# Ray-Casting Time-varying Volume Data Sets with Frame-to-frame coherence

D. Tost*
Computer Graphics Division.
CREBEC
UPC, Spain

S. Grau†
Computer Graphics Division.
CREBEC
UPC, Spain

M. Ferre ‡
Mathematics and Computer
Science Department
URV, Spain

A. Puig §
Mathematics Department
UB, Spain

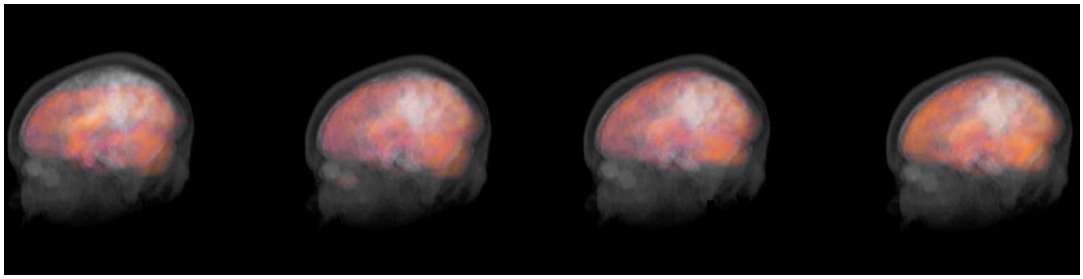

Figure 1: Four sample frames of the animations of the MR-SPECT dataset.

**ABSTRACT**

The goal of this paper is the proposal and evaluation of a ray-casting strategy that takes advantage of the spatial and temporal coherence in image-space as well as in object-space in order to speed up rendering. It is based on a double structure: in image-space, a temporal buffer that stores for each pixel the next instant of time in which the pixel must be recomputed, and in object-space a Temporal Run-Length Encoding of the voxel values through time. The algorithm skips empty and unchanged pixels through three different space-leaping strategies. It can compute the images sequentially in time or generate them simultaneously in batch. In addition, it can handle simultaneously several data modalities. Finally, an on-purpose out-of-core strategy is used to handle large datasets. The tests performed on two medical datasets and various phantom datasets show that the proposed strategy significantly speeds-up rendering.

**CR Categories:** I.3.3 [COMPUTER GRAPHICS]: Picture/Image Generation, Viewing Algorithms—; I.3.7 [COMPUTER GRAPHICS]: Three-Dimensional Graphics and Realism - Visible line/surface algorithms, Ray Tracing—; I.4.8 [IMAGE PROCESSING AND COMPUTER VISION]: Time-varying imagery—;

**Keywords:** Time-varying volume datasets. Direct volume rendering. Ray-casting. Run-Length Encoding. Temporal coherence.

## 1 INTRODUCTION

The fast rendering of volume data that evolve along time is today one of the major challenges of Visualization [18]. This work focuses on biomedical applications in which we have identified two major uses of rendering time-varying datasets: (i) animation of functional data, such as SPECT, PET and fMR which are inherently dynamic and, (ii) analysis of the evolution of pathologies, such as bone fractures or tumors, captured with CT, MR or MRA over a larger period of time. In the former case, the capture of the temporal sequence of data is generally realized in the same session and it guarantees the alignment of the data. The number of frames may vary between 2 to 30 frames and the changes from frame to frame can be important and spread through the volume. In the latter case, there are typically fewer datasets. The data are not aligned from frame to frame and the changes are localized in the specific region of interest under study. In both cases, rendering data along time provides useful information for the exploration. However, because of the large size of the datasets, the animation is slow. Moreover, many medical studies require the analysis of several modalities, but the involved amount of data is a serious drawback for a multimodal animation.

Different strategies have been proposed in the bibliography for time-varying rendering. We herein address *Direct Volume Rendering*. Thus, existing work on extracting isosurfaces from temporal sequences of volume data for *Indirect Volume Rendering* ([28],[24],[26], [3]) fall out of the scope of this paper. Direct volume animation strategies can be classified into two main categories: one that treats the time-varying data as a special case of an n-D model ([2], [5], [21], [32]), and the other, specifically designed for animation, that treats separately the time dimension from the spatial dimensions ([33], [25], [24] [1], [29], [15]). Both strategies produce a sequence of frames corresponding to the volume at different instants of time. An alternative to this approach is the *Chronovolumes* method described in [31]. It consists of integrating the volume through time and rendering it with a conventional ray-casting. The resulting frame represents the evolution of the volume through time.

The medical applications described above require a frame-to-frame analysis of the data rather than an integration over time. Moreover, in these animations, the number of frames is small in comparison to the number of samples of the datasets. Because of this asymmetry, a 4-D representation is not convenient. Therefore, the approach followed in this paper belongs to the methods that separate time from space. Our goal is to provide a strategy to reduce the computational cost of animation for the biomedical unimodal

*e-mail:dani@lsi.upc.edu
†e-mail:sgrau@lsi.upc.edu
‡e-mail:maria.ferre@urv.net
§e-mail:anna@maia.ub.es

and multimodal applications mentioned above in conventional PC architectures. Therefore, previous work on high performance animations using parallel supercomputers [16] fall outside of the scope of this paper. Previous approaches can be further classified into two sub-categories: methods that compress and decompress the volume for animation [29], [9] and methods that adapt static direct volume rendering strategies to time-resolved sequences. Our approach belongs to this last group, and therefore, we review its papers with more depth in Section 2.

The main contributions of this paper are: (i) the analysis of the suitability of a Run-Length Encoding (RLE) of the voxels values along time (Temporal RLE) instead of in space, (ii) the use of this encoding to predict changes in the voxel values from frame to frame, (iii) the proposal of a frame-to-frame coherent ray casting based on this predictive capability that allows to only cast modified rays, and (iv) the proposal and comparation of three different time and space leaping methods along the recomputed rays.

