Time-varying volume visualization

D. Ayala J. Campos M. Ferre S. Grau A. Puig
D. Tost

July 22, 2005

Abstract

Volume rendering is a very active research field in Computer Graphics because of its wide range of applications in various sciences, from medicine to flow mechanics.

In this report, we survey a state-of-the-art on time-varying volume rendering. We state several basic concepts and then we establish several criteria to classify the studied works: IVR versus DVR, 4D versus 3D+time, compression techniques, involved architectures, use of parallelism and image-space versus object-space coherence. We also address other related problems as transfer functions and 2D cross-sections computation of time-varying volume data.

All the papers reviewed are classified into several tables based on the mentioned classification and, finally, several conclusions are presented.

Contents

1 Introduction **1**

2 Basic concepts **1**
2.1 Changes through time 1
2.2 Data characteristics 3
2.3 Coherence Metrics 3
2.4 Model 5

3 Classification **5**
3.1 Rendering versus other manipulations 5
3.2 IVR versus DVR 6
3.3 4D models versus 3D+time models 7
3.4 Compression 7
3.5 Architecture 11
3.6 Use of parallelism 12
3.7 Temporal coherence 13

4 Survey **14**
4.1 IVR methods 14
4.1.1 3D + time models 14
4.1.2 4D models 16
4.2 DVR methods 17
4.2.1 Ray-casting 17
4.2.2 Splatting 22
4.2.3 Hardware-driven texture mapping 22
4.2.4 Shear-warp 24
4.3 Other uses 25
4.3.1 Tranfer functions for Time-Varying Data 25
4.3.2 Volume animation based on skeletons 26
4.4 Rendering in the transformed space 27

5 Conclusions **27**

A Acronyms **30**

1 Introduction

Volume rendering is a very active research field in Computer Graphics because of its wide range of applications in various sciences, from medicine to flow mechanics.

One of the major problems of Volume rendering is the huge volume of data that must be manipulated in order to obtain meaningful visualizations. The size of the datasets conditions the necessary time for rendering. In general, the larger the datasets, the slower the rendering. This is the reason why many research efforts have been put on speeding up rendering of large datasets with software as well as hardware-based solutions. From a software point-of-view, several techniques and data structures have been designed that provide an efficient access to the relevant data while skipping empty space *(space-leaping)* [DH92] [SH94] [WSK02]. Other strategies address the I/O bottleneck when the data do not fit into memory *(out-of-core)* [SCES02] [SGS95] [PPL+99] [CS97] and [CSS98]. Hardware-based speeding strategies, on the other hand, can be further subdivided into categories: those that exploit the capacity of current consumer hardware, such as 3D texture based rendering [MGS02] and parallelization [GPR+94], and those that design hardware specifically customized for volume rendering [PK96] [PHK+99].

In spite of these contributions, the problem of the efficiency in rendering continues being serious because, as the input devices improve, the size of the datasets increases. In addition, the necessity prevails more and more to work with data coming from different devices (*multimodal datasets*). Furthermore, there is a great demand of applications able to show the evolution of data throughout time (*time-varying datasets*).

In this last case, it is possible to accelerate the visualization, considering that a continuity in time or *temporal coherence* exist: successive images look like. Therefore, part of the calculations made for one image, could be used for the following one.

The idea of taking advantage of temporal coherence to speed-up rendering was early introduced by Hubshman and Zucker [HZ82]. Since then, many techniques have been proposed to speed up visibility computations [Bad88] [Gla89] [CCD90] [Tos91] [GWP91] [HBS03] [CT99] and global illumination [BP01] [WKB+02] [MPT03] [HDM03] of polygonal scenes. However, there are still relatively few works addressing this problem for volume rendering. The goal of this report is to survey the state-of-the-art on the use of temporal coherence for volume datasets. We first define some basic concepts (Section 2). Then, in Section 3, we describe the classification criteria that we have used to present the existing works which are discussed in Section 4. Finally, we present our conclusions in Section 5.

