The Visual Computer

Multiresolution volume visualization with a texture-based octree

Imma Boada[1], Isabel Navazo[2], Roberto Scopigno[3]

[1] Institut d'Informàtica i Aplicacions, Universitat de Girona
[2] Departament Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya
[3] CNUCE – Institute of the Italian National Research Council (C.N.R.)
e-mail: imma@ima.udg.es, isabel@lsi.upc.es, r.scopigno@cnuce.cnr.it

Although 3D texture-based volume rendering guarantees image quality almost interactively, it is difficult to maintain an interactive rate when the technique has to be exploited on large datasets. In this paper, we propose a new texture memory representation and a management policy that substitute the classical one-texel per voxel approach for a hierarchical approach. The hierarchical approach benefits nearly homogeneous regions and regions of lower interest. The proposed algorithm is based on a simple traversal of the octree representation of the volume data. Driven by a user-defined image quality, defined as a combination of data homogeneity and importance, a set of octree nodes (*the cut*) is selected to be rendered. The degree of accuracy applied for the representation of each one of the nodes of the cut in the texture memory is set independently according to the user-defined parameters. The variable resolution texture model obtained reduces the texture memory size and thus texture swapping, improving rendering speed.

Key words: Volume rendering – Octree – 3D Texture mapping – Multiresolution representation and rendering

Correspondence to: I. Boada

1 Introduction

In recent years, many researchers have searched for new rendering methods that can support real-time exploration of large volume datasets. The use of the specialized hardware features of new graphics subsystems has gained great importance in direct volume rendering [3, 12]. In particular, 3D texture-mapping hardware has become a common volume rendering method on high-end graphics workstations. Although the adoption of this hardware-enhanced approach guarantees high-quality images almost interactively, the limited size of the texture memory on standard graphics subsystems makes it difficult to maintain an interactive frame rate when large datasets have to be rendered. When the technique has to be exploited on a large dataset that does not fit into the available texture memory, a decomposition in bricks has to be applied to the data. In this case, the texture memory load-in and swap-out required for the visualization of the entire volume reduce the rendering performances substantially. Reducing the amount of texture memory needed to render a volume dataset, and thus reducing the texture loading overhead, is the main focus of our proposal.

In this paper, a new texture memory-management policy is presented. It is based on a multiresolution representation of the volume dataset in the 3D texture space. The proposed algorithm assumes an octree representation for the original dataset that could be computed during preprocessing. At rendering time, a set of octree nodes is selected. The selection is driven by user-defined criteria about the accuracy of the data representation and the importance of the data. More importantly, the degree of resolution (number of texels) required to represent each node in texture space is fixed. Note that the selected nodes determine how the volume has to be decomposed, i.e., the bricking decomposition. Note also that nodes of the same size could be represented by different numbers of texels. The variable resolution texture model obtained is not view dependent and takes advantage of data homogeneity and data importance, which reduces the texture loading and swapping cost while it guarantees the user-desired quality for the final images.

The paper is organized as follows. In Sect. 2, a short overview of the 3D texture-based volume-rendering approaches is given. The proposed octree-based representation and rendering algorithm is introduced in Sect. 3, and subsequently a complete description is given in Sect. 4. Results achieved with the proposed technique are presented in Sect. 5. Finally,

The Visual Computer (2001) 17:185–197

in Sect. 6, concluding remarks and future work are described.

2 Volume rendering based on 3D texture mapping

Hardware 3D texture-based rendering was first mentioned by Akeley [1] in 1993, and then used and enhanced in other works [3, 6, 20]. Fundamentally, this technique renders a volume dataset by a back-to-front composition of a set of planes that slices and samples the volume dataset that is loaded into the texture memory of the graphics subsystem. The resulting set of planes is drawn as a set of textured polygons that are blended together to obtain the final image. Efficiency is gained by automatic volume-data sampling on the cutting planes via the trilinear interpolation capability supported by the graphics subsystem. Texture mapping and compositing operations are performed by the hardware very quickly. Therefore, the rendering time is negligible compared to that of other software-based approaches.

Since the introduction of 3D texture-based volume visualization, several improved approaches have been proposed. The main differences between them are summarized here.

- *Voxel-texel relation.* The common mapping policy to represent the volume information in texture memory is one texel (texture element) per voxel. This strategy guarantees good image quality [14–16], but the limited size of the texture memory often makes impossible to store all of the dataset at once in the texture memory. Some approaches have been proposed to take this constraint into account. The method is generally based on the decomposition of the voxel space and the loading of chunks of texture data (often called *bricks*) into texture memory on request. A *brick* corresponds to a subset of voxel data that, by definition, is smaller than or equal to the available texture memory. Each brick is processed independently, and the final rendering is obtained from these contributions. Unfortunately, texture loading and swapping degrades the frame rate drastically. The bricking process can be performed with various strategies, which are based on one of the following criteria:
  * *Straightforward data partition.* The volume can be decomposed into the minimal number of bricks with maximal size. This criterion reduces redundant data that has to be stored between adjacent bricks to preserve continuity [5].
  * *No empty voxel representation.* The brick decomposition tries to reduce the percentage of empty voxels represented in the texture memory. Empty regions are detected during the preprocessing [14, 15].
  * *Multiresolution representation.* The brick decomposition is based on the evaluation of multiple user-defined parameters, such as the degree of interest, degree of homogeneity, distance from the viewer, etc. [2, 8].