## 2 RELATED WORK

The idea of taking profit of frame-to-frame coherence to speed-up rendering in time-varying scenes was early introduced by Hubshman and Zucker [12]. Since then, many frame-to-frame techniques have been proposed for non-volumetric scenes in the context of visibility culling [4], [10] and global illumination [19],[11]. As mentioned above, temporal coherence has also been exploited for volume scenes in the extraction of isosurfaces from time-varying volumes ([28],[24],[26], [3], as well as for *Direct Volume Rendering* ([33], [25],[17], [2], [5], [21], [32], [24], [1], [27],[14]). We next survey the last group of papers, which are the most related to our work.

Yagel and Shi [33] propose a frame-to-frame coherent ray-casting that stores in a *C-Buffer* the coordinates of the first non-transparent voxel encountered by the ray emitted at each pixel. If the light conditions or the transfer function changes in successive frames, the ray sampling can start at this location and skip the previous empty voxels. Moreover, the *C-Buffer* can be re-used if the model rotates by reprojecting the intersection points. Actually, Wan et al. [27] found that the original point-based reprojection method can create artificial hole pixels, that can be corrected using a cell-reprojection scheme. Both approaches speed up ray casting computations when the camera or the transfer function change. However, when the property varies inside the voxels and the empty voxels change along time, the *C-Buffer* must be recomputed. The reprojection technique has also been used to track points across frames in order to reduce temporal aliasing [20].

Shen and Johnson [25] focuses on exploiting ray coherence when the property values inside the voxels change along time and the camera remains static. Given the initial data sets, this method constructs a voxel model for the first frame and a set of incremental models for the successive frames, composed of the coordinates of the modified voxels and their values. The first frame is computed from scratch. The next frames are computed by determining which pixels are affected by the modified voxels of the corresponding incremental file, updating the voxel model and recasting only the modified rays. This strategy produces a significant speed-up of the animation if the incremental files are small, i.e., the number of modified voxels is low. However, the incremental files do not keep the spatial ordering of the voxel models. Therefore, the method is not suitable to visualize sub-models or specific features in a model. In a recent paper, Liao at al. [14] propose an improvement of this technique consisting of computing an additional differential file, called SOD (Second Order Differential file) that stores the changed pixels positions. At each frame, the rays are either computed following Shen at al.'s strategy [25] or using the SOD, i.e. accessing directly to the modified pixels, avoiding the cost of projection of the modified voxels.

Reinhard et al. [23] address the I/O bottleneck of time-varying fields in the context of ray-casting isosurfaces. They propose to partition each time step into a number of small files containing a small range of iso-values. They use a multiprocessor architecture such that, during rendering, while one processor reads the next step time, the other ones render the data currently in memory. Their results show that partitioning data is an effective out-of-core solution.

Ma et al. [17] explore the use of a BON (Branch-On-Need) octree [30] for time-varying data. The construction of the tree consists of three steps: quantization of the volume, construction of a BON for every instant of time and merging of the subtrees that are identical in successive BONs. This data structure is rendered with ray-casting by processing the first BON completely, and only the modified subtrees of the following BONs. To do so, an auxiliary octree, called the *compositing tree*, is constructed, similar to the BON, that stores at each node the partial image corresponding to the subtree. At successive frames, when a subtree changes, its sub-image is recomputed and composited at its parent level in the hierarchy. The TSP *Temporal Space Tree* [24] is a spatial octree that stores at each node a binary tree that represents the evolution of the subtree through time. The TSP tree can also store partial sub-images to accelerate ray-casting rendering. It has also been used to speed-up texture-based rendering [5]. This structure is particularly suitable for datasets in which most of the volume remains almost static and only specific regions vary through time. Otherwise, the tree can be highly subdivided and only few partial images can be re-used. The extension of these octree-based methods to multimodality would require the datasets to be aligned. In addition, the sub-images would hardly be re-usable, since the fusion of different modalities must be done at the sample level and not at sub-images in order to preserve correct depth integration.

The shear-warp technique proposed by Anagnostou et al. [1] uses an incremental Run-Length Encoding (RLE) of the volume. Whenever a change is detected over time, the RLE is updated by properly inserting the modified runs in the volume scan-line. In addition, the volume is processed by slabs, recomputing only the modified slabs and compositing them with the unchanged slabs.

Finally, Lum et al's approach [15] is based on hardware assisted texture mapping. The time-varying volume over a given span of time is compressed using the Discrete Cosine Transform (DCT). Every sample within the span is encoded as a single index. The volume is represented as a set of 2D paletted textures. The textures are decoded using a time-varying palette. In order to keep a constant frame rate, the texture slices re-encoding at the end of each time-span is interleaved. A parallel implementation is also described.

The strategy proposed in this paper is based on ray-casting. We have chosen this rendering method because it can be easily adapted to non-aligned multimodal datasets. Our goal is to exploit spatial and temporal coherence in image-space as well as in object-space in order to speed rendering. Spatial coherence in image-space allows us to skip rays that do not intersect the volume of interest, and in object-space to avoid samples that are not relevant for rendering. Similarly, we use temporal coherence in image-space to skip pixels that haven't changed since the previous frame, and in object-space to skip voxels that have kept the same value.

