IEEE/EG International Symposium on Volume Graphics (2010)
R. Westermann and G. Kindlmann (Editors)

Split-Voxel: A Simple Discontinuity-Preserving Voxel Representation for Volume Rendering

Marco Agus, Enrico Gobbetti, José Antonio Iglesias Guitián, and Fabio Marton

Visual Computing Group, CRS4, Pula, Italy – http://www.crs4.it/vic/

Abstract

The most common representation of volumetric models is a regular grid of cubical voxels with one value each, from which a smooth scalar field is reconstructed. However, common real-world situations include cases in which volumes represent physical objects with well defined boundaries separating different materials, giving rise to models with quasi-impulsive gradient fields. In our split-voxel representation, we replace blocks of N^3 voxels by one single voxel that is split by a feature plane into two regions with constant values. This representation has little overhead over storing precomputed gradients, and has the advantage that feature planes provide minimal geometric information about the underlying volume regions that can be effectively exploited for volume rendering. We show how to convert a standard mono-resolution representation into a out-of-core multiresolution structure, both for labeled and continuous scalar volumes. We also show how to interactively explore the models using a multiresolution GPU ray-casting framework. The technique supports real-time transfer function manipulation and proves particularly useful for fast multiresolution rendering, since accurate silhouettes are preserved even at very coarse levels of detail.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture and Image Generation—; Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—.

1. Introduction

The most common representation of volumetric models is a regular grid of cubical voxels with one value each, from which a smooth scalar field is reconstructed. This regularity enables efficient rendering, but is not well adapted to all data distributions. Common real-world situations include cases in which volumes represent physical objects with well defined boundaries separating different regions, giving rise to models with quasi-impulsive gradient fields. Boundary regions are prominently visible in volume rendered images, and the sampling nature of the voxelized representation can lead to aliasing, since voxel size restricts size and location of rendered details.

Contribution. In order to overcome these limitations, we introduce a volumetric primitive, that we call *split-voxel*, which replaces blocks of N^3 voxels by one single voxel that is split by a feature plane into two regions with constant values. The feature plane provides a linear approximation to the strongest value discontinuity in the block, while the two values represent medians or averages which are not blurred over the discontinuity. This description is exploited in a multiresolution GPU ray-casting framework able to handle large out-of-core datasets, and is used to represent both leaves and inner levels. Specifically, the main contributions of this paper are the following:

- the split voxel volumetric primitive that encodes scalar data together with edge detection information;
- a hierarchical approach for converting a standard mono-resolution voxelized representation into an out-of-core multiresolution structure based on split voxels, both for labeled and continuous scalar volumes;
- a simple and efficient ray-casting accumulation scheme exploiting split-voxels incorporated within a GPU accelerated out-of-core renderer providing real-time exploration with dynamic transfer function editing.

Advantages. This representation has little overhead over storing precomputed gradients, and has the advantage that feature planes provide minimal geometric information about the underlying volume regions that can be effectively exploited for volume rendering. In particular, split voxels are able to track material interfaces, as they occur in many physical objects, and, when employed in a multiresolution representation, provide accurate silhouettes even at very coarse levels of detail, reducing the data and time required to render understandable images (see figure 1). We show that the split-voxel primitive can efficiently model volume datasets containing intensity scalar values, as well as material labels. Since the method is applied to scalar values, the renderer is able to change the transfer function in real-time without the need to reprocess the data.

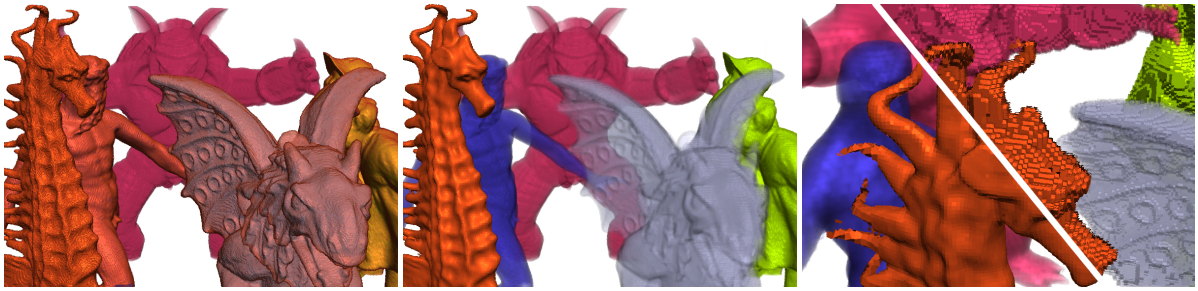


Figure 1: ***Rendering of labeled scene.*** *Left: The usual linear interpolation method suffers from color bleeding, since interpolated values generate false colors when accessing transfer functions even when using pre-integration. Middle: using split-voxels, all segmented models keep their color. Right: a close-up of the scene with low pixel tolerance shows how the split-voxel method preserves boundaries even at very low levels of details, instead the nearest method suffers from low quality edge representation.*

Limitations. The split-voxel model is tuned for scalar fields containing sharp boundaries between different near uniform zones. The representation is non-continuous, and volumes exhibiting smoothly varying gradient fields, e.g., the results of gas/fluids/fire simulations, are not managed as efficiently.

