

# Designing Perspectively-Correct Multiplanar Displays

Pawan Harish and P. J. Narayanan  
Center for Visual Information Technology  
IIIT, Hyderabad, India

**Abstract**—Displays have remained 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 to one's natural surroundings. In this paper, we present a system to design, render, and build view-dependent *multiplanar displays*, of arbitrary shape built using planar, polygonal facets. Our system provides high quality, interactive rendering of 3D environments to a head-tracked viewer on arbitrary planar display shapes. We develop a novel rendering scheme that creates exact image and depth at each display facet. The facets thus align exactly at boundaries without inconsistencies in comparison with existing methods. Our approach scales well to large numbers of display facets. This is achieved using a single-pass rendering of all facets using a parallel, per-frame, view-dependent binning and prewarping of scene triangles. The method places no constraints on the scene or display and allows for fully dynamic scenes to be rendered at high resolutions using a single pass of rasterization. These are implemented efficiently on the GPUs. A general realization of our system envisions a display of an arbitrary shape built using polygonal facets. The display is driven using one or more quilt images into which the the pixels are packed. We present a few prototype displays to establish the scalability of our system to different 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 is shown in terms of both quality and rendering speeds of our system for increasing scene and display facet sizes. A subjective user study is also presented to evaluate the user experience using a walk-around display to a flat panel in a game-like setting.



**Index Terms**—Non-rectangular displays, fish tank virtual reality, arbitrary shaped displays, 3D visualization, view-dependent rendering, fast 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. These technologies have improved various aspects of a display system, including power consumption, color gamut, vertical refresh rate and pixel resolution amongst others. The shape and planarity of a display, however, have not changed. Displays still remain flat, inactive and rectangular stand-alone windows feeding images to an observer viewing it. This passive viewing platform has restricted displays in many ways. These include non-intuitive interaction, lack of focus+context and lack of three dimensional viewing to name a few. 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 at making displays more active.

Simultaneously, much work has gone into capturing and creating 3D content. A bulk of content shown on displays today is three-dimensional, ranging from games to complex

CAD simulations. With the advent of powerful computing methods, real-time 3D content capture and creation is now possible [1]. As a result much work has gone into understanding and improving interaction with 3D content on displays. 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 real and displayed objects. This notion has two parts: (a) the display itself being part of the viewer space and (b) content displayed on it being accurate depictions. If the shape of the display is arbitrary, the first part of the notion is resolved. The display can take shape of any object in the user's surrounding. Rendered images on such a display must also be accurate such that the depiction agrees with the natural perspective of the viewer. Interaction must also be retained along with quality rendering to provide a consistent view of the virtual world.

In this article, we present a framework to design, build and render to *multiplanar displays*, which are built out of multiple polygonal *facets*. We show perspectively correct 3D on to such a display to a single head tracked viewer. In effect, our framework extends fish tank virtual reality (FTVR) displays to generic polygonal shapes. Our system scales to arbitrary shapes, which sets it apart from other FTVR displays which are limited to a few number of facets. Rendering to FTVR displays has a tradeoff between achievable quality and the number of facets. Projective Texture

Mapping (PTM), has been shown to produce FTVR effect on any surface, albeit 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 facet boundaries, generating a collective view of the virtual world. Our rendering scheme produces artifact free images as shown in comparison with existing methods. Rendering cost is reduced using a novel parallel binning of scene triangles to the display geometry implemented on commodity GPUs, which helps the system scale to arbitrary display shapes. To reduce the rendering load we render all facet images in a single pass of rasterization, generating a quilt of all facet images which can be un-mapped to the display geometry either using multiple VGA outputs or using specialized hardware. Our system is implemented on commodity GPUs and can scale to display shapes consisting of well over a thousand facets. Our design provides both quality and interactivity needed in many applications such as medical visualization, computer aided design, simulations and games. The framework contributes (a) a system to drive a generic multiplanar display through one or more rectangular quilt images which can be generated by standard graphics systems retaining pixel density, (b) a method to correctly render images to a set of arbitrarily oriented planar shapes to provide a perspective correct view across facets boundaries (the image as well as the depth buffer are correct, guaranteeing consistent views on facets), and (c) a scalable, single-pass rendering mechanism that sorts the scene polygons to display facets in parallel and prewarps the scene, 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 1600 facets at 35Hz. 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 article. 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 multiple previous works including multiplanar, non-planar, curved and FTVR displays. We re-

view the literature related to these in this section, segregated into multi-surface displays, FTVR displays and rendering to FTVR displays.

**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 non-tiled arrangements such as displaying a high-resolution focus window within a lower resolution context [8]. Non-planar arrangements of displays have also been explored, an example is the fish eye view generated using multiple projectors projecting on to 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 in using multiple projectors to create high-resolution appearance editing [12]. Such advancements have spawned much work in auto-calibrating systems for multi projector displays [13], both in terms of color and geometric corrections. Other non-planar displays that enable users to interact with information content from all angles have also been showcased [14], [15], [16], [17]. 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. Multisurface displays provide novel interaction and appearance onto arbitrary shapes, but are limited to two dimensional or information content.

