Ibero-American Symposium on Computer Graphics - SIACG (2006)
P. Brunet, N. Correia, and G. Baranoski (Editors)

# Efficient focus+context visual exploration of volume datasets

J. Campos [1], A. Puig [1] and D. Tost [2]

[1] Departament de Matemàtica Aplicada i Anàlisi, UB, Barcelona, SPAIN
[2] Centre de Recerca en Enginyeria Biomèdica (CREB), UPC, Barcelona, SPAIN

**Abstract**

*We propose a hierarchical representation of volume data sets based on a user-given definition of features. The model is general and it can be used for various types of feature definitions such as rendering queries, topological skeletons and property-based classification. We use this model to interactively select features of the volume and to render them providing* focus+context *information. In addition, our model provides a mechanism to index directly the feature voxels, therefore it considerably speeds up rendering.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image generation-Display algorithms; I.3.6 [Computer Graphics]: Methodology and techniques - Graphics data structures and data types H.5.2 [Information interfaces and presentation]: User interfaces- Theory and methods

## 1. INTRODUCTION

The recent technological advances in acquisition devices as well as in computing power have yield to an explosion in data production. Today, human capacity of generating data, either by simulation or by measurements, is largely bigger than our ability to understand them. Data visualization is an effective way to get insight in the data and extract useful knowledge from them, because the human visual system constitutes a fundamental part of our intelligence. According to the experts, more than 50% of human brain neurons are devoted to visual processing!

Visualization started as a recognized research and development field in 1987 [MDB87] and since then, it has been evolving in an impressive way, in hardware technology as well as in software solutions. Many visualization methods have been developed that can be used in standard consumer PC's, something almost unimaginable a decade ago. However, in the last two years, leading researchers have expressed their concerns about the future of visualization [vW05]. Data production is still growing faster than graphics hardware development and visualization methods. Therefore, on one hand, improving rendering efficiency to keep interactivity is still a major goal. On the other hand, since classical visualizations may convey too much information to be understandable, new visual idioms must be designed to help users to detect and focus on relevant features without loosing a global perspective on the context. Finally, new interaction metaphors must be investigated to help users manipulating data.

These three goals have been addressed in the bibliography from different perspective. On one hand numerous papers focus on accelerating rendering using software [MHB*00] and hardware optimizations [NM05]. On the other hand, *focus-plus-context* techniques have been proposed that show different parts of the data at different resolutions according to their *degree of interest* (DOI) [Fur86] [DGH03]. Finally, several works address user interaction in visualization, especially, user specification of transfer functions [JKM01] [KKH01].

In this paper, we explore means of reaching these three goals by using structural information of the data. Specifically, we propose to extract a hierarchical model of the inner structure of the data. We use this model in the visualization to provide contextual information and to interactively select specific features to focus on. Moreover, our model provides a mechanism to index directly the feature voxels, therefore it considerably speeds up rendering.

## 2. PREVIOUS WORK

The idea of using structural information in volume rendering is not new. It has been used to ease the specification of rendering parameters [MLK03], [FPT05], to help animation and deformation [NS01] [SSC03] and to speed up rendering computations [HMBG01] [LMK03]. We next survey the previous papers most related to ours. Paying attention to: the type of structure they propose, how they construct them, if they can attach voxels to the structure to visit them separately from the rest of the volume, and which rendering algorithm they support.

A simple idea to isolate features in rendering is to classify the volume into boxes so that only the selected boxes are visited during rendering [PTN00] [LMK03]. The main advantage of this strategy is that the spatial ordering of the voxels inside the boxes is preserved. Puig et al. [PTN00] compute the discrete medial axis transform of the inner surface of blood vessels from magnetic resonance angiographies, and they extract from it a labeled skeletal model of the cerebral vascular. They associate to each segment of the skeleton a volume bounding box. The voxels in the box that do no belong to the segment have a zero value. The boxes overlap in such a way that each voxel may belong to many boxes but it has a non-zero value in only one of them. Any rendering approach can be used to render each feature separately, but rendering simultaneously various features requires a depth sorting of the bounding boxes. Three-dimensional texture slicing could be applied, but it would require a specific treatment of the masks which is not addressed in the paper. Li et al. [LMK03] propose to keep features into disjoint boxes, so their relative order can be preserved using an orthogonal BSP tree. The preservation of separate boxes increases the number of boxes. Therefore, one feature can correspond to many boxes. In the paper, it is unclear how this correspondence is managed and how the boxes are structured. Rendering is done by 3D texture slicing without problem since the boxes do not overlap.

