Scalable Rendering of Massive Triangle Meshes on Light Field Displays ⋆

Fabio Bettio Enrico Gobbetti Fabio Marton Giovanni Pintore
CRS4, POLARIS Ed. 1, 09010 Pula (CA), Italy – www.crs4.it/vic/

Abstract

We report on a multiresolution rendering system driving light field displays based on a specially arranged array of projectors and a holographic screen. The system gives multiple freely moving naked-eye viewers the illusion of seeing and manipulating 3D objects with continuous viewer-independent parallax. Multi-resolution techniques which take into account the displayed light field geometry are employed to dynamically adapt model resolution to display capabilities and timing constraints. The approach is demonstrated on two different scales: a desktop PC driving a 7.4Mbeams TV-size display, and a cluster-parallel solution driving a large (1.6x0.9 meters) 35Mbeams display which supports a room-size working space. In both cases, massive meshes of tens of millions of triangles are manipulated at interactive rates.

Key words: 3D displays, Massive datasets, Out-of-core algorithms, Parallel graphics, Level of detail

1. Introduction

The accurate reproduction of 3D light fields requires generating a large number of light beams of appropriate origin, direction, and color. Recent advances in 3D display design have demonstrated that interactive high resolution light field display technology is practically achievable [1,2]. Even though considerations on human vision specifics can drastically reduce the amount of data that has to be encoded in a reconstructed light field, rendering still remains a complex and computationally intensive task, which has limited until very recently the applicability of such displays to presentation of static images, prerecorded movies, or small graphics models. In this article, we report on a specialized out-of-core multiresolution real-time rendering system able to render massive geometric models on light field displays. The system gives multiple viewers the illusion of seeing virtual objects floating at fixed physical locations situated in a human–scale working volume (see figure 1). Different viewers see the scene from their point of view and enjoy continuous horizontal parallax without specialized viewing devices. The displays are based on a specially arranged array of projectors emitting light beames onto a holographic screen, which then makes the necessary optical transformation to compose these beams into a continuous 3D view. The displays are driven by a multiresolution renderer able to correctly project geometries onto the display and to dynamically adapt model resolution by taking into account the particular spatial accuracy characteristics of the display. The method is a light field display-aware version of our Adaptive TetraPuzzles technique [3]. The efficiency and scalability of this approach is demonstrated by an application supporting interactive manipulation of massive colored highly tessellated models on both a TV sized 7.4Mbeams display and a large (1.6x0.9 m) 35Mbeams display that is designed to be used in a room-size working space. The small-scale system can be driven by a single desktop PC, while the large-scale display is driven by a cluster-parallel implementation of the adaptive multiresolution renderer. The large display is an evolution of the 50Mbeams display used in our earlier work [4], and it is based on LED projectors, thus improving color reproduction and reducing energy consumption at the cost of slightly reduced luminance and 2D resolution. Many of the details of this framework were presented in [4]. We provide here a more thorough exposition, but also significant new material, including the discussion of a single PC solution and of the projection method.

⋆ Extended version of a paper presented at the Eurographics Symposium on Parallel Graphics and Visualization (Manno, Switzerland, May 2007)

Email addresses: fabio@crs4.it (Fabio Bettio), gobbetti@crs4.it (Enrico Gobbetti), marton@crs4.it (Fabio Marton), gianni@crs4.it (Giovanni Pintore).

2. Related work

Developing a scalable light field display system which targets interactive manipulation of massive models is a big engineering effort, which requires the combination of state-of-the-art results in a number of technological areas. In the following, we briefly discuss the approaches most closely related to ours.

Fig. 1. Interactive massive model rendering on light field displays.

Light field displays. A number of approaches have been proposed to support naked-eye stereoscopic rendering. For a review on the subject of display technology we refer the reader to [5]. The displays used in this work [1,6,7] adopt the distributed image generation approach of untracked projector-based multi-view technology, but remove some of the intrinsic optical limitations by offering a fully continuous blend among views. Typical multi-view displays, often based on an optical mask or a lenticular lens array, show multiple 2D images in multiple zones in space. They support multiple simultaneous viewers, but at the cost of restricting them to a limited viewing angle. Matusik et al. [8], for instance, demonstrated a prototype based on this technology and assembled with sixteen 1024x768 projectors and a lenticular screen. As in our case, the setup requires one projector per view. However, their screen achieves vertical diffusion not by diffusing light vertically from the screen as we do, but by focusing light horizontally onto a diffuse surface, yielding a different projection geometry. A 3D stereo effect is obtained when the left eye and the right eye see different but matching information. The small number of views of multi-view systems based on masks or lenticulars produces, however, cross-talks and discontinuities upon viewer's motion [9]. Instead, the solution employed here exploits the light shaping capabilities of a holographic screen, and presents a continuous image to many viewers within a large workspace, due to the many smoothly blended view-dependent pixels that contribute to a single image. The light field display recently presented by Jones et al. [2] also uses an anisotropic diffuser, which, however, covers a rapidly spinning mirror illuminated by a single high speed video projector. This setup allows for 360° viewing, but is suitable only for limited image sizes and model complexity.