**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 outwards, 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 increasing rendering load as the number of facets increase. Iwata implemented a non-cubic, rhombic dodecahedral display using projectors to cover the full solid angle [24].

**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 multi-perspective 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 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 pre-warping 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, microprojector, 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 facets. Each pixel of the display is addressed using a three-dimensional 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 (Figure 1). Un-mapping of facet images given the quilt image can be achieved using multiple VGA outputs or specialized hardware. We consider the un-mapping to be a part of the display hardware. 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 inputs to our system are the display configuration (as a mesh model), mapping to the quilt image, scene to

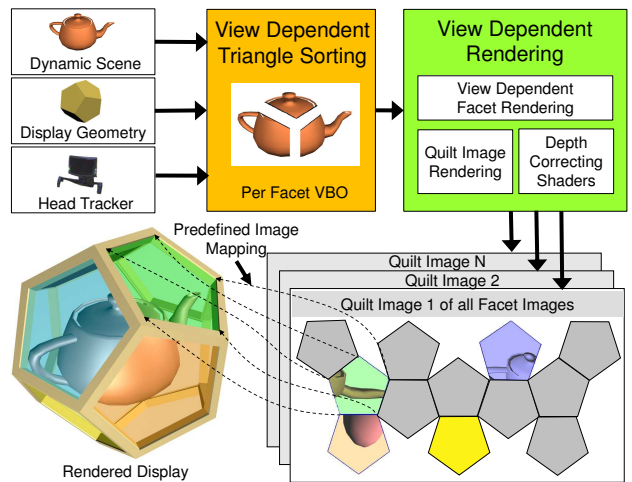


Fig. 1. The multiplanar display system. Top: Rendering pipeline. Bottom: Driving the display using quilt images.

be rendered and the user location. The system generates the quilt images based on the user location and sends these to the display hardware for un-mapping. The process is described in Figure 1. As the first step, the scene is sorted to display facets in parallel (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 pre-warping 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 Figure 2. The view-dependent virtual plane intersects multiple facets of the display. Each facet  $i$  can be assigned a facet camera  $C_i$ ; its image  $I_i$  can then be related to the viewer camera image  $I_v$  by a  $3 \times 3$

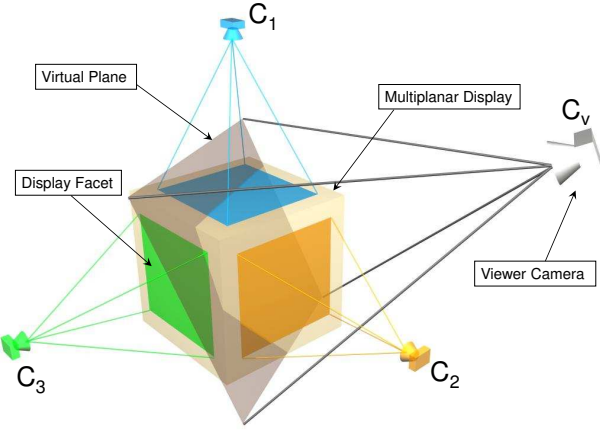


Fig. 2. Viewer camera  $C_v$  viewing the display and the *virtual plane* passing through the center of the display.  $C_1$ ,  $C_2$ ,  $C_3$  are cameras corresponding to each display facet.

homography matrix as given below [30].

$$I_i = H_{iv} I_v, \quad \forall i \leq \#facets \quad (1)$$

Each visible facet will have such a homography relating its image to the virtual plane. Rendering  $I_i$  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  $I_i$  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  $C_i$  and  $C_v$ , normal  $n$ , distance  $d$  of the virtual plane, and intrinsic parameters  $K_i$  and  $K_v$  of the cameras,  $H_{iv}$  can be computed directly as  $H_{iv} = K_v[R - Tn^T/d]K_i^{-1}$  [30] (Figure 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  $C_i$  using a thread for each facet.

**Facet Homography in Canonical Space:** Facet images,  $I_i$ , 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

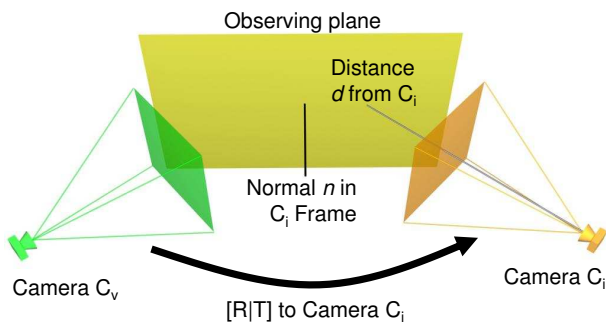


Fig. 3. Computing Homography transformation between two cameras.