Our approach shares the idea of Shen and Johnsons [25] and Liao at al [14] of re-casting only modified rays when the camera is static. By opposite to their approach, we use a global representation of the voxel model instead of an incremental model over time. This avoids updating the model at each frame and it allows us to do space leaping along the rays. In addition, we use an image-space buffer that avoids computing the modified pixels given the set of modified voxels. This buffer is somewhat similar to the *C-Buffer* proposed by Yagel and Shi [33] and [27], but instead of storing the positions

of the first non-empty voxels, it stores a fitted sampling interval and the next instant in time at which the pixel should be recomputed. Thus, our approach is valid when the property value changes in the model, and the position of the first empty voxel along a ray varies along time. Nevertheless, our approach still allows changing the viewpoint following the reprojection technique proposed in these papers. In comparison to the 4D octree proposed in [17] and the TSP tree described in [24], we use an object-space Temporal Run-Length Encoded (TRLE) model. However, our Run-Length Encoding differs from the classical RLE used in static volumes for shear-warp factorization [13] and from the incremental RLE proposed in [1] for time-varying data. Instead of encoding voxels in space, our TRLE encodes the voxels through time. For each voxel, it stores a set of codes composed of the value of the voxel and the number of frames in which this value remains constant. Finally, as Reinhart at al. [23], we partition data in order to reduce the I/O bottleneck. Instead of partitioning for instant of time, though, we subdivide data both in space and time. Our approach emphasizes the use of spatial and temporal coherence and it is suitable for multimodal studies.

## 3 Overview

Volume datasets are composed of empty space and relevant structures. Usually, during data exploration, not all the structures are rendered simultaneously. In multimodal rendering specially, users rarely choose to select all the non-empty voxels of all the modalities, because it is too much visual information to process. On the contrary, they typically prefer to perform various visualizations selecting different combinations of property ranges [7]. In Section 7, we show two examples of this type of selection: the animation of high SPECT values merged with MR brain and the animation of any SPECT value in the left brain hemisphere. Skipping not only empty but also non-selected voxels can drive to substantial computational time savings for static data and moreover for time-varying ones. From now on, we will call *selected samples*, the ray samples that are inside a selected region.

The goal of this paper is to propose a frame-to-frame coherent strategy such that, if the camera is static during animation and the sampling rate is constant, at each frame, only the rays whose contribution to the image has changed in relation to the previous frame are computed. Moreover, empty and non-selected samples are skipped along these rays. If the camera changes through animation, the reprojection technique proposed in [33] and improved in [27] can still be used in order to skip empty space.

The basic idea is illustrated in Figure 2. A ray through a volume dataset is depicted at four instants of time $T_1$ to $T_4$ for a static camera. The values of the traversed voxels at each instant of time are written inside them as integer values. For simplicity, in the figure, the ray is axis-aligned and sampled at regular intervals of one sample per voxel, but our algorithm supports any ray orientation and sampling rate. We consider three types of voxels: (i) those, coloured in white in the figure, that belong to an empty region, (ii) those in blue, that are non-empty and non-selected and (iii) those, magenta, that are selected. Let suppose that for every sample at $T_1$ we know the next instant of time at which the sample value changes. In the figure, considering for clarity that the property value is constant inside each voxel, the next instant of change of the first sample of the ray is $T_3$. It is $T_2$ for the second sample, $T_3$ for the third and so on. Then, when casting the ray at $T_1$, it is possible to predict how long will the contribution of the ray to the image remain constant. For the ray, the pixel intensity computed at $T_1$ will remain constant at $T_2$, since changes in the selected (magenta) voxels occur after the maximum opacity is reached, at sample 4. Moreover, when the ray is cast at $T_3$, ray integration does not need to start at the first voxel, but rather at the first selected voxel.

In general, let $r$ be a ray cast through the volume, and let $s_1$, $s_2$, ..., $s_k$ be the samples computed Front-To-Back (FTB) along $r$ until the maximum opacity is reached. Let $tcur$ be the current instant of time. Only a subset of the samples $s_i$ is rendered at $tcur$, those which belong to an non-empty selected region. The sample values change along time, but the only changes that are relevant for rendering are those that affect the selected region, i.e. samples that remain or enter in the selected region or at the contrary, become non-selected. Let $tnext(s_i, tcur)$ be the next instant of relevant changes of sample $s_i$ from instant $tcur$. Let $I(r, tcur)$ be the contribution of ray $r$ to the image intensity at $tcur$. We define $tnext(r, tcur)$ as the next instant at which the contribution of ray $r$ to the image changes:

$\forall t: tcur \leq t < tnext(r, tcur) : I(r, t) = I(r, tcur)$.

It is straightforward that $tnext(r, tcur)$ is the minimum of the $tnext(s_i, tcur), i = 1..k$. Therefore, at every instant $t$, the only the rays that effectively change, i.e. those such that $tnext(r, tcur) = t$.

In order to implement this idea, we encode the time-varying volume as a Temporal Run-Length. This encoding allows us to quickly compute at any time the next instant of change of any voxel. The algorithm computes the frames incrementally skipping the pixels that don't change and casting the modified rays only. We have developed three different strategies for the computation of the sampling interval along rays in order to skip as much as possible non-relevant

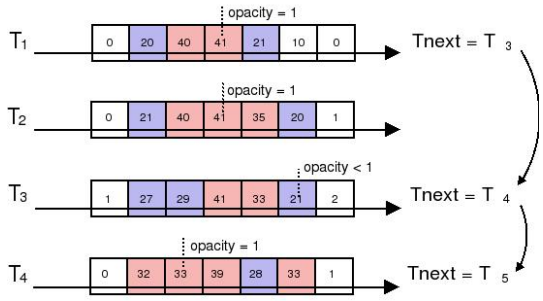


Figure 2: Time-leaping: the same ray at four consecutive frames ($T_1$ to $T_4$) is shown. The value of the voxel is depicted as an integer. The data values are classified into three sets: empty voxels in white, a non selected region in blue and a selected region in magenta. Changes in the white and blue regions are not taken into account to compute the next instant of time. Changes in the magenta region (selected voxels) between $T_1$ and $T_2$ occur after the maximum opacity is reached. Therefore, tnext($T_1$) is set to $T_3$. Instead, tnext($T_3$) $= T_4$ because changes in voxels occur between these two frames before reaching the maximum opacity.

voxels. In addition, we have designed an out-of-core strategy to split the datasets into smaller sets that allows us to handle large datasets.