  Data complexity has often been managed by the design of hierarchical data structures and adaptive methods [4, 7, 19] that trade quality for speed. In [14] an octree is proposed to deal with large datasets by skipping empty regions of the volume; a similar approach has been used in [15] to ensure that only nonvoid regions will be loaded into texture memory. In these methods, the final texture representation of the volume assigns one texel per voxel, thus, for large datasets, the texture memory limitation could still be a problem. A hierarchical data organization has been adopted in the context of a texture-based volume rendering system in [8], where texture bricks at different resolutions can be assigned to sections of the original dataset. The tiling solution is based on dual criteria: a static one (distance from a center of focus) and a view-dependent one (width of the brick silhouette in view space). This policy reduces the texture memory required for the volume representation, improving the technique when it has to be applied to large datasets, but it might have an impact on image quality. An improved rendering solution, based on the use of spherical shells instead of view-aligned planes, is also proposed in [8].
- *Shading or not shading texel values.* Commonly, the 3D texture is obtained from the application of a transfer function and/or a look-up table that maps the field values of the voxels into $RGB\alpha$ values. In this case, the texel assigned to the volume information represents an $RGB\alpha$ value. This texture representation becomes view dependent [20] if shading is computed on the voxels to enhance the visualization of the isosurfaces implicitly contained in the data. This slightly increases the voxel-to-texture conversion phase (the transfer-function mapping has to be followed

by an explicit texel-shading step based on voxel gradients), and, more dramatically, adds a substantial data transmission overhead (a new 3D texture has to be loaded into the texture memory of the graphics subsystem for each different frame, corresponding to a modified orientation of the voxel set with respect to the scene lighting).
A new strategy, in which texel values represent density values and gradients has been proposed by [11, 17]. In this case, the texture representation is no longer view dependent, and shaded images can be produced more efficiently.

The approach presented in this paper is similar to the one in [8]. The main differences are: the volume data representation and the criteria used to organize the data; the approach devised for the selection of the dataset partition (bricks) and for determining the resolution at which each brick is represented. Our algorithm does not consider view-dependent criteria (which are generally based on the perspective distortion and shrinking of farthest data sections) because, in volume rendering, the differences in the projecting ratio are very limited. Because of the limited spatial extent of a volume dataset, all voxels have an approximately similar projection size. Moreover, parallel projections are very often used. The focus is therefore on the design of a view-independent multivalued criterion that should take both data homogeneity and data importance into account in an integrated manner.

3 Efficient rendering of large datasets

The purpose of this paper is to speed up the texture-based rendering of large datasets. We consider "large" a volume dataset whose size does not fit the available texture memory. The common bricking process applied to render such datasets (partition in bricks, each one loaded and processed independently) reduces the frame rate and makes rendering insufficiently interactive. The multiple texture memory loads and swaps required to render the complete volume are the main reason for the overhead.
Since the earliest work from Meagher [10] concerning 3D data representation based on the octree scheme, many applications of this hierarchical representation have appeared over the years [13], e.g., to improve rendering speed, to manage large data sets easily, or to obtain error-controlled rendering. They have also often been adopted in volume rendering, with the object of avoiding the evaluation of noncritical regions and optimizing the rendering speed [7, 9, 18, 19].
In this paper, we present a method that is based on a hierarchical representation of the data (using the octree scheme) and that performs a selective traversal of the octree representation to obtain a user-driven adaptive representation of the dataset in the 3D texture space.
The octree representation is built in a preprocessing phase. To do this, some attributes that characterize the corresponding voxel sub-region in terms of data homogeneity are stored in each octree node. The octree construction phase is described in detail in Sect. 4.1. Given some user-defined constraints, the octree is traversed at rendering time and a set of nodes, which we call the cut, is selected during the traversal. This set of nodes determines the current decomposition to be used to render the volume. The resolution of the texture brick used to render each selected octree node is also determined at traversal time (see Fig. 1). All these processes are view independent. An efficient algorithm to visualize the cut from the view position is presented in Sect. 4.3.
Given an octree representation of a volume dataset, we formally define a cut on the octree as a set of nodes $C = \{n_0, n_1, ..., n_{nb}\}$ such that, for each possible root-leaf path, one and only one of its nodes is contained in the cut.
The set of texture bricks (number and spatial extent in the voxel space) that will be used in rendering directly depends on the selected cut. For performance reasons (see details of the rendering algorithm in Sect. 4.3), the number of bricks cannot be too large, and the spatial extent in voxel space cannot be too small (otherwise the number of textured polygons will increase excessively and aliasing could become massive). The voxels of the original dataset could be represented at a different accuracy. The voxels of the original dataset could be represented at different accuracy, by assigning texture bricks of different resolution to each node in the cut.
For each node n_i of the cut, we should determine an optimal texture brick B_i that corresponds to the hexahedral region covered by this node. The resolution of B_i is set according to the user-defined image quality parameters, i.e., homogeneous content (threshold on the representation accuracy) and/or importance of the corresponding region (defined by the selection of focus regions), which determine the degree of accuracy (maximal error ε^*) admitted for its render-

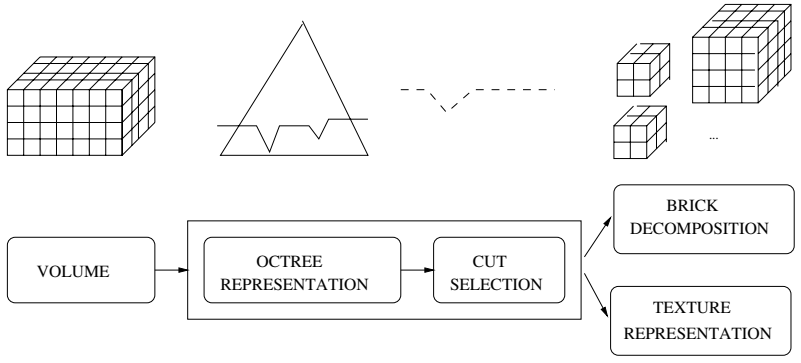


Fig. 1. Given a volume data set greater than the available texture memory, a set of nodes of its octree representation is selected. These nodes determine the current bricking of the volume, which also minimizes the texture memory required to confirm to some user-selected criteria

ing. This parameter generally gets different values for the different bricks (see Sect. 4.2 for a detailed description).

The final result is that the selection of the cut C also implies the selection of a brick decomposition $B_v = \{B_0(d_o), ..B_{nb}(d_{nb})\}$, where nb corresponds to the number of nodes of the cut and $B_i(d_i)$ represents the texture representation of node n_i in C at the selected resolution d_i.

A compressed representation of the dataset is thereby obtained. Depending on the dataset size and the parameters selected by the user, the resulting set of bricks fits into the available texture memory. Obviously, the rendering speed-up is obtained at the expense of rendering quality: the compression of the data is lossy (this depends on the user-selected thresholds), and some aliasing could be introduced into the images.

