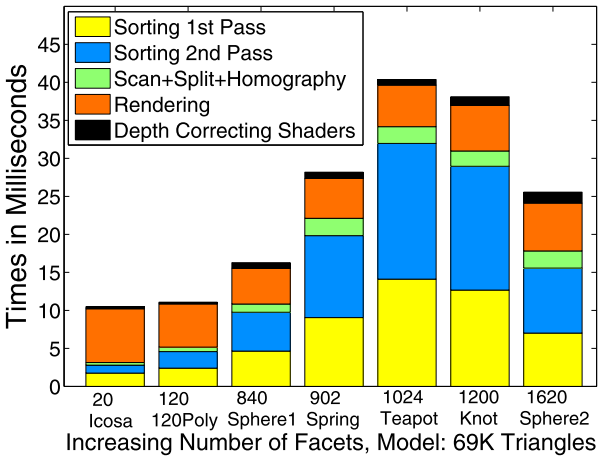


Fig. 19. Time breakup for our system, showing time taken by each step of our rendering pipeline.

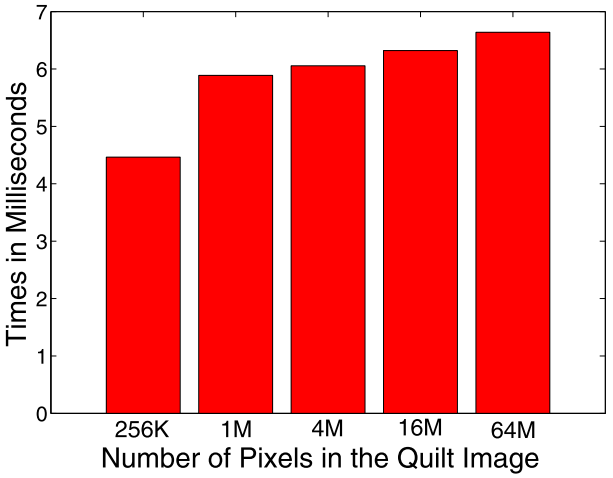


Fig. 20. Rendering speed w.r.t. increasing quilt image size for a scene consisting of 69K triangles and a display comprising of 1,200 facets.

rendered per facet increased by a factor of 2 compared to our sorting. Fast sorting on today's GPUs make our method faster than a spatial hierarchy-based method, which has more irregular operations that may not map well to the data-parallel hardware.

Fig. 18 shows an increase in rendering time as the number of facets increase for both octree hierarchy and the independent rendering approach. Our method, however, exhibits uneven per-frame times due to view dependent sorting. The topology of the display surface and visible facets decide the sorting time, which dominates the overall time taken by our system as shown in Fig. 19. We see that even on a larger display with 1,600 facets (sphere2) our method can take less time than on a display with fewer number of facets (teapot, 1,024 facets) because of a more even topology of the former. In case of teapot the sorting times vary with the viewpoint when the handle and the nose are in view. This observation can help design better displays. Almost equal number of facets across various views will provide a better performance.

Fig. 19 gives the time breakup of our system for the same experiment reported in Fig. 18. We see the sorting time dominates the overall pipeline, more specifically the second pass of sorting takes maximum time. This is because of the expensive triangle-facet overlap tests performed in this step. Even though the number of threads used is less, this pass requires projections of facets and triangles and computes intersection of these projections in camera space. The times also vary with the viewer location as more facets and triangles can come in view at various viewpoints. The figure clearly shows triangle sorting to be critical in our scheme. We also note that depth correcting shaders are not expensive as one additional parameter is interpolated by the rasterizer and only one division is computed per pixel. Similarly, the homography computation, transformation of vertices and splitting of triangles to facet VBOs are not limiting factors.

Fig. 20 examines the rendering time for increasing number of pixels for our method. Rendering depends on facet packing and typically for a closed shape only about 25 to 40 percent pixels are rendered per frame. The fill rate of current generation GPUs is high and thus even increasing the quilt image size to very large dimensions does not affect the rendering times by much.