Parallel rendering for multi-projector displays. Driving our displays requires generating appropriate images for each projector, which is done for our large scale setup using a cluster-based solution. Such an approach to creating large multi-view displays has gained a lot of interest, because these architectures are economical, scalable in performance and resolution, and easy to upgrade [10–12]. Chen et al. [10] classify cluster-based display setups into two approaches: master-slave and client-server. In the master-slave setup, the dataset is mirrored across all the nodes and multiple instances of a program run in parallel on each cluster node. Each process has identical behavior and manages the entire dataset but renders only a certain portion of it. Representative examples of this approach are VR Juggler [13] and Net Juggler [14]. In the client-server setup, instead, a server distributes appropriate data to a number of rendering nodes and handles the synchronization among them. The approach is more general, and data distribution can follow sort-first, sort-last or hybrid strategies [15,16]. Data can be distributed in a transparent manner by intercepting calls at the graphics API level [11] or at the display manager level [17], as well as by implementing data distribution features at the scene graph level, as in OpenSG [18], Szyzygy [19], Blue-C [20], and Garuda [21]. Managing data distribution at the scene description level requires more application programmer effort, but offers more optimization opportunities, since data transfers can be performed at coarser scales, and high-level object structures can be exploited by culling algorithms to reduce network requirements. Our approach follows an object-based server-push philosophy, similar to the one employed in Garuda [21], that exploits client and server object caches and multicasting to reduce network load. However, our system is tailored to render massive models on a light field display, in which all rendering clients almost fully share the same view, and must use specialized techniques to adapt and project geometry onto the display. Load balancing strategies for sort-first [22,23] and sort-last [16,15,24,25] rendering algorithms have been proposed in the past with the goal of maximally exploiting hardware resources. They rely on primitive count heuristics and efficient image compositing methods to redistribute the images computed by rendering nodes to the appropriate display devices. Since all the projectors in our display typically look at the same portion of the displayed object, we take the straightforward approach of using a single LOD/culling front-end and statically assigned per-projector back-ends. Instead of focusing on per-view frustum culling, which are effective in a tiled display configuration, we focus on display-aware LOD selection and employ a multiresolution mesh structure instead of a general scene graph. Various GPU-oriented multiresolution structures have been recently proposed which obtain maximum performance by moving the granularity of the LOD representation from triangles to triangle patches [3,26,27]. In this work, we adapt one of these methods [3] to take into account the 3D display characteristics when producing variable accuracy approximations, and demonstrate its suitability for a cluster-parallel environment. The multiview image generation approach has also been taken for other 3D displays (see, e.g., [28]) but not applied to interactive massive model rendering.

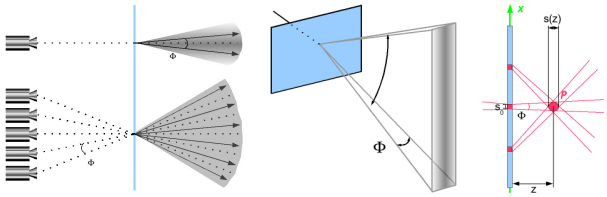
3. Light field display technology overview

The displays employed in this work are based on technology developed by Holografika [1] . In this section, we illustrate the main concepts behind them, which we exploited to develop our scalable massive model renderer. We refer the reader to [1] for more information.

A set of projectors is densely arranged behind a holographic screen. Each of them projects a specific image onto it to contribute to building up a light field (see figure 2(a)). Each projector emits light beams toward a subset of the points of the holographic screen. At the same time, each screen point is hit by more light beams coming from different projectors. By positioning mirrors at the sides of the display, it is possible to reflect back onto the screen the light beams that would otherwise be lost, thus creating virtual projectors that increase the display field of view. The holographic screen is the key element in this design, as it is the optical element enabling selective directional transmission of light beams. It is a holographically recorded, randomized surface relief structure able to provide controlled angular light divergence. The light diffusion characteristic of the screen is the critical parameter influencing the angular resolution of the system, which is very precisely set in accordance with the system geometry. In the ideal case, each point of the screen should be able to produce light beams at an arbitrary angle within the hemisphere bounded by the plane tangent to the surface at that point. As a first limitation to the display complexity, one can require the display to emit light beams only within a cone of angle ϕ, therefore limiting the effective 3D viewing to a conical *Field of View* of angle ϕ. For the displays presented here, $\phi = 50°$. Since our eyes are displaced horizontally and we move mainly in horizontal directions, the requirements on the display can be relaxed further by removing vertical directional selection, thus removing vertical parallax while maintaining the horizontal one. This obviously has a limited effect on motion parallax, a very important factor in setting up depth cues, since our head movements are mainly in the horizontal direction. In the horizontal parallax design, the projectors are arranged in a horizontal linear array and the angular light distribution profile induced by the screen is strongly anisotropic (see figure 2(b)). Horizontally, the screen surface is sharply transmissive, in order to maintain a sub-degree separation between views. Vertically, the screen scatters widely, so the projected image can be viewed from virtually any height. The angular light distribution profile introduced by the holographic screen, with a wide plateau and steep Gaussian slopes precisely overlapped in a narrow region in the horizontal direction, results in a homogeneous light distribution and continuous 3D view with no visible crosstalk within the field of depth determined by the angular resolution. If one assumes that each beam has a finite angular size Φ, with $\Phi = \phi/n$, n being the number of beams hitting a given screen pixel, we ensure a continuous coverage of the emission angle thus approximating continuous parallax. This, however, introduces a finite resolution effect – which is independent from the screen pixel resolution – in the reconstructed 3D scene. In fact, the size s of the smallest voxel than can be reproduced will depend (see figure 2(c)) on the distance z of its center from the screen and from the beam angular size Φ

$$s(z) = s_0 + 2\|z\| \tan(\frac{\Phi}{2}) \quad (1)$$

where s_0 is the pixel size on the screen surface. In other words, the achievable spatial resolution decreases with the distance from the screen. This is intuitive, because the illusion of the existence of a particular spatial point is generated by pyramidal beams crossing at a specific 3D position. This fact also practically limits the field-of-depth of the display, i.e., the maximum distance from the screen at which objects are faithfully reconstructed. For instance, the accuracy of the large display varies from $s_0 = s(0) = 1.50mm$ on screen to $s(300mm) = 5.69mm$, while for the small display the accuracy varies from $s(0) = 1.25mm$ on screen to $s(300mm) = 5.38mm$. As we will see, the particular light field geometry has important implications for the design of the rendering system.