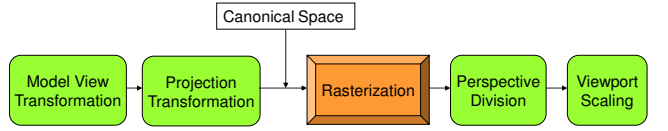


Fig. 4. Graphics pipeline with canonical space. The perspective division and viewport scaling stages only modify the scale of the point in canonical space.

facet image as pixels in  $I_v$  may not map to any pixel in  $I_i$ , 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,  $I_i$ . 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 (Figure 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 correction of an off-axis projection by Raskar [28]. The process of transforming a scene point  $\mathbf{X}$  to the facet  $i$  can now be given as

$$I_i = VP_d H_{iv} P M_v \mathbf{X}, \quad (2)$$

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:

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

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 Figure 5(a). 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 Figure 5(b).

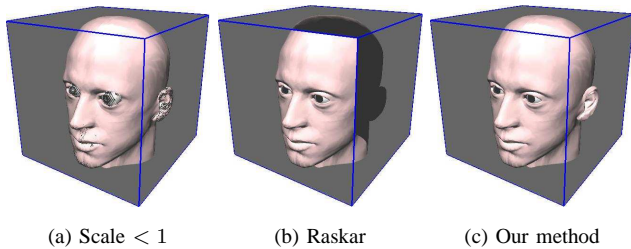


Fig. 5. The depth problem: comparison of approaches. (a) depth errors due to reduced  $z$ -range at eye, lip and ear. (b) error due to near plane clipping artifacts. (c) our method without artifacts

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

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

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. Figure 5(c) shows the correct image using our scheme. Early- $z$  culling is avoided by setting a constant depth value of  $-1$  at all vertices. 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:
  - 7: {**Fragment\_Shader(V)**}
  - 8: Perform fixed-pipeline operations for color
  - 9: Transform  $V_c$  to canonical space as  $V'_c$
  - 10: Set depth as the ratio of  $z$  coordinate of  $V'_c$  and  $V_c$ .
- 

### 3.1.1 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, both on the facet and the combined effect from the user's perspective. 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 (PTM) can also be used to generate a similar effect to that

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.

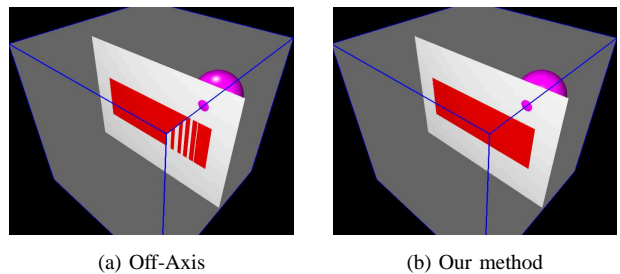


Fig. 7. Comparing off-axis projection with our rendering scheme. Scene consists of two proximate rectangles and a push-through sphere.

of our system. It produces an intermediate image which is mapped to multiple facets using interpolation. This can lead to quality degradation due to image re-sampling. A detailed visual comparison with Projective Texture Mapping can be found in the Appendix at the end of this article.

Off-axis rendering [25] has been the rendering mechanism for FTVR based displays like the CAVE, Cubby, etc. The method renders each facet using a view plane parallel to the facet using oblique frustum boundaries (Figure 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 (Figure 6). This results in visual artifacts that can be seen across facet boundaries. Figure 7(a) shows the rendering of two proximate planes using off-axis rendering. Depth errors can be seen as opposed to our rendering scheme (Figure 7(b)). Figure 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. Multiplanar display rendering to arbitrary display shapes requires this property to avoid artifacts. Correct image and depth at each facet ensures quality rendering needed for visualization

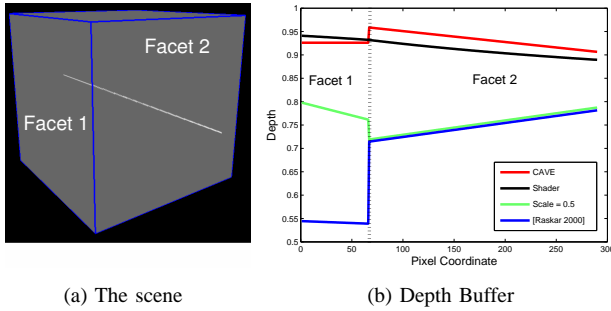


Fig. 8. Comparing depth buffers for various approaches at the boundary of two facets. Note no sudden change in depth values for the shader approach, ensuring continuity at facet boundaries.

applications. The rendering load for our scheme is minimal as evaluated in Section 4.2, Figure 19.

### 3.2 Scaling to a Large Number of Display Facets

Current FTVR rendering methods need to 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: (i) A parallel scene sorting algorithm which sorts the triangles of the scene to the display facets quickly on the GPU per frame. (ii) A rendering scheme that renders the quilt of multiple facet images in a single pass of rasterization. These can be combined with our rendering mechanism to 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.