In Fig. 21, we examine the scalability of our system with increasing number of scene triangles (16-871K) and increasing number of facets (20-1,620). Increasing the number of scene triangles increases the number of threads needed for sorting stages. This results in larger sort times and increases the overall rendering time. At 871K triangles we see that the frame rate drops to about 7-10 frames per second. For larger models the sorting load can be distributed to multiple GPUs driving thousands of facets. It can be seen that our system produces real-time frame rates only up to 200K triangles for various display configurations on a single GPU.

4.3 Limitations of the Display

The main drawback of our framework is its view dependence. Correct perspective is available only to a single viewer. Others see a distorted image due to a view dependent homography being applied to facet images. This is, however, a common feature of all head-tracked displays. The scene triangle sorting step is the most time consuming of all steps, especially the second pass. Performance can, however, be improved using multiple GPUs to sort the

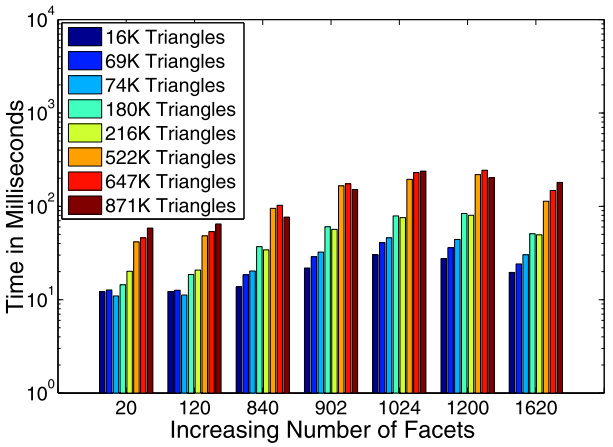


Fig. 21. Scalability of our system with increasing scene complexity for various display configurations.

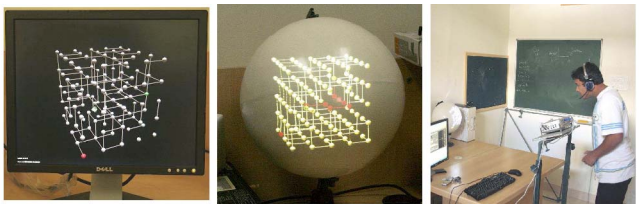
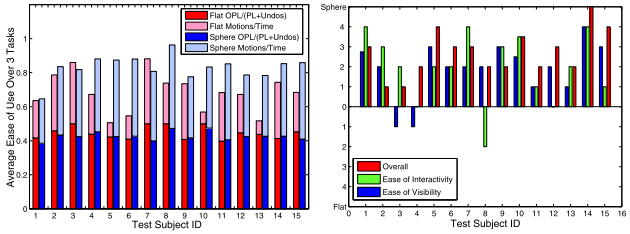


Fig. 22. Hollow cube structure used for user evaluation. Goal is to find a path from green to red node



(a) Ease of Use for various test subjects.

(b) Subjective user ratings

Fig. 23. Results of our user study. (a) Shows the evaluated ease of use based on recorded parameters. (b) Shows the results of the questionnaire.

scene to multiple facets and to generate multiple quilts, as the steps are highly parallel. An implementation of our system requires hardware unmapping of the quilt image at the display end. A display manufacturer can easily create such a setup using suitable electronics. Our LCD-based system used different VGA outputs instead. Our projected system sacrificed resolution to avoid this unmapping. Our framework is highly scalable inspite of these issues.

5 User Study: Utility of a Spherical Walk-Around Display

We study user interaction with a spherical display to evaluate the ease of use of a walk-around monoscopic display in this section. Rendering is not evaluated since the display used in this study is a projected setup—which only provides a mock platform for look and feel of our system if implemented using proper electronics. Please see Section 3.1.1 for comparison of various rendering methods. The aim of this study is to see if walk-around displays provide a more natural way to view and interact with virtual objects as compared to flat screens. The focus is on user experience and thus a full implementation of our system is nonessential. The spherical shape is chosen for its natural viewing properties. A WiiMote is used to interact with the display, to move the cursor in camera space and also to select objects shown inside the display. We design a simple path-finding task with a hollow connected cube structure with marked start and finish nodes (Fig. 22). The goal is to find a path between two specially marked nodes through the edges of the cube. Moving around the display is essential to evaluate a path in this exercise. The same experiment is also conducted on a flat display with the same mode of input, with additional buttons assigned to move user viewpoint. The rotation angle and the rotation speed are also restricted to simulate walking around scenario on the flat panel. Each user is given the same three tasks to perform on LCD and Sphere with a mock task on each to get familiarized with the display and input modes. We store the selected path length (PL), number of backtracking steps (Undos), viewpoints and the overall time taken to perform the task that are later used to evaluate a subjective measure defining ease of use. A questionnaire comprising three aspects of the experiment is also rated by the user: 1) Ease of visibility, which rates how easy it is to perceive the object/path based on moving around the object on a spherical display as opposed to rotating the object on a flat panel, 2) Ease of interactivity rates the user input, moving viewpoints using buttons as opposed to physically moving around, and 3) an overall rating which states the user preference.