Gagvani and Silver [NS01] extract a topological medial surface skeleton of articulated bodies using the distance transform. The skeleton is deformed by users using standard animation tools. The deformed volume is then reconstructed about the skeleton using the distance field. This method is primarily aimed at deforming the volume. It cannot be used to render separate features, but only the whole reconstructed volume. This method was later enhanced by Singh et al. [SSC03] who designed the VolEdit system. In VolEdit, skeletons are automatically extracted from the volume using a parameter-controlled volume thinning algorithm. The skeleton associates to each skeletal segment a rectangular bounding box enclosing the segment. The interface of the application (VolEdit) allows animators to pick segments, and rotate and scale them. Rendering is done by texture slicing all the bounding boxes. The main advantage of this approach is that it does not require the reconstruction of all the volume. However, at the overlapping between bounding boxes, texture interpolations are unnecessarily repeated. Moreover, some artifacts may appear at the redundant texels if alpha blending is applied in the slices. In a recent paper [WJ06] on non-linear deformation of volumetric datasets the reconstruction stage is done during rendering. The authors use 1D-skeletal curves called *wires* to specify the deformation. They associate to each voxel of the model an index to its corresponding wire and the value of voxel's parameter on the wire. They render the volume using a GPU implementation of ray-casting. For each sample on the ray, they compute a point on the wire indexed by the sample's nearest voxel, they deform this point and they use the property of the nearest voxel to the deformed point to shade the ray sample. The main drawback of this approach is its enormous storage requirement (more than 5 bytes per voxel for only 128 wires).

Mueller et al. [MLK03] describe a rendering system for pre-segmented datasets. Instead of having binary tag volumes, they migrate the density range of the features to private ranges surrounded by a smooth boundary. This strategy presents two advantages: first, it reduces the aliasing problems that may appear at the boundary between two features [TSH98] [VHN*05], next, it makes easier the specification of rendering parameters via transfer functions using the new migrated feature ranges. The authors propose an interface that shows the hierarchical structuring of the density ranges *(feature map)* along with user-defined descriptive labels of the features. The main drawback of this method is that it may require more bits per property value than the original voxel model if there are too much features in the model to find non-overlapping ranges. The authors do not give details on the rendering technique they use. However, their system is conceived to render the volume as a whole, since it doesn't have an indexing scheme for the feature voxels. Therefore, in principle, any algorithm can be used to render it. However, if *view-aligned-splatting* [MMI*98] is used and the voxels are organized in a per-value sorted-list as suggested by Ihm et al. [IL95], the bucket construction stage is efficient, because a binary search of the selected feature range can be applied.

Ferré et al.'s work [FPT05] is inspired in the Information Theory field [GS88]. They propose to construct a *rendering decision tree* that represents the hierarchical process of decision done by users at the specification stage of rendering. They classify the data into categories and construct three run-length encoding of the voxel model according to these categories. The traversal of the rendering decision tree for a specific query gives as output the list of selected categories. The run-length encoding is then traversed, all the codes are checked and only those that correspond to selected categories are processed. This strategy has the overhead cost of the run-length traversal, but it considerably reduces the number of visited voxels. It supports object-aligned splatting and it can be used for the construction of 3D textures and depth-sorted buckets for view-aligned plane splatting.

Finally, the RTVR system [MH01] uses an intermediate array of slice-sorted *RenderLists* for each object that stores the voxels of the slices that are relevant for rendering. The structure must be updated if the selection is modified. It requires the position of the non-empty samples to be stored, via a de-referencing mechanism, so it sacrifices the benefit, in terms of memory requirements, of the implicit spatial arrangement of the voxel model. The *RenderLists* are also used in the *Two-level rendering* proposed by Hauser et al. [HMBG01] [HBH03] that combines object-order traversal and image-order ray-casting. The *RenderLists* are essentially designed for rendering. They are simple features lists and not a true abstract model of the volume structure.

The existing approaches either do not provide a hierarchical structure of the data or they are restricted to a specific type of structure (topological skeleton [SSC03] or decision query [FPT05]). The feature map structure is more general, but it does not provide a direct access to the feature voxels. Feature isolation with bounding boxes [PTN00], [LMK03], [SSC03] reduces the rendering computations but it still requires visiting empty voxels in the boxes and it may give problems if boxes overlap. Our approach is based on a general purpose structuring of the volume and a voxel indexing scheme that does not require bounding boxes.