(a) Schematic diagram (b) Holographic optical transformation (c) Voxel size

Fig. 2. Light field display principles

4. Parallel and sequential adaptive rendering

Reconstructing the light field of a rendered scene amounts to precomputing the projection parameters associated to each of the projectors and to using them for generating rendered views. Given the high pixel count of the display, and the high triangle count of the target models, appropriate techniques must be employed in order to meet timing constraints in interactive applications. In particular, it is of primary importance to parallelize image generation as well as to dynamically adapt rendering complexity by employing a multiresolution technique which takes into account the characteristics of the display.

Even though the general approach has similarities with current parallel view-dependent level-of-detail approaches, there are some extremely important differences. First of all, each frame is composed of many very similar views, and it is therefore appropriate to amortize level-of-detail selection costs over all views instead of repeating a view-dependent

[1] www.holografika.com

render call for each of the projector images. It is also important, for image continuity reasons, that all views agree on the same level-of-detail representation. This leads to an approach in which a common front-end system chooses a single per-frame level-of-detail. Moreover view-frustum culling can also be performed globally by generating a single set of visible nodes shared by all projectors, since all of them look at an almost identical portion of the displayed object. Moreover, geometry cannot be projected onto the display simply following a standard pinhole projection approach, because of the transformation made by the holographic screen on the rays that traverse it. Modeling this transformation requires a multiple-center-of-projection perspective, which cannot be achieved using the standard linear graphics pipeline and requires the use of appropriate vertex shaders. In addition, the multiresolution renderer cannot exploit the position of a particular viewer to select a level of detail, since an unlimited number of viewers must be free to move in a very large workspace in front of the display. In our approach, adaptive rendering exploits the finite spatial resolution of the display to perform adaptation. Finally, nonlinear geometric and color correction must be performed to undo distortions due to lenses and approximate mechanical calibration, as well as to correct the different color, contrast, and intensity response of the projectors. This leads to an approach in which the usual rendering pipe-line is fine-tuned by a nonlinear warping and photometric correction pass, as in standard multi-projector display systems [29]. The various components of our system are described in the following subsections.

4.1. *Renderer design*

Even though, in principle, it is possible to use, for maximum performance, one PC per projector, benefit/cost analysis leads to a configuration in which multiple projectors are controlled by a single PC. On current architectures, for instance, a dual twin-view solution is very well supported by most desktop mainboards. For this reason, controlling 4 outputs/machine does not require any special hardware but just the connection of two dual-DVI graphics board on the PCI Express bus (no need for SLI). This solution reduces the number of PCs by a factor of 4, and the graphics performance to less than 4 because of the reduced synchronization needs. The large scale display described here uses such a configuration.

Our parallel rendering method uses a distributed image generation system that can be implemented on a cluster, with a front-end client PC selecting the level of detail from the multiresolution structure and multicasting graphics commands to back-end PCs. The characteristics of multiresolution techniques based on coarse grained adaptation are exploited to efficiently distribute data to back-end nodes as well as to efficiently pass them to the GPU through preferential paths. With this overall design, the display driven by a single PC can be thought of as a particular instance of the parallel version, where the back-end and the front-end are collapsed into a single machine that sequentially renders all projector images. In this case, a DVI connection can be used to drive the display, which must, however, contain specific electronics to decode the DVI stream and dispatch images to the appropriate projectors.

The overall architecture of the parallel rendering system is depicted in figure 3, and its main components are discussed in the following sections.

4.2. *Multiresolution structure overview*

The TetraPuzzles structure uses a conformal hierarchy of tetrahedra generated by recursive longest edge bisection to spatially partition the model in a preprocessing step. Each tetrahedral cell contains a precomputed simplified version of the original model. The representation is constructed offline during a fine-to-coarse parallel out-of-core simplification of the surface contained in diamonds (sets of tetrahedral cells sharing their longest edge). Appropriate boundary constraints are introduced in the simplification process to ensure that all conforming selective subdivisions of the tetrahedron hierarchy lead to correctly matching surface patches (see [3] for more details). The main advantage of the method is its ability to rapidly produce seamless variable accuracy reconstructions by assembling precomputed patches.

In a first construction phase, the original input triangle soup is partitioned in a top-down fashion. The end result is a DAG of diamonds, and a set of triangle buckets associated with leaf tetrahedra that cover the mesh. After the partitioning phase, the inner nodes are constructed in parallel by fine-to-coarse diamond-by-diamond constrained simplification. In this work, a number of modifications were made with respect to the original TetraPuzzles implementation. First of all, instead of simply partitioning input triangles according to centroid location, we clip them at diamond boundaries and, also, subdivide input triangle edges that are longer than 3/2 the input mesh average edge length. This approach ensures an input model with small near uniform triangles, and simplifies boundary management during the coarse-to-fine simplification step, since locked vertices during diamond simplification are simply the endpoints of boundary edges lying on diamond faces. We thus do not require the extraction of the original mesh boundary and the maintenance of boundary flags, as in [3]. Moreover, since rendering for the 3D display requires the adaptation of vertices to very small (voxel-sized) triangles, controlling triangle shapes during simplification to reduce triangle counts in nearly flat areas is no longer important. Thus, instead of performing high-quality (quadric based) simplification as in [3], we construct diamonds with a simplification method that produces (roughly) uniformly tessellated meshes, and use edge length as a measure of tessellation accuracy. This approach allows us to manage colored meshes by simply using a color-per-vertex representation.

