

View-dependent Exploration of Massive Volumetric Models on Large Scale Light Field Displays

José Antonio Iglesias Guitián · Enrico Gobbetti · Fabio Marton

Received: February 2, 2010 / Accepted: March 19, 2010

Abstract We report on a light-field display based virtual environment enabling multiple naked-eye users to perceive detailed multi-gigavoxel volumetric models as floating in space, responsive to their actions, and delivering different information in different areas of the workspace. Our contributions include a set of specialized interactive illustrative techniques able to provide different contextual information in different areas of the display, as well as an out-of-core CUDA based raycasting engine with a number of improvements over current GPU volume raycasters. The possibilities of the system are demonstrated by the multi-user interactive exploration of 64GVoxels datasets on a 35MPixel light field display driven by a cluster of PCs.

1 Introduction

Volumetric datasets are growing at incredible rates in terms of number and size, resulting in two challenges: maintaining performance and extracting meaningful information. These two challenges are closely related, since user interaction, which imposes real-time constraints, is a key to volumetric understanding. In the last few years, GPU accelerated methods have emerged as a viable solution for achieving real-time performance on moderately sized datasets. Recent results on out-of-core techniques [8,6] have shown how to extend these results to datasets of potentially unlimited size. The images produced by such techniques are impressive, but often contain many overlapping structures, leading to cluttered depiction of shape information. As a result, volume rendering images are often hard to interpret. Enhancing depth and shape perception in volumetric rendering is thus

a very active research area, which is tackled under many angles. In this paper, we report on an illustrative multiresolution volume rendering system capable to interactively drive large multi-projector light field displays offering continuous horizontal parallax in a room-size workspace. Our contributions include: a virtual environment enabling multiple naked-eye users to perceive detailed multi-gigavoxel models as floating in space and responsive to their actions; a set of interactive illustrative techniques able to provide different contextual information in different areas of the display; an adaptive out-of-core CUDA rendering engine with a number of improvements over current GPU volume renderers and integrated within a cluster-parallel system. The approach delivers 3D visualizations of unprecedented quality, which enhance volumetric understanding, especially in the presence of depth-oblivious rendering techniques.



Fig. 1 Multi-user interactive exploration of a 64Gvoxel dataset on a 35Mpixel light field display. Users freely mix and match 3D tools creating view-dependent illustrative visualizations. Objects appear floating in the display workspace, providing correct parallax cues while delivering direction-dependent information.

2 Related work

Our system extends and combines state-of-the-art results in a number of technological areas. In the following, we only discuss the approaches most closely related to ours.

Rendering for multi-projector light field display. The display hardware employed in this work is based on commer-

cially available technology developed by Holografika¹. Our image generation methods take into account the display characteristics in terms of both geometry and resolution of the reproduced light fields. In particular, we extend a multiple-center-of-projection technique [12, 2] with a depth compression factor. We use the common sort-first parallel rendering approach, but rely on an adaptive out-of-core GPU renderer rather than fully replicating data across render nodes. The assignment between rendering processes and images is static, even though load balancing strategies based on image partitioning could be implemented on top of our framework (e.g., [16]).

GPU accelerated out-of-core volume rendering. GPU volume rendering is a well established approach (see, e.g., a recent survey [18]). Most modern techniques perform full ray traversal on the GPU. In this context, large volume data is handled by compressing it using adaptive texturing schemes to fit entire datasets into GPU memory [22], or by using flat [14] or hierarchical [8, 6] multiresolution structures in conjunction with adaptive loaders. Our method builds on the ability to rapidly traverse an octree structure, and is based on the stackless ray traversal method for kd-trees [9, 17], recently adapted to GPU volume rendering [8]. In this paper, the method is extended to support preintegration, a more flexible marching and compositing engine, and adaptive view-dependent blending among multiple levels of details. In addition, our CUDA algorithm better takes advantage of visibility information gathered during traversal to avoid loading occluded data. The scheme of Gobbetti et al. [8] is based on occlusion queries, scheduled so as to reduce end-to-end latency. Crassin et al. [6] exploit multiple render targets to store a subset of the traversed nodes for each pixel. Spatio-temporal coherence has to be exploited to try not to miss visible nodes. Instead, we take advantage of the scatter memory write capability of CUDA threads to the visibility of all rendered octree blocks.

Reducing image generation costs. We improve rendering performance by reducing the amount of pixels fully recomputed at each frame. The methods employed are similar in spirit to classic methods such as adaptive frameless rendering [7]. In our case, however, we do not rely on random or error driven sparse pixel sampling, but we group pixel batches in sub-frames aligned on a quincunx lattice, in order to avoid destroying neighboring ray coherence. Our method also bears a number of similarities with interleaved sampling [20, 23, 10], which reconstructs a smooth approximation of some shading component from an estimate based on a subset of the pixels.

Illustrative visualization. Our flexible renderer employs both photorealistic and illustrative techniques. The accumulation

scheme is similar to MIDA [4], even though we accumulate importance rather than intensity. The usage of an importance transfer function has also been advocated by Viola et al. [21]. In order to deal with high frequency transfer functions, we employ preintegration for all attributes, rather than only for colors as is usually done. Similarly to [13], we use extinction-weighted colors and reconstruct opacity on the GPU, but we employ a numerically stable approach with small memory requirements. Our tools bear a number of similarities with previous context-preserving techniques [3, 15], but extend them with view-dependent effects tuned for 3D displays.

3 Virtual environment overview

Our integrated system enables multiple naked-eye users to perceive detailed multi-gigavoxel models as floating in space, responsive to their actions, and providing different information in different areas of the workspace. The system concept is illustrated in Fig. 2. Presented images combine ray-traced volumetric data with rasterized surface-based information, such as labels or user interface tools. Visualization is both view-dependent and view-specific. In this system, which naturally supports multi-user discussion, much of user interaction is replaced by simple, natural head motions, reducing interface complexity and user burden. We improve over recent work in this area [2, 1] by proposing a set of specialized illustrative techniques based on a flexible compositing engine and designed to exploit the view-dependent characteristics of the display to provide different contextual information in different viewing areas.