## 3. THE SEMANTIC LABELS HIERARCHICAL MODEL

### 3.1. Definition

Our approach is based on a hierarchical model of the inner structure of the data. We define a *feature* as a relevant structure of the dataset, this is a structure that users may query for rendering at some time. Let $f_{all}$ be a the feature representing all the relevant structures of the volume. Let $F = \{f_i, i = 1...n\}$ be the set of $n$ semantical features identified in a voxel model $V$ including $f_{all}$ (such that $\exists! i : 1 \leq i \leq n : f_i = f_{all}$). We define $l$ as a labelling function that associates to any voxel $v$ of $V$ and any feature $f_i$ of $F$ a boolean value indicating if $v$ belongs to the feature: $l(v, f_i) = \{v \in f_i\}$. We define the *Set of Relevant Voxels* of $V$ ($SRV(V)$) as set of voxels of $V$ that belong to one or more features of the model: $\forall v : v \in SRV(V) : \{\exists f_i : f_i \in F : l(v, f_i)\}$. The complementary set of non relevant voxels $SNRV(V)$ is defined as: $SNRV(V) = V - SRV(V)$. Finally, we define $SOV(f_i)$ as the *Set Of Voxels* belonging to a feature $f_i$. Thus, $\forall v : v \in SOV(f_i) : l(v, f_i)$. Observe that the $SOVs$ are not disjoint. Moreover, a $SOV$ can be completely included into another, indeed, the $SOV$ of $f_{all}$ includes all the other $SOVs$. This idea is the basis of our hierarchical representation.

Let $\oplus$ be a binary relation between features such that $f_i \oplus f_j$ iff $SOV(f_i) \subset SOV(f_j)$. The set of features $F$ and the relation $\oplus$ define a partially ordered set (*poset*) that can be represented with a Hasse diagram [Ski90]. More precisely, since $f_{all}$ is a supremum of the set, the poset can be considered as a join-semilattice.

The data structure that we derive from these definitions is an acyclic directed graph (digraph) $G$ that represents $SRV(V)$, the set of relevant voxels of $V$. It is composed of $n$ nodes such that:

- Each node $i$ of the digraph corresponds to a unique feature $f_i$ of $F$;
- A directed arc $a(i \longrightarrow j)$ exists between two nodes $i$ and $j$ if $f_j \oplus f_i$, which is same as $SOV(f_j) \subset SOV(f_i)$, and no other node $k$ exists such that $f_k \oplus f_i \wedge f_j \oplus f_k$.
- There is a unique node called $root(G)$ to which no arc arrives.

Therefore, the feature associated to $root(G)$ is $f_{all}$ in $F$ such that $SOV(f_{all}) = SRV(V)$. The $SOV$ of a node of $G$ is included in the $SOVs$ of all the nodes of $G$ having an arc directed to it and, recursively, in the $SOVs$ of the nodes that have directed arc to those. Nodes from which no arc exits are called *leaf nodes*. Finally, the path between two nodes is unique, because an arc between two nodes cannot exist if there is an intermediate node between them.

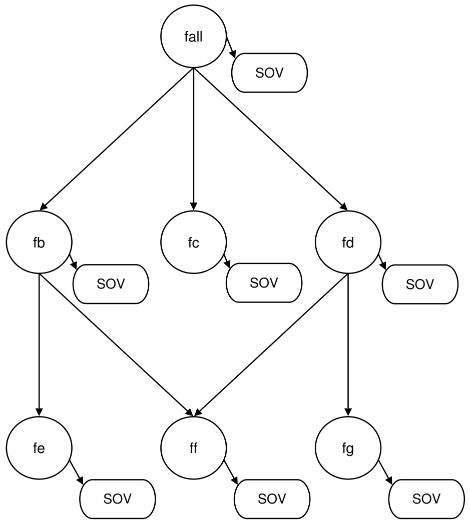


**Figure 1:** *The features digraph.*

Each node of the digraph stores information on:

- the label of the corresponding feature $f_i$
- the $SOV(f_i)$
- *focus+context* information

Figure 1 illustrates the definition of the digraph.

We next describe the representation of the $SOVs$ and of the *focus+context* data.

## 3.2. Indexing mechanism of the *SOVs*

The representation of the *SOVs* associated to the nodes provides access to all its voxels. In order to avoid redundancies we differentiate between leaf and intermediate nodes.

Let the set of voxels that belong to a certain set of features be defined as: $voxels(f_1 \dots f_n) = \{\forall v : v \in SRV(V) : \bigwedge_{i=1}^{n} l(v, f_i)\}$. And let the set of voxels that *uniquely* belong to a certain set of features be defined as: $uniqueVxl(f_1 \dots f_n) = \{\forall v : v \in voxels(f_1 \dots f_n) : \{\forall j : j \neq 1 \dots \neq n : !l(v, f_j)\}\}$.

Accordingly, the $root(G)$ only stores $uniqueVxl(f_{all})$.