#### 3.2.1 Visibility Determination and Triangle Sorting

We sort the geometry of the scene at the triangle level, with triangles of the scene sorted 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 display facets. Per vertex ray-casting implemented on the GPU is used to achieve this in real-time. Figure 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 Figure 10. In most scenes, case 1 yields maximum number of triangles. The second and third cases yield fewer number of triangles, but require more expensive computations. We

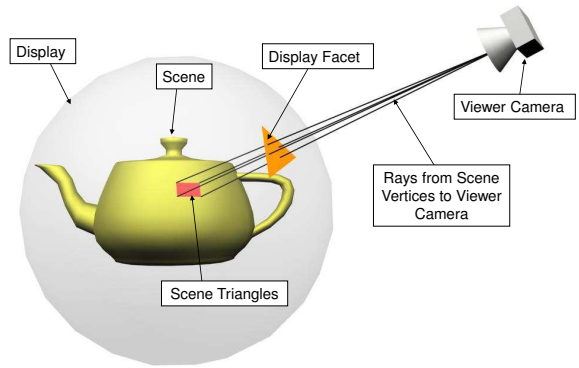


Fig. 9. Triangle sorting based on per vertex raycasting

divide our sorting into two passes based on this observation. The first pass handles the first case while the second pass deals with residual triangles.

**Pre-Processing:** To reduce the number of tests performed per ray, we pre-process 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% of the total scene triangles. The remaining triangles have different facet ids at their vertices and undergo the second pass. Figure 11(a) 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**
- 

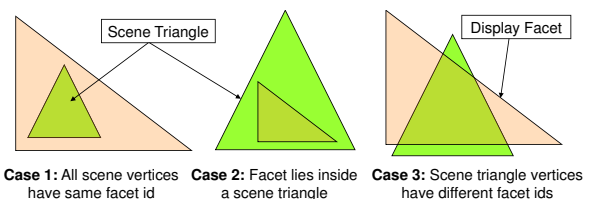


Fig. 10. Cases occurring from per vertex ray casting

**Algorithm 3** Second Pass Kernel

---

```

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

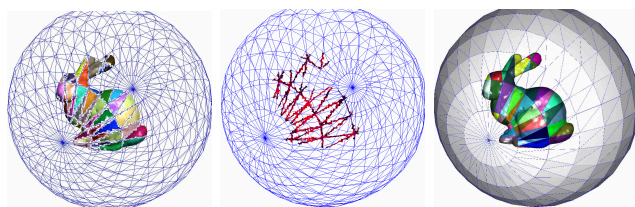
```

---

**Second Pass:** The second pass examines all triangles that remain (Figure 11(b)). 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 Figure 10 is 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. We also reduce the work done by the GPU for the second pass of the algorithm using thread compaction [33], 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 [34] 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 Figure 11(c), 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 reported in Section 4.2. An average increase of 4 – 5% is seen for the number of triangles.