2. Related work

Reconstructing the original data, as accurately as possible from a discretized representation, while maintaining a reasonable execution speed and limited memory needs, is a fundamental problem in volume graphics. The most common representation of volumes in (GPU) volume rendering applications is a regular grid of scalar values. The most direct way to reconstruct data from the sampled representation is to use a polynomial filter. Constant and linear filters, directly supported by the hardware, are by far the most commonly used representations, but have obvious limits in their reconstruction power. In case of rapidly varying data, too many samples are required to avoid aliasing problems, with problems in terms of memory occupancy and traversal efficiency. Sparser representations can in principle be obtained with higher order filters. Even though these filters (typically the tricubic one) can be smartly implemented on top of linear interpolation to reduce their computational cost [SH04], their run-time evaluation remains costly for an interactive renderer due to the many texture fetches. Ringing artifacts may also appear, and sharp boundaries are not directly represented. Moreover, with these continuous representations, sampling has still to be performed at a high frequency to avoid missing sharp value changes. Various approaches for reducing sampling rates are known in the literature. Pre-integrated volume rendering [EKE01, RGW*03, LWM04, KSS08] deals with high frequencies in transfer functions rather than in the scalar volume itself. Adaptive sampling methods solve the boundary detection problem in real-time by adjusting sampling frequency during ray casting. For instance, Hadwiger et al. [HSS*05] combined rasterization of min-max blocks with adaptive sampling and a secant method solver to ray cast discrete iso-surfaces on the GPU, while Knoll et al. [KHW*09] employ a peak finding strategy which explicitly solves for iso-values within the volume rendering integral. The underlying sampled function is still considered continuous, and due to the cost of repeated evaluation, is at most a low order polynomial. A number of systems deal with segmented datasets (e.g., [HBH03, RGW*03]), but the focus is on blending separate gridded representations using different pre-determined transfer functions. Our method, instead, encodes minimal boundary information in the split-voxel primitive. This way, the primitive can be used for constructing multiresolution structures while preserving boundary details even for low resolution scales, and the model can easily be employed in all kinds of accumulation strategies commonly considered for volume ray casting. The underlying concept of our split voxel has been explored in different contexts before. In two dimensions, geometrical wavelets, which explicitly encode discontinuities, were introduced to properly catch edges often present in images [Don99, WN03], and they have been applied to approximation and compression problems. Representations that enrich images with codes for local boundaries have also been introduced with the main purpose of combining vector and raster image features [Sen04, TC04, RBW04]. Modified 2D interpolation rules requiring multiple fetches per texel are used to support up to four linear discontinuities in every pixel. The Pinchmap [TC05] improves over these methods by proposing a GPU-optimized 2D texture representation which separates the encoding of discontinuities from the encoding of the signal. The two-colored pixel approach of Pavic and Kobbelt [PK10], developed in parallel to our work, also uses a representation using two colors per pixel separated by a feature line, and extends it to 3D for a video retargeting applications. Other methods are instead targetted to extract feature preserving surface representations from volume data [KBSS01, JLSW02]. While these approaches conceptually share similarity to ours, they cannot applied to volume rendering. In the context of volume rendering, Sereda et al. [SBSG06] introduced the LH space as a transfer function domain. It starts from the assumption that every voxel lies either inside a material or on the boundary between two materials with lower intensity FL and higher intensity FH, respectively. They show that the LH histogram conveys information about dataset boundaries in a more compact and robust way than common intensity-gradient histograms and therefore seems to be well-suited for volume exploration. Praßni at al. [PRH09] improved this approach by deriving an effi-

cient technique for the computation of LH values, which is fast enough to support post-classification at interactive frame rates. The underlying assumption is similar to our split-voxel concept, but while these methods deal with the generation of efficient transfer functions, we instead focus on modeling and rendering volume data. With respect to the construction of the split-voxel model, our approach deals with the approximation of a small cubic scalar volume with two materials and a separation surface. To this end, various statistical methods can be employed to perform this classification, such as, for example, the well-known Support Vector Machines [Vap95] or the enumeration strategy of Pavic and Kobbelt [PK10]. The split-voxel construction deals also with the 3D edge-detection problem, which has been investigated in depth in the literature. Our technique shares similarity with the moment-based operator [LHDC93]. Finally, the split-voxel model is incorporated into a multiresolution CUDA rendering framework, whose data structures and rendering strategy are similar to those described by Iglesias Guitián et al. [IGM10].

3. Split-voxel model

We consider a generic scalar regular volume, i.e., a discrete representation over a cartesian grid of a continuous scalar function $s(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^3$. In this representation, the scalar value associated to each voxel is derived from some sort of measurement apparatus or simulation or digital processing, and it takes the form of an intensity measured value, in the case, for example, of data directly coming from medical scanning devices, or the form of a material label, when some classification has been performed over data. In order to enhance the appearance of surface structures in the volume, it is common to exploit the gradient ∇s to evaluate a surface shading model. High quality volume renderers, thus often precompute gradient values at each voxel position. Precomputing the gradients, rather than evaluating them at run-time, allows the renderer to use high quality filters with kernels larger than those which can be afforded for on-the-fly computations. Moreover, since the gradient is a bad predictor for the surface normal orientation in nearly homogeneous regions due to the increased influence of noise, gradients with low magnitude are typically filtered out. For instance, Bruckner el al. [BG09] filter out all gradients whose norm is less than $\frac{1}{8}$ of the maximum gradient magnitude in the dataset. In order to better handle situations in which the gradient changes abruptly, rather than simply precomputing and prefiltering gradients, we incorporate in our split voxel representation a minimal approximation of the locations of the strongest value discontinuity within the split voxel region. Specifically, a **split-voxel** is defined as a volumetric primitive representing the interface between two intensity values (or label values in the case of segmented volume data). The following information is encoded inside a split-voxel (a 2D schematic representation is showed in Fig. 2): two scalar values F and B, and the equation of an oriented feature plane $\Sigma = (\alpha, \beta, \gamma, \delta)$ indicating the boundary between F and B. The same representation is also valid for voxels containing uniform materials, that are encoded by imposing that front material is equal to back material and by zero-ing the separation plane. The representation has the following properties:

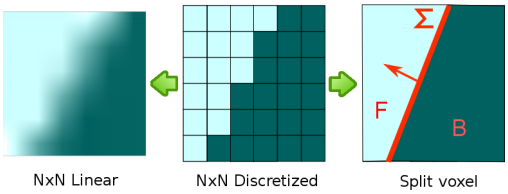


Figure 2: ***Split-voxel primitive.*** *A split-voxel replaces a block of N^3 voxels by one single voxel that is split by a feature plane into two regions with constant values. The feature plane provides a linear approximation to the strongest value discontinuity in the block, while the two values represent medians or averages which are not blurred over the discontinuity.*

- the feature plane Σ provides a linear approximation to the strongest value discontinuity in the block. It replaces the gradient as an indication of the local surface normal, and the availability of a plane constant provides additional information on the location of the discontinuity;
- the values F, B represent medians or averages which are not blurred over the discontinuity. Hence, they can be used for sharp value evaluation and exploited for implementing multiresolution filtering schemes. In particular, a sharp value for a point $P(x,y,z)$ inside the block can be obtained by computing a signed distance $d = \alpha x + \beta y + \gamma z + \delta$, and selecting F if $d > 0$, and B otherwise;
- the storage overhead with respect to a typical precomputed gradient representation is limited to two values: an additional scalar and the plane constant; moreover, the additional information permits to render at coarser levels of detail, leading to reduced memory needs and frame rendering times. Each split voxel can thus replace blocks of N^3 voxels.

The representation thus trades continuity of the reconstruction with the ability to rapidly track value changes at negligible cost during rendering algorithms. Boundary sharpness with split voxels is infinite, and will not depend on the density of sample points as it might with plain voxels. It is clear that only a single discontinuity can appear in a single split-voxel, but this limitation is unavoidable for fixed-size representations. Limiting the representation to be linear makes it fast to compute and efficient for rendering.

4. Construction

The split-voxel construction replaces gradient precomputation and data filtering in the volume preprocessing pipeline of our volume rendering framework. The goal is to convert a standard mono-resolution voxelized representation into a out-of-core multiresolution structure based on split voxels, both for labeled and continuous scalar volumes. The process is hierarchical and constructs an octree of volume bricks, in which each brick element is a split voxel. We first detail how a single split voxel is constructed from a gridded representation, and

then we explain how the procedure is hierarchically applied to complete the multiresolution hierarchy.

4.1. Constructing a split voxel

Since a split-voxel represents a generic small cubical volume portion, it can be derived from a standard gridded representation by considering a voxel block of N^3 with resolutions starting from $N = 2$. In order to increase continuity among adjacent split-voxels, a boundary overlap of M voxel layers can be also considered during construction (typically $M = 1$ or 2), thus resulting in grids containing $(N+2M)^3$ voxels (see figure 3). The split-voxel construction procedure specifically consists of identifying two main values (in the case of intensity based volumes) or labels (in the case of segmented volumes), and the plane which better separates the two material clusters.

Discrete data. In the case of segmented data, material labels are intrinsically defined inside the volume grid, and no class identification is needed. Hence, value classification consists of identifying the one or two most frequent labels inside the block. When more than two materials are present inside the volume, region-growing approaches could be also considered, but here we decided to limit the definition of split-voxel to the two most important materials in terms of frequency. Once classes are identified, if the volume can be represented by a single value, the associated split-voxel is assigned the same front and back materials and a null separation plane, otherwise the separation plane is rapidly estimated by considering the geometric cluster barycenters $\mathbf{p_f}$ and $\mathbf{p_b}$ of the two identified materials, and the two material frequencies n_f and n_b. Specifically, the split surface is defined as the plane with normal $\mathbf{n}$ and passing through $\mathbf{p}$, where

$$\mathbf{n} = \frac{\mathbf{p_f} - \mathbf{p_b}}{\|\mathbf{p_f} - \mathbf{p_b}\|} \quad (1)$$

$$\mathbf{p} = \mathbf{p_b} + (\mathbf{p_f} - \mathbf{p_b})\frac{n_b}{n_f + n_b} \quad (2)$$

as depicted in figure 3. If the distance between $\mathbf{p_f}$ and $\mathbf{p_b}$ is too small (a fraction of the voxel size), it is considered that the region is not easily linearly separable and we assign a uniform split-voxel with the most frequent label. This happens, for example, in the case of noisy regions, of small convex clusters almost entirely wrapped by another material. In most situations, however, a good linear approximation of the separating surface exists. While more elaborate solutions are possible, we found that this method provides a good approximation of the orientation and location of the strongest discontinuity (see Fig. 4 left). **Sampled data.** When volume data are intensity scalars, we choose the approach to locally convert the data to a segmented representation before applying the same procedure used for the discrete case. To this end, we employ the classical iterative k-means algorithm [Llo03] to cluster the intensity values. Specifically, two initial means are defined as the minimum and the maximum intensity values available in the grid. The iterative procedure creates the two clusters by associating every intensity value in the grid to the nearest mean, and