Moreover, the intermediate nodes store only the voxels that belong to its associated feature ($f_{node}$) and also belong to the features of all ascendent nodes ($f_{parent}, \dots, f_{all}$), $uniqueVxl(f_{node}, f_{parent}, \dots, f_{all})$. Consequently, intermediate nodes store only the voxels of their *SOV* that do not belong to any descendent node, and provide access to all their other voxels following the descendent nodes. Therefore, in the particular case of features defined as a union of other features, no voxels are stored at intermediate levels.

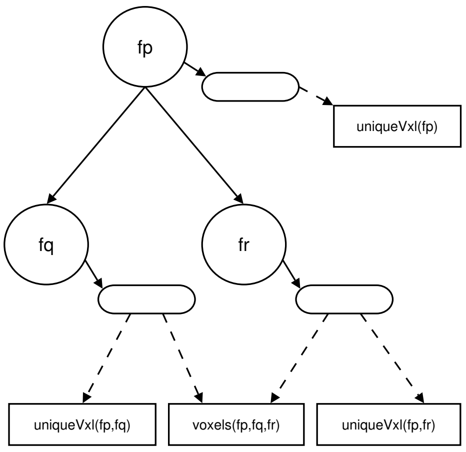


**Figure 2:** *Implementation of SOV to avoid redundancies.*

Finally, the leaf nodes store voxels in different arrays: one for the voxels that belong only to that feature, $uniqueVxl(f_{leaf}, f_{parent}, \dots, f_{all})$ and one for each set of voxels shared with other leaf nodes. In particular, in Figure 2, node $f_q$ points to $uniqueVxl(f_q, f_p)$ and $voxels(f_q, f_p, f_r)$. Thus, the *SOV* of a leaf node is implemented as an array of pointers to these different arrays and therefore the voxels representation in the digraph is unique. Again, we have designed this structure to avoid redundancies between non-disjoint SOVs of leaf nodes.

Another issue related to the presented model is how the *SOVs* are codified. The simplest approach is to store the coordinates of each involved voxel. As it will be shown in the Section 5, this encoding may require a great amount of memory. However, since the *SOVs* can be stored in different files, efficient out-of-core traversal strategies can be applied. Moreover, some compression techniques such as run-length encoding could be applied. This has not been addressed in this work.

## 3.3. Focus+context data

The proposed model provides a general purpose structuring of the volume. It can be used to emphasize more or less given features during rendering. We define the selection of features of $F$ that are considered relevant for rendering at a given instant as $Sel(F)$. Let the degree-of-interest of a feature $doi(f_i), f_i \in Sel(F)$ be a value in a discrete range $R$. The range represents the number of level-of-details that can be used to render the features of the model. We define the set of rendering parameters used to render a feature for a given degree-of-interest as: $vis(f_i, r), f_i \in Sel(F), r \in R$. Finally, for each rendering, users must specify: the selection, the degrees-of-interests of all the features and the visual parameters. We define this specification as a query $Q$ such that $Q = \{\forall f_i : f_i \in Sel(F) : (doi(f_i) \times vis(f_i, doi(f_i)))\}$

The definition of a query allows us to apply *focus-plus-context* techniques, because each feature is rendered at the level-of-detail corresponding to its degree-of-interest. In the particular case of a binary range of *dois*, features with the highest range are the *focus*, rendered at full resolution. Features with the lowest range are the *context*. These feature may be rendered without even accessing to their voxels by using graphical symbolical representations. In order to apply these techniques, each node stores focus and context related data (see Figure 1) such as transfer functions, polygons and lines.

## 3.4. Construction

Given a set of features, the construction of the proposed model involves three different tasks: (1) to define the labelling functions (2) to create the hierarchy and bind the *SOV* to the corresponding nodes, and (3) to fill the focus/context data. The naive approach to create the digraph consists of sequentially creating all the *SOVs*. For each voxel, we check if it has already been inserted in the voxel array of another *SOV*. If it is not the case, the voxel is directly inserted in its *SOV*. Otherwise, the voxel is shared by two or more *SOV*s. In this case, we get the list of *SOVs* to which the voxel belongs and eventually we have to split some of them and recompute the indexes to voxel arrays. Once all the *SOVs* and voxel arrays are computed, the hierarchy is created according to the definition of the digraph. A node is created for every *SOV* and directed arcs are created between nodes such that the *SOV* of one node totally includes the *SOV* of the other.

In practice, this brute force strategy may be simplified depending on the set of features. In Section 5, we show two different applications of our structure. In the first example, we define the features as the segments of a simplified topological skeleton. In this case, the labelling functions are based on the distance map of each skeleton segment, the hierarchy is automatically extracted from the skeleton and the focus+context information contains a wire-frame representation of the skeleton automatically computed from it. In the second example, we define the features as anatomical regions of a human brain. The labelling functions are defined after the voxel model has been classified, a fairly complex process, assisted by a physician and that uses property values, voxel positions and an auxiliary atlas model. The output of the classification is a labeled voxel model such that each voxel belongs to a unique class, plus a tree that indicates relationships of inclusion between classes. This tree is used to define our hierarchy. Finally, the *focus+context* data are wire-frame representations of edges between the features bounding boxes.