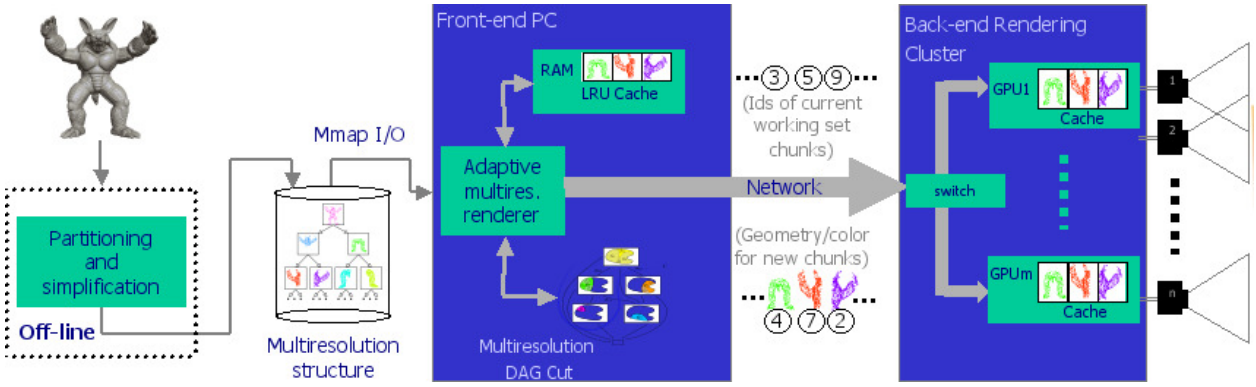


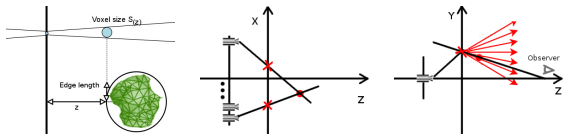
Fig. 3. Rendering architecture.

4.3. *Rendering front-end: selection of levels of detail*

As in [3], the nested subdivision hierarchy is encoded as a forest of binary trees, and we employ a saturation technique [30] to extract conforming meshes without requiring neighbor finding. Therefore, each tree is stored as a memory mapped linear array, and each of its nodes, corresponding to a particular tetrahedron, contains just the following information: a reference to the associated patch data (vertex attributes and connectivity in stripified form) in a patch repository; the tight bounding sphere for the patch; the saturated model space average edge length and bounding sphere of the neighborhood (maximum among diamond's tight values and saturated values for children); the index of child nodes in the linear arrays, which corresponds to the two tetrahedra generated by bisection. With such a structure, variable resolution rendering is implemented by simple stateless top-down traversals of the binary trees used to encode the tetrahedron hierarchy, which combine view-frustum and contribution culling. The traversal is performed once per 3D frame, and consequently generates as a result the set of patches that needs to be rendered for all the views. The standard view-dependent technique must thus be adapted to become the required spatial accuracy-dependent technique.

As we recurse the hierarchy, we test whether the current node is invisible by checking the tight bounding sphere of the associated patch against the spatial display working volume, determined by screen dimension, viewing angle, as well as achievable field of depth. If a node is found out of the working volume, we simply stop recursion, culling away the entire branch of the tree. If the node is potentially visible, we test whether its patch is an accurate enough representation by measuring its saturated spatial tessellation accuracy, which depends on its position within the volume. If so, we can add the associated patch to the active patch set for the frame, otherwise we continue the recursive refinement with the node's children.

Saturated spatial tessellation accuracy is the quantity that guides refinement, to achieve a target of (no more than) one vertex per voxel. Since the method exploits error saturation to encode dependencies, particular care must be taken to ensure that view-dependent measures are monotonically decreasing as we descend in the hierarchy and produce the same value for all tetrahedra in the same diamond. In our system, we obtain a consistent upper bound on the view-dependent error by measuring the apparent size of a sphere equal in diameter to the saturated average edge lengths of the patch and centered at the saturated bounding sphere point closest to the display screen (see figure 4(a)). If this value is higher than the display voxel resolution at that same position, computed from equation 1, the node needs refinement, otherwise we can safely stop refinement and consider the node for rendering.



(a) Position-dependent accuracy control (b) Horizontal projection geometry (c) Vertical projection geometry

Fig. 4. Projecting graphics onto the display.

At the end of the traversal, all nodes required for holographic rendering have been identified, and rendering can proceed by generating the projector images. In the single PC configuration, all views are rendered sequentially and all the images are sent to the projectors through the DVI interface. From a software point of view, the front-end can be collapsed to just the refinement module, which extracts the view dependent representation of the model to be rendered and manages a RAM LRU of the selected patches, while the back-end module takes care of rendering the model representation for all the different views and manages a GPU LRU of the current patches. In the cluster version, rendering is instead done in parallel by all back-end PCs. Since all nodes share almost the same view, we use a pure sort-first distribution approach in which all patches to be rendered are broadcast to all rendering PCs without any sorting or filtering at the source. In order to save bandwidth, the LRU cache maintained in the front-end is exploited by sending only patches not already in cache and referring to already sent patches by patch id. Because of space-time coherence, only few patches per frame need to be coarsened or refined

and therefore caching aggressively reduces bandwidth requirement. In the parallel implementation, since all back-end nodes receive the same list of patches to be rendered, a multicast protocol can be effectively used to further reduce network load. In the current system, communication between front-end and back-end nodes goes through a dual gigabit switch supporting GIMP snooping. At the end of the frame, the front-end synchronizes with the back-end nodes through a barrier operation, performed before calling the hardware swap-buffers on all rendering nodes. Optionally, for control purposes, the front-end node can also render a frame by using the selected patches and showing them in a standard 2D window by taking the picture from a central viewpoint.