## 4 DATA STRUCTURES

The *T-Buffer* is a buffer of the same size as the rendered image, that stores for every pixel the next frame at which the corresponding color of the image buffer is expected to change. It is computed from scratch at the first frame and updated at successive frames only at the pixels whose value in the T-Buffer coincides with the current frame.

The *Temporal Run-Length* (TRL) representation of the different modalities stores for every voxel $v_i$ a sequence of codes composed of the voxel value and the next frame at which this value changes: $codes(v_i) = < value_k, tnext_k >, k = 1...ncodes(v_i)$. For each modality, a different TRL is computed.

The query for the value of a voxel $v_i$ at a given frame $t$ requires, with this structure, a search in $codes(v_i)$ of the code whose time span contains frame $t$. In order to avoid this search and access directly to the searched code, we add to the structure a pointer that is set to the first code at the beginning of the sequence and that is updated to the current code during the traversal. Therefore, assuming that a simple byte is sufficient to store the number of frames of the codes and that the pointer to the current code is also a byte, the occupancy in bytes of the TRL structure for a modality $m$ occupying $nb_m$ bytes per voxel of a voxel model composed of $nv_m$ voxels is:
$Occup(TRL_m) = \sum_{i=1}^{nv_m}(1 + ncodes(v_i) * (nb_m + 1))$

This occupancy can be compared to the occupancy of the regular voxel model along time:
$Occup(VoxelModel_m) = nv_m * nf_m * nb_m$, being $nf_m$ the number of frames of modality $m$.

We call $r_{occ}$ the ratio between the occupancy of the TRL and the regular voxel model:
$r_{occ} = \frac{Occup(TRL)}{Occup(VoxelModel)}$.

This ratio has a direct relationship with the temporal coherency of the voxel model, since it depends on the number of voxels that change. As it is obvious, the TRL cannot be constructed on static models, composed of one frame, because it would triplicate their occupancy. In the worst case, for an animated model all the voxels change at every frame and, thus, the ratio of occupancy is: $r_{occ} = 1 + \frac{nb_m + nf_m + 1}{nb_m * nf_m}$, more than 2 when the bytes per property of modality $m$ is $nb_m = 1$. However, this is less than the worst case of the incremental model proposed in [25] which can be four times more the original one.

Nevertheless, as it will be shown in the tests, if the temporal coherency is high, this ratio can be very small (less than 0.13 in the *rabbit* dataset). In these cases, the TRL is a compressed representation of the temporal evolution of the model.

The TRL is computed in a pre-process that first loads the voxel model corresponding to the first frame and initializes the list of codes for every voxel. Next, the voxel models at the following frames are loaded one-by-one and traversed. For every voxel, the value of the current code in the TRL is compared to the value of the loaded voxel model. If the two values are equal, the frame of the current code is updated, otherwise a new code is constructed. Variations of the property values of empty voxels are not considered for the creation of new codes. Therefore, if a voxel has a variable value, but empty through all the sequence, it has a unique code. This preprocessing has a cost complexity $Cost_{PP} = O(nv_m * nf_m)$.

## 5 THE ALGORITHMS

We have designed two algorithms based on these data structures: the emphframe-to-frame coherent one, which computes one image after the other, and the *simultaneous* one that computes all the images at the same time.

### 5.1 Frame-to-frame coherent algorithm

This algorithm proceeds as follows. First, it sets the pointer to the current code of every voxel to the first code of the voxel. Next, for every frame, it traverses the *T-Buffer* and casts rays only at the pixels whose tnext value in the *T-Buffer* is equal to the current frame. Spatial coherency is exploited with three different space-leaping strategies whose efficiency is tested in Section 7. The three strategies are illustrated in Figure 3.

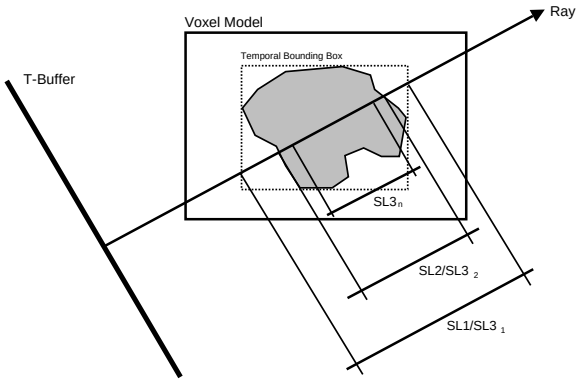


Figure 3: Space-leaping: The gray region represents the set of voxels that have a relevant value at some frame of the sequence. The temporal bounding box encloses this region. The intersection of the ray with the temporal bounding box defines the space-leaping interval of SL1 strategy. The intersection of the ray with the gray region defines the sampling interval of SL2. Finally, the sampling interval of SL3 is refined through time: at the first frame it is the same as SL1 ($SL3_1 = SL1$), at the second frame, it is the same as SL2 and after successive frames it diminishes to $SL3_n$

.

**Space-Leaping 1 (SL1): temporal bounding box**

During the pre-process of construction of the TRL, the bounding box of the voxels that are non-empty at some instant during the temporal sequence is computed. When the rendering starts, the sampling interval of every ray is computed as its intersection with the temporal bounding box (see the dot box in Figure 3) . The cost of the pre-processing is still $O(nv_m)$ but it increases with a comparison per voxel. However, the bounding box is valid for any camera position. This strategy provides space-leaping over empty voxels, but it does not skip non-selected voxels. Since the temporal bounding box is computed in the pre-process, it cannot take into account user's queries.