4 Importance-driven rendering based on octree traversal

There are three steps in the proposed algorithm: (1) The octree construction, (2) the cut selection, and (3) the cut rendering. These steps are all described in detail here.

4.1 Octree construction

The octree construction needs the definition of criteria to determine whether a region of the volume is important or not in order to decide if an internal node must be subdivided. These criteria vary from the simplest one that evaluates the "presence" or "absence" of significant data to a more sophisticated one that determines whether the volume samples associated with a node vary according to a given reconstruction kernel applied to selected voxel values. This is done to detect nearly homogeneous zones. Once a criterion has been defined, the *accuracy* of the node represents the degree of satisfaction of the given evaluation rule. Let us consider data homogeneity. A large dataset can contain large data sections where the field values are either uniform or variable in a very smooth manner. In all these cases, the dataset section associated with an octree node can also be represented, with a certain degree of approximation, by the eight values on the corners of the associated region of space (and all internal data can be approximated by adopting an interpolation kernel, e.g., trilinear interpolation).

In our approach, a space-efficient hierarchy (the octree) is required, not only to identify nearly homogeneous regions, but also to evaluate and store the *accuracy* associated with alternative representations at different resolutions of the same node region. It must also integrate a user-dependent criterion based on the on-line selection of focus areas (regions of greater interest where the representation at maximal resolution is always preferred).

The octree is built from the input voxel set. Voxel values are not represented explicitly by the octree nodes, but implicitly (the extent of the corresponding

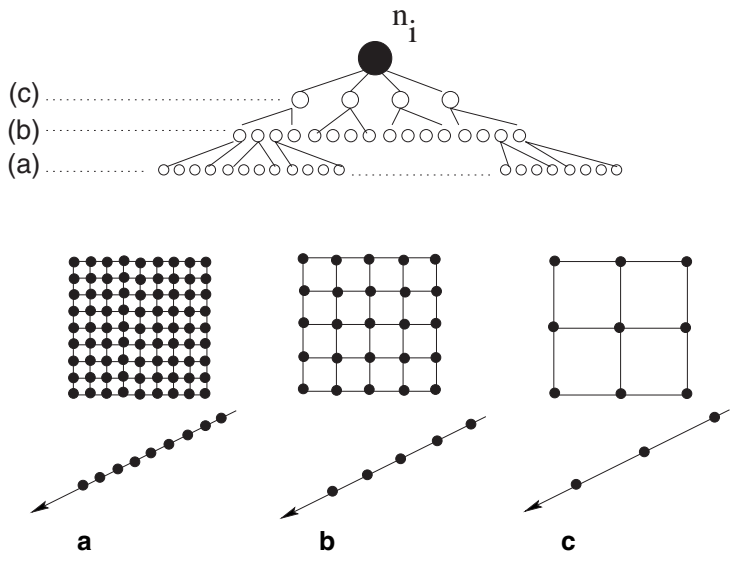


Fig. 2. Given a node n_i of one of the possible cuts on the octree, the corresponding subtree gives us multiple choices for assigning a corresponding texture chunk that is defined by subsampling the associated voxel region

region in the voxel space are directly computed from the node index). The octree is initially complete, and then all sibling nodes that can be merged in the parent node without introducing an approximation error are purged (bottom-up traversal). For each node n_i at level k of an octree with depth $l_{\max}$, our octree representation stores a list of *nodal errors* $\varepsilon_0, .., \varepsilon_{l_{\max}-k}$ such that:

- ε_0 Is the *nodal error* introduced in the volume representation when all the voxel values contained in the region associated with the octree node are simply replaced by a (trilinear) interpolant *TrilInt*() based on the eight corner values of n_i. Given $s(v_j)$, the sampled field value of a voxel v_j contained in the region associated with node n_i, and $s_{\max}$, the voxel field range, this error is computed as:

$$\varepsilon_0(n_i) = \frac{\sqrt{\frac{1}{2^{3(l_{\max}-k)}} \sum_{j=1}^{2^{3(l_{\max}-k)}} (s(v_j) - TrilInt_{n_i}(v_j))^2}}{s_{\max}} \quad (1)$$

- Analogously, each one of the other *nodal error* values $\varepsilon_1, .., \varepsilon_{l_{\max}-k}$ of node n_i is computed by assigning the maximum of the ε_0 nodal errors of those descendant nodes at the octree level j ($j = k+1 \,..\, l_{\max}$).

These nodal errors give a measure of the degree of homogeneity in the area covered by the node: the degree of homogeneity of the region covered by n_i increases if the nodal error ε_0 tends to 0; conversely, homogeneity decreases as it tends to 1. In practice, for each node, the *nodal error* ε_m ($m = 0 \ldots l_{\max} - k$) gives us a measure of the accuracy supported when we assign to node n_i a texture chunk of resolution $2^m * 2^m * 2^m$.

4.2 Octree cut selection

Before introducing details on the cut selection phase and the bricks definition, we analyze the various policies of representing an octree node in texture memory, and we define the optimal one in terms of accuracy and memory savings.

4.2.1 Node texture representation

To represent the region covered by a node n_i in texture memory we can consider the classical assignment policy of the one-texel-per-leaf node (a voxel in a complete octree), or a more synthetical representation in which one texel can represent multiple terminal nodes (Figs. 2 and 3). In the first case, the values in the texture memory correspond to original volume data values. The second case occurs when a more compact representation is associated with a given node, either because the