## 4. RENDERING

The visualization stage implies three steps. First, the query is defined through an interactive widget which shows the context data stored in the digraph $G$ (see Color Plate 1.b). Second, the model is traversed in order to compute the selected voxel set to be rendered, $SOV_{sel} = \{SOV(s), \forall s \in Sel(F)\}$. Finally, all the $SOVs$ of $SOV_{sel}$ are rendered according to their visual parameters and degree of interest.

It should be noted that voxels may be sent to the rendering pipeline as many times as selected nodes to which they belong. The user-defined rendering preferences determine how these shared voxels should be rendered: if one of the features to which the voxels belong is preferred or if a fusion of optical properties should be realized. Since shared voxels are stored in separate arrays indexed by the $SOVs$, it is not necessary to filter these special cases at each voxel. When a $SOV$ is traversed, the rendering preferences are set once, at the beginning of each separate array indexed by the $SOV$. Moreover, when no fusion of optical properties is applied at the shared voxels, once a shared array has been visited, it can be marked as so, in order to avoid revisiting it through another of its $SOVs$. In the particular case of a digraph constructed on the basis of a classification of the voxel values, voxels at the boundary between two features are assigned to the two corresponding $SOVs$, which allows us to apply smoothing boundary strategies during rendering.

Figure 3 summarizes this procedure. A traversal through the digraph $G$ is performed to obtain all the nodes $f_i$ of the selection $Sel(F)$, by using an iterator. Depending on $doi(f_i)$ of the node, $SOV(f_i)$ may be traversed or not. The traversal is realized if $vis(f_i, doi(f_i))$ requires accessing to the voxels. Otherwise the node is rendered according to the context data. In the former case, for each voxel array of the $SOV$ we check if it is necessary or not to visit it. If so, we traverse all the voxels to render them according to $vis(f_i, doi(f_i))$.

It should be observed that our traversal does not preserve the order of the original voxel model, so it cannot be used for object-aligned splatting. Thus, a standard GL-based zbuffer approach is followed to keep the correct depth order in the visualization (see Figure 3). It must be noted that on the traversal of the digraph $G$, the context data of all nodes can be rendered, without increasing too much the computational cost of the z-buffer process, since accessing to the context data does not necessarily require traversing the voxels.

```
selNodes = digraph->getSelectedNodes(query);

foreach node in selNode
{
  doi  = query->getDoi(node->label);
  vis  = query->getVis(node->label, doi);

  data = node->getFocusContextData(doi);

  if ( doi->visitVoxels() )
  {
    foreach pArray in node->getSOV()
    {
      if ( not pArray->alreadyVisited() )
      {
        foreach voxel in pArray->getVoxels()
        {
          renderVoxel(voxel, doi, vis, data);
        }
        pArray->updateVisited();
      }
    }
  } else {
    renderNode(node, doi, vis, data);
  }
}
```

**Figure 3:** *Zbuffer traversal of the digraph G.*

The main drawback of the z-buffer approach is that it requires a depth pre-ordering in order to keep correctly the translucently. Alternatively, an image-aligned sheet-buffered splatting is applied, that eliminates this drawback by processing the selected voxels with slabs, or sheet-buckets, aligned parallel to the image plane. Then, at the rendering stage, the digraph $G$ is traversed to insert the voxels of the $SOV_{focus}$ into the buckets. The traversal algorithm is very similar to the one shown in Figure 3. Finally, after the traversal the composition of all sheets in FTB order is performed.

## 5. RESULTS

We have implemented and tested the different algorithms in our software platform *Hipo* with two datasets: *Teddy bear* and *MR-brain*. In both cases, the ranges of the $doi()$ are binary: $[focus, context]$. Moreover, the user queries always include all the available features, classified as focus or context.

The resulting image are obtained with two rendering algorithms: z-buffer and view-aligned-splatting. In all the cases, the focus is rendered using different transfer functions for each feature, and the context is rendered as wireframes.