The light field display used in this work is based on projection technology and combines a holographic screen with a specially arranged projector array driven by a PC cluster. By appropriately modeling the display geometry, the light beams leaving the various pixels can be made to propagate in specific directions, as if they were emitted from physical objects at fixed spatial locations. As is the case with other multi-screen displays, we use a distributed image generation system implemented on a cluster, with a front-end PC coordinating many rendering back-end PCs. The front-end PC is connected to 3D input devices and manages the user-interface, which supports, in addition to the standard viewing operations, specialized view-dependent volume exploration tools (see Fig. 1 and Sec. 4). The system uses a sort-first parallel rendering approach, in which each back-end PC is responsible only for the images associated to the connected projectors. 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 using multiple

¹ www.holografika.com

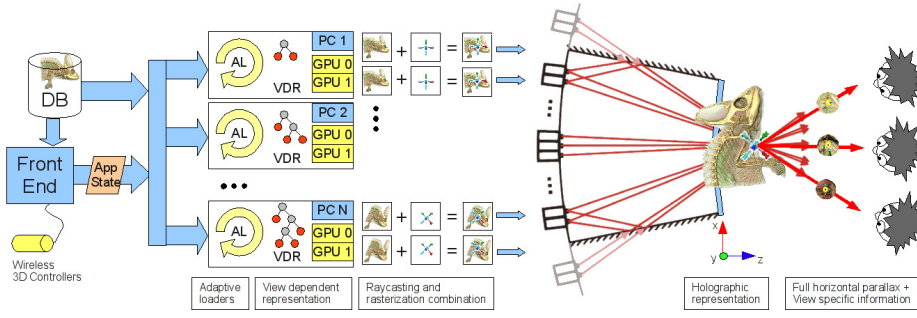


Fig. 2 Virtual environment concept. Presented images combine ray-traced volumetric data with rasterized surface-based information, such as labels or user interface tools. Visualization is both view-dependent, since it provides full parallax, and view-specific, since it delivers different information in different areas of the workspace.

graphics boards. Each back-end process controls a frame-buffer portion, and is capable to overlay a standard OpenGL rendered image, e.g., for 3D GUI objects, on top of a volume rendered image. Partitioning images rather than data simplifies the use of an unrestricted rendering kernel. This approach is feasible even when rendering massive models, since our out-of-core single pass volume raycaster with visibility feedback provides output-sensitive performance through occlusion culling and levels-of-detail. Thus, by controlling image size we also control data complexity. Scalability is ensured by a CUDA multiresolution rendering engine which builds on and improves over state-of-the-art massive volumetric renderers [8,6] (see Sec. 5).

4 Illustrative rendering for the light field display

The view-dependent characteristics of the display can be exploited to develop specialized interactive illustrative techniques designed to improve spatial understanding. With such techniques, simple head motions can reveal new aspects of the inspected data. In the following, we will first describe the overall traversal and compositing framework, based on a depth oblivious technique, and then illustrate examples of techniques that can be implemented in our framework by defining appropriate “color enhancers” associated to tools manipulated with a 3D cursor. As we will see in Sec 5 and Sec. 6, thanks to the performance and flexibility of our system, these techniques can be also freely mixed and matched within a single interactive application.

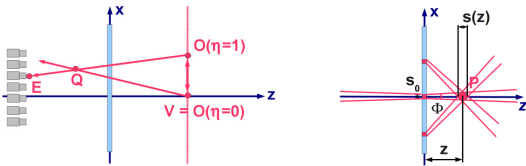


Fig. 3 Light field display projection. Left: we smoothly transition from standard single view perspective rendering (with $\eta = 0$) to full horizontal parallax (with $\eta = 1$). Right: the display has a depth-dependent spatial resolution determined by Φ .

Projecting graphics to the display. Following [12,2], we employ a multiple-center-of-projection technique for generating images with good stereo and motion parallax cues.

We have extended the approach by introducing a multi-view factor which enables us to optionally compress depth ranges to fit large scenes within the display workspace. Our technique is based on the approach of fixing the viewer’s height and distance from the screen to those of a virtual observer in order to cope with the horizontal parallax only design. We assume that the screen is centered at the origin with the y axis in the vertical direction, the x axis pointing to the right, and the z axis pointing out of the screen. Given a virtual observer at $\mathbf{V}$, the ray origin passing through a point $\mathbf{Q}$ is then determined by:

$$\mathbf{O} = ((1 - \eta)(V_x) + \eta(E_x + \frac{Q_x - E_x}{Q_z - E_z}(V_z - E_z)), V_y, V_z) \quad (1)$$

where $\mathbf{E}$ is the position of the currently considered projector, and η is a interpolation factor, which allows us to smoothly transition from standard single view perspective rendering (with $\eta = 0$) to full horizontal parallax (with $\eta = 1$). In the latter case, the solution is exact for all viewers at the same height and distance from the screen as the virtual observer and proves in practice to be a good approximation for all other viewing positions in the display workspace. When $0 \ll \eta < 1$, the 3D effect persists and the visible depth range of the presented images is compressed. This reduces the direct mapping between real and virtual world, as the image is not exact for any observer position, but allows us to map large objects within the restricted display range while maintaining a 3D effect for better spatial understanding (see accompanying video). Knowing how to compute $\mathbf{O}$ for every 3D point $\mathbf{Q}$ allows us to determine where $\mathbf{Q}$ should be drawn on a given projector to produce a perspective correct image using either a raycaster or a rasterizer. It is also important to note that the display has a depth-dependent spatial resolution determined by Φ , approximated by $s(z) = s_0 + 2\|z\| \tan(\frac{\Phi}{2})$, where s_0 is the pixel size on the screen surface. We employ $s(z)$ to determine the level of detail at which volumes have to be sampled and the amount of blending for MipMapping by quadrilinear filtering (see Fig. 3 and Sec. 5).

Flexible ray traversal and compositing. Our volume renderer employs scalar transfer functions to associate optical properties to scalar values. The renderer works with extinction weighted colors, and we thus have a transfer function

$\tau(s) \in [0, \infty[$ for the extinction coefficient and a transfer function $c(s)$ for the color, that has to be multiplied by $\tau(s)$ to yield an actual color intensity $\tilde{k}(s) = \tau(s)c(s)$ for a given scalar value s . In addition, we have a transfer function $\iota(s) \in]0, 1[$ for the importance of scalar value s , and $\nu(s) \in]0, \infty[$ for the refraction index. In order to deal with high frequency transfer functions, we use a preintegration based technique. In our approach, each transfer function is stored in a $2 \times N$ 2D texture. The first row of the texture contains a look-up table for a function $f(s)$, while the second row stores a look-up table for the function $F(s) = \int_0^s f(s) ds$, f being one of the functions $\tau, \tilde{k}, \iota, \nu$. We actually maintain a 4-components texture for $\tilde{k}(s)$, $\tau(s)$, and two 1-component ones for $\iota(s)$ and $\nu(s)$.