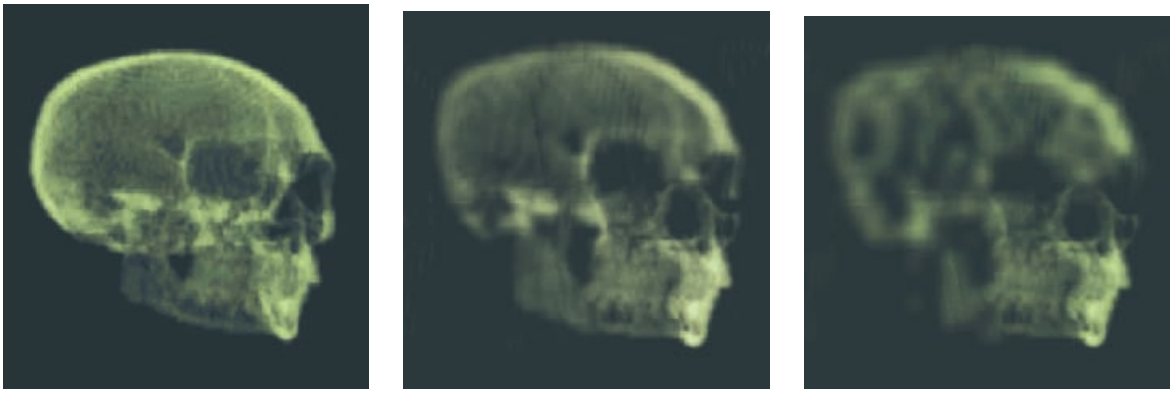


Fig. 3. Different rendering obtained on the skull dataset by adopting different voxel-to-texel ratios: from left to right: 1 : 1, 1 : 2, 1 : 4 (at each step, texture size becomes 1/8 of the previous one)

node corresponds to a nearly homogeneous region or because it is contained in a region of low importance.

For each node n_i at level k, we have a corresponding subtree of depth $l_{\max} - k$ and a corresponding voxel region of resolution $2^{l_{\max}-k} * 2^{l_{\max}-k} * 2^{l_{\max}-k}$. We can represent the voxel region associated with n_i by using any one of the regularly subsampled representations built on the initial voxel values. These are not explicitly stored in the octree, but can be simply extracted from a hierarchical representation of the voxel dataset (a pyramid of 3D voxel sets, each one obtained by regularly subsampling the previous level).

Therefore, if we select a set at the deepest level, we end up with the case we have described (one texel for each leaf node). In all other cases, we select a more compact representation corresponding to nodes that are p levels below in the octree structure (with $p = 0 \cdots l_{\max} - k$). In this case, we should consider that:

- The accuracy of the texture-based representation is encoded in the node's corresponding *nodal error*, $\varepsilon_R(n_i) = \varepsilon_p(n_i)$.
- The *texture memory* required to represent the node n_i is $2^p * 2^p * 2^p$ and some extra texture memory is required to store the boundary information between neighbor bricks. This value varies with the filtering scheme used; in the case of a trilinear interpolation kernel, we should add a one-voxel-thick layer of voxels in each direction.

4.2.2 The octree traversal

The octree is traversed top-down to select the nodes that will compose the cut. The main constraints to be satisfied are the following:

> *We should determine a valid cut such that the corresponding brick decomposition B_v guarantees that all dataset regions corresponding to the nodes n_i in the cut have a representation in B_v whose approximation error ε_R satisfies the user-selected error threshold ε_u, i.e. $\varepsilon_R \leq \varepsilon_u$ and such that the size of the texture representation associated to the cut and the cardinality of the brick decomposition are minimized.*

Obviously, the degree of compression that we can achieve by simply considering the possible homogeneous subregions of the dataset is limited. For this reason, we also take into account two user-defined parameters: the error threshold and the node importance.

Given ε_u, the user-desired accuracy threshold, the *optimal* texture representation of n_i with respect to ε_u is the smaller texture representation that satisfies a nodal error less than ε_u. We denote this representation as $S_{\text{opt}}(n_i, \varepsilon_u)$. An example is shown in Fig. 4: the subtree rooted in a given node n_i at octree level k. After an explicit node condensation (because of uniform values in the dataset), levels $k + d_1$, $k + d_2$ and $k + d_3$ (represented as dashed lines) are three descendent levels that can guarantee a texture error representation less than the user-tolerated error (i.e.,

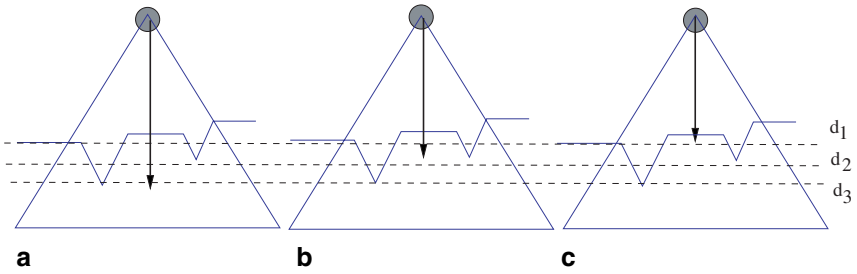


Fig. 4. The optimal texture representation of a node for the smallest subsampled texture representation that satisfies the required accuracy

$\varepsilon_{d_1}(n_i) \leq \varepsilon_u$, $\varepsilon_{d_2}(n_i) \leq \varepsilon_u$ and $\varepsilon_{d_3}(n_i) \leq \varepsilon_u$). Therefore, the optimal texture representation $S_{\text{opt}}(n_i, \varepsilon_u)$ is extracted from the volume information stored in descendent nodes of level d_1.

To evaluate the importance of a node, we also define an *importance function*, denoted as $Imp(n_i, f)$, that evaluates the degree of satisfaction of a focusing operator f defined by the user over the dataset domain. This function weights a node's importance by taking into account a region of interest (ROI), which in turn can be defined by the user with the selection of a simple 3D subregion of the dataset domain or a particular field value subrange, identified by maximal and minimal value bounds. By definition, the importance of a node n_i with respect to a function f will be a value in the [0,1] interval. As the importance varies according to the current function f adopted (which can be dynamically adjusted by the user), the importance value is not stored in the octree, but is computed on-line during the octree traversal. Moreover, let us also define a second function $Unif(n_i, f)$, having again in the co-domain the interval [0,1], which gives us a measure of how much the value of $Imp(n_i, f)$ is uniform in the space associated with node n_i. Therefore, a small value of the $Unif(n_i, f)$ function can highlight situations where only a section of the space associated with the given node is contained in the focus region. It can be used to drive the selection of a decomposition of the space (e.g., the one corresponding to its descendant nodes), which could lead to a better representation of the region.