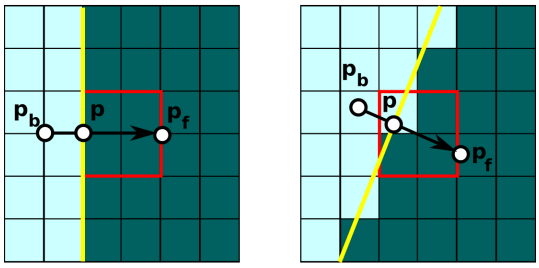


Figure 3: ***Split-voxel construction scheme for segmented data.*** *Two examples of split-voxel construction. Split-voxel width N is* 2 *and boundary overlap M is also* 2*. Red part is the split-voxel, while the separation plane is depicted in yellow, and is computed according to equations 1, 2*

it updates the means by computing the centroid of the clusters. Clustering stops when the the assignments to clusters no longer change or after a fixed large number of iterations. Generally, after few iterations the algorithm converges with two resulting intensity values representing the two materials. At this point, if these intensity values are similar (their difference is less than a given threshold), we assume that a uniform material is contained inside the volume, otherwise we classify the grid voxels with respect to the two final means and we refer back to the same procedure employed for label volumes, by computing the separation plane according to equations 1 and 2. The front and back values are then defined by each cluster median values. We verified that the planes found by this simple method provide in practice good local approximations of the surface (see Fig. 4 right).

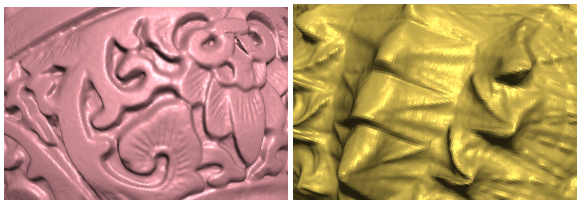


Figure 4: ***Close-up views.*** *The left image shows a segmented dataset (*$1911 \times 1908 \times 1813$ *resolution), while the right one shows a post-classified scalar dataset (*$512 \times 512 \times 400$ *resolution). Note the nice normals reconstructed by the simple split-voxel construction technique and the good matching between neighboring planes.*

Encoding. In the case of typical 8-bit datasets, each split-voxel is stored in a 6-byte structure: the plane is represented in 4 bytes, while front and back values use a byte each one. For the sake of simplicity, and to maintain regularity in the structure and avoid too many indirections, we store uniform voxels using the same representation. The split-voxel dataset size has thus in the worst case at most 50% overhead over the typical scalar dataset stored with precomputed gradients at the same resolution (6 bytes in place of 4 bytes per element). However, since split-voxels are normally used to represent N^3-voxel resolution volume grids, the actual occupancy ratio is roughly $3/(2N^3)$. For example, in the case of split-voxels representing just 2-voxel width grids, the occupancy ratio is $3/16$, i.e. 18.75% with respect to the original dataset size.

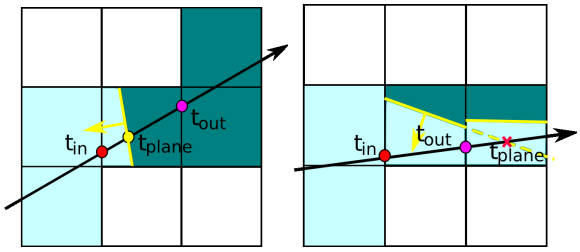
4.2. Multiresolution hierarchy construction

Our multiresolution structure is a coarse-grained octree whose nodes are bricks of split-voxels, which are built with a bottom-up procedure. Each brick is a cubical block with an edge width of some tens of split-voxels [GMG08]. Leaf bricks are computed by sampling the input volume data and by encoding each brick component into a split-voxel representation using the technique of Sec. 4.1. In order to construct non-leaf octree bricks, instead, we first resample into a gridded representation the split voxels of the next finer level which fall within the area of interest for construction. In this resampling procedure, each voxel receives the most dominant among the front and the back value of the associated split voxel (estimated by sampling the voxel). Once this procedure is terminated, the volume subsumed by the brick (plus M layers of overlap), is represented as a regular voxel grid. The procedure detailed in Sec. 4.1 is then applied to convert it to the split-voxel representation. The procedure is hierarchically repeated until we reach the octree root. It should be noted that the effect of the procedure is dual. First of all, each discontinuous split-voxel ends up having an associated feature plane. Moreover, values are low pass filtered throughout the hierarchy while not blurring values over discontinuities. The procedure thus applies edge-preserving filtering for level of detail construction.

5. Rendering

We have integrated the split-voxel primitive in a GPU accelerated out-of-core multiresolution renderer based on single pass octree ray-casting [GMG08, CNLE09, IGM10].