Fig. 4 Continuous refraction. With refraction enabled (right), ray direction is changed at each integration step according to Snell’s law, and fine details in transparent objects become visible. The view-dependent effect is evident when seen on the light field display.

The final color for each pixel is computed by performing compositing operations while following a view ray $\mathbf{r} = \mathbf{O} + t\mathbf{v}$, where the origin $\mathbf{O}$ and direction $\mathbf{v}$ are computed using Eq. 1. At each integration step i , we consider a segment of length d_i along $\mathbf{r}$, sample the scalar texture at the end of the segment to retrieve a scalar $s_i = s(t_i)$, and compute each of the attributes f_i we require for compositing by:

$$f_i = \begin{cases} \frac{F(s_i) - F(s_{i-1})}{s_i - s_{i-1}} & \text{if } \|s_i - s_{i-1}\| > \varepsilon \\ f(s_i) & \text{otherwise} \end{cases} \quad (2)$$

where s_{i-1} is the scalar value at the beginning of the segment, which is recorded from a previous step together with $F(s_{i-1})$, and $0 < \varepsilon \ll 1$, e.g., $\varepsilon = 10^{-4}$, is used to detect constant opacity areas. The overall method is numerically stable and the main assumption behind it is that the scalar data s along the small segment is well approximated by a linear interpolation between s_{i-1} and s_i . Once the average attributes for the segment have been computed, the extinction weighted color $\tilde{k}_i$ and the extinction τ_i are transformed into opacity weighted colors $\tilde{c}_i$ and opacity α_i using the relation $\alpha_i = 1 - e^{-\tau_i d_i}$. We then compute the gradient ∇s_i using central differences at $t = \frac{t_{i-1} + t_i}{2}$. As for GigaVoxels[6], we can exploit multilevel access to the hierarchy (see Sec. 5) to sample both s_i and ∇s_i at multiple levels in the octree hierarchy, and blend them using quadrilinear interpolation for continuous filtering. In addition, we employ the multiple values and gradients for multilevel shading effects, such as exaggerated shading [5] and normal unsharp masking [19].

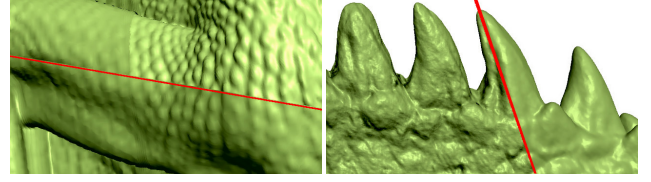


Fig. 5 Multilevel rendering. Left: using a single LOD per brick may lead to discontinuities at brick boundaries, while quadrilinear interpolation ensures continuous rendering; Right: Tiny details can be enhanced at run time by unsharp masking of the gradient field.

All the computed attributes, the current ray position and direction, and the material and light properties (ambient, diffuse, and specular factors, plus shininess exponent) are then made available to “enhancer” objects, which, much as shaders in a rasterization pipeline, can freely modify them to implement a variety of effects (see Sec. 4 for examples). All the active enhancers are chained together, each receiving the result of the prior one as input. After the enhancers have produced their final values, the actual shading and compositing steps can be performed. The shading step simply takes colors and gradients and computes a shaded color. The accumulation and compositing scheme is similar to MIDA [4], even though in our case we accumulate importance rather than intensity, leading to a *Maximum Importance Difference Accumulation* (MImDA) approach. First, we compute the current importance difference $\delta_i = \max(\iota_i - \iota_{\max_i}, 0)$, where $\iota_{\max_i}$ is the current maximum importance value along the ray. We then compute $\beta_i = 1 - \gamma\delta_i$, and perform the accumulation according to the following equation:

$$[c_i^* \alpha_i^*] = [c_{i-1}^* \alpha_{i-1}^*]\beta_i + (1 - \beta_i\alpha_{i-1}^*)[c_i \alpha_i] \quad (3)$$

where c^* and α^* denote accumulated colors and opacity, and $\gamma \in]0, 1[$ is a parameter that allows us to smoothly interpolate from plain volume rendering (with $\gamma = 0$) to full maximum importance difference accumulation (with $\gamma = 1$). When $\gamma > 0$, more important content shines through previously accumulated values. Eq. 3 only differs from standard DVR compositing by the additional weighting with β_i of previously accumulated colors and opacity. Higher prominence is assigned to local maxima of the importance, letting more important content shine through less relevant material (Fig. 6). Since Eq. 3 does not ensure monotonically increasing opacity, early ray termination criteria must be revised. We thus estimate the max future opacity delta by $\Delta\alpha_i^* = 1 - (1 - \gamma(\iota_{\max}^{(tf)} - \iota_{\max_i}))\alpha_i^*$, where $\iota_{\max}^{(tf)}$ is the max importance in the transfer function, and stop when $\Delta\alpha_i^* < \varepsilon$. This reverts to the usual accumulated opacity test for $\gamma = 0$.

Our framework permits the creation of specialized interactive illustrative techniques that provide different contextual information in different areas of the display. The underlying idea is to use the point of view relative to the display as a variable controlling some aspects of the visualizations. A simple example, shown in the accompanying video, consists

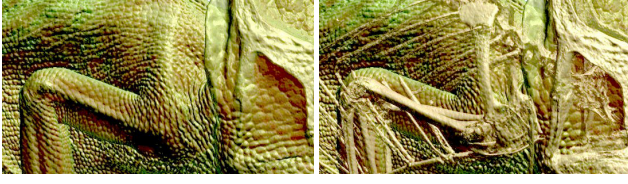


Fig. 6 Maximum importance difference accumulation. In the left image, all materials are given the same importance. In the right image, bone is given a higher importance and thus shines through previously accumulated material layers.

in adaptively showing different registered layers of information by controlling a layer’s opacity as a function of the view direction. More elaborate examples, discussed below, consist in changing the visualization of the volumetric model to reveal different information from different view angles.