For multimodal studies, the temporal bounding boxes are computed separately for each modality. Each ray is intersected with these bounding boxes. Thus, every pixel of the *T-Buffer* stores instead of one sampling interval, a list of sampling intervals. Each of these intervals corresponds to a combination of modalities. During the integration of an interval, only the TRL of its modalities are consulted.

**Space-Leaping 2 (SL2): temporal ray sampling intervals**

The second strategy adds a view-dependent preprocessing that computes the sampling interval of each pixel of the *T-Buffer* as the position of the first and the last samples along the ray that have a relevant value at some frame. This preprocessing consists of a traversal of all the samples codes for every ray. It should be re-done

for any change in the viewpoint, but precisely because of that, it can skip not only empty voxels but also non-selected ones. This strategy has a higher pre-processing cost but it provides a more efficient space-leaping for the successive frames. In multimodal datasets, these intervals are computed separately for each modality and processed in the same way as SL1 strategy.

**Space-Leaping 3 (SL3): run-time adjustment of ray sampling intervals**

The third space-leaping strategy computes ray sampling intervals as in the SL2 strategy, but instead of a per-view pre-process, it does it during the rendering samples traversal. Whenever a non-relevant voxel is sampled along a ray, its next relevant value is searched. If the voxel is never relevant during the sequence, its current code is set to the last frame, independently from its possible variation through time. Then, it is not processed in the successive frames. The ray integration interval is defined by the first and last non-relevant voxels encountered along the ray in the previous casting of that ray. This interval is refined at each frame. By opposite to SL2, this strategy does not require a costly per-ray pre-process. Since the intervals are successively refined, the cost per frame tends to diminish. However, at the first frames, this cost can be greater than in the SL2 strategy. This is illustrated in Figure 3: the sampling interval $SL3$ at the first instant ($SL3_1$) is larger than $SL2$, but it is smaller than $SL2$ at the $n-th$ instant ($SL3_n$). This method adapts to multimodality as the previous two ones.

### 5.2 Simultaneous algorithm

The *simultaneous* method is a process that constructs simultaneously all the frames of the sequence. It is based on the TRL of the volume, but it does not require the *T-Buffer*. It processes all the rays along time sequentially. Every ray is cast at the first frame. Then, the computed pixel value is copied to all the following frames until the frame in which a change in the ray integration has been predicted. Then, the ray is recast at this frame and a new prediction on its next change is done. During the ray integration, the next sampling interval along the ray is also computed in order to provide space-leaping. This interval is defined by the position of the first and last relevant voxel at the next change, similarly to SL3 strategy.

This type of rendering is not interactive, but it is useful if the sequence can be generated in batch.

## 6 Out-of-core rendering

When the datasets are larger than the main memory, input/output communication becomes the bottleneck of rendering [6]. In the context of our investigation, this is serious problem since our datasets are multimodal and vary in time. Even though the Run-Length Encoding compresses efficiently the model in time, the size of the models and the number of bytes per voxel even for a one-frame dataset may be larger than the memory. As shown in [23], subdividing the volume can solve this problem. In order to overcome the I/O bottleneck while keeping the benefits of the proposed method, we subdivide the models into blocks both in space and time. First, long sequences are subdivided into smaller time spans. Next, the models are subdivided following the BON (Branch-On-Need) octree strategy [30]. The leaf nodes are the voxel model blocks. The criterion to divide a node of the octree is its memory occupancy. Thus, the more temporal variation of the voxels into a node, the more subdivided it is. The octree hierarchy is used only for the creation. Only leaf nodes (blocks) are stored in disk. The files are labelled according to the position of the blocks into the octree in order to allow us a sorted traversal. Therefore, during rendering, the leaf nodes are processed orderly: read from disk and rendered. The resulting subimages are composed FTB. For the simultaneous algorithm, each block of a time span is stored in one file and read only once. For the frame-to-frame algorithm, the blocks must be read from disk at each frame. Therefore, they are stored into various files, such that only the current codes of the voxels are read from disk at a given frame. Besides the capacity of handling large datasets, this strategy brings the advantage of improving time and space leaping: if the pixels of the subimage of a block in the main buffer have already reached the maximum opacity at a given frame, the block is skipped. Moreover, the block is rendered only if at least one of the pixels of its subimage changes at that frame.

| Dataset | size | nb | nf | $r_{sel}$ | $r_{coh}$ | $r_{occ}$ |
|---|---|---|---|---|---|---|
| Fireball | 100x100x100 | 1 | 10 | 0.52 | 0.52 | 0.61 |
| Movingball | 512x512x512 | 1 | 100 | 0.19 | 0.03 | 0.36 |
| MR-Head | 256x256x166 | 1 | 1 | 0.35 | – | – |
| SPECT | 256x256x166 | 3 | 8 | 0.19 | 0.14 | 0.49 |
| SRabbit | 256x256x256 | 1 | 16 | 0.92 | 0.37 | 0.47 |
| LRabbit | 387x513x640 | 1 | 16 | 0.86 | 0.17 | 0.44 |

Table 1: : Characteristics of the datasets: original size in number of voxels, original number of bytes per voxel (nb), number of frames (nf), ratios of selection ($r_{sel}$), coherency ($r_{coh}$) and occupancy ($r_{occ}$).

## 7 Tests

### 7.1 Description

We have performed tests on four different datasets:

- two phantom datasets that help us to better understand the behaviour of the algorithms,
- a real multimodal dataset of the brain composed of a dynamic SPECT and a static MR.
- a real dataset corresponding to the temporal evolution of a biomaterial implant in a rabbit femur at two different spatial resolutions (SRabbit and LRabbit).