The first dataset is based on a simplified topological skeleton of a Teddy bear (see Color Plate 1). The original dataset has been labeled with features corresponding to a simplified version of its topological skeleton. This correspondence has been computed automatically from the proximity of each voxel to the skeleton segments plus some user adjustments to profile the joints. Afterwards, the digraph representing the features has been extracted from the labeled model and the *SOVs* of each feature have been computed. The skeleton joints are represented by nodes without *SOV* and they are arcs with all the nodes representing connected skeleton segments. The direction of the arcs is determined by the inverse kinematics (IK) structure. The focus and context data stored at these nodes contain the coordinates of the joints. The skeleton segments are nodes with *SOV* and they store the transfer functions in the focus and context data. Finally, all the model has been presented to the user as shown in Color Plate 1.a. In Color Plate 1.b we show the *DOI widget* offered to the user, where some of the segments are selected as focus (in yellow). Color Plate 1.c shows the rendering of all the features according to their $doi()$.

The second dataset is based on pre-classified anatomical structures of a MRI brain (see Color Plate 2). The original dataset was already labeled with features corresponding to 35 regions. Thus, the digraph representing the features and their hierarchy has been automatically extracted as indicated in Section 3.4. Like the first dataset, the focus and context data of each node contained the different transfer functions to apply to each region. In addition, these data also contained the coordinates of the center of each spatial region. Thus, it was possible to graphically represent the digraph of features related to anatomical regions. Finally, the steps shown in the Color Plate 2 are the same as the ones realized with Teddy bear dataset: Color Plate 2.a shows the full model using the different transfer functions, Color Plate 2.b shows the DOI widget and Color Plate 1.c shows the rending of all features according to their $doi()$.

Tables 1 and 2 show the times needed to perform different renderings of the Teddy bear and the MRI brain datasets. All tests were run on an AMD Athlon(tm) 64 Processor 3200+ with 1MB of RAM and a NVidia GeForce 6600 PCI-E 256MB, inside a window with a viewport of 775x720 pixels. The columns express the percentage of selected voxels, the number of projected voxels (*#voxels*), the cost in time of filling the splatting buckets (*S.Buckets*), the cost of splatting the buckets' voxels (*S.Render*) and the rendering cost using a z-buffer algorithm (*ZBuffer*). The times into brackets in the rows indicate the time consumed to realize the same rendering operation without using the digraph, this is verifying for each voxel if is selected or not. In table 1, there are two sections in the rows, one for a low resolution dataset of the Teddy bear ($128^3$), and the other for the full resolution dataset ($512^3$). The MRI brain dataset has a geometry of 190x220x178 voxels.

| Bear | #voxels | S.Buckets | S.Render | ZBuffer |
|---|---|---|---|---|
| SMALL DATASET $128^3$ | | | | |
| **100%** | 357,633 | 0.15 [0.21] | 0.13 | 0.26 0.36 |
| **80%** | 324,124 | 0.13 [0.21] | 0.12 | 0.24 0.33 |
| **60%** | 288,900 | 0.12 [0.19] | 0.11 | 0.21 0.32 |
| **40%** | 195,720 | 0.08 [0.16] | 0.10 | 0.14 0.24 |
| LARGE DATASET $512^3$ | | | | |
| **100%** | 24,842,270 | 10.2 [15.22] | 133.94 | 18.27 23.57 |
| **80%** | 20,899,916 | 8.82 [14.49] | 103.93 | 15.19 21.93 |
| **60%** | 14,925,731 | 6.11 [13.54] | 75.42 | 10.83 19.97 |
| **40%** | 9,916,539 | 4.05 [11.86] | 52.2 | 7.23 15.58 |

**Table 1:** *Times in seconds to render a simplified topological skeleton dataset (Teddy bear): 100% means all the non-empty voxels, 80% means non-empty voxels without the right leg, 60% means non-empty voxels without both legs and 40% means non-empty voxels without the body.*

| Brain | #voxels | S.Buckets | S.Render | ZBuffer |
|---|---|---|---|---|
| **100%** | 1,366,905 | 0.56 [0.97] | 0.66 | 0.83 1.30 |
| **50%** | 700,395 | 0.29 [0.62] | 0.62 | 0.52 0.89 |
| **10%** | 114,033 | 0.05 [0.61] | 0.58 | 0.49 0.86 |

**Table 2:** *Times in seconds to render a pre-classified anatomical structures dataset (a MRI brain): 100% means all the non-empty voxels of the brain, 50% means non-empty voxels of the right brain plus cerebellum, 10% means non-empty voxels of the left cerebral exterior and right cerebellum structures.*

In summary the results show that the model allows better rendering performance: (1) for large datasets, observe how the ratio of *S.Render* using the digraph vs. not using it increases in all cases from the small to the large Teddy bear datasets; (2) for the small focus renderings, observe how the same ratio also increases in each dataset as the percentage of selected voxels decreases.

## 6. CONCLUSIONS

We have designed and implemented an abstract hierarchical model that provides a symbolical representation of various aspects of the inner structure of the datasets such as topology and functional distribution. Our model allows users to interactively select focus features of the data for rendering while preserving context information through the visualization of the skeleton. Moreover, our voxel indexing scheme provides means of efficiently access to the voxels of a selected feature, reducing the computational cost of rendering.