Taking into account all these restrictions, we evaluate the importance of the octree nodes during the octree traversal. The approximation error ε^* tolerated in the texture representation of a single node n_i is obtained by weighting the user-selected threshold with the node *importance*:

$$\varepsilon^* = \frac{\varepsilon_u}{Imp(n_i, f)} . \tag{2}$$

In this form, each node is associated with its own user-defined ideal error ε^*. Regions of little interest will have small ε^* values, which allows the use of a coarse representation for the associated brick (i.e., a compact $S_{\text{opt}}(n_i, \varepsilon^*)$).

The brick decomposition of the volume is obtained with a top-down traversal function $OTcut(IN: OT\_node, \varepsilon_u, f, T_{\max}, minPartSize; OUT: brick\_set)$ where:

- ε_u Represents the user-defined tolerance error.
- f Represents the interest function.
- $T_{\max}$ Is the maximum size of 3D texture memory available on the graphics subsystem.
- *minPartSize* Is the minimal partition size that can be produced in the volume decomposition process. This parameter is related the maximal depth we are allowed to reach in the traversal of the octree. For rendering reasons, it is not possible to generate a large number of texture bricks (to maintain rendering overhead and data redundancy at an acceptable level). Therefore, we limit the level of the nodes that can be used in rendering to the interval $[0, l_{\max} - minPartSize]$, where $l_{\max}$ is the depth of the initial octree. In this manner, in the worst case, the dataset is subdivided into no more than $(2^{(l_{\max} - minPartSize)})^3$ bricks, and each one of them represents a volume subset of at least $(2^{minPartSize})^3$ voxels; please note that we can assign a smaller texture brick to one of these

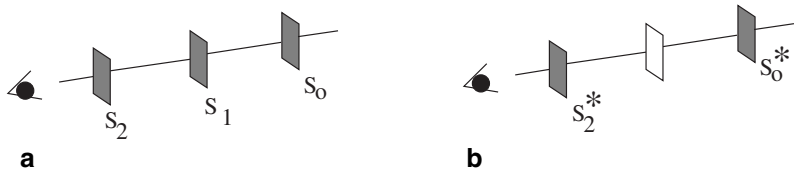


Fig. 5. To render textures at a different resolution, a number of planes proportional to the texture resolution is generated (and thus a number of samples)

partitions (e.g., when the corresponding voxel set is sufficiently uniform).
- *brick_set* Is the texture representation produced in the output.

The *OTcut*() function traverse the octree evaluating the root node first. If the region associated with the root node is sufficiently uniform (i.e., $Unif(n_i, f) \geq$ unif_threshold, with *unif_threshold* a user-defined parameter) and the representation of minimal size that satisfies ε^* is smaller than or equal to the maximum texture $T_{\max}$, then this representation is produced in the output and rendered. Otherwise, we recursively apply *OTcut*() to the descendants of the current node, with the constraint that the sum of the texture representations chosen for all the descendants node should be less than $T_{\max}$.
The traversal of the octree proceeds until we find a cut where each node satisfies both the data uniformity criterion and the texture-based representation accuracy. Given this cut, it can happen that the total size of the brick-based decomposition is still larger than the available texture memory $T_{\max}$. In this case, the accuracy constraint is released, and some of the nodes are rendered with more compact representations (giving priority to the nodes having a lower *Imp*() value).

4.2.3 From samples to texels: opacity correction factor

Once a cut has been selected and a subsampled representation has been chosen for each node in the cut, we have to convert voxel values into texels. Some issues have to be considered when the texture is defined. Firstly, a conductor is imposed by hardware and it forces texture data loaded into texture memory to be packed into a rectangular parallelogram, each dimension constrained to be a power of two. Secondly, the fact that image quality must be preserved forces each texture to share its boundary with its neighboring texture nodes. Thirdly, we must consider the opacity assigned to the texels. Usually, a *field_to_RGBα* conversion, based on the current transfer function mapping, is done. When a subsampled representation of the raw voxels is assigned to a node, an opacity correction factor has to be applied to preserve image quality. This opacity correction factor is calculated as follows.
The accumulated opacity α_{out} of a volume element [9] can be expressed as

$$\alpha_{\text{out}} = \alpha + \alpha_{\text{in}}(1-\alpha), \tag{3}$$

where α is the opacity of the volume element and α_{in} is the entering opacity of this element. The α_{out} value depends directly on the number of samples that are considered, which, in our case, depends on the number of planes used in the texture-based rendering. Of course, it is desirable that images obtained with texture at a different resolution (obtained by subsampling) be very similar. To reduce the difference between the various texture resolutions selected for a given octree cut, an *opacity correction factor* opc is applied to every subsampled representation. Let us analyze the two situations of Fig. 5 (textures of different resolutions assigned to the same node): α_0, α_1, and α_2 are the opacities of samples s_0, s_1, and s_2 taken into consideration for a given view ray in Fig. 5a, and α_0^* and α_2^* are the opacities of samples s_0^* and s_2^* accumulated on the same ray in Fig. 5b. Using (3), we find that the opacity accumulated by the back-to-front composition of these samples is, respectively:

$$\alpha_a = \alpha_2 + \alpha_1(1-\alpha_2) + \alpha_0(1-\alpha_1)(1-\alpha_2) \tag{4}$$

$$\alpha_b = \alpha_2^* + \alpha_0^*(1-\alpha_2^*), \tag{5}$$

where $\alpha_0^* = \alpha_0$ and $\alpha_2^* = \alpha_2$.

The accumulated opacities are the same only if $\alpha_0^* = \alpha_0 + \alpha_1(1-\alpha_0)$. Thus, the optimal op_c is $op_c = \alpha_1(1-\alpha_0)$, where $\alpha_0 = \alpha_0^*$ and α_1 is approximated according the criterion applied for the octree construction. If the octree construction criterion is that a node is *homogeneous when all the samples are nearly equal*, then $\alpha_1 = \alpha_0 + \epsilon'$, where ϵ' depends on the approximation error of the node; if the octree construction criterion considers a node must be *homogeneous when internal samples can be obtained by interpolation of the external ones*, α_1 can be approximated by $\frac{\alpha_2 - \alpha_0}{2} + \epsilon'$. Obviously, the complexity of the computation of this op_c value depends on the criterion applied for the octree construction. Nevertheless, because coarser representations are applied only at regions of lesser interest, it has been experienced that the $\alpha_1 = \alpha_0 + \epsilon'$ rule gives sufficiently good results. Although this approximation introduces some error, this error only affects less critical, nonfocus regions.