Clip plane with view-dependent context. Frequently, the use of clipping planes is the only way to uncover otherwise hidden details of a volumetric data set or to visualize non-orthogonal slices of the objects. Clip planes, however, provide most of their information when viewers are facing them, but offer little insight in most other situations. In particular, when view direction is parallel to the plane, no information is provided to the viewer (besides the removal of a portion of the object). Thus, we propose a formulation that supports traditional cut away visualization, when our view direction is orthogonal to the clip plane, while offering more helpful contextual information in other situations (see Fig. 7). We compute the distance from the plane $\delta_i = \mathbf{n} \cdot \mathbf{x}_i + p$. If the distance is positive, we simply modify the opacity of samples by multiplying it by a view-dependent correction factor $\mu_i = \text{smoothstep}(1 - \max(0, -\mathbf{n} \cdot \mathbf{v}), f_l, f_h)$, where the smoothstep function is a cubic polynomial that smoothly transitions from zero to one as the view dependent factor varies between f_l and f_h . If, otherwise, $-\delta_{\text{thickness}} < \delta_i < 0$, we smoothly vary the opacity of the plane and shading parameters from the original ones at $\delta_i = -\delta_{\text{thickness}}$ to full opacity and ambient plus emission shading at $\delta_i = 0$, emphasizing the tissue sliced by the plane. The other samples on the back of the plane are left unchanged.

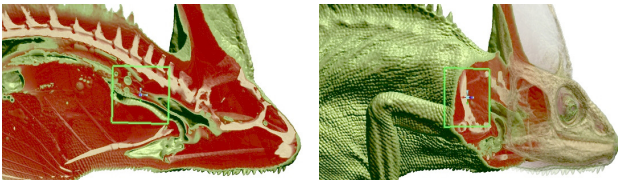


Fig. 7 Clip plane with view-dependent context. As the view becomes less and less orthogonal to the plane, more and more contextual information appears.

Context-preserving probe. Our context-preserving probe provides a means of interactively inspecting the interior of a volumetric data set in a feature-driven way which retains context information (see Fig. 8). Our spherical probe is positioned in space, and has an associated enhancer, which

modifies sample opacity and color by multiplying them with two correction factors. Both correction factors have a dependency on two weighting functions, ω_d , which is a quadratic polynomial equal to one on the plane passing through the probe center and oriented towards the viewer and zero at a distance from this plane equal to the probe radius, and ω_a , computed similarly to the view-dependent factor of our clip plane and is equal to one when the probe axis is aligned with the view direction and quickly goes to zero for other directions. The opacity correction factor $\mu_i^{(\alpha)}$ and the intensity correction factor $\mu_i^{(i)}$ are computed by:

$$\mu_i^{(\alpha)} = \begin{cases} \text{lerp}(0, \eta_i, \omega_d) & d < d_{\max} \\ \text{lerp}(\eta_i, 1, 1 - \omega_d) & d \geq d_{\max} \end{cases} \quad (4)$$

$$\mu_i^{(i)} = \begin{cases} 1 - \eta_i & d < d_{\max} \\ \text{lerp}(1 - \eta_i, 1, 1 - \omega_d) & d \geq d_{\max} \end{cases} \quad (5)$$

where $\eta_i = \omega_a \text{smoothstep}(t_i \frac{\|\nabla s_i\|}{\nabla_{\max}}, g_l, g_h)(1 - \frac{(\nabla s_i \cdot \mathbf{v})}{\|\nabla s_i\|})^4$. By scaling opacity by a function of gradient norm, we emphasize edges within contextual information. Since the gradient is a bad predictor for the surface normal orientation in nearly homogeneous regions due to the increased influence of noise, we filter its magnitude through the smoothstep function, which smoothly transitions from zero to one as the (rescaled) gradient varies between g_l and g_h . Scaling gradient norm by the importance t_i allows us to have edges within less important tissues fade out quickly. η_i is proportional to the alignment of both the tool axis and the plane defined by the local gradient with the view direction. Rather flat surfaces that are oriented towards the viewer will thus fade out in the context region. At the same time, edges will only be visible if looking straight at the tool, and clutter due to contextual information can be removed by simple head motions. The distance-based modulation forces everything to transparent when entering the probe, and gradually increases the opacity modulation factor to one on the probe back. Thus, the material within the probe becomes less and less opaque with increasing tool penetration. The effect of the color correction factor is to perform silhouette darkening in the context region, while gradually blending to the original material in the back of the probe.

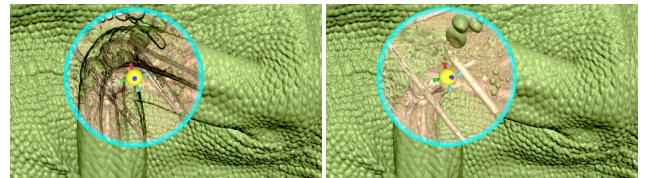


Fig. 8 Context-preserving probe. Material becomes less and less opaque with increasing tool penetration. Contextual edges are visible only if looking straight at the tool (left), and clutter due to contextual information can be removed by a simple head motion (right).

Band picker. Our band picker provides a means to enhance the importance of a currently selected band in the transfer

function. We assume, as usual, that a band is parametrized by four ordered values, which define a trapezoid function. The values delimiting the current band can be determined by accessing an existing transfer function, or by using a region growing approach similar to [11]. In addition, the area within which the modification is applied is made view- and tool-dependent by weighting the importance and opacity modification by factors proportional to the alignment between the view direction and the tool main axis (see Fig. 9). In our band picker, we set the importance of the samples within the selected band to one, and the importance of other samples to zero. At the same time, we modulate the opacity of the other bands as a function of the view angle. When looking straight at the probe, only the selected band is visible, while when looking at an angle other bands provide contextual information.

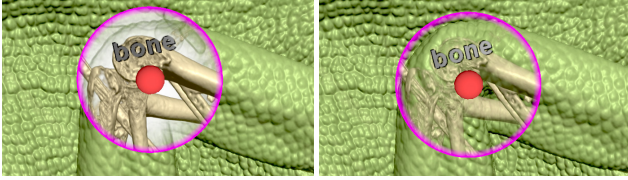


Fig. 9 View-dependent band picker. The tool selectively highlights the selected material. When looking straight at the probe (left), only the selected band is visible, while when looking at an angle (right) other bands provide contextual information using MImDA.