This work continues in several directions. First, in our current implementation, we have only tested two degrees of interest. We plan to add more degrees and to investigate means of smoothing the transition between different levels of detail in a same rendering. In addition, we want to explore the use of our model in volume animation applications, when the digraph represents the topological skeleton of the data. Given a user-defined geometrical transformation to apply to a part of the model, the corresponding *SOVs* and voxel arrays must be recomputed and the digraph arcs updated. We are currently analyzing how to perform this task efficiently. The other application in which we plan to continue working is the interactive classification of the datasets. Users define the set of materials that are inside the model; a first automatic classification is performed; the digraph is computed and rendered. Users can interactively refine this classification playing with the focus+context parameters and specifically with the way shared voxels are combined during rendering. Depending on the combination preferred, a voxel can be removed from a *SOV* and the digraph arcs updated. Our aim is to explore the usability and effectiveness of this classification.

## 7. ACKNOWLEDGEMENTS

This work has been funded by the project MAT2005-07244-C03-03 from the Ministerio de Educación y Ciencia and the spanish research network IM3:Imagen molecular y multimodalidad from the Instituto de Salud Carlos III.

### References

[DGH03] DOLEISCH H., GASSER M., HAUSER H.: Interactive feature specification for focus+context visualization of complex simulation data. In *Proceedings of the* $5^{th}$ *Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2003)* (2003), pp. 239–248.

[FPT05] FERRÉ M., PUIG A., TOST D.: Decision trees for accelerating unimodal, hybrid and multimodal rendering models. *The Visual Computer* (2005), 305–313.

[Fur86] FURNAS G. W.: Generalized fisheye views. In *CHI86: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 1986), ACM Press, pp. 16–23.

[GS88] GOODMAN R., SMYTH P.: Decision tree design from a communication theory standpoint. *IEEE Trans. on Information Theory 34*, 5 (1988), 979–994.

[HBH03] HADWIGER M., BERGER C., HAUSER H.: High-quality two-level volume rendering of segmented data sets on consumer graphics Hardware. In *IEEE Visualization '03* (2003), IEEE Computer Society Press, pp. 40–45.

[HMBG01] HAUSER H., MROZ L., BISCHI G., GRÖLLER M.: Two-level volume rendering. *IEEE Trans. on Visualization and Computer Graphics 7*, 3 (2001), 242–252.

[IL95] IHM I., LEE R.: On enhancing the speed of splatting with indexing. In *IEEE Visualization '95* (1995), IEEE Computer Society Press, pp. 69–76.

[JKM01] JANKUN-KELLY T., MA K.-L.: A study of transfer functions generation for time-varying volume data. In *Proceedings of the Joint IEEE TCVG and Eurographics Workshop on Volume Graphics 2001* (2001), Mueller K., Kaufman A., (Eds.), IEEE TCVG and Eurographics, Springer-Verlag, pp. 51–68.

[KKH01] KNISS J., KINDLMANN G., HANSEN C.: Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the conference on Visualization 2001* (2001), IEEE Press., pp. 255–262.

[LMK03] LI W., MUELLER K., KAUFMAN A.: Empty space skipping and occlusion clipping for texture based volume rendering. In *IEEE Visualization 2003* (2003), pp. 317–324.

[MDB87] MCCORMICK B., DEFANTI T., BROWN M.: Visualization in scientific computing. *ACM Computer Graphics 21*, 6 (1987).

[MH01] MROZ L., HAUSER H.: RTVR: a flexible *j*ava library for interactive volume rendering. In *IEEE Visualization'01* (2001), IEEE Computer Society Press, pp. 279–286.

[MHB*00] MEISSNER M., HUANG J., BARTZ D., MUELLER K., CRAWFIS R.: A practical evaluation of popular volume rendering algorithms. In *IEEE Visualization'2000* (2000), pp. 81–91.

[MLK03] MUELLER K., LAKARE S., KAUFMAN A.: Volume exploration made easy using feature maps. In *Workshop on Scientific Visualization* (2003).

[MMI*98] MUELLER K., MÖLLER T., II J. E. S., CRAWFIS R., SHAREEF N., YAGEL R.: Splatting errors and antialiasing. *IEEE Trans. on Visualization and Computer Graphics 4*, 2 (1998), 178–191.

[NM05] NEOPHYTOU N., MUELLER K.: GPU accelerated image aligned splatting. In *Volume Graphics* (2005), Fujishiro I., Gröller E., (Eds.), pp. 197–205.

[NS01] N.Gagvani, Silver D.: Animating volumetric models. *Graph. Models 63*, 6 (2001), 443–458.