Hardware accelerated out-of-core multiresolution rendering. CPU and GPU strictly cooperate to manage octree traversal and ray-casting. Specifically, a CPU thread, given the current viewing parameters, has the responsibility of identifying an octree cut, by refining bricks whose projected voxel size is higher than a user-selected threshold, while updating the associated LRU cache of bricks in the GPU by loading data from out-of-core storage. The GPU cache is updated incrementally frame by frame, and each brick can be reused over several frames, exploiting temporal and spatial coherence, thus limiting the required frame bandwidth. Moreover, the CPU thread builds at each frame an octree spatial index for the current cut, which allows the GPU to perform the traversal of the current uploaded octree cut. The GPU performs ray-casting, using the spatial index to enumerate the current cut leaf nodes traversed by the ray in front-to-back order. If the visited node is empty, it is simply skipped. If the node is non-empty, the associated brick is selected in the texture cache, and all the relevant split-voxels are traversed,while accumulating color and opacity contributions according to the optical model employed. Octree traversal terminates when the ray leaves the volume bounding box, or when full opacity is reached. A variety of accumulation schemes can be employed for split voxels. In this work, we describe the classic DVR scheme using a 1D transfer function to map values to colors and opacities. Since our scheme exploits variable step sizes, the renderer works with



(a) Plane intersection inside voxel (b) Plane Intersection outside voxel

Figure 5: ***Ray DDA traversal.*** *Scheme of voxel traversal using the DDA algorithm to enumerate intersected voxels. In figure 5(a) the ray intersects the plane of the central voxel inside $[t_{in}, t_{out}]$. The front material is accumulated along $[t_{in}, t_{plane}]$, the back material along $[t_{plane}, t_{out}]$, and shading is computed for both by considering the plane normal. In figure 5(b) the ray-plane intersection lies outside the voxel: $t_{plane} > t_{out}$ and only front material is accumulated.*

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. We actually maintain a single 4-components texture for $\tilde{k}(s), \tau(s)$. The opacity of a segment of length δt is derived from an extinction coefficient τ_i by $\alpha_i = 1 - e^{-\tau_i \cdot \delta t}$.

Non-empty brick traversal and split-voxel accumulation. Color and opacity accumulation is performed during non-empty brick traversal. We exploit a 3D digital differential analyzer (3DDAA) scheme [FI85] to traverse all intersected split voxels in front-to-back order. For a given ray $\mathbf{r} = \mathbf{o} + \mathbf{d}t$, after an initialization step performed upon brick entry, all traversed voxels are enumerated, and at each step the intersection abscissae t_{in} and t_{out} between the ray and the current voxel are updated, and the length $\delta t = t_{out} - t_{in}$ of the current ray segment is computed. If the current split-voxel is uniform ($F = B$), we simply accumulate its unshaded contribution weighted with the ambient component. In order to do so, we fetch $\tilde{k}(F) = \tau(F)c(F)$, and convert them to opacity weighted colors associated to the length δt. When the split-voxel contains a feature plane, a shaded contribution is instead applied for the front and back values. We thus fetch both $\tilde{k}(F) = \tau(F)c(F)$ and $\tilde{k}(B) = \tau(B)c(B)$ and separately use them for color accumulation. First, we order materials with respect to the relative orientation of the plane normal $\mathbf{n}$ and of the viewing ray direction $\mathbf{d}$. When $\mathbf{n} \cdot \mathbf{d} \leq 0$, the front material is traversed first, otherwise the back material is traversed first. Second, we intersect the ray with the voxel plane to derive the t_{plane} abscissa value, and we accumulate the first material from t_{in} to t_{plane}, and the second one from t_{plane} to t_{out}, shading both the contributions by using the plane normal in the lighting equation, as indicated in figure 5(a). When t_{plane} lies outside the voxel, only one of the two materials is accumulated: the second material, if $t_{plane} < t_{in}$, the first one otherwise, as shown in 5(b). The rendering scheme thus trades continuity of the reconstruction with the ability to rapidly track value changes at negligible cost during rendering algorithms. Unrelated values are not combined together,

and boundary sharpness is infinite and does not depend on resolution. In many cases, this provides an improvement with respect to standard volume rendering techniques, which suffer when adjacent voxels contain very different intensity values, since interpolation tends to blur abrupt variations and to generate material values that are not physically present in original data in that position, often producing a color bleeding effect, as shown in figures 1 and 6. Finally, another advantage of our rendering scheme is that split-voxels can be projected to large screen areas, also for the coarsest level of details, since they are able to maintain a nice silhouette by shaping the boundaries with the voxel planes, as indicated in figure 6.

6. Results

A prototype software system implementing the presented techniques has been developed on a Linux system using C++ and CUDA. The out-of-core octree structure has been implemented on top of Berkeley DB, exploiting the LZO compression library to reduce memory occupancy of each split-voxel brick. We have tested our system with a variety of high resolution models and settings. In this paper, we discuss the results obtained with the processing and inspection of three 8-bit datasets: a $512 \times 512 \times 400$ micro-CT of a Mata Mata turtle specimen (Source: Digital Morphology Project, the CT-Lab and the Texas Advanced Computing Center, University of Texas, Austin), a $1911 \times 1908 \times 1813$ synthetic labeled volume containing various surface models of statues which have been voxelized, and a $404 \times 474 \times 512$ labeled volume containing a segmented leg reconstructed from MRI acquisitions (Source: MiraLab, Geneva).