In general, given a node $n_i \in C$ that represents 2^{l^3} samples of the original volume dataset, and given q, the selected level for its representation in texture space, where $q \leq l$, the α_i^* opacity values of the texels are substituted with $\alpha_i^* + op_c$, where

$$op_c = (1-\alpha_0)((1-\alpha_0)^{q-1} - 1) .$$

4.3 Octree cut rendering

The octree cut rendering, which only considers orthographic projections in our implementation, is based on an iterative application of a node visualization function that computes the contribution of each brick to the final image. The composition order of the bricks is obtained directly from a back-to-front octree traversal, driven by the viewing direction.

4.3.4 Node visualization function

The node visualization function is based on the algorithm proposed in [16]; all texture space transformations and clipping plane texturing/compositing are implemented with standard graphics library functions.

To describe the node visualization function, we consider that it is applied to a node n_i of the cut. The texture representation of this node is T_i, and np_i is the number of polygons that will be used for the node's rendering[1]. The function can be decomposed in the following steps:

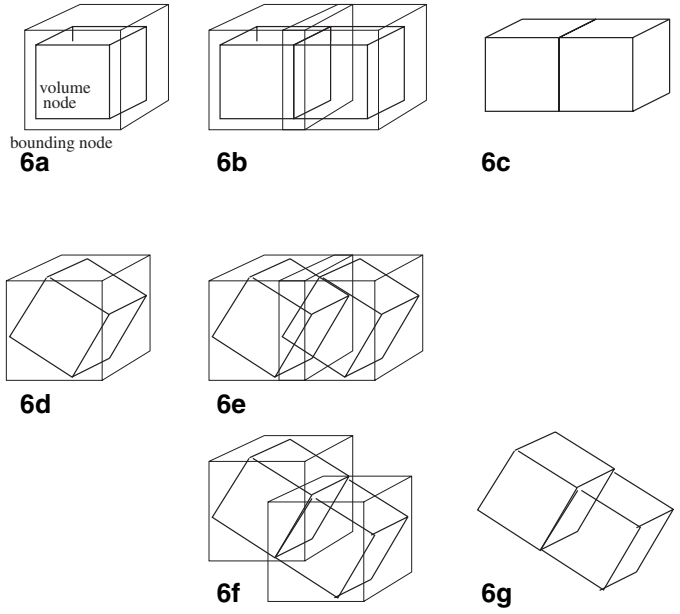
1. *Bounding cube definition.* A bounding cube centered on the center of the node n_i is defined. We call it bn_i. The corresponding polygons np_i to be rendered form series of slices through bn_i, parallel to the xy face.
2. *Clipping planes activation.* The set of np_i polygons to be rendered always remains parallel to the projection plane, and extends to the bounding cube. The node's volume is represented by T_i and rotates into bn_i depending on the viewing direction (Fig. 6a and d). To guarantee that only the volume information represented in the texture will contribute to the final image (i.e., external areas of the node contained in bn_i are eliminated) a set of clipping planes is defined. The clipping planes are positioned according the viewing direction at each one of the node's volume faces.
3. *Node's positioning.* Once the clipped polygons that represent the volume node are computed, we determine where they have to be projected. The position of these polygons is fixed by the node's octree position and by the viewing direction. These parameters determine the current set of geometrical transformations. A first geometrical transformation determines the position of the bn_i according to the octree position (Fig. 6b and e). A new orientation requires a rotation of the texture space, keeping bn_i stationary. This rotation is independent for each node and is dependent on the viewing direction. To maintain continuity between nodes, a second transformation (Fig. 6f) translates the bounding node.
4. *Texture mapping.* Finally, texture coordinates are assigned to the transformed and clipped polygons, according to [16].

All these steps produce the final I_i contribution to the image (Fig. 6c and g).

5 Experimental results

Our method has been implemented on a SGI Octane workstation using a single 270 MHz/R12000 processor with 4 MB of texture memory. The size of the

[1] np Is computed by finding the longest ray through the node (for any viewing direction), transforming the voxel values found along that ray to the texels.



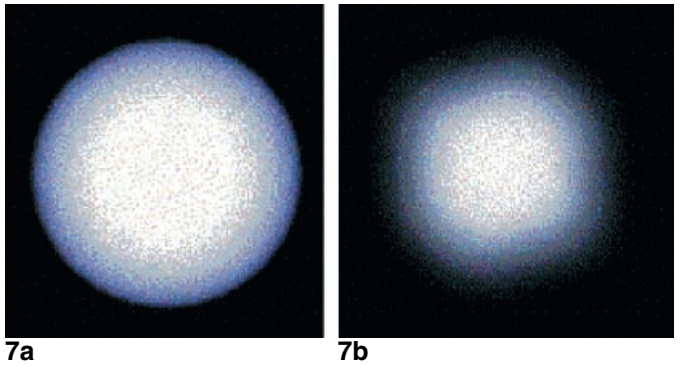


Fig. 6a–g. This illustrates an orthographic projection of two 3D bounding nodes, the first in the original orientation (**a**) and the second one rotated (**d**). A neighbor node is correctly positioned by a set of transformations, (**b**), (**e**), and (**f**). Panels (**c**), (**g**) correspond to the final image contribution of the node

Fig. 7a,b. Rendering obtained from a 64^3 3D texture that satisfies $\varepsilon_u = 0$ (**a**); rendering obtained from a 8^3 3D texture that satisfies $\varepsilon_u = 0.3$ (**b**)

texture cache is $128 \times 128 \times 64$, thus the size of the maximal texture brick is 64^3.