5 Volume rendering architecture

The mentioned illustrative rendering techniques, especially in the context of the multi-user applications naturally supported by the display, require a rendering system which is both flexible enough to support mixing and matching of several enhancing methods and performing enough to enable visualization and manipulation of extremely massive volumetric models. In our cluster-parallel implementation, we run one separate rendering back-end process per board, and connect each of them to a coordinator process running on the front-end PC using MPI. A separate rendering is performed for each projector view. Each back-end process controls a portion of the framebuffer, and is capable to overlay a standard OpenGL rendered image, e.g., for 3D GUI objects, on top of a volume rendered image rendered with CUDA. By default, no depth combination is done (and, thus, the OpenGL objects shine through volumetric objects). It is also possible, but only without refraction, to depth-combine the two images, using the depth buffer generated by OpenGL to clip rays followed in the volumetric renderer.

Each of the volume rendering processes uses an adaptive out-of-core renderer combining out-of-core data management with level-of-detail and visibility culling techniques.

As in other recent massive volume renderers [8, 6], we separate the creation and maintenance of the rendering working set from the actual rendering. In our case, adaptive working set maintenance is performed on the CPU starting from data stored on disk in octree form, while rendering is fully performed on the GPU by a single pass CUDA based stack-less raycaster traversing an adaptively maintained working set of volume bricks. The CUDA raycaster is designed for supporting plugging in of enhancer objects.

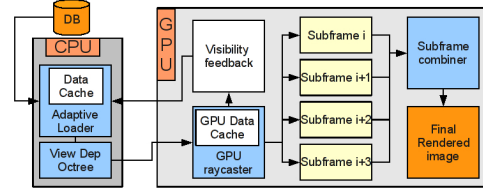


Fig. 10 Volume rendering architecture. Structure of the rendering subsystem for one single view.

Data preprocessing and compressed data representation.

Our input model is a full rectilinear scalar volume, with 8 or 16bits/voxel. At preprocessing time, the original volumetric model is decomposed into small cubical bricks of fixed size (typically 32^3) with 2 voxels overlap in every direction organized in an octree. The data contained in each brick is an 8bit value obtained by linearly mapping each voxel value to the range of values contained in the brick and quantizing the resulting scalar number. This simple lossy compression technique allows the renderer to use hardware accelerated trilinear interpolation, while increasing cache effectiveness with respect to the uncompressed version.

GPU cache and spatial index. At run-time, an adaptive loader updates a view- and transfer function-dependent working set of bricks incrementally maintained on CPU and GPU memory by asynchronously fetching data from the out-of-core octree. The adaptive refinement method is guided by suitably computed node priorities, determined using information fed back from the renderer (see below). Following Gobbetti et al. [8], our adaptive loaders stores in a 3D texture a cache of recently used volume bricks, and constructs at each frame a spatial index for the current working set in the form of an octree with neighbor pointers. Our spatial index consists in a single $6 \times N$ 2D texture with 32bit components. Texture columns index the N nodes of a linearized octree, while each row contains a different attribute. The first row contains spatial index nodes, with a 16bit index pointing to the node's parent, and a 16bit index pointing to the first of eight consecutive children (null for leaves). The second row contains the index of associated data in the current brick pool, or null for empty nodes. The third row contains the minimum and maximum values used for mapping the 8bit data values contained in the referenced bricks to the input domain. The fourth, fifth, and sixth rows contain the $(-x, +x)$, $(-y, +y)$, $(-z, +z)$ neighbor pointers. The cost of

this representation is of 192bits/node, improving over the 256bits/node of Gobbetti et al. [8], even though we store data ranges, parent pointers, and data pointers for all tree nodes, as they enable us to perform high-quality filtering. The GigaVoxel approach, which also supports multilevel access to the hierarchy, uses only 64bits/node (but without data ranges), but forces the usage of kd-restart, leading to many redundant traversal steps.

Stackless octree raycaster. The spatial index is traversed by a raycaster implemented in CUDA, with a CUDA thread associated to each ray. Traversal starts by transforming the ray to volume coordinates and by clipping it to the volume extent. Empty and non-empty bricks are then enumerated in front-to-back order, adapting sampling density to brick resolution, and stopping as soon as possible. The brick enumeration strategy improves the stackless approach of Gobbetti et al. [8] by tracking the location of a small number of coarser level bricks above the leafs (in this paper: the current node, its parent, and its grandparent). The basic concept behind the stackless traversal is to start by performing a down traversal of the octree for locating the leaf node that contains the current sampling position, which, at the start, is the position at which the ray enters the volume. During the down traversal, a small queue maintains references to parent and grandparent data. Then, the leaf node is processed by accumulating data according to the current rendering mode, stepping through the associated brick if it contains data, or performing analytical integration if it is empty. If the ray does not terminate because maximum opacity is reached, the algorithm determines the face and the intersection point through which the ray exits the node. Then traversal continues by following the neighbor pointer of this face to the adjacent node, eventually performing a down traversal to locate the leaf node that contains the exit point, which is now the entry point of the new leaf node. When moving to a neighbor, parent and grandparent data are updated only if their index changes. Pseudo-code of the method is provided in a paper addendum. We compared the performance of this approach with GigaVoxel’s kd-restart [6]. In our implementation, full volume rendering (including data communication) using the stackless method is about 10% faster in static views and 4% faster in dynamic views on a NVIDIA 8800GTS 640 (G80 GPU) due to the reduction in traversal steps.

Visibility feedback and adaptive loader. The adaptive loader maintains in-core a view- and transfer-function- dependent cut of the out-of-core octree and uses it to update the GPU cache and spatial index. The basic principle of the method is to update at each frame the visibility status of the nodes in the graph based on rendering feedback, and, during the refinement cycle, only refine nodes that were marked as visible during the previous frame and are considered inaccurate and non-empty according to the current transfer function. Under

this approach, the available GPU texture slots are used for visible nodes, and load requests will not even be posted for invisible ones (see Fig. 11).