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

Fig. 11. The stages of our triangle sorting algorithm

**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

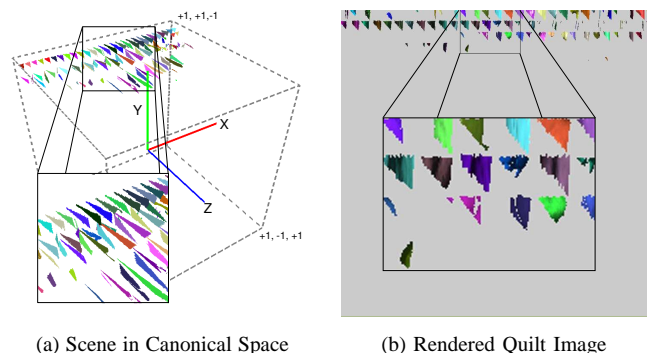
```

---

## 3.2.2 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 trade-off 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 Equation 2, we observe that each display facet has a different projection matrix ( $H_{iv}P$ ). 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 pre-warping each point  $\mathbf{X}$  for each facet. The mapping for a facet image  $i$  in the quilt image is a fixed 2D transformation (Figure 1). Let  $Y_i$  denote this in the canonical space. The point  $\mathbf{X}$  per facet is modified by the facet-specific quilt image mapping ( $Y_i$ ), homography ( $H_{iv}$ ) and the projection and modelview matrices ( $PM_v$ ) to  $\{Y_i H_{iv} PM_v \mathbf{X}\}$ . 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  $i$ , but the pipeline



(a) Scene in Canonical Space

(b) Rendered Quilt Image

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

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

$$Quilt = VP_d I_{M_v} I_P \sum_{i=1}^{\# facets} Y_i H_{i_v} P M_v \{VBO_i\}, \quad (5)$$

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  $i$  can be pre-warped to  $Y_i H_{i_v} P M_v \mathbf{X}$  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. Figure 12(a) 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 Figure 12(b). For a display with 1600 facets, a 15 – 20% decrease in rendering time is seen using this method compared to setting up the graphics pipeline for each VBO at the same facet resolution.

### 3.2.3 Packing Facets Into the Quilt Image

Facet packing transformation  $Y_i$  is a 2D transformation matrix per facet that maps a facet image to the quilt image. This can be done without any interpolation using equal number of pixels in each.  $Y_i$  differs from texture mapping as a result. Optimal packing of polygonal shapes in a 2D plane is an NP hard problem [35]. We can use a heuristic packing arrangement that attempts to utilize the quilt image space optimally. This is a preprocessing step done while designing the display configuration. A simple packing 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 Figure 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 Figure 13.

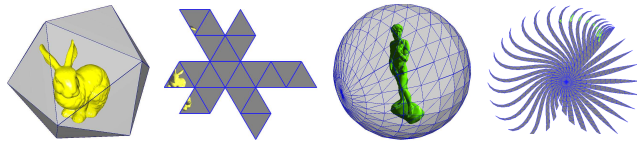


Fig. 13. Display as seen from user's point of view and its facet packing into the quilt image using a BFS tree

## 3.3 The Rendering Pipeline

Figure 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

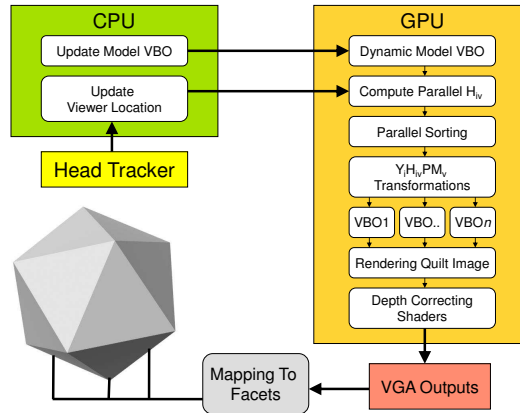


Fig. 14. The Overall Rendering Process

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 1600 facets on current generation GPUs. For larger scenes the framerate drops to below interactive rates as shown in Figure 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 (a) the generality of the multiplanar display framework, (b) the correctness of displaying 3D information using our method on challenging display configurations, and (c) the scalability of the approach to arbitrarily large displays. We use three types of display prototypes for this. Examples showing dynamic three dimensional scenes and information on our prototypes can be found in the accompanying video.

#### 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





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

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 Figure 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 [36], intended for use in gaming applications. With a refresh rate of 120Hz the latency is minimized and the location of the viewer is given within an error of 5 – 10%.