2 Basic concepts

2.1 Changes through time

Different elements of a scene can vary through time: the objects, their internal properties, the camera and rendering parameters such as transfer functions and lighting

conditions. The strategies that can be used depend on theses cases. For clarity, in this work, we will distinguish each case using the following nomenclature.

- **Volume animation:** when the volume datasets move. This movement can be an affine transformation of translation and rotation of all the model or a non-uniform displacement of the sample points. The former case happens whenever the volume is part of a huger scene, composed of other objects, either polygonal or volumetric [KK99]. In the later case, the volume is actually deformed.

- **Time-varying animation:** when the volume is static but the properties inside the cells vary. Two typical applications of this type of scenes are (i) a temporal series of SPECT of a patient's brain and (ii) the simulation of a fluid flowing in a fixed section of a channel.

- **Fly-through navigation:** when the volume dataset is static and its properties constant but the viewer's position and direction varies though time, because it navigates through the data. Typical examples of fly-through navigations are the virtual cateterism [PTN97], virtual colonoscopy [HKW+95] [WTK+99] and bronchoscopy [BSG+01], [MHT+96]. Some of these papers perform the navigation through surface models previously extracted from volume datasets, while others [BJNN98] actually navigate through the volume. The major problem addressed in this type of navigation is how to efficiently perform visibility culling and to bring into memory the portions of volumes that fall in the current viewing frustrum. In this report, we do not address fly-through navigation.

- **Fly-around navigation:** when the volume dataset is static and its properties constant but the viewer's position and direction varies, because it moves around the volume, without entering inside it. [YS93]

Obviously, these cases can be combined. As an example, if the viewer moves around a volume whose properties vary through time, we will talk about a *Fly-around navigation of time-varying volume data*. If the viewer navigates inside the same volume, it will be a *Fly-through navigation of time-varying volume data*. If the volume moves and its properties vary, we will talk about a *Volume animation of time-varying data*.

We do not enclose in this taxonomy the case in which only user-defined rendering parameters such as lighting conditions and transfer functions change. This has much to do with interactivity than actually animation and its variants.

In most of these cases, rendering consists of generating a sequence of images of the volume at different instants throughout a period of time. For clarity, we will call these time instants *key-instants* and the corresponding images *key-frames*. A different approach is that of the *Chronovolumes* [WS03]. Inspired on the early photographic methods of Marey and Muybridge, this technique performs an integration through time and produces a single view that captures the essence of various key-instants of the

sequence. It is suitable for time-varying voxel models. The Chronovolume is a voxel model such that every voxel is computed by integrating all the voxel values throughout time for a given transfer function. The chronovolume is rendered as a regular volume in order to produce 2D images.

2.2 Data characteristics

Volume rendering strongly depends on the application in which it is used and on the characteristics of volume data: the size of the data sets, the distribution of the sample points, the type of property associated to each sample and the way they vary along time. In medical applications, the grid structures of the sample points are generally regular, based on a Cartesian lattice, also called rectilinear grid. Typical medical dataset sizes are 256^3 or 512^3. On the contrary, computer simulations of vector fields such as *Computational Fluid Dynamics* (CFD) produce curvilinear and irregular grids.

The property values associated to each sample can be unique, coming from one input modality (*unimodal rendering*), or coming from different input modalities (*multimodal rendering*). Properties can be scalar values, such as density, or vectorial values, such as velocity. Scalar values can be rendered directly using emission and absorption and surface scattering shading models. Vector field rendering requires to constructing geometric primitives such as pathlines from the data. An efficient computation of trajectories based on texture discretization, the *Unsteady Flow Advection Convolution* (UFAC) is described in [WEE03]. This type of work fall beyond the scope of this report.

Finally, the speed with which the property values vary determines the degree of similarity of the volume between successive key-instants.

2.3 Coherence Metrics

In order to evaluate the spatial coherence and the temporal coherence on the data sets, some metrics have been defined in different papers. *Spatial metrics* indicate the amount of coherence on the volume domain and *temporal metrics* measure the amount of coherence or variability within a series of volumes. These metrics have been used to analyze the variability of the original data sets as well as to define an error measure on the compressed volume representations used to improve the rendering stage. Thus, the metrics can be defined based on the voxels' scalar values (*scalar-based* metric) and if it is computed from the actual color of the voxels it is called *color-based* metric.