Preprocessing. Datasets were processed on a Linux PC Intel Core 2, 2.66 GHz. The construction of the octree from source data was performed using octree bricks of 32^3 split voxels. Each split voxel was constructed from a discretized grid of 2^3 voxels with a 2 layers overlap (i.e., a 6^3 sampling grid). For the statue dataset, data processing took 7 hours and produced a $360MB$ octree database starting from an uncompressed source size of $3.1GB$, while for the turtle dataset, data processing took 15 minutes and produced a $60MB$ octree database from an uncompressed source size of $100MB$, and finally for the leg dataset data processing took 10 minutes and produced a $10MB$ octree database from an uncompressed source size of $100MB$ Processing times are comparable to those of other systems using high quality gradient precomputation (e.g., a Sobel 5^3 kernel) [GMG08]. The percentage of the split-voxel in the statue and leg dataset is 2% with respect to 98% of constant voxels. On the other side, for the generation of the turtle dataset we considered a separation threshold of 10 and we got a percentage distribution completely different: 72% split-voxels, with respect to 28% constant voxels. This distribution difference explains also the difference in compression ratio between label and scalar volumes. For comparison purposes, we also built multiresolution datasets to be used with nearest and trilinear rendering. Datasets to be used with the nearest technique were produced by considering a median filter to reconstruct a voxel from its 8 children, while an average filter was employed for datasets to be used with the linear technique.

Rendering. The performance of our rendering system prototype was evaluated on a Linux PC Intel Core 2, 2.66 GHz, equipped with an Nvidia GTX 280. We considered a number of interactive inspection sequences using the three 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 using a window size of 800×600 pixels. Our technique efficiently supports real-time transfer function manipulation: please refer to the accompanying video for interactive sequences recorded live. When using transfer functions ranging from moderately opaque to highly transparent, we experienced that the frame rate of typical inspection sequences varies between 10Hz for extreme close-up views with transparency to over 40Hz for overall views. Interactive rates are thus guaranteed even in the most demanding situations. With respect to image quality, we compared our split-voxel rendering method to common direct volume rendering strategies, employing nearest filtering and trilinear interpolation filtering. The nearest filtering strategy is able to separate between different materials, but it has the problem that reconstruction quality is intrinsically poor and needs high resolutions to get good quality images. On the opposite side, trilinear filtering increases reconstruction quality at the cost of losing boundary features. Furthermore, it suffers from color bleeding, since interpolated values generate false colors when accessing transfer functions even when using pre-integration. Instead, our rendering method exploiting split-voxels is able to keep separation between materials, so that each material keeps its color, also in the presence of an impulsive transfer function. Furthermore, boundaries are well preserved even at very low levels of details. Figure 6 compares the three rendering techniques applied to the datasets considered with respect to the ability of avoiding color artifacts, and the ability to reconstruct correct object silhouettes at various resolutions. In fact, for the segmented data, which is a detail of the statues label dataset, in the case of trilinear interpolation other parts of the transfer function modify the color of the model, while the split-voxel method keeps the correct color for each object in the scene (see also Fig. 1 and Fig. 8). Similarly, for the turtle intensity-based dataset, when using trilinear interpolation the boundary between bone and air is degraded by the interpolated value, especially for the simplified model. Figure 7 shows two semi-

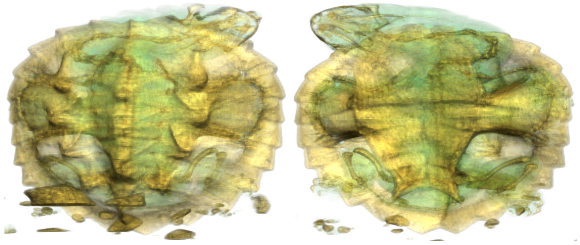


Figure 7: ***Semitransparent volume rendering.*** *Two views of the turtle dataset.*