The first phantom dataset *fireball* consists of a sphere initially full of variable voxel values that *freeze* along time from the external layers of the sphere to the internal ones. Once a layer is freezed, it remains constant through time. The second phantom dataset *multi-ball* is multimodal. It is composed of the *fireball* and the model *movingball* composed of a ball of constant values moving across the volume. Four different frames of this animation are shown in Figure 4. The SPECT and MR datasets depicted in the header Figure 1 belong to the Whole Brain Atlas of the Harvard medical department. Finally, the rabbit dataset shown in Figure 5 has been captured at the European Sincrotron Radiation Facility (ESRF, Beam Id17).

In Table 1, we show the characteristics of the different datasets: name, size, number of frames (nf), selection ratio ($r_{sel}$), temporal coherence ratio ($r_{coh}$), occupancy ratio ($r_{occ}$). The selection ratio $r_{sel}$ is defined as the total number of selected voxel values divided by the number of voxels multiplied by the number of frames. The temporal coherence ratio $r_{coh}$ is the number of codes of all the voxels, divided by the number of voxels multiplied by the number of frames. Finally, as explained in Section 3 the occupancy ratio $r_{occ}$ is the size of the RLE divided by the size of the original models. In the MR dataset, these two last parameters are not depicted because, being static, it is not represented as a TRL.

Tables 2 and 3 show the results of two unimodal and two multimodal tests on the datasets. The results correspond to executions on a Pentium-4 at 3.06 GHz with 1G memory and a Video

| Method | PP | First fr. | next fr. | gain |
|---|---|---|---|---|
| BF-SL1 | 0.31 | 6.62 | 7.38 | 1 |
| BF-SL2 | 2.53 | 3.51 | 4.29 | 0.61 |
| Coh-SL1 | 0.31 | 6.52 | 2.74 | 0.43 |
| Coh-SL2 | 2.53 | 3.50 | 2.20 | 0.36 |
| Coh-SL3 | 0.48 | 7.15 | 1.85 | 0.33 |
| Simul | 0.47 | 0.99 | 0.99 | 0.14 |

Table 2: : Results of the tests on phantom data *Fireball*. *PP* is the pre-processing cost, *First fr.* is the cost in seconds of the first frame, *next fr.* is the average cost in seconds of the remaining frames, and *gain* the performance of the method measured as the overall cost of the animation in relation to its cost with the reference method BF-SL1. Times are expressed in seconds and they correspond to a 800x800 image.

| Method | fireball | multiball | MR-SPECT | SRabbit | LRabbit |
|---|---|---|---|---|---|
| BF-SL1 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| BF-SL2 | 0.61 | - | 0.41 | 0.83 | - |
| Coh-SL1 | 0.43 | 0.04 | 0.71 | 0.58 | 0.16 |
| Coh-SL2 | 0.36 | - | 0.37 | 0.58 | - |
| Coh-SL3 | 0.33 | 0.04 | 0.34 | 0.44 | 0.15 |
| Simul | 0.14 | 0.03 | 0.35 | 0.36 | 0.09 |

Table 3: : Results of the simulations on the different datasets. The values shown express the ratio between the overall cost of the animation and its cost with BF-SL1.

Card Nvidia FX5600. The proposed method has been implemented with the three different space-leaping strategies and the simultaneous method. They are labeled in the tables as *Coh-SL1*, *Coh-SL2*, *Coh-SL3* and *simul*, respectively. In order to provide a better understanding of the results, we have also implemented the brute-force sequential method that simply processes the volume frame after frame without temporal coherence. This method has been implemented with the two first space-leaping strategies (SL1 and SL2). They are labeled in the tables as *BF-SL1*, *BF-SL2*. Tests on large datasets have used the out-of-core strategy which is incompatible with the SL2 space leaping method. Specifically,*Multiball* has been subdivided into 10 spans and 8 blocks, while *LRabbit* has been subdivided in space only, into 16 blocks.

Table 2 shows the results of the tests of the five methods on phantom data *Fireball*. Times are expressed in seconds, and the gain is computed as the total time of the animation divided by the total time of the animation with the basic method *BH-SL1*. The higher pre-processing times correspond to SL2 both in the sequential and the coherent methods. However, it is a third of the cost of a frame in BF-SL1. The higher cost in the first frame occurs in Coh-SL3, because it is when most of the space-leaping interval is computed. The lowest average cost of the next frames is in Coh-SL3, because the space-leaping interval is better fitted. The cost per frame diminishes along time in Coh-SL3, while it is almost constant in the other strategies. Space-leaping SL2 provides a considerable saving in the sequential and the coherent strategies. The more efficient method is the simultaneous.

Table 3 shows the results of the tests of the five methods on all the datasets. Only the ratio of the method in relation to the sequential one is shown. For the out-of-core processing, the ratio includes both reading from disk and rendering times. All the tests correspond to images of 800x800. The cost of reading from disk is small in relation to the cost of rendering, therefore, the out-of-core results follow the same pattern as the others.

### 7.2 Results

In all the datasets the occupancy ratio of the TRL in relation to the regular model is low. The relative occupancy of the TRL varies between 12% and 61%, depending on the number of selected voxels that change along time. The cost of construction of the TRL is low, less than the lowest cost of rendering of one frame in the best frame-to-frame coherent strategy.

In all the tests, the four frame-to-frame coherent algorithms reduce considerably the processing time of the animation in comparison to the sequential methods. This is due to the high number of rays that are skipped, because they cross empty or non-relevant regions or because they keep the same value as in the previous frame. Overall, the number of skipped rays ranges between 60% and 88%.