Figure 7 shows two different images rendered from a 64^3 sphere dataset. The first image (Fig. 7a) has been obtained from a 64^3 full resolution texture (i.e., the dataset is represented by the values of the maximal division nodes). The error of the representation satisfies $\varepsilon_u = 0$. The second image has been obtained by selecting a uniform error threshold $\varepsilon_u = 0.3$ on the entire data domain; a 8^3 3D texture has been generated in this case, and rendered in Fig. 7b.

Three real medical datasets have been used for the tests: a CT-scanned jaw model of resolution $256 \times 256 \times 128$, a MRI-scanned brain of resolution $256 \times 256 \times 128$ and a CT-scanned vertebra of resolution 256^3.

The experimental results are listed in Table 1, where the (*nodes*, *res*) column represents the number of nodes that compose the rendered cut and the resolution of the corresponding B_v texture representation assigned; $S(B_v)$ represents the required texture size for the B_v representation; $T_P(B_v)$ represents the data processing time for the cut selection and the texture generation time, which is executed once for each update of the user-selected parameters; and T_R is the rendering time.

The image shown in Fig. 8 was obtained by selecting the region associated with the front-most tooth

Table 1. Experimental results

Row	Data set	(nodes,res)	$S(B_v)$ in megabytes	$T_P(B_v)$ in seconds	T_R in seconds
1	MRI brain	(32,64^3)	32	$71+5$	2.8
2	MRI brain	(4,64^3)	4	$57+0.7$	0.9
3	CT jaw	(32,64^3)	32	$71+5$	2.8
4	CT jaw	(144,multiresolution)	11	$66+1.4$	1.3
5	CT vertebra	(64,64^3)	64	$79+10$	6.4
6	CT vertebra	(360,multiresolution)	42	$81+2.5$	3.5

Fig. 8. A variable resolution rendering of the CT jaw model (the current region of interest is centered on the front teeth)

area as the region of maximal interest. The other parameters drive the octree traversal: $\varepsilon_u = 0.1$, and *minPartSize* $= 4$ (corresponding to a 16^3 voxel region). Only nodes inside the ROI have $Imp(n_i, f)=1$. The cut obtained is composed of 144 nodes represented at different resolutions. Nodes of the cut that correspond to empty regions of the volume are not represented in texture memory (i.e., the corresponding octree nodes are purged). The required texture space for this cut representation reduces the texture memory required by more than half for the *classical texture* representation in which all the volume is represented at high resolution (see rows 3 and 4 of Table 1).

In the case of the MRI brain data, the whole dataset was considered to be of maximal interest, i.e., $Imp(n_i, f)$ is always 1. We can either render this dataset using bricks at maximal accuracy ($\varepsilon_u = 0$), which would require a texture memory eight times larger than the available one or multiple texture loads/swaps, or we can render the same model using an $\varepsilon_u = 0.35$ that allows the entire cut to fit into the available texture memory (four nodes that can be represented at the selected accuracy by bricks at resolution 64^3). The corresponding times are presented in rows 1 and 2 of Table 1.

Finally, images in Fig. 9a and b are obtained from the CT vertebra dataset. The homogeneous interior structure of the vertebra allows a simplification of the texture representation of the model with the minimal error, $\varepsilon_u = 0.1$, with respect to the higher resolution representation (see rows 5 and 6 in Table 1). The cut used to produce these images is composed of 360 nodes represented at different resolutions (where *minPartSize* $= 4$, corresponding to a 16^3 voxel region).

6 Conclusions and future work

The algorithm introduced in this paper focuses on the 3D texture volume rendering of large datasets. Based on an octree representation of the volume data, the algorithm can produce a concise texture representation that takes advantage of data homogeneity and data importance in order to reduce the required texture memory and thus to improve rendering speed.

The basis of the algorithm is a new texture management policy in which texture memory is assigned according to the degree of importance of the region represented and the data content of the voxel dataset. Driven by a user-defined combination of data, interpolation error, and data importance, the octree is traversed and a set of nodes is selected. These nodes determine how the volume has to be decomposed and represented in texture memory. Experimental results show that good quality visualizations can be obtained while the rendering speed is improved.

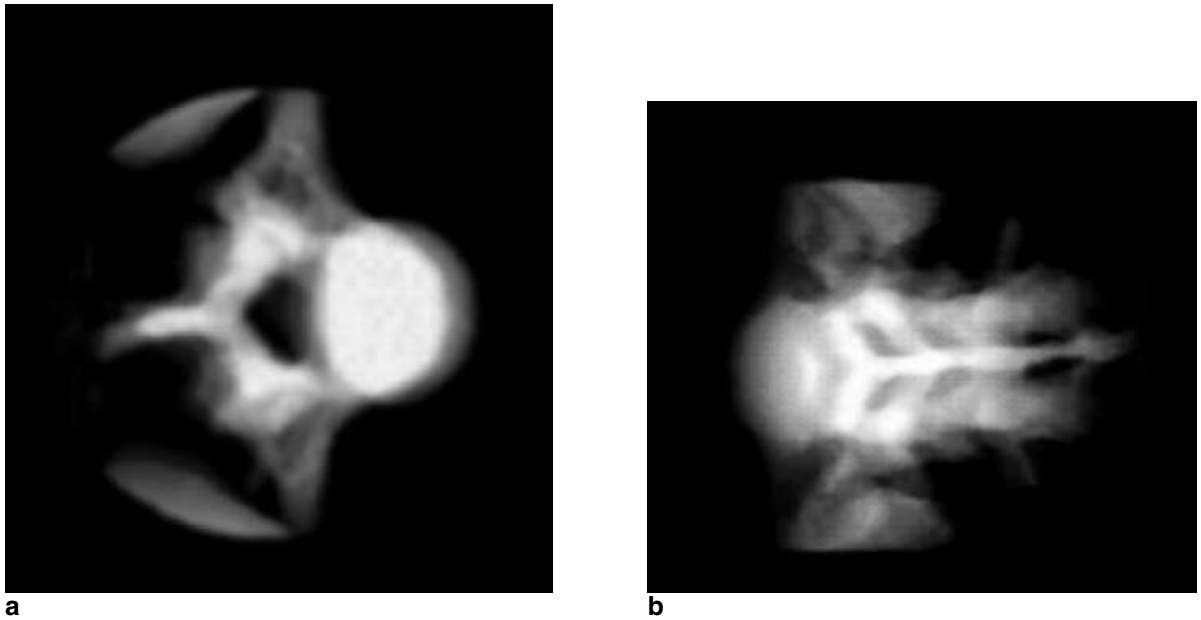


Fig. 9. Multiresolution rendering of the CT vertebra dataset

"Future work" is addressed to analyze different heuristics to select the *interest function* that better satisfies user-defined restrictions. Furthermore, we want to address the problem of the combining surface and volume information.