In order to provide visibility information back to the loader, we employ a writable CUDA array storing an 8bits value per spatial index node. The array is initialized at zero before rendering each frame, and each time a non-empty leaf brick is traversed to accumulate its contribution, the traversal kernel updates the corresponding array entry by setting the value to one. Array entries with a value of zero after rendering correspond to bricks that have not been traversed, and are thus occluded or out of view. At the beginning of each frame, the visibility feedback array of the previous frame is copied from the device into the host and the visibility status of the in-core nodes is updated accordingly. Since the array only stores data for visited leaves, visibility data is pulled up to inner nodes by recursively combining visibility information of child nodes. The visibility feedback and octree refinement processes are extremely fast, given the coarse granularity of the octree of bricks. While it is true that scatter writes to global memory are costly, they occur only upon traversal of non-empty leaf bricks. The cost, amortized over full brick traversal, is negligible. We did not find a measurable difference between a kernel with the scatter write and a kernel not using it. Transferring the visibility map is also fast, since the array size is limited by the maximum size of the current view data (typically a few thousands nodes), not by the maximum dataset size. The combined cost of downloading the visibility map to the CPU and pulling up the visibility information is less than 1ms on a NVIDIA 8800GTS 640MB.

In our parallel implementation, each of the rendering processes has access to the full model stored out-of-core, but only loads the portion associated to each projector using our adaptive loader. Since a single graphics board is, in general, rendering images for more than one projector, we amortize refinement time and reduce memory needs by merging the refinement and loading step of all views. We thus maintain a single shared cache and spatial index per process, and, during the refinement phase, we define the octree cut by using the finest resolution required among the different views. We also use a single combined visibility feedback array obtained by zeroing out the visibility feedback array only once per frame rather than once per rendering kernel call.

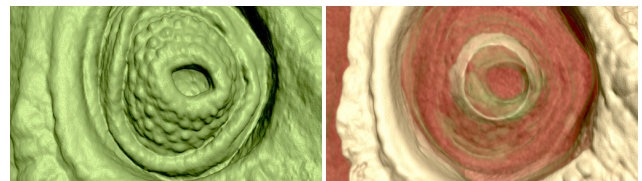


Fig. 11 Visibility culling. Images with different transfer functions, rendered on a 1024×576 window. Visibility culling reduces the working set from 4553 bricks to 1554 bricks in a almost opaque case, and from 4620 bricks to 3290 bricks when surfaces get more transparent.

Custom compositing behavior. The above rendering structure requires particular care in implementation to ensure reasonable flexibility and performance. In our design, the rendering kernel is a generic function whose template parameters define the stepping behavior within non-empty bricks and a series of up to three chained enhancing behaviors. A multiple dispatching code run at each kernel launch, transforms all dynamic types into static types before kernel invocation (pseudo-code is provided in a paper addendum). An enhancer is seen as a class providing three device functions of the current accumulated and sampled state that define the behavior at ray start, at each sampling step, and at ray termination. These device functions operate on the rendering state, and also have access to a specific enhancer’s constant and variable state. The constant state is a structure containing the current enhancer parameters, which is stored in device constant memory and shared among all threads. The enhancer variable state, instead, is a per-thread structure stored in device shared memory, and initialized by the kernel before ray traversal using a specific enhancer’s entry point. In our design, a specific kernel is generated (and optimized by the compiler) for each of the possible combinations of steppers and enhancers. A special class, with empty constant and variable data and no operations performed in device functions implements the default pass-through behavior.

Adaptive frame reconstruction. Rendering performance for large pixel count displays can be further improved by taking into account spatio-temporal coherence. In this work, we reduce the amount of pixels generated at each frame, and reconstruct full images based on samples acquired at different times. Our aim is to increase responsiveness, while not compromising the ability to render static images at full quality. Our method is purely image-based and works without making assumptions about the rendered scene and rendering process. No cooperation from the renderer is required, besides the ability of computing images. The basic idea is to use an adaptive spatio-temporal filter, which strives to provide spatially detailed imagery where the viewed scene is static, and low-latency (if blurred) imagery where it is dynamic. In order to fully exploit parallel rendering performance and to avoid destroying coherence among neighboring rays, we do not perform adaptive sampling, but group pixel batches in sub-frames aligned on a quincunx lattice. We explicitly avoided using more elaborate sparse/adaptive schemes because we want to reduce CPU/GPU communication. Moreover, it is generally better to trace many groups of very nearby rays to reduce thread divergence than selecting fewer sparse rays to trace. Screen pixels are thus assigned to N sub-frames (4 in our current implementation) made of interleaved pixels on x and y axes (see Fig. 12 left), which are sequentially rendered and buffered. At each frame presented to the viewer, only one sub-frame is re-rendered, and

frame reconstruction is performed for presenting the image. The reconstruction process starts by copying the pixels of the just rendered sub-frame to the appropriate locations in the final image. All other pixel values are then generated by a spatio-temporal filter. The basic idea is to compute a new not-rendered pixel Q'_t by combining its previous version in time, Q_{t-1} , with its four cross neighbors in space, A_t, B_t, C_t, D_t (see Fig. 12 right). We first compute a spatial approximation $Q_t^{spatial}$ by averaging the two opposite neighbors with lower color difference (i.e., A_t, C_t or B_t, D_t). Then, we estimate the temporal change ΔQ of the pixel by taking the maximum absolute color difference among $(A_t, A_{t-1}), \dots, (D_t, D_{t-1})$. The new pixel color is then computed by $Q'_t = Q_{t-1} + \text{smoothstep}(\Delta Q, \Delta_l, \Delta_h)(Q_t^{spatial} - Q_{t-1})$, where the smoothstep function smoothly transitions from zero to one as ΔQ varies between Δ_l and Δ_h . The process is then repeated for all other missing pixels, by suitably shifting and rotating the reconstruction grid when applying the same reconstruction process. With such an approach, where the image is dynamic, more recent samples are emphasized, resulting in up-to-date, but possibly slightly blurred images. Where the image is more static, older samples dominate reconstruction, leading to sharp images. In static image areas, the method is guaranteed to converge to the same image that would have been computed by performing a full-resolution ray-casting. We have purposely decided not to include motion compensation or reprojection steps, since we prefer increased blur in motion areas over artifacts when dealing with semi-transparent objects.