In general, the simultaneous method has the better performance. This is basically due to the fact that it saves the cost of management of the *T-Buffer* and that the memory access is more efficient than in the other methods. The performance is much better for large model, since the blocks are read only once.

The cost of pre-processing in space-leaping SL2 is not very high in relation to the rendering cost per frame, between one third and one half the cost of rendering of one frame in the sequential method. In fact, the size of the images affects the cost of the pre-process, but globally, it does not have a visible influence on the overall cost. Therefore, in the datasets that have a lot of empty space, SL2 produces considerable time savings even for a low number of frames. This is visible in the gains of the sequential method BF-SL2 in relation to BF-SL1 and Coh-SL2 in relation to Coh-SL1.

In most cases, the best frame-to-frame strategy is Coh-SL3. Although the first frame cost is larger than in the other strategies, it benefits from the adaptive space-leaping and the average cost of the remaining frames is lower than in other methods. In fact, starting from the second frame, the space-leaping interval is at most as large as in SL2. In large datasets where the out-of-core strategy has been applied, the ratios are even better than in the small datasets. This is due to the fact that some blocks are pruned before loading because of ray and temporal coherency.

## 8 Conclusions

The encoding of the volume through time as a Temporal Run-Length provides an important compression of the voxel model while keeping its spatial order. In addition, it allows us to fast predict changes in voxel values along time. In this paper, we have shown that this prediction capability can be conveniently exploited in a time-varying ray casting. The Temporal Run-Length together with a temporal buffer and three different space-leaping strategies provide important savings in the computational cost of the animation. The method can be used on small datasets that fit into the main memory and it can be adapted to large datasets using the out-of-core strategy presented in the paper.

Several new research line start from this work. First, we believe that the Temporal Run-Length Encoding of the volume could be used in other direct rendering methods such as axis aligned splatting and 3D texture mapping. Next, the out-of-core strategy could be complemented with memory bricking as suggested in [8] and [22].

**Acknowledgments:** This work has been partially funded by the projects MAT2002-04297-C03-02 and *IM3: Imagen Molecular y Multimodalidad* from the spanish government and by the CREBEC from the catalan government. The capture of the rabbit dataset has been funded by the ESFR. We thank Alberto Bravin and Paul Tafforeau for their help in the capture process. We would like to thank Eduard Vergés for his help in processing the rabbit dataset.

## REFERENCES


[1] K. Anagnostou, T. Atherton, and A. Waterfall. 4D volume rendering with the Shear-Warp factorization. *Proc. Symp. Volume Visualization and Graphics'00*, pages 129–137, 2000.

[2] C. L. Bajaj, V. Pascucci, G. Rabbiolo, and D. Schikorc. Hypervolume visualization: a challenge in simplicity. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 95–102. ACM Press, 1998.

[3] P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurfacing in higher dimensions. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 267–273. IEEE Computer Society Press, 2000.

[4] S. Coorg and S. Teller. Temporally coherent conservative visibility. *Comput. Geom. Theory Appl.*, 12(1-2):105–124, 1999.

[5] D. Ellsworth, L.J. Chiang, and H.W. Shen. Accelerating time-varying hardware volume rendering using tsp trees and color-based error metrics. In *VVS '00: Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 119–128. ACM Press, 2000.

[6] R. Farias and C. T. Silva. Out-of-core rendering of large, unstructured grids. *IEEE Comput. Graph. Appl.*, 21(4):42–50, 2001.

[7] M. Ferré, A. Puig, and D. Tost. A fast hierachical traversal strategy for multimodal visualization. *Visualization and Data Analysis 2004*, pages 1–8, 2004.

[8] A. Savopoulos G. Sakas, M. Grimm. Optimized maximum intensity projection (mip). *6th Eurographics Workshop on Rendering*, pages 81–93, June 1995.

[9] S. Guthe and W. Strasser. Real-time decompression and visualization of animated volume data. In *Proceedings of the conference on Visualization 2001*, pages 349–356. IEEE Press, 2001.

[10] V. Havran, J. Bittner, and H.P. Seidel. Exploiting temporal coherence in ray casted walkthroughs. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, pages 149–155, New York, NY, USA, 2003. ACM Press.

[11] V. Havran, C. Damez, and H.P. Myszkowski, K. Seidel. An efficient spatio-temporal architecture for animation rendering. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 106–117, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[12] H. Hubschman and S. W. Zucker. Frame-to-frame coherence and the hidden surface computation: constraints for a convex world. *ACM Trans. Graph.*, 1(2):129–162, 1982.

[13] P. Lacroute and M. Levoy. Fast volume rendering using a Shear-Warp factorization of the viewing transformation. *ACM Computer Graphics*, 28(4):451–458, July 1994.

[14] S.K. Liao, Y.C. Chung, and J.Z.C. Lai. A two-level differential volume rendering method for time-varying volume data. *The Journal of Winter School in Computer Graphics*, 10(1):287–316, 2002.

[15] E.B. Lum, K.L. Ma, and J. Clyne. A hardware-assisted scalable solution for interactive volume rendering of time-varying data. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):286–301, 2002.

[16] K. Ma and D.M. Camp. High performance visualization of time-varying volume data over a wide-area network status. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, page 29. IEEE Computer Society, 2000.

[17] K. Ma, D. Smith, M. Shih, and H.W. Shen. Efficient encoding and rendering of time-varying volume data. *Technical Report ICASE NASA Langsley Research Center*, pages 1–7, 1998.

[18] K.L. Ma. Visualizing time-varying volume data. *Computing in Science and Engg.*, 5(2):34–42, 2003.

[19] I. Martin, X. Pueyo, and D. Tost. Frame-to-frame coherent animation with two pass radiosity. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):70–84, 2003.