It must be noted that the TetraPuzzles structure offers additional front-end parallelization possibilities. In particular, since level of detail selection is implemented through a stateless recursive visit of a forest of binary trees, concurrent traversal is straightforward to implement. However, in our current implementation, we decided to keep the front-end sequential, since the typical size of the traversed coarse grained hierarchy is very small (typically one to few hundreds tetrahedra/frame) and traversal time is negligible in comparison to back-end nodes rendering times.

4.4. *Rendering back-end: controlling the displays*

The rendering back-end drives the 3D display by decoding the stream received by the front-end. It can range from a single graphics PC (for the purely sequential version), to an array of PCs connected to the front-end through high-speed networking components. Each of the back-end PCs is connected to the display by DVI connections and runs a server agent that controls an OpenGL framebuffer. The server is responsible for generating the images associated to a fixed subset of the display projectors from the original stream (matrix transforms and patches). As explained above, each back-end node typically controls more than one projector (all of them for the sequential version), and must thus render each of the patches in the working set several times (see fig. 5(a)).

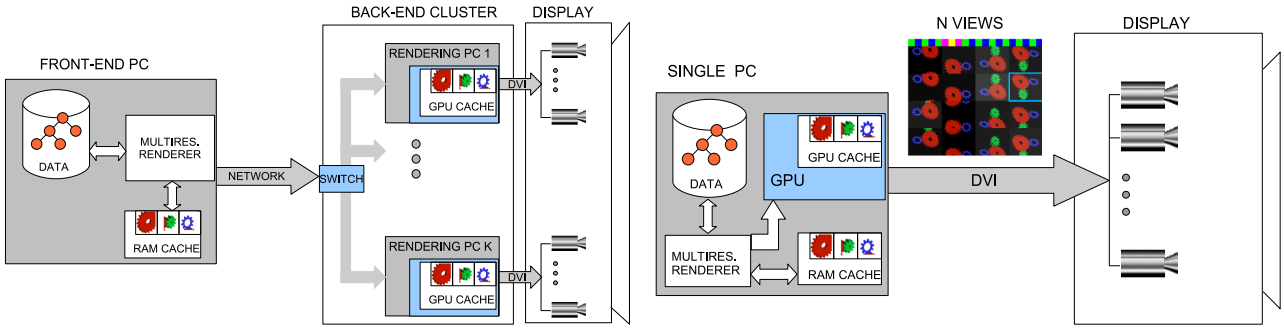
Each of the back-end PCs thus buffers all the commands received from the front end before starting the rendering operation. When all commands for a given frame have been received, a rendering loop iterates on all associated projectors. All identified patches are then traversed and rendered from the projector's point of view in a projector's viewport. In this step, multiple center of projection perspectives implemented in vertex shaders are used to take into account the anisotropic diffusion behavior of the holographic screen. Because of the horizontal parallax-only approach, it is not possible to reconstruct a full light field. Providing a good illusion of seeing floating objects requires the application to take decisions on how to deal with the missing vertical degree of freedom. In our approach, we consider a solution which is exact for observers lying at a fixed height and distance from the screen. For projecting geometry onto the display, we thus introduce a virtual observer at a fixed height and distance from the display, and project graphics onto the display using a multiple-center-of-perspective projection that has the projector position as the center of projection for horizontal coordinates (see figure 4(b)) and the virtual observer for the vertical ones (see figure 4(c)). The solution is exact for all users at the same distance from screen and height as the virtual observer and proves to be a good approximation for other positions in the display workspace.

For both the parallel and the sequential implementation, in order to take advantage of spatial and temporal coherency among views also in back-end nodes, each back-end node contains a memory manager, based on the same LRU strategy used in the front-end, which explicitly manages graphics board memory, using OpenGL *Vertex Buffer Objects* to store patches. Each time we need to render a patch, we reuse the cached version if present, otherwise we render it and cache its representation in place of the oldest one. Least-recently used patches are deleted when the cache becomes full. By making sure that back-end caches are at least as big as the front-end one, we ensure that front-end and back-end caches remain properly synchronized (i.e., the front-end will never refer to a deleted object). Moreover, since all projectors share the same active object set, cache misses for a given patch can happen at most once per spatial frame. Therefore, rendering N projector views costs less than N times the rendering of a single view. At end-of-frame, as in all tiled projector displays (see, e.g., [31]), non-linear photometric and geometric corrections are applied in a post-pass before synchronizing with the frame barrier and swapping buffers.

In the high-end parallel version, each DVI output controls a single projector. Thus, generating an image for a projector is just a matter of defining the correct viewport on a full-screen window. The sequential desktop version, instead, has to control many projectors with a single DVI output (fig. 5(b) In this case, there is a need to distribute projector images over multiple DVI frames, and to dispatch these images to the correct projectors. This is done by grouping projector images for a given frame into successive batches, and using color codes in reserved scan lines of the DVI streams to associate sub-images to specific projectors.