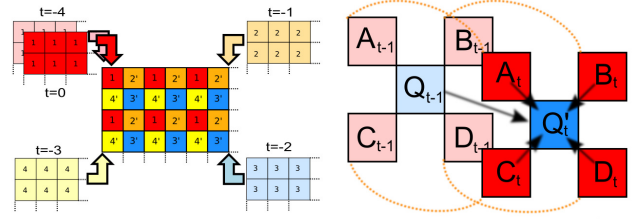


Fig. 12 Frame reconstruction. Left: sub-frame pixels are arranged according to a quincunx pattern. The last sub-frame, as well as the four previously rendered ones, contribute to the full image. Right: a spatio-temporal filter reconstructs a full image.

The frame reconstructor is fully implemented in a single CUDA kernel. Each thread reconstructs a quad of 2×2 pixels using a three phase process. First, threads within the same thread block cooperate to load into shared memory the tile of present and past pixel values required for reconstruction, and copy to the output buffer the single up-to-date pixel associated to them. Then, each thread reconstructs the diagonal neighbor of that pixel. Finally, the horizontal and vertical neighbors are reconstructed. A *syncthreads()* barrier at the end of each phase ensures that the four neighbors of the next reconstructed pixels are up-to-date.

6 Implementation and results

An experimental system has been implemented on Linux using OpenGL and NVIDIA CUDA 2.1. The out-of-core octree structure is implemented on top of Berkeley DB, with LZO compression applied to data nodes. OpenMPI 1.2.6 is used for interprocess communication in the rendering cluster. When running on a desktop configuration, the rendering kernel is embedded directly in a single-process application, using a separate thread for asynchronous data loading.

We have tested our system with a variety of high resolution models and settings. In this paper, we discuss the results obtained with the inspection of two 16bit/sample micro-CT datasets: a Veiled Chameleon ($1024 \times 1024 \times 1080$) and a Pichi Armadillo ($1024 \times 1024 \times 999$) specimens². In order to test the system with extremely large and demanding datasets, the original data has been artificially zoomed by $4\times$ in a preprocessing step (i.e., to $\approx 4K^3$ voxels) by tricubic interpolation plus procedurally added detail. The construction of the octree from source data was performed using octree bricks of 32^3 , instructing the preprocessor to use the compressed 8bit format and to filter out data with a value lower than 6400, in order to discard most of the noisy empty space. For the chameleon, data preprocessing took 12.3hours on a Linux PC 2.4GHz CPU and produced a 15GB octree database starting from an uncompressed source size of 135GB. The RMS error was 14.6, and the AMAX error was 116. The armadillo dataset produced similar results (13.8GB octree database, $RMS = 15.8$, $AMAX = 122$).

Standard desktop rendering. We evaluated single PC rendering performance on a number of interactive inspection sequences using the 4K models and measuring actual frame rates (i.e., not only raw rendering times, but frame-to-frame times). We report here on the results obtained when rendering on a PC with NVIDIA GTX 280 using a window size of 1024×576 pixels and 1voxel/pixel accuracy to drive the adaptive renderer. When using the moderately opaque transfer function of Fig. 11 left, the frame rate of typical inspection sequences of the chameleon dataset varies between 23Hz for extreme close-up views to over 60Hz for overall views. Making the model highly transparent, as in Fig. 4, reduces the performance to 11Hz – 39Hz. Adding refraction leads to rendering rates of 7Hz – 32Hz. Using interactive tools, which activate enhancers in the CUDA raycaster, decreases the performance by about 10%. Interactive rates are thus guaranteed even in the most demanding situations. Part of the performance is due to the frame reconstruction approach, which guarantees a speed-up of $2.15 \times - 3.3 \times$ depending on the situation. The speed-up is sub-linear, since



Fig. 13 Multi-user interactive exploration of 64Gvoxel datasets on a 35Mpixel light field display. Users freely mix and match 3D tools creating view-dependent illustrative visualizations. Objects appear floating in the display workspace, providing correct parallax cues while delivering direction-dependent information.

we reduce the number of traced pixels by 4. The additional overhead is mostly due to the reduced coherence of rays traced from a sparser pixel grid, leading to increased divergence among threads traversing the spatial index structures.

Large scale light field display. Our 3D display is capable of visualizing 35Mpixels by composing images generated by 72 SVGA LED commodity projectors illuminating a 160×90 cm holographic screen. The display provides continuous horizontal parallax within a 50° horizontal field-of-view with 0.8° angular accuracy. The pixel size on the screen surface is 1.5mm. The rendering back-end is currently running on an array of 18 Athlon64 3300+ Linux PCs equipped with two NVIDIA 8800GTS 640MB (G80 GPU) graphics boards running in twin-view mode. Each back-end PC has thus to generate $4 \times 800 \times 600$ pixels using two OpenGL graphics boards with CUDA capability 1.0 and based on an old G80 chip. It is obviously impossible to fully convey the impression provided by our system on paper or video. As a simple illustration of our system's current status and capabilities, we analyze the behavior of the multi-user application that allows users to interactively position, scale, and explore volumetric models using Intersense IS900 trackers. We recorded the performance using a hand-held video camera freely moving in the display workspace. Representative video frames are shown in Fig. 1 and Fig.13. Note the parallax effects, the view-dependent illustrative visualizations, and the good registration between displayed object space and physical space, which demonstrate the multi-user capability of the display. Thanks to stereo and motion parallax, MImDA images proved easy to understand on the 3D display even though they provide mixed depth cues. The various view-dependent illustrative tools proved intuitive to use and able to reveal important volumetric details. Combining view dependent rendering mechanisms with stereo projections inherently leads to the issue of presenting two images to the viewer that differ by more than just the view point. However, we did not notice problems related to this fact. This is because our visualizations have a strong correlation between views, and subtle differences are extremely well tolerated by the human visual system. Even though the rendering hardware is definitely not high end, the application remained interactive for large models. The average

² Source: Digital Morphology Project, the CTLab and the Texas Advanced Computing Center, University of Texas, Austin

frame rate during the recorded interaction sequences was of 9.8fps. During the entire inspection sequences, the resident set size of each rendering process is maintained well within the 600 MB/board of pre-allocated cache size by the out-of-core data management system. The average octree size in the render nodes was of 1549 bricks/process, while the most loaded process handled 2087 bricks. Thus, introducing a load balancing strategy could (slightly) improve the system's performance by $\approx 30\%$. The good load balancing achieved by the static image distribution is caused by the geometry of the display, with all projectors typically looking at the same portion of the displayed object. The visibility feedback mechanism has proved to be able to reduce working set size, especially when using transfer functions with moderate to high opacity. It should be noted that the graphics boards used in our cluster are two generations older than the one used for desktop testing, leading to a noticeable slowdown, that we have quantified to $2.3 \times - 3.2 \times$, the higher figures being for the most demanding situations. Please refer to the accompanying video for further results.