[20] M. Martin, E. Reinhard, P. Shirley, S. Parker, and W. Thompson. Temporally coherent interactive ray tracing. *Journal of Graphics Tools*, (72):41–48, 2003.

[21] N. Neophytou and K. Mueller. Space-time points: 4D splatting on efficient grids. In *Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 97–106. IEEE Press, 2002.

[22] S. Parker, M. Parker, Y. Livnat, P.P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):238–250, 1999.

[23] E. Reinhard, C.Hansen, and S.Parker. Interactive ray-tracing of time varying data. In *Eurographics Workshop on Parallel Graphics and Visualisation 2002*, pages 77–82. Eurographics, 2002.

[24] H. Shen, L. Chiang, and K. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. *Proc. IEEE Visualization*, pages 371–377, 1999.

[25] H.W. Shen and C.R. Johnson. Differential volume rendering: a fast volume visualization technique for flow animation. In *Proceedings of the conference on Visualization '94*, pages 180–187. IEEE Press, 1994.

[26] P. Sutton and C. Hansen. Accelerated isosurface extraction in time-varying field. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):98–107, 2000.

[27] M. Wan, A. Sadiq, and A. Kaufman. Fast and reliable space leaping for interactive volume rendering. In *VIS'02: Proceedings of the conference on Visualization '02*. IEEE Computer Society, 2002.

[28] C. Weigle and D. Banks. Extracting iso-valued features in 4-dimensional scalar fields. *Proc. Symposium on Volume Visualization*, pages 103–110, October 1998.

[29] R. Westermann. Compression domain rendering of time-resolved volume data. In *VIS '95: Proceedings of the 6th conference on Visualization '95*, page 168. IEEE Computer Society, 1995.

[30] J. Wilhems and A. Van Gelder. Multidimensional trees for controlled volume rendering and compression. *Proc ACM Symposium on Volume Visualization*, 11:27–34, October 1994.

[31] J. Woodring and H.W. Shen. Chronovolumes: a direct rendering technique for visualizing time-varying data. In *VG '03: Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, pages 27–34. ACM Press, 2003.

[32] J. Woodring, C. Wang, and H-W Shen. High-dimensional direct rendering of time-varying volumetric data. In *IEEE Visualization 2003*, pages 417–424, 2003.

[33] R. Yagel and Z. Shi. Accelerating volume animation by space-leaping. In *VIS '93: Proceedings of the 4th conference on Visualization '93*, pages 62–69, 1993.

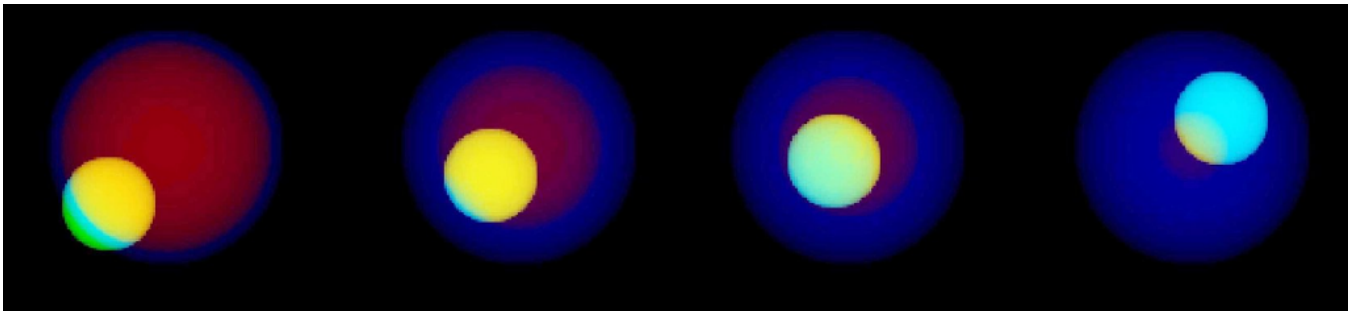

Figure 4: Four sample frames of the animations of the multiball dataset.

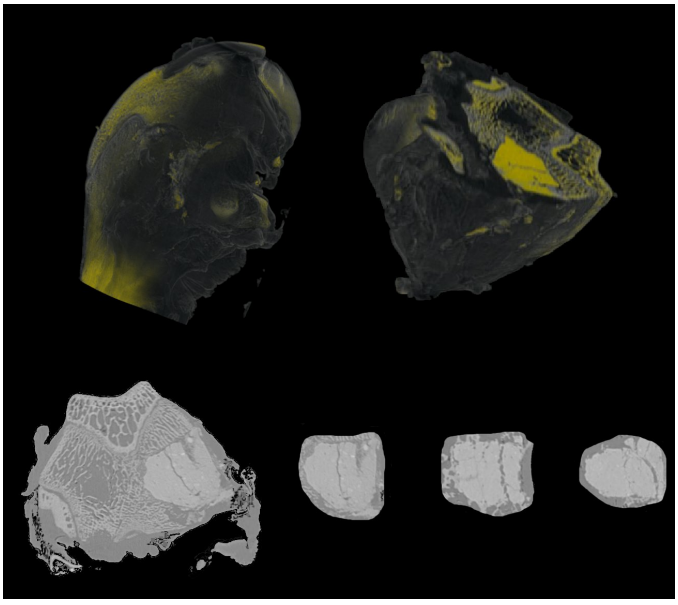

Figure 5: The rabbit datasets: at top two frames of the 3D rendering, the leftmost one shows the full model, and the rightmost one is the model clipped, showing the bioimplant; at bottom left a slice of the bone showing the bioimplant, at bottom right, three slices of the bioimplant at different instants of time.