We define *Ease Of Use* (EOU) based on recorded parameters using a penalizing and rewarding mechanism. Time taken should be penalized along with deviation from the Optimum Path Length (OPL) for each task. Motions help in perception and thus should be rewarded. Backtracking should also be penalized. Considering these we define ease of use as follows:

$$EOU = \frac{OPL}{PL + Undos} + \frac{Motions}{Time\ Taken}. \tag{6}$$

The first term captures the perceptual aspect whereas the second term captures interaction with the display. Both terms are normalized to have a maximum value of 0.5 and are given equal weightage. The metric gives a measure of how easy it is to move around, interact and perceive objects in a display based on our experiment. Fifteen test subjects evaluated EOU on an LCD and on the spherical display. Fig. 23a states their achieved EOU averaged over three tasks. It can be seen that both flat panel and spherical display show nearly the same deviation from the optimal path length. This is expected as both use monoscopic viewing and are only aided by motion for depth cues. It can also be seen that it is easier to move around the sphere than on the LCD. Motions should also help in depth cues and improve perception. However, because of lower resolution on the sphere, users complained about low quality of viewing, adversely affecting perception on the sphere. Users also indicated that perception might also be affected because of their familiarity with LCD. Fig. 23a favors motion on the sphere for 70 percent of the users indicating that it is easier to move around a spherical display.

Subjectively, an overwhelming majority of users preferred sphere over LCD as shown in Fig. 23b. They indicated it was easier to plan the path on the sphere, and depth variations were much clearer on the sphere. Interaction is also favored suggesting moving around a display to be more comfortable than moving using buttons, also confirmed by the result of Fig. 23a.

6 Conclusions and Future Work

We presented a framework to correctly render 3D scenes to multiplanar displays with a large number of facets. Our approach produces correct rendering and maintains interactivity of the application even when the facet count increases to over a thousand facets. We also demonstrated the scalability of our system with increasing resolution, scene complexity, and number of facets. The framework

facilitates rendering to virtually any display configuration as shown in simulation. Practical setups using LCD panels are not hard to build into any shape using appropriate electronics. This can provide a whole new interaction paradigm with the virtual world. With current display technology and advancements in motion-in-gaming our framework ideally suits the needs of interactive applications at minimal cost. Users like the additional interactivity of such displays over flat panels.

Visualization applications can benefit from quality rendering and interactive frame rates provided by our system. Medical visualization, design prototyping, molecular interactions, etc. require high-quality rendering. Our system can provide a glass box interactive display for such applications. Participative games is another area that general polyhedral displays can invigorate, especially walk-around displays. With the advent of new motion capture technologies like the Microsoft Kinect, Sony Move, and Nintendo Wii, games benefit greatly from novel user interactions. Head tracking is integral to these technologies and hence can drive 3D displays for a single viewer. Pairing this technology with flat displays limits its potential.

We would like to explore interactions on such displays using touch panels. Such a setup could then enable natural user interfaces for currently challenging problems. For example, with the use of a touch panel spherical walk-around display an artist could sculpt a virtual object as though he were actually working on a real statue. Many 3D displays are about to become practical in the coming years. Our framework combined with these can provide a truly enhanced visual experience of 3D environments and interactivity to the users of the future.

REFERENCES

[1] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-Time Dense Surface Mapping and Tracking," *Proc. IEEE 10th Int'l Symp. Mixed and Augmented Reality (ISMAR '11)*, pp. 127-136, 2011.