Acknowledgements. This work was partially financed by grant TIC-980586-C03-01 of the Spanish Goverment.

References

1. Akeley K (1993) Reality engine graphics. Comput Graph 27:109–116
2. Boada I, Navazo I, Scopigno R (2000) A 3D texture-based octree volume visualization algorithm. In: Skala V (ed) Short Communication papers of the 8th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media '2000, University of West Bohemia, Plzeň, Czech Republic, pp 1–8
3. Cabral B, Cam N, Foran J (1994) Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: Kaufman A, Krueger W (eds) Proceedings of the 1994 Symposium on Volume Visualization, Washington, DC, pp 91–98
4. Danskin J, Hanrahan P (1992) Fast algorithms for volume ray tracing. Proceedings of the Workshop on Volume Visualization, ACM Press, New York, pp 91–98
5. Grzeszczuk R, Henn C, Yagel R (1998) Advanced geometric techniques for ray casting volumes. ACM SIGGRAPH '98, Course Notes n. 4, pp 1–239
6. Guan S-Y, Lipes R (1994) Innovative volume rendering using 3D texture mapping. In Proceedings Medical Imaging 1994 – Image Capture, Formatting and Display, vol 2164, SPIE, pp 382–392
7. Laur D, Hanrahan P (1991) Hierarchical splatting: a progressive refinement algorithm for volume rendering. (SIGGRAPH '91) Comput Graph 25:285–288
8. La Mar EC, Hamann B, Joy KI (1999) Multiresolution techniques for interactive texture-based volume visualization. In: Ebert D, Gross M, Hamann B (eds) Proceedings of the 1999 IEEE Conference on Visualization (VIS-99). ACM Press, San Francisco, CA, pp 355–362
9. Levoy M (1990) A hybrid raytracer for rendering polygon and volume data. IEEE Comput Graph Appl 10:33–40
10. Meagher D (1982) Geometric modeling using octree encoding. Comput Graph Image Process 19:129–147
11. Meissner M, Hoffmann U, Straßer W (1999) Enabling classification and shading for 3D texture mapping based volume rendering. In: Ebert D, Gross M, Hamann B (eds) Proceedings of the 1999 IEEE Conference on Visualization (VIS-99), ACM Press, San Francisco, CA, pp 207–214
12. Pfister H, Hardenbergh J, Knittel J, Lauer H, Seiler L (1999) The VolumePro real-time ray-casting system. Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH '99) Los Angeles, CA, pp 251–260
13. Samet H (1990) Applications of spatial data structures. Addison Wesley, Reading, Mass

14. Srinivasan R, Fang S, Huang S (1997) Rendering by template-based octree projection. In: Lefer W, Grave M (eds) Proceedings of the 8th Eurographics Workshop on Visualization in Scientific Computing, Boulogne-sur-Mer, France, Eurographics, pp 155–163
15. Tong X, Wang W, Tsang W, Tang Z (1999) Efficiently rendering large volume data using texture mapping hardware. In: Gröller E, Löffelmann H, Ribarsky W (eds) Data Visualization '99, Eurographics. Springer, Vienna, pp 121–132
16. Van Gelder A, Kim K (1996) Direct volume rendering with shading via three-dimensional textures. In Crawfis R, Hansen C (eds) 1996 Symposium on Volume Visualization, San Francisco, CA, pp 23–30
17. Westermann R, Ertl T (1998) Efficiently using graphics hardware in volume rendering applications. In: Cohen M (ed) Conference Proceedings, Annual Conference Series (SIGGRAPH '98), Addison Wesley, pp 169–178
18. Wilhelms J, van Gelder A (1992) Octrees for faster isosurface generation. ACM Trans Graph 11:201–227
19. Wilhelms J, van Gelder A (1994) Multi-dimensional trees for controlled volume rendering and compression. In Kaufmann A, Krueger W (eds) Proceedings of 1994 Symposium on Volume Visualization. Washington, DC, pp 27–34
20. Wilson O, van Gelder A, Wilhems J (1994) Direct volume rendering via 3D textures. Technical Report UCSC-CRL-94-19, University of California, Santa Cruz, Calif

IMMA BOADA is an Assistant Professor of the University of Girona and a PhD student at the Computer Science Department of the Politechnical University of Barcelona. Her research interests includ scientific visualization, volume rendering and surface reconstruction. She received an advanced degree in Computer Science in 1992 from the Autonoma University of Barcelona.

ISABEL NAVAZO is an Associate Professor of the Computer Science Department at the Polytechnical University of Catalonia in Barcelona (UPC) and a Senior Researcher at the Institute of Robotics and Industrial Informatics run jointly by UPC and the Consejo Superior de Investigaciones Científicas (CSIC). She received her engineering degree from the UPC in 1982 and her PhD in Computer Science from the same University in 1986. She is the coordinator of the Graduate Programmes of the Computer Science Department. Her research interest include volume and solid modeling, surface fitting, octree representation, virtual reality, and visibility in complex scenes. She is the Project leader of projects in these areas funded by the Spanish grant-funding state agency (CICYT) and private companies.

ROBERTO SCOPIGNO is a senior research scientist at the Istituto CNUCE of the National Research Council in Pisa, Italy. He is currently engaged in research projects concerned with scientific visualization, volume rendering, web-based graphics, multiresolution data modeling and rendering, 3D scanning, and applications of 3D computer graphics to cultural heritage. He received an advanced degree (Laurea) in Computer Science from the University of Pisa in 1984. He is member of the IEEE, Eurographics, and Siggraph.