As the scalar-based metrics only depend on the data, they can be precomputed and saved in an auxiliary structure. Although color-based metrics are more accurate because they are more closely related to the image than the scalar value, they must be recomputed when the transfer function is changed.

- scalar-based metrics: There are a variety of algorithms in the literature that measure 2D image similarity such as Simple Differencing, the Likelihood ratio method, Adaptive Threshold method and others based on feature extraction.

In the time varying volume data sets, [AAW00], use the χ^2 criterion to detect changes between a data set A and a data set B, and it is formulated as:

$$\chi^2 = \sum_{data} \frac{(f_i(data_A) - f_i(data_B))^2}{2\sigma^2}$$

If χ^2 is less than a certain threshold T, the two data sets are similar, otherwise they differ.

[SCM99] defines the *Coefficient Of Variation* (COV), which is the standard deviation divided by the mean. They define COV in a single frame for all voxels, COV_{ss}, as well as for each voxel in a time span, COV_{st}:

$$COV_{ss} = \frac{\sigma}{v}$$

$$COV_{st} = \frac{1}{n}\sum \frac{\sigma_i}{v_i}$$

where σ is the voxel's standard deviation and v is the voxel's mean. σ_i is the per-voxel standard deviation and v_i the per-voxel mean. COV_{ss} measures the spatial coherence of the volume in a single frame and COV_{st} measures the temporal coherence of a single voxel.

$$v = \frac{1}{N}\sum_{i,t} v_{i,t}$$

$$\sigma = sqrt(\frac{1}{N}\sum_{i,t}(v_{i,t} - v)^2))$$

$$v_i = \frac{\sum_{t=t_1}^{t=t_2} v_{i,t}}{t_2 - t_1 + 1}$$

$$\sigma_i = sqrt(\frac{\sum_{t=t_1}^{t=t_2}(v_{i,t} - vi)^2}{t_2 - t_1 + 1}$$

These metrics have been used in other later approaches ([ECS00] and [JKM01].

- color-based metrics: [ECS00] extends the COV metric to two different color-based metrics: the reference color-based and the approximate color-based. The former one is the natural extension of the COV metric defined above to the $RGB\alpha$ domain. They define distances in RGB space in order to compute the standard deviation. Thus, the color-based mean m and the color-based standard deviation σ are:

$$m = sqrt(\alpha(r^2 + g^2 + b^2) + \alpha^2)$$

$$\sigma = sqrt(\frac{1}{N}\sum d(c_{i,t}, c))$$

where $d(c_1, c_2)$ is the squared distance function defined as:

$$d(c_1, c_2) = \alpha_1[(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2] + (\alpha_1 - alpha_2)^2$$

Then $COV_{cs} = \frac{\sigma}{m}$. The COV_{ct} is defined in similar way than COV_{st} using the extended distances on the RGB space.

As the calculations of these color based metrics are very slow, [ECS00] also defines an approximate color-based metric that can be computed faster. The new metric computes the difference between each transfer function entry and the estimated mean color values. Also, they assume that the frequencies of values are normally distributed. With this metric, they achieve a significant performance improvement when the transfer functions change during the rendering process without less quality on the final image.

2.4 Model

The model that represents a 3D region of volume depends strongly on the type of data grid. In general, regular Cartesian grids are well represented with voxel models which are *Cubic Cartesian* (CC) grids [Kau90], but Theussl at al. [TMG01] have shown that the *Body Centered Cubic* (BCC) grid can save up to 30% of memory requirement and of rendering performance. Irregular data are represented with tetrahedral cells [CFM+94]. Table 1 presents the papers surveyed in Section 4 classified according to the model.

3 Classification

We herein describe different criteria of classification that we have used in our study. Table 5 and the next ones show the classification of all the references surveyed in Section 4 according to these criteria.