[2] O. Bimber, G. Wetzstein, A. Emmerling, and C. Nitschke, "Enabling View-Dependent Stereoscopic Projection in Real Environments," *Proc. IEEE/ACM Fourth Int'l Symp. Mixed and Augmented Reality*, pp. 14-23, 2005.

[3] K. Akeley and J. Su, "Minimum Triangle Separation for Correct Zbuffer Occlusion," *Proc. 21st ACM SIGGRAPH/EUROGRAPHICS Symp. Graphics Hardware (GH '06)*, pp. 27-30, 2006.

[4] A.J. Sabri, R.G. Ball, A. Fabian, S. Bhatia, and C. North, "High Resolution Gaming: Interfaces, Notifications, and the User Experience," *Interacting with Computers*, vol. 19, pp. 151-166, Mar. 2007.

[5] Nirnimesh, P. Harish, and P.J. Narayanan, "Garuda: A Scalable Tiled Display Wall Using Commodity Pcs," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 5, pp. 864-877, Sept./Oct. 2007.

[6] L. Shupp, R. Ball, B. Yost, J. Booker, and C. North, "Evaluation of Viewport Size and Curvature of Large, High-Resolution Displays," *Proc. Graphics Interface*, pp. 123-130, 2006.

[7] U. Hahne, J. Schild, S. Elstner, and M. Alexa, "Multi-Touch Focus+Context Sketch-Based Interaction," *Proc. Sixth Eurographics Symp. Sketch-Based Interfaces and Modeling (SBIM '09)*, pp. 77-83, 2009.

[8] P. Baudisch, N. Good, and P. Stewart, "Focus Plus Context Screens: Combining Display Technology with Visualization Techniques," *Proc. 14th Ann. ACM Symp. User Interface Software and Technology (UIST '01)*, pp. 31-40, 2001.

[9] R. Raskar, G. Welch, and H. Fuchs, "Seamless Projection Overlaps Using Image Warping and Intensity Blending," *Proc. Fourth Int'l Conf. Virtual Systems and Multimedia*, 1998.

[10] P. Bourke, "Idome: Immersive Gaming with the Unity Game Engine," *Proc. Computer Games and Allied Technology (CGAT '09)*, pp. 265-272, 2009.

[11] R. Raskar, G. Welch, K.-L. Low, and D. Bandyopadhyay, "Shader lamps: Animating Real Objects with Image-Based Illumination," *Proc. 12th Eurographics Workshop Rendering Techniques*, pp. 89-102, 2001.

[12] A. Law, D. Aliaga, B. Sajadi, A. Majumder, and Z. Pizlo, "Perceptually-Based Appearance Modification for Compliant Appearance Editing," *Computer Graphics Forum*, vol. 30, pp. 2288-2300, 2011.

[13] B. Sajadi and A. Majumder, "Autocalibration of Multiprojector Cavelike Immersive Environments," *IEEE Trans. Visualization and Computer Graphics*, vol. 18, no. 3, pp. 381-393, Mar. 2012.

[14] S. Kettner, C. Madden, and R. Ziegler, "Direct Rotational Interaction with a Spherical Projection," *Proc. Interaction: Systems, Practice and Theory*, 2004.

[15] S. Schkolne, M. Pruett, and P. Schröder, "Surface Drawing: Creating Organic 3D Shapes with the Hand and Tangible Tools," *Proc. SIGCHI Conf. Human Factors in Computing Systems*, pp. 261-268, 2001.

[16] Microsoft, *Microsoft Surface 2.0*, http://www.microsoft.com/surface/, 2007.

[17] H. Benko, A.D. Wilson, and R. Balakrishnan, "Sphere: Multi-Touch Interactions on a Spherical Display," *Proc. 21st Ann. ACM Symp. User Interface Software and Technology (UIST '08)*, pp. 77-86, 2008.

[18] S.S. Fisher, "Viewpoint Dependent Imaging: An Interactive Stereoscopic Display," *Proc. SPIE*, vol. 367, pp. 41-45, 1982.

[19] C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti, "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the Cave," *Proc. SIGGRAPH '93*, pp. 135-142, 1993.