5. Implementation and Results

We have implemented a software system based on the design discussed in this paper, and tested on both the displays. The display hardware and software components have been produced by Holografika. The multiresolution renderer discussed here has been designed and implemented by the authors. Both multiresolution rendering front-ends run on Linux on an Athlon64 3300+ PC with a NVIDIA7800GT graphics board and a local SATA disk for storing models. The test models are two high resolution laser scan datasets:



(a) Parallel rendering architecture

(b) Sequential rendering architecture

Fig. 5. Driving large and small scale displays.

a colored version of Michelangelo's David 1mm model (56M triangles) and a colored wooden statue model (6M triangles) acquired by combining laser scan data with high resolution digital photographs.

Model preprocessing. The multiresolution hierarchies were generated on a cluster of 15 PCs running Linux 2.4. Each PC has two CPU Athlon 2200+ CPUs, 1GB DDR memory, a 70GB ATA 133 hard disk, and an Ethernet 100 Mbps network connection. We constructed all multiresolution structures with a prescribed maximum leaf size of 4000 triangles/tetrahedron for the partitioning phase and an average non-leaf size of 2000 triangles/tetrahedron for the bottom-up construction phase. Processing time for the wooden statue model is 142s (42K triangles/s), while it is 2722 s (21K triangles/s) for the David model, which has a larger I/O overhead. The speed-up with respect to the original TetraPuzzles implementation [3] is consistent (1.2x to 1.4x), even though this version handles color data and is due to the simpler simplification strategy as well as to the improved boundary management technique.

Sequential rendering on the small scale display. The main goal of the single PC/small display solution is to demonstrate that current desktop graphics components provide enough computing power to drive a simple Holodeck-style massive model manipulator. The small scale display is able to visualize 7.4Mbeams originating from 96 320x240 LCD fast displays. It has an horizontal resolution of 0.8 degrees and covers a field of view of 50 degrees, providing continuous horizontal parallax, with a screen size of 400 by 500 mm. In order to comply with the single DVI interface of the display, consisting in a stream of 1280x1024 images, the images required for a full update of a holographic frame are grouped into 8 successive batches. The top row of each rendered image encodes the ids of the updated modules using a color code. A dedicated board in the display decodes each DVI frame and dispatches sub-images to the projectors depending on the color code. Since the update frequency of the monitor is 75 Hz, and after each image batch we perform a a swap buffer operation synchronized with the v-sync signal, the 3D renderer can not achieve frequencies higher than $75/8 = 9.375$ fps, when connected to the display through a single DVI. We have tested the small scale display with a hand tracking system using a simple markerless vision based gesture recognition mechanism that allows a user to intuitively interact with an object which is placed in the light field display working volume [32]. Our main target was to give the user the ability to position and scale the synthesized object in space with the same gestures that he would make to position a real 3D object in a real environment. We adopted a two-hand solution for positioning objects rendered by the display because this approach gives a more intuitive interface for scaling and rotating an object than an equivalent interface based on a single 3D cursor. Three types of two-hand gestures are recognized: translation, rotation and scaling. A movement starts when both hands are in the working area of the two cameras and we detect that they are closed (as it would be done to grab a real object), and stops when the hands release the object or when they move out of the working area (see figure 6). In order to guarantee interactive performance, we decided to adopt two thresholds for the LOD selection: one for static high quality images and one for dynamic situation, thus allowing user interaction. Our frame rates are about 2 Hz when performing rendering at full accuracy (1 triangle/voxel) with a precision of 1.25mm on the screen, while with a 4 times reduced precision during interaction frame rates range from 5 to 9 Hz.

Parallel rendering on the large scale display. The large scale display is capable of visualizing 35Mbeams by composing images generated by 72 SVGA LED commodity projectors. The display provides continuous horizontal parallax within an approximately 50-degree horizontal field-of-view. The parallel rendering back-end, which drives the 72 projectors, is currently running on an array of 18 Athlon64 3300+ Linux PCs equipped with two NVIDIA8600GTS graphics boards running in twin-view mode (i.e., each back-end node controls four projectors through four DVI outputs). It is obviously impossible to fully convey the impression provided by our system on paper. As a simple illustration of our system's current status and capabilities, we analyze the behavior of an application that allows users to interactively translate, rotate, and scale models in 3D

Fig. 6. Successive moments of interactive manipulation of the David 1mm dataset (56M triangles) using our 7.4Mbeams display coupled with the vision based hand tracker and driven by a single PC.

space using a Logitech 3D mouse input device for direct manipulation. We recorded the performance using a hand held video camera freely moving in the display workspace. Representative video frames are shown in figure 7. Note the parallax effects and the good registration between displayed object space and physical space, which demonstrate the multi-user capability of the display (see figure 8). The perceived image is fully continuous. This is qualitatively very different from other contemporary multiview display technologies, which force users into approximately fixed positions, because of the abrupt view-image changes that appear at the crossing of discrete viewing zones [8]. The performance of the application is obviously much higher than for the small display test case bacause of the parallel solution. In the large display case, we are thus able to display the models at maximum accuracy (1 triangle/voxel) even during interaction. During typical interaction tasks, the frame rate ranges from 14 to 42 Hz, depending on the scale of the object and on the number of new patches created, with an average throughput of $10 Mtri/s$. Only few patches per frame need to be updated when the object is rotated or translated. On the other hand scaling the model typically requires updating most of its representation, since the triangle-size/voxel ratio rapidly changes. This implies the creation and transfer to the back-end GPUs of new patches, which is the most critical work done by the renderer. Both test models have about 2000 triangles per patch and each VBO patch takes up about $30KB$ (1000 vertices with 3 float per position, 3 short per normal and 4 bytes per color, plus about 3500 indices stored as short). Zooming the David model from the minimum scale (60 patches) to the maximum scale ($314 patches$) requires the creation of about 1000 new patches to go through all intermediate representations from the farthest to the nearest view. We tested the back-end, which is able to create up to 1400 VBO patches per second. From the network point of view, matching the graphics performance would require a bandwidth of $336 Mbps$ in multicast mode and $6 Gbps$ in unicast to transfer data from the front-end to the 18 back-end nodes. Reaching such a performance is not strictly required for the application. If we perform this zooming operation in about 5 seconds, the renderer has to create 200 patches per second, which means that we roughly need a bandwidth of $50 Mbps$ in multicast (and $900 Mbps$ in unicast), which is less than the maximum throughput of our network. Nonetheless, during rapid zooming users can perceive a decrease in feedback smoothness, since patch creation is not evenly distributed, and some frames can incur substantial delays (up to $0.5s$). In order to have a fully interactive rendering we plan to move to a dual queue algorithm, such as the one presented in BMT [27], where the cut extraction process is interruptible and the renderer can adapt within a time budget.