**Calibration:** We calibrate the cube using a simple procedure based on ARToolkit [37] 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 [38]. 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 4-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 (Figure 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 system using projection based setups. These setups use our pipeline to generate the quilt image by following the stages of parallel sorting, facet rendering,

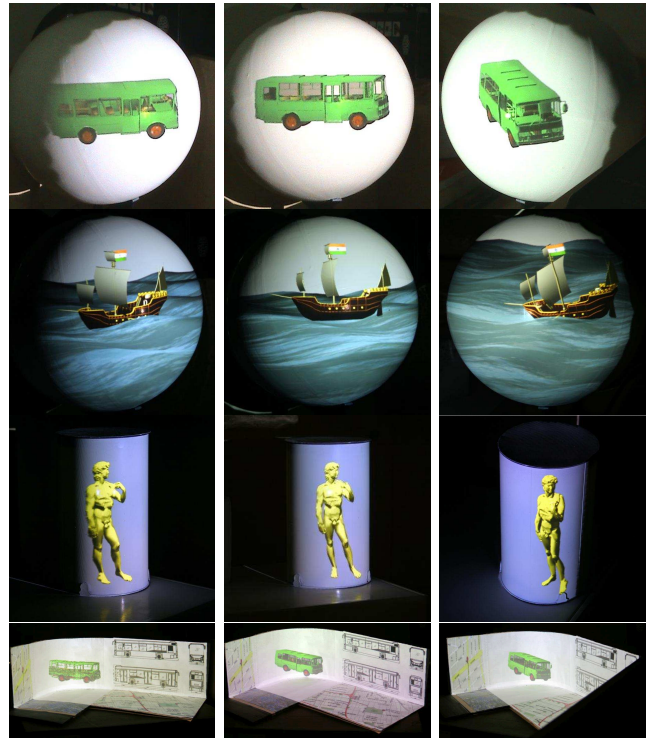


Fig. 16. Projection Based setups showing various scenes on sphere, cylinder and desktop form factors

quilt image rendering and depth correcting shaders. The inverse mapping stage is absent as there is no hardware un-mapping 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% 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 64M 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. Figure 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

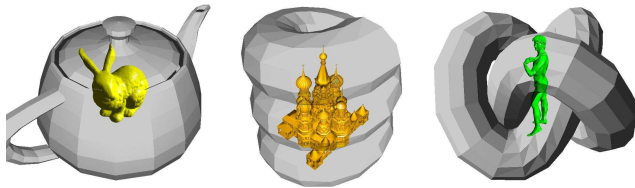


Fig. 17. Simulated setups showing various scenes on teapot, spring and knot form factors

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 64M pixel quilt image is rendered for each of these setups since the final image is viewed on a low resolution display. Figure 17 shows form factors shaped like a teapot (1024 facets), spring (902 facets) and a knot (1200 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 1.5GB of RAM on an Intel Core i7 930 processor with 4GB RAM is used as a testbed for the following experiments. All experiments are reported at 64 mega pixel quilt image resolution with times are averaged over a 1000 frame walk-through.

**Comparison with Spatial Hierachies:** To compare our method with spatial hierachies, 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 hierachies to be slower than our method due to the increase in computation as the number of facets increase, as shown in Figure 18. The number of triangles 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.

Figure 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

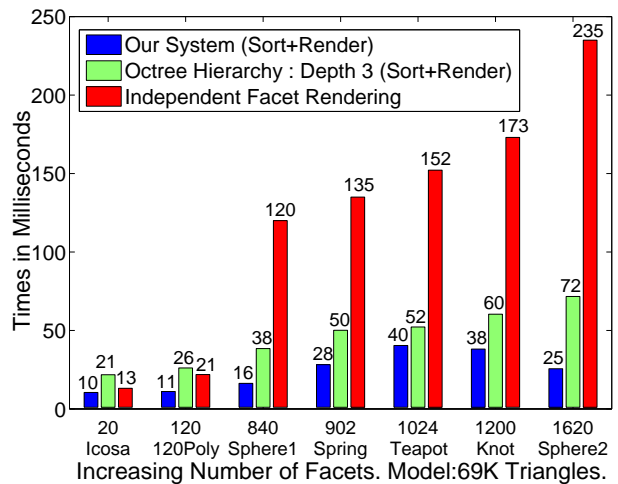


Fig. 18. Comparing performance of our system with spatial hierarchy and independent rendering for increasing number of facets.

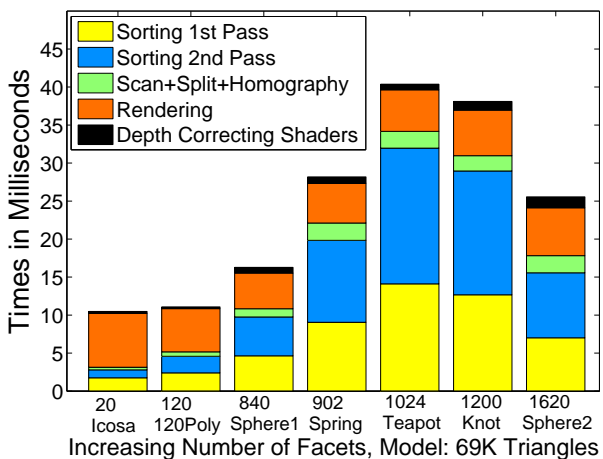


Fig. 19. Time breakup for our system, showing time taken by each step of our rendering pipeline.

time taken by our system as shown in Figure 19. We see that even on a larger display with 1600 facets (sphere2) our method can take less time than on a display with fewer number of facets (teapot, 1024 facets) because of a more even topology. 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 a better display. Almost equal number of facets across various views will provide a better performance.

Figure 19 gives the time breakup of our system for the same experiment reported in Figure 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

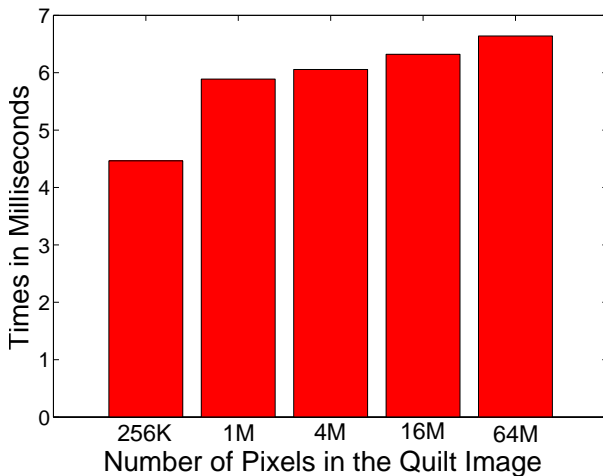


Fig. 20. Rendering speed w.r.t. increasing quilt image size for a scene consisting of 69K triangles and a display comprising of 1200 facets.

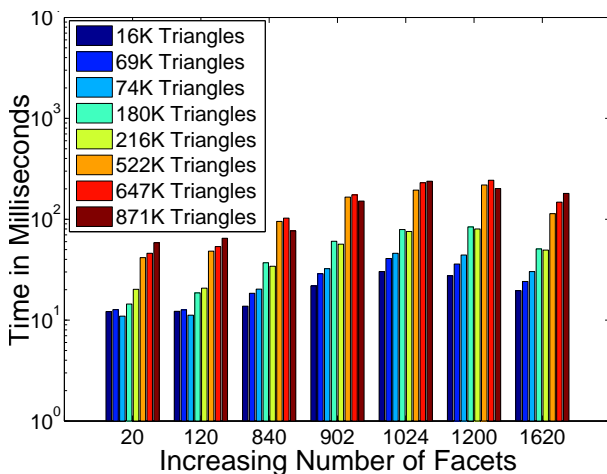


Fig. 21. Scalability of our system with increasing scene complexity for various display configurations.

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.

Figure 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% 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 Figure 21, we examine the scalability of our system with increasing number of scene triangles (16K–871K) and increasing number of facets (20–1620). 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 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 scene to multiple facets and to generate multiple quilts, as the steps are highly parallel. An implementation of our system requires hardware un-mapping 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 un-mapping. 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 non essential. 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 (Figure 22). The goal