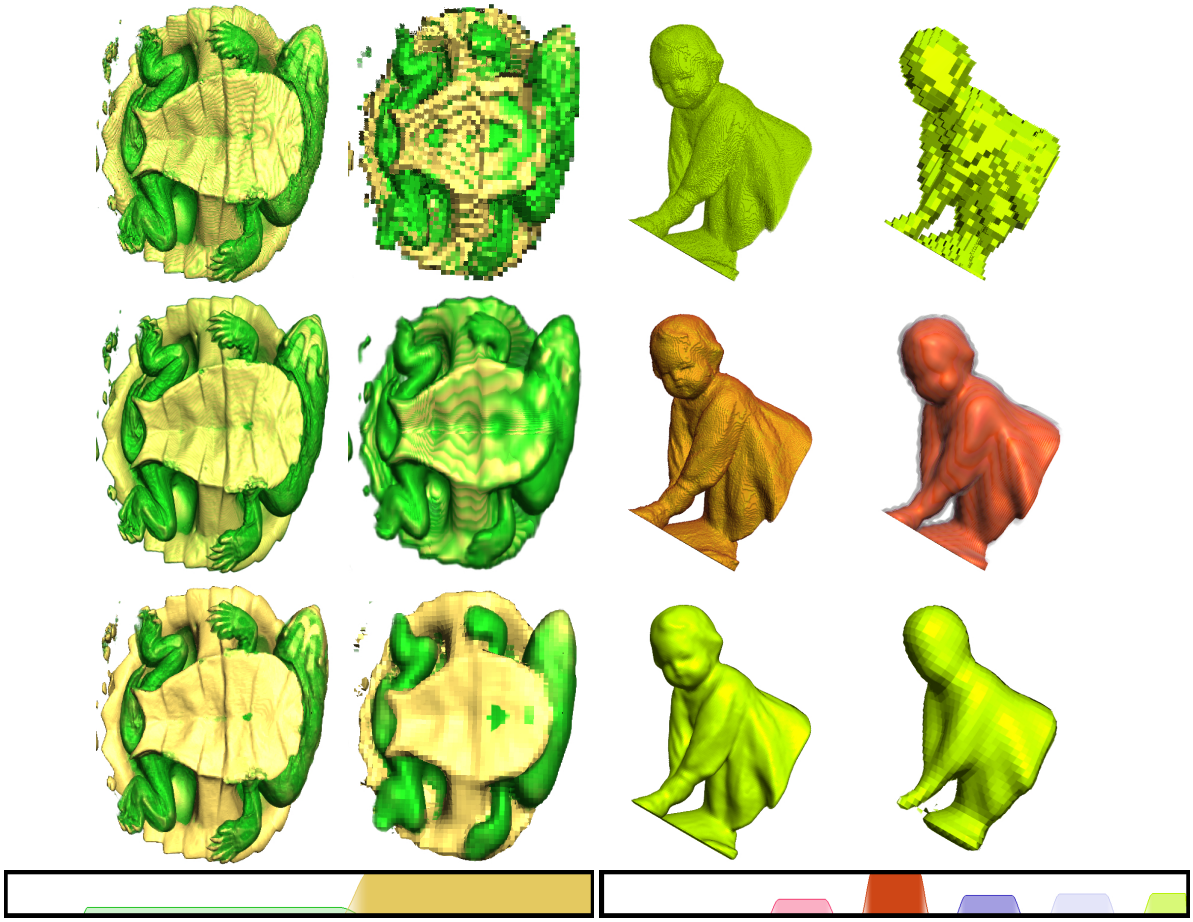


Figure 6: ***Rendering quality comparison.*** *Volume rendering of a continuous intensity based dataset of a turtle CT scan, and a close-up view of the discrete statues volume (first columns pixel tolerance 2, second columns pixel tolerance 8). Top row: nearest method is not able to preserve features as resolution decreases, but produces the right color. Middle row: linear method keeps a smooth silhouette, but color bleeding effect is present and increases for decreasing resolution, and the boundary tends to blur for the coarser representations. Base row: split-voxel method preserves objects silhouette and sharp separation among different layers even at very low resolutions, while maintaining the correct colors. Bottom: transfer functions. The right transfer function is the same one as employed for Fig. 1.*

transparent views of the turtle dataset. Figure 6 compares the three techniques also with respect to the ability to preserve features at low levels of detail. It appears evident that the nearest method suffers when resolution decreases, while trilinear interpolation keeps a smooth but thicker silhouette and introduces a color-bleeding problem, and finally the split-voxel method reconstructs nice silhouettes even for very low levels of detail, preserving the correct color associated by the transfer function. The low-resolution representations use very little memory. At extreme magnification levels, the discontinuous nature of the representation becomes evident, but the images remain understandable. The nice quality of low levels of details can be an advantage in a number of applications, e.g., for streaming and remote rendering.

7. Conclusions and Future Work

We presented a novel volumetric description, which trades continuity with the ability to model infinitely sharp value changes. This representation has little overhead over storing precomputed gradients. Separation planes provide in fact minimal geometric information about the strongest discontinuity in the underlying volume regions, which can be effectively exploited for multiresolution data filtering and volume rendering. In particular, we are able to loosely track material interfaces, as they occur in many physical objects, avoiding the mixing of unrelated values. When employed in a multiresolution representation, nice silhouettes are preserved even at very coarse levels of detail, reducing the data and time required to render understandable images. We have shown that the split-voxel primitive can be applied to volume datasets containing intensity scalar values, as well as material labels. Since the method is applied to scalar values, the renderer is able to change the transfer function in real-time without the need to reprocess the data. Even though our implementation can be improved in many aspects, our approach is the first attempt, to our knowledge, to model discontinuities inside a voxel prim-

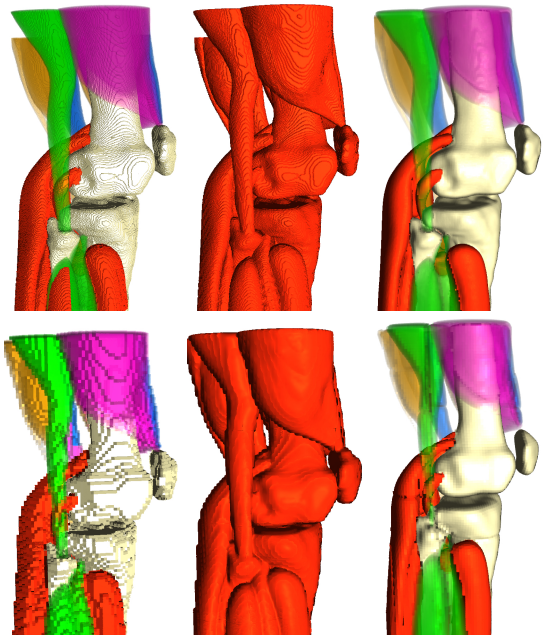


Figure 8: ***Knee segmented model.*** *Top row: knee model at pixel tolerance 2, base row same model at pixel tolerance 8. As explained on figure 6 the nearest method (left) preserves proper colors but presents jagged outlines, the linear one (center) preserves boundaries but produces color-bleeding effect, while the split-voxels (right) produces proper colors while maintaining nice silhouettes also for the coarsest pixel tolerance.*

itive in the context of a ray casting framework. The potential of the split-voxel representation has not yet been adequately explored. To this end, we plan to investigate other compositing strategies, in order to find more meaningful visualization metaphors, such as layer extraction, or non-photorealistic illustrative methods. Moreover, when dealing with massive models, we plan to further evaluate the compression capabilities of the split-voxel primitive.