6. Conclusions and Future Work

The main take home message of this work is that light field display technology is starting to be mature enough to support complex applications, and that coarse-grained view-dependent multiresolution rendering techniques designed for single user view-dependent rendering can be effectively transformed into 3D display-aware spatially adap-

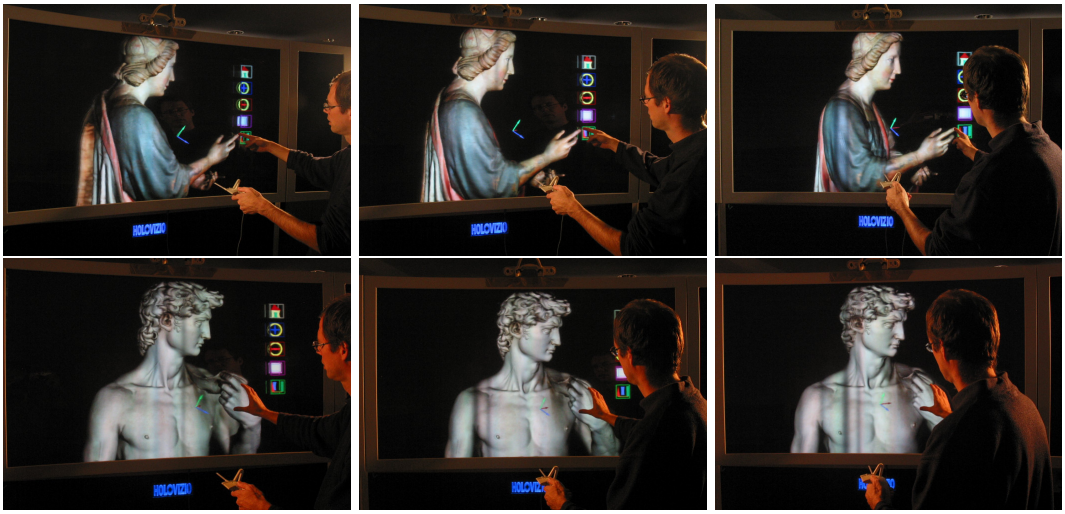


Fig. 7. Images were taken from different point of views during interactive manipulation of multi-million triangles colored datasets on the 35Mbeams display driven by a cluster of PCs.

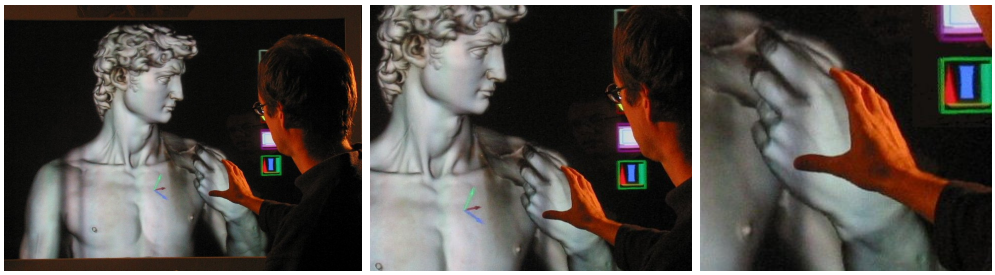


Fig. 8. Object inspection closeups: during manipulation objects appear to be floating in the display workspace.

tive sort-first cluster parallel renderers. State-of-the-art 3D display technology can thus be exploited to explore extremely complex and highly detailed datasets. The applications discussed here are clearly meant to work as an enabling technology demonstrator, as well as a testbed for integrated 3D interaction, massive model rendering, and display research.

Acknowledgments. This research is partially supported by the CYBERSAR project. The authors are grateful to the Holografika team for the design and implementation of display hardware and low-level software. Test models are courtesy of the Digital Michelangelo Project and CNR ISTI VCG. We also acknowledge the key technical contributions of Eric Bouvier and Marco Agus.

References

[1] T. Balogh, T. Forgacs, T. Agocs, O. Balet, E. Bouvier, F. Bettio, E. Gobbetti, G. Zanetti, A scalable hardware and software system for the holographic display of interactive graphics applications, in: Proc. Eurographics Short Papers, 2005.

[2] A. Jones, I. McDowall, H. Yamada, M. Bolas, P. Debevec, Rendering for an interactive 360 degrees light field display, ACM TOG 26 (3) (2007) 40.

[3] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, R. Scopigno, Adaptive TetraPuzzles – efficient out-of-core construction and visualization of gigantic polygonal models, ACM TOG 23 (3) (2004) 796–803.