[PTN00] Puig A., Tost D., Navazo M.: *A hybrid model for vascular tree structures.* Springer Verlag Computer Science, Eds. W. de Leeuv, R. Van Liere, 2000, ch. Data Visualization, pp. 125–135.

[Ski90] Skiena S.: *Implementing Discrete Mathematics: Combinatorics and Graph Theory With Mathematica.* Addison-Wesley, 1990.

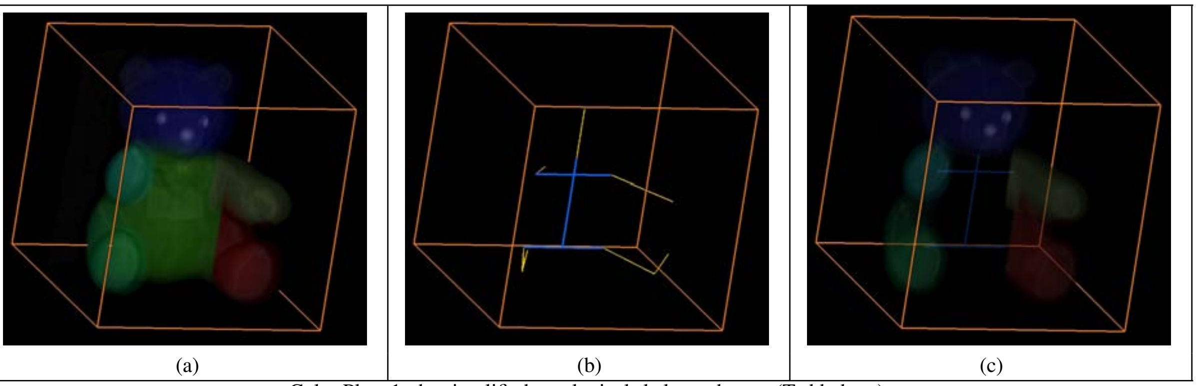
[SSC03] Singh V., Silver D., Cornea N.: Real-time volume manipulation. In *Volume Graphics* (2003), pp. 45–52.

[TSH98] Tiede U., Schiemann T., Hohne K.: High quality rendering of attributed volume data. *Proc. IEEE Visualization* (1998), 255–262.

[VHN*05] Vega F., Hastreiter P., Naraghi R., Fahlbusch R., Greiner G.: Smooth volume rendering of labeled medical data on consumer graphics hardware. In *Proceedings of SPIE Medical Imaging 2005* (2005), pp. 13–21.

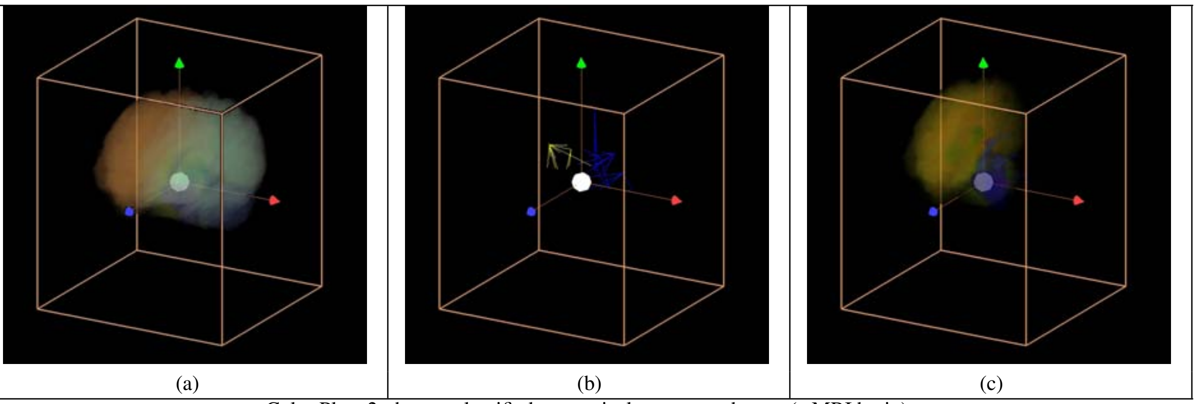
[vW05] van Wijk J.: The value of visualization. In *IEEE Visualization'05* (2005), IEEE Press, pp. 76–86.

[WJ06] Walton S., Jones M.: Volume wires: a framework for empirical non-linear deformation of volumetric datasets. *The Journal of WSCG 14* (2006), 81–88.



(a) (b) (c)

Color Plate 1: the simplified topological skeleton dataset (Teddy bear).



(a) (b) (c)

Color Plate 2: the pre-classified anatomical structures dataset (a MRI brain).