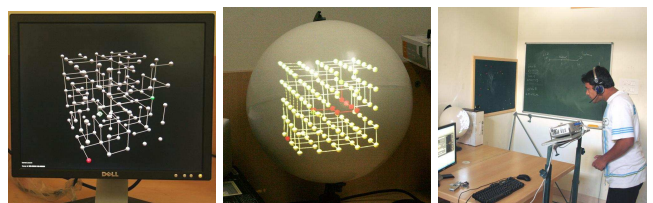


Fig. 22. Hollow cube structure used for user evaluation. Goal is to find a path from green to red node

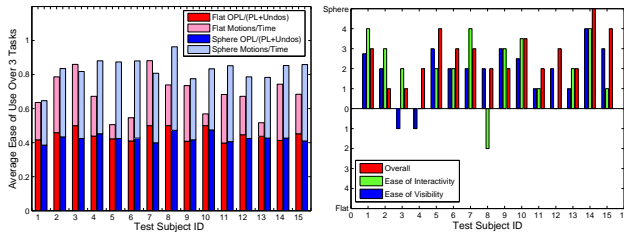
is to find a path between two specially marked nodes through the edges of the cube. Moving around the display is essential to evaluate a path in this exercise. The same experiment is also conducted on a flat display with the same mode of input, with additional buttons assigned to move user viewpoint. 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: (i) 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, (ii) Ease of interactivity rates the user input, moving viewpoints using buttons as opposed to physically moving around and (iii) 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

$$EOU = \frac{OPL}{PL + Undos} + \frac{Motions}{Time Taken} \quad (6)$$

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. Figure 23(a) states their achieved EOU averaged over three tasks. It can be seen that both flat panel and



(a) Ease of Use for various test subjects.

(b) Subjective user ratings

Fig. 23. Results of our user study

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. Figure 23(a) favors motion on the sphere for 70% 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 Figure 23(b). 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 Figure 23(a).

## 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," in *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, ser. ISMAR '11. IEEE Computer Society, 2011, pp. 127–136.
- [2] O. Bimber, G. Wetzstein, A. Emmerling, and C. Nitschke, "Enabling view-dependent stereoscopic projection in real environments," in *IEEE/ACM ISMAR*, ser. ISMAR '05, 2005, pp. 14–23.
- [3] K. Akeley and J. Su, "Minimum triangle separation for correct z-buffer occlusion," in *ACM SIGGRAPH/EUROGRAPHICS GH '06*, 2006, pp. 27–30.
- [4] A. J. Sabri, R. G. Ball, A. Fabian, S. Bhatia, and C. North, "High-resolution gaming: Interfaces, notifications, and the user experience," *Interact. Comput.*, vol. 19, pp. 151–166, March 2007.
- [5] Nirmimesh, P. Harish, and P. Narayanan, "Garuda: A scalable tiled display wall using commodity pcs," *IEEE TVCG*, vol. 13, pp. 864–877, 2007.
- [6] L. Shupp, R. Ball, B. Yost, J. Booker, and C. North, "Evaluation of viewport size and curvature of large, high-resolution displays," in *Proceedings of Graphics Interface 2006*, ser. GI '06, 2006, pp. 123–130.
- [7] U. Hahne, J. Schild, S. Elstner, and M. Alexa, "Multi-touch focus+context sketch-based interaction," in *Eurographics SBIM*, ser. SBIM '09, 2009, pp. 77–83.
- [8] P. Baudisch, N. Good, and P. Stewart, "Focus plus context screens: combining display technology with visualization techniques," in *ACM UIST*, ser. UIST '01, 2001, pp. 31–40.
- [9] R. Raskar, G. Welch, and H. Fuchs, "Seamless projection overlaps using image warping and intensity blending," in *Virtual Systems and Multimedia*, 1998.
- [10] P. Bourke, "idome: Immersive gaming with the unity game engine," in *CGAT 09*, ser. CGAT '09, 2009, pp. 265–272.
- [11] R. Raskar, G. Welch, K.-L. Low, and D. Bandyopadhyay, "Shader lamps: Animating real objects with image-based illumination," in *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*. Springer-Verlag, 2001, pp. 89–102.
- [12] A. Law, D. Aliaga, B. Sajadi, A. Majumder, and Z. Pizlo, "Perceptually-based appearance modification for compliant appearance editing," *Computer Graphics Forum*, 2011.
- [13] B. Sajadi and A. Majumder, "Autocalibration of multiprojector cave-like immersive environments," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 3, pp. 381–393, 2012.