[4] F. Bettio, E. Gobbetti, F. Marton, G. Pintore, Multiresolution visualization of massive models on a large spatial 3D display, in: Proc. EGPGV, 2007.

[5] N. A. Dodgson, Autostereoscopic 3D display, Computer 38 (8) (2005) 31–36.

[6] T. Agocs, T. Balogh, T. Forgacs, F. Bettio, E. Gobbetti, G. Zanetti, A large scale interactive holographic display, in: Proc. IEEE EDT, 2006.

[7] T. Balogh, T. Forgacs, O. Balet, E. Bouvier, F. Bettio, E. Gobbetti, G. Zanetti, A scalable holographic display for interactive graphics applications, in: Proc. IEEE EDT, 2005.

[8] W. Matusik, H. Pfister, 3D TV: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes, ACM TOG 23 (3) (2004) 814–824.

[9] N. A. Dodgson, Analysis of the viewing zone of the cambridge autostereoscopic display, Applied Optics: Optical Technology & Biomedical Optics 35 (10) (1996) 1705–1710.

[10] H. Chen, D. W. Clark, Z. Liu, G. W. K. Li, Y. Chen, Software environments for cluster-based display systems, in: Proc. IEEE International Symposium on Cluster Computing and the Grid, 2001.

[11] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, J. T. Klosowski, Chromium: a streamprocessing framework for interactive rendering on clusters, in: Proc. SIGGRAPH, 2002, pp. 693–702.

[12] Viswall high resolution display wall, http://www.visbox.com/wallMain.html.

[13] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, C. Cruz-Neira, VR Juggler: a virtual platform for virtual reality application development, in: Proc. IEEE VR 2001, 2001, pp. 89–96.

[14] J. Allard, V. Gouranton, L. Lecointre, E. Melin, B. Raffin, Net Juggler: Running VR juggler with multiple displays on a commodity component cluster, in: Proc. IEEE VR, 2002, pp. 273–274.
[15] R. Samanta, T. Funkhouser, K. Li, Parallel rendering with K-way replication, in: Proc. IEEE PVG, 2001, pp. 75–84.
[16] R. Samanta, T. Funkhouser, K. Li, J. P. Singh, Hybrid sort-first and sort-last parallel rendering with a cluster of PCs, in: Proc. ACM/EG Graphics Hardware, 2000, pp. 97–108.
[17] Distributed multihead X project (DMX), http://dmx.sourceforge.net.
[18] G. Voss, J. Behr, D. Reiners, M. Roth, A multithread safe foundation for scene graphs and its extension to clusters, in: Proc. PGV, 2002, pp. 33–37.
[19] B. Schaeffer, C. Goudeseune, Syzygy: Native PC cluster VR, in: Proc. IEEE VR, 2003, pp. 15–22.
[20] M. Naef, E. Lamboray, O. Staadt, M. Gross, The Blue-C distributed scene graph, in: Proc. EGVE, 2003, pp. 125–133.
[21] Nirnimesh, P. Harish, P. Narayanan, Garuda: A scalable, geometry managed display wall using commodity PCs, IEEE Transactions on Visualization and Computer Graphics 13 (5) (2007) 864–877.
[22] C. Mueller, Hierarchical graphics databases in sort-first, in: Proc. IEEE Symposium on Parallel Rendering, 1997, pp. 49–58.
[23] R. Samanta, J. Zheng, T. Funkhouser, K. Li, J. P. Singh, Load balancing for multi-projector rendering systems, in: Proc. ACM/EG Graphics Hardware, 1999, pp. 107–116.
[24] W. T. Correa, J. T. Klosowski, C. T. Silva, Out-of-core sort-first parallel rendering for cluster-based tiled displays, in: Proc. EGPGV, 2002, pp. 89–96.
[25] X. Cavin, C. Mion, Pipelined sort-last rendering: scalability, performance, and beyond, in: Proc. EGPGV, 2006.
[26] S.-E. Yoon, B. Salomon, R. Gayle, D. Manocha, Quick-VDR: Interactive view-dependent rendering of massive models, in: Proc. IEEE Visualization, 2004, pp. 131–138.
[27] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, R. Scopigno, Batched multi triangulation, in: Proc. IEEE Visualization, 2005, pp. 207–214.
[28] W.-S. Chun, J. Napoli, O. Cossairt, R. Dorval, D. Hall, T. Purtell, J. Schooler, Y. Banker, G. Favalora, Spatial 3-D infrastructure: Display-independent software framework, high-speed rendering electronics, and several new displays, in: Proc. SPIE-IS&T Electronic Imaging, Vol. 5664, SPIE, 2005, pp. 302–312.
[29] R. Yang, D. Gotz, J. Hensley, H. Towles, M. S. Brown, PixelFlex: a reconfigurable multi-projector display system, in: Proc. IEEE Visualization, 2001, pp. 167–174.
[30] M. Ohlberger, M. Rumpf, Adaptive projection operators in multiresolution scientific visualization, IEEE Transactions on Visualization and Computer Graphics 4 (4) (1998) 344–364.
[31] J. Binns, G. Gill, M. Hereld, D. Jones, I. Judson, T. Leggett, A. Majumder, M. McCroy, M. Papka, R. Stevens, Applying geometry and color correction to tiled display walls, in: IEEE Visualization - Poster sessions, 2002.
[32] F. Bettio, A. Giachetti, E. Gobbetti, F. Marton, G. Pintore, A practical vision based approach to unencumbered direct spatial manipulation in virtual worlds, in: Eurographics Italian Chapter Conference, 2007.