[20] J. Djajadiningrat, C. Overbeeke, and P. Stappers, "Cubby: A Unified Interaction Space for Precision Manipulation," *Proc. IEEE Transportation Electrification Conf. and Expo (ITEC)*, pp. 24-26, 2001.

[21] J.W. Frens, J.P. Djajadiningrat, and C.J. Overbeeke, "Cubby+: Exploring Interaction," *Proc. Designing Interactive Systems (DIS' 02)*, pp. 135-140, 2002.

[22] I. Stavness, F. Vogt, and S. Fels, "Cubee: A Cubic 3D Display for Physics-Based Interaction," *Proc. SIGGRAPH '06*, p. 165. 2006,

[23] I. Stavness, B. Lam, and S. Fels, "Pcubee: A Perspective-Corrected Handheld Cubic Display," *Proc. Conf. Human Factors in Computing Systems*, pp. 1381-1390, 2010.

[24] H. Iwata, "Full-Surround Image Display Technologies," *Int'l J. Computer Vision*, vol. 58, pp. 227-235, 2004.

[25] M. Deering, "High Resolution Virtual Reality," *SIGGRAPH Computer Graphics*, vol. 26, no. 2, pp. 195-202, 1992.

[26] X. Hou, L.-Y. Wei, H.-Y. Shum, and B. Guo, "Real-Time Multi-perspective Rendering on Graphics Hardware," *Proc. EUROGRAPHICS Symp. Rendering*, pp. 93-102, 2006.

[27] M. Ashdown, M. Flagg, R. Sukthankar, and J.M. Rehg, "A Flexible Projector-Camera System for Multi-Planar Displays," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognitation (CVPR)*, pp. 165-172, 2004.

[28] R. Raskar, "Immersive Planar Display Using Roughly Aligned Projectors," *Proc. IEEE Virtual Reality (VR '00)*, 2000.

[29] P. Harish and P.J. Narayanan, "A View-Dependent, Polyhedral 3D Display," *Proc. Eight Int'l Conf. Virtual Reality Continuum and Its Applications in Industry (VRCAI)*, pp. 71-75, 2009.

[30] R.I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, second ed. Cambridge Univ. Press, 2004.

[31] Nvidia, *CUDA Programming Guide for CUDA Toolkit 3.2*, 2011.

[32] T. Möller, "A Fast Triangle-Triangle Intersection Test," *J. Graphics, GPU, and Game Tools*, vol. 2, no. 2, pp. 25-30, 1997.

[33] S. Patidar and P.J. Narayanan, "Scalable Split and Gather Primitives for the GPU," technical report IIIT/TR/2009/99, 2009.

[34] S. Liao, M.A. Lopez, and D. Mehta, "Constrained Polygon Transformations for Incremental Floorplanning," *ACM Trans. Design Automation of Electronic Systems*, vol. 6, pp. 322-342, 2001.

[35] NaturalPoint, *TrackIR5*, www.naturalpoint.com/trackir/products/trackir5/, 2009.

[36] ARToolkit, www.hitl.washington.edu/artoolkit/, 2002.

[37] A. Majumder, "Contrast Enhancement of Multi-Displays Using Human Contrast Sensitivity," *Proc. Computer Vision and Pattern Recognition*, pp. 377-382, 2005.

Pawan Harish received the bachelors degree in computer science and engineering from Uttar Pradesh Technical University, Lucknow, India, in 2005. Currently, he is working toward the PhD degree in computer science at the Center for Visual Information Technology, International Institute of Information Technology, Hyderabad, India. His research interests include novel displays, computer graphics, visualization, parallel algorithms, and GPU computing.

P.J. Narayanan received the bachelors degree from the IIT, Kharagpur and the PhD degree from the University of Maryland. He is a professor and dean of research at the IIIT, Hyderabad. He was a research faculty member at the Robotics Institute of Carnegie Mellon University from 1992 to 1996 and a scientist at the Center for Artificial Intelligence and Robotics, Bangalore till 2000. His research interests include computer vision, computer graphics, and GPU Computing. He was made a CUDA Fellow in 2008. He was the general chair of ICVGIP 2000 and the program co-chair of ACCV 2006 and ICVGIP 2010. He currently co-chairs the ACM India Council.