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

References

[BG09] BRUCKNER S., GRÖLLER M. E.: Instant volume visualization using maximum intensity difference accumulation. *Computer Graphics Forum 28*, 3 (2009).

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proc. I3D* (2009), pp. 15–22.

[Don99] DONOHO D.: Wedgelets: nearly minimax estimation of edges. *Ann. Statist. 27*, 3 (1999), 859–897.

[EKE01] ENGEL K., KRAUS M., ERTL T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proc. Graphics Hardware* (2001), pp. 9–16.

[FI85] FUJIMOTO A., IWATA K.: Accelerated ray tracing. In *CG Tokio* (1985).

[GMG08] 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. *The Visual Computer 24*, 7–9 (2008), 797–806.

[HBH03] HADWIGER M., BERGER C., HAUSER H.: High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proc. Visualization* (2003), pp. 301–308.

[HSS*05] HADWIGER M., SIGG C., SCHARSACH H., BÜHLER K., GROSS M. H.: Real-time ray-casting and advanced shading of discrete isosurfaces. *Comput. Graph. Forum 24*, 3 (2005), 303–312.

[IGM10] IGLESIAS GUITIÁN J. A., GOBBETTI E., MARTON F.: View-dependent exploration of massive volumetric models on large scale light field displays. *The Visual Computer* (2010). To appear.

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. *ACM Trans. Graph. 21*, 3 (2002), 339–346.

[KBSS01] KOBBELT L. P., BOTSCH M., SCHWANECKE U., SEIDEL H.-P.: Feature sensitive surface extraction from volume data. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM, pp. 57–66.

[KHW*09] KNOLL A., HIJAZI Y., WESTERTEIGER R., SCHOTT M., HANSEN C., HAGEN H.: Volume ray casting with peak finding and differential sampling. *IEEE Trans. Vis. Comput. Graph.* (2009).

[KSS08] KYE H., SHIN B.-S., SHIN Y. G.: Interactive classification for pre-integrated volume rendering of high-precision volume data. *Graph. Models 70*, 6 (2008), 125–132.

[LHDC93] LUO L., HAMITOUCHE C., DILLENSEGER J.-L., COATRIEUX J.-L.: A moment-based three-dimensional edge operator. *IEEE Trans Biomed Eng 40*, 7 (1993), 693–703.

[Llo03] LLOYD S.: Least squares quantization in pcm. *Information Theory, IEEE Transactions on 28*, 2 (January 2003), 129–137.

[LWM04] LUM E., WILSON B., MA K.-L.: High-quality lighting and efficient pre-integration for volume rendering. In *Proceedings of Joint Eurographics-IEEE TVCG Symposium on Visualization* (May 2004), pp. 25–34.

[PK10] PAVIC D., KOBBELT L.: Two-colored pixels. *Computer Graphics Forum* (2010). To appear.

[PRH09] PRASSNI J.-S., ROPINSKI T., HINRICHS K. H.: Efficient boundary detection and transfer function generation in direct volume rendering. In *Proceedings of the 14th International Fall Workshop on Vision, Modeling, and Visualization (VMV09)* (nov 2009), pp. 285–294.

[RBW04] RAMANARAYANAN G., BALA K., WALTER B.: Feature-based textures. In *Proc. Rendering Techniques* (2004), pp. 265–274.

[RGW*03] ROETTGER S., GUTHE S., WEISKOPF D., ERTL T., STRASSER W.: Smart hardware-accelerated volume rendering. In *Proc. VisSym* (2003), pp. 231–238.

[SBSG06] SEREDA P., BARTROLI A. V., SERLIE I. W. O., GERRITSEN F. A.: Visualization of boundaries in volumetric data sets using lh histograms. *IEEE Trans. Vis. Comput. Graph. 12*, 2 (2006), 208–218.

[Sen04] SEN P.: Silhouette maps for improved texture magnification. In *Proc. Graphics Hardware* (2004), pp. 65–74.

[SH04] SIGG C., HADWIGER M.: Fast third-order texture filtering. In *GPU Gems*, vol. 2. Morgan-Kaufmann, 2004.

[TC04] TUMBLIN J., CHOUDHURY P.: Bixels: Picture samples with sharp embedded boundaries. In *Proc. Rendering Techniques* (2004), pp. 255–264.

[TC05] TARINI M., CIGNONI P.: Pinchmaps: textures with customizable discontinuities. *Computer Graphics Forum 24*, 3 (Sept. 2005), 557–568.

[Vap95] VAPNIK V. N.: *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[WN03] WILLETT R., NOWAK R.: Platelets: A multiscale approach for recovering edges and surfaces in photon-limited medical imaging. *IEEE Trans. Med. Imag. 22* (2003), 332–350.