- [14] H. Benko, A. D. Wilson, and R. Balakrishnan, "Sphere: multi-touch interactions on a spherical display," in *UIST*, ser. UIST '08, 2008, pp. 77–86.
- [15] S. Kettner, C. Madden, and R. Ziegler, "Direct rotational interaction with a spherical projection," in *In Interaction: Systems, Practice and Theory Proceedings*, 2004.
- [16] S. Schkolne, M. Pruetz, and P. Schröder, "Surface drawing: creating organic 3d shapes with the hand and tangible tools," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, ser. CHI '01, 2001, pp. 261–268.
- [17] Microsoft, "Microsoft Surface 2.0, <http://www.microsoft.com/surface/>," 2007.
- [18] S. Fisher, "Viewpoint dependent imaging: An interactive stereoscopic display," *SPIE*, p. 367, 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," in *SIGGRAPH '93*, 1993.
- [20] J. Djajadiningrat, C. Overbeeke, and P. Stappers, "Cubby: A unified interaction space for precision manipulation," in *ITEC 2001*, ser. ITEC 2001, 2001, pp. 24–26.
- [21] J. W. Frens, J. P. Djajadiningrat, and C. J. Overbeeke, "Cubby+: exploring interaction," in *DIS*, ser. DIS '02, 2002, pp. 135–140.
- [22] I. Stavness, F. Vogt, and S. Fels, "Cube: a cubic 3D display for physics-based interaction," in *SIGGRAPH '06 Sketches*, 2006, p. 165.
- [23] I. Stavness, B. Lam, and S. Fels, "pcube: a perspective-corrected handheld cubic display," in *Human factors in computing systems*, ser. CHI '10, 2010, pp. 1381–1390.
- [24] H. Iwata, "Full-surround image display technologies," *Int. J. Comput. Vision*, vol. 58, pp. 227–235, 2004.
- [25] M. Deering, "High resolution virtual reality," *SIGGRAPH Comput. Graph.*, 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," in *EUROGRAPHICS Symposium on Rendering*, ser. EGSR '06, 2006, pp. 93–102.
- [27] M. Ashdown, M. Flagg, R. Sukthankar, and J. M. Rehg, "A flexible projector-camera system for multi-planar displays," in *EEE CVPR*, 2004, pp. 165–172.
- [28] R. Raskar, "Immersive planar display using roughly aligned projectors," in *IEEE VR '00*, 2000.
- [29] P. Harish and P. J. Narayanan, "A view-dependent, polyhedral 3d display," in *VRCAI*, ser. VRCAI '09, 2009, pp. 71–75.
- [30] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.
- [31] Nvidia, "CUDA Programming Guide for CUDA Toolkit 3.2," 2011.
- [32] T. Möller, "A fast triangle-triangle intersection test," *Journal of graphics, gpu, and game tools*, vol. 2, no. 2, pp. 25–30, 1997.
- [33] P. Harish, V. Vineet, and P. J. Narayanan, "Large Graph Algorithms for Massively Multithreaded Architectures," Tech. Rep. IIT/TR/2009/74, 2009.
- [34] S. Patidar and P. J. Narayanan, "Scalable Split and Gather Primitives for the GPU," Tech. Rep. IIT/TR/2009/99, 2009.
- [35] S. Liao, M. A. Lopez, and D. Mehta, "Constrained polygon transformations for incremental floorplanning," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 6, pp. 322–342, 2001.
- [36] TrackIR5, "NaturalPoint TrackIR5, [www.naturalpoint.com/trackir/products/trackir5/](http://www.naturalpoint.com/trackir/products/trackir5/)," 2009.
- [37] ARToolkit, "[www.hitl.washington.edu/artoolkit/](http://www.hitl.washington.edu/artoolkit/)," 2002.
- [38] A. Majumder, "Contrast enhancement of multi-displays using human contrast sensitivity," in *CVPR'05*, ser. CVPR '05, 2005, pp. 377–382.



**Pawan Harish** received his bachelors degree in computer science and engineering from Uttar Pradesh Technical University, Lucknow, India, in 2005. He is currently working towards 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** is a Professor and Dean of Research at the IIIT, Hyderabad. He got his bachelors from IIT, Kharagpur and his PhD from the University of Maryland. He was a research faculty member at the Robotics Institute of Carnegie Mellon University from 1992 to 1996 and a scientist

at the Centre 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.