7 Conclusions

We have described a flexible multiresolution volume rendering system capable to interactively drive large scale multi-projector light field displays which can provide stereo and motion parallax cues in a setting supporting collaborative use. The system improves over recent work by proposing specialized interactive illustration techniques based on a flexible compositing engine. The illustrative techniques are designed to exploit the view-dependent characteristics of the display to provide different contextual information in different viewing areas.

Acknowledgments. This work is partially supported by the EU Marie Curie Program under the 3DANATOMICALHUMAN project (MRTN-CT-2006-035763).

References

1. Agus, M., Bettio, F., Giachetti, A., Gobbetti, E., Iglesias Guitián, J., Marton, F., Nilsson, J., Pintore, G.: An interactive 3d medical visualization system based on a light field display. *Visual Computer* **25**(9), 883–893 (2009)
2. Agus, M., Gobbetti, E., Guitián, J.A.I., Marton, F., Pintore, G.: GPU accelerated direct volume rendering on an interactive light field display. *Computer Graphics Forum* **27**(2), 231–240 (2008)
3. Bruckner, S., Grimm, S., Kanitsar, A., Gröller, M.E.: Illustrative context-preserving exploration of volume data. *IEEE Trans. Vis. Comput. Graph* **12**(6), 1559–1569 (2006)
4. Bruckner, S., Gröller, M.E.: Instant volume visualization using maximum intensity difference accumulation. *Computer Graphics Forum* **28**(3) (2009)
5. Cignoni, P., Scopigno, R., Tarini, M.: A simple normal enhancement technique for interactive non-photorealistic renderings. *Computer & Graphics* **29**(1), 125–133 (2005)
6. Crassin, C., Neyret, F., Lefebvre, S., Eisemann, E.: Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In: *Proc. I3D*, pp. 15–22 (2009)
7. Dayal, A., Woolley, C., Watson, B., Luebke, D.: Adaptive frameless rendering. In: *Rendering Techniques*, pp. 265–276 (2005)
8. Gobbetti, E., Marton, F., Guitián, J.A.I.: A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *Visual Computer* **24**(7–9), 797–806 (2008)
9. Havran, V., Bittner, J., Sára, J.: Ray tracing with rope trees. In: *Proc. Spring Conf. Comput. Graph.*, pp. 130–140 (1998)
10. Herzog, R., Eisemann, E., Myszkowski, K., Seidel, H.P.: Spatio-temporal upsampling on the GPU. In: *Proc. I3D* (2010)
11. Huang, R., Ma, K.L.: RGVis: region growing based techniques for volume visualization. In: *Proc. Pacific Conf. Comput. Graph. and Applic.*, pp. 355–363 (2003)
12. Jones, A., McDowall, I., Yamada, H., Bolas, M.T., Debevec, P.E.: Rendering for an interactive 360 degree light field display. *ACM Trans. Graph* **26**(3), 40 (2007)
13. Kraus, M.: Pre-integrated volume rendering for multi-dimensional transfer functions. In: *IEEE/EG Symposium on Volume and Point-based Graphics*, pp. 97–104 (2008)
14. Ljung, P.: Adaptive sampling in single pass, GPU-based raycasting of multiresolution volumes. In: *Proc. Volume Graphics*, pp. 39–46 (2006)
15. Luo, Y., Guitián, J.A.I., Gobbetti, E., Marton, F.: Context preserving focal probes for exploration of volumetric medical datasets. In: *Second 3D Physiological Human Workshop* (2009)
16. Niski, K., Cohen, J.D.: Tile-based level of detail for the parallel age. *IEEE Trans. Vis. Comput. Graph.* **13**, 1352–1359 (2007)
17. Popov, S., Günther, J., Seidel, H.P., Slusallek, P.: Stackless kd-tree traversal for high performance GPU ray tracing. *Computer Graphics Forum* **26**(3), 415–424 (2007)
18. Ropinski, T., Rezk-Salama, C., Hadwiger, M., Ljung, P.: Gpu volume raycasting with advanced illumination. In: *Eurographics Tutorial 4* (2009)
19. Rusinkiewicz, S., Burns, M., DeCarlo, D.: Exaggerated shading for depicting shape and detail. *ACM Trans. Graph.* **25**(3), 1199–1205 (2006)
20. Segovia, B., de Iehl, J.C., Mitanchey, R., Péroche, B.: Non-interleaved deferred shading of interleaved sample patterns. In: *Graphics Hardware*, pp. 53–60 (2006)
21. Viola, I., Kanitsar, A., Grollier, M.E.: Importance-driven feature enhancement in volume visualization. *IEEE Trans. Vis. Comput. Graph* **11**(4), 408–418 (2005)
22. Vollrath, J.E., Schafhitzel, T., Ertl, T.: Employing complex GPU data structures for the interactive visualization of adaptive mesh refinement data. In: *Proc. Volume Graphics*, pp. 55–58 (2006)
23. Yang, L., Sander, P.V., Lawrence, J.: Geometry-aware framebuffer level of detail. *Computer Graphics Forum* **27**(4), 1183–1188 (2008)



José Antonio Iglesias Guitián is a Marie Curie Early Stage Researcher in the Visual Computing group of the Center for Advanced Studies, Research and Development in Sardinia (CRS4). He obtained his Masters degree in Computer Science in 2006 at the University of A Coruna, Spain.



Enrico Gobbetti is the director of the Advanced Computing and Communications Program and of the Visual Computing group at the Center for Advanced Studies, Research, and Development in Sardinia (CRS4). His research spans many areas of computer graphics and is widely published in major journals and conferences. Enrico holds an Engineering degree (1989) and a Ph.D. degree (1993) in Computer Science from the Swiss Federal Institute of Technology in Lausanne (EPFL).



Fabio Marton is a researcher in the Visual Computing group at the Center for Advanced Studies, Research, and Development in Sardinia (CRS4). He holds a Laurea (M.Sc.) degree (1999) in Computer Engineering from the University of Padua, Italy. His current research interests include out-of-core data processing, multiresolution modeling and time-critical rendering.