3.1 Rendering versus other manipulations

Although the main application on volume datasets is rendering, some work has been published that addresses related problems, such as the design of transfer functions [JKM01] and the computation of 2D cross sections from time-varying data. We survey them in Section 4.3.

Regular model (voxels)	Irregular model (tetras)
[Wes95]	[GSDJ04]
[GS01]	[WEE03]
[LMC02]	[She98]
[WB98]	[SB05]
[BWC00]	
[AAW00]	
[CD99]	
[SCM99]	
[BPRS98]	
[NM02]	
[WWS03]	
[YS93]	
[SJ94]	
[WSK02]	
[LCL02]	
[RHP02]	
[ECS00]	
[BAS02]	
[WS03]	
[SH99]	

Table 1: Regular data sets versus irregular data models.

3.2 IVR versus DVR

There are two major strategies to visualize volumes:*Indirect Volume Rendering* (IVR) and *Direct Volume Rendering* (DVR). In the former approach, one or more polygonal isosurfaces are first extracted from the volume [LC87] and then rendered using the polygon-based hardware-assisted pipeline. In the bibliography, we have found several works addressing the extraction of surfaces from time-varying volume datasets (see Table 2). We survey them in Section 4.1. Strategies to speed up the second part of the IVR pipeline, i.e. rendering the extracted surfaces, would fall in the category of temporal coherence for polygonal scenes [HDM03] which is out of the scope of this report.

The latter approach (DVR) renders directly the volume using one of these four methods [MHB+00]:

- ray-casting [Lev90]
- splatting, either volume-aligned [Wes90] or image-aligned [KMC99]
- shear-warp [LL94]

IVR	Direct Volum Rendering (DVR)			
	RayCasting	Splatting	Shear-warp	Texture Mapping
[WB98]	[Wes95]	[BPRS98]	[GS01]	[GS01]
[BWC00]	[SCM99]	[NM02]	[CD99]	[LMC02]
[GSDJ04]	[YS93]		[AAW00]	
[SH99]	[WSK02]		[ECS00]	
[BAS02]	[SJ94]			
[She98]	[LCL02]			
[SB05]	[RHP02]			
	[WS03]			
	[WWS03]			

Table 2: IVR versus DVR.

- 3D hardware-assisted texture-mapping [WE98]

Table 2 presents the papers surveyed in Section 4.2 classified according to the algorithm on which they are based.

3.3 4D models versus 3D+time models

Time-varying datasets can be treated specifically as three-dimensional models that evolve throughout time or as particular cases of *n*-dimensional models [Neo03]. Strategies developed in the former case typically exploit temporal coherence, whereas *n*-D techniques treat time as one dimension more. In [WWS03] different interpretations of a 4D projection are analyzed and in [BPRS98] a graphical user interface is presented to deal with parallel projections of n-dimensional data. Table 3 shows a list of both types of approaches.

3.4 Compression

The huge size of volume models compromises the efficiency of rendering. The use of compression techniques can reduce this problem. The compression schemes can be classified into two categories: those based on spatial data structures and those based on the space of values. The purpose of spatial data structures is to reduce the complexity of traversal algorithms, which is generally of the order of the mesh size. By grouping cells of the models, either hierarchically or not, spatial data structures provide means of avoiding costly cell-to-cell processing. On the contrary, algorithms based on the values space rearrange the data according to function space coherency [BAS02], but they sacrifice spatial coherency. Hybrid approaches working in both spaces can take advantage of space and value coherencies.

Spatial data structures

4D model	3D+time model
[WB98]	[Wes95]
[BWC00]	[GS01]
[SCM99]	[LMC02]
[BPRS98]	[CD99]
[NM02]	[AAW00]
[WWS03]	[She98]
	[SB05]
	[GSDJ04]
	[SJ94]
	[LCL02]
	[RHP02]
	[ECS00]
	[SH99]

Table 3: 4D models versus 3D+time models.

In DVR, spatial data structures are often used to encode the location of non-empty cells in a volume. They provide means to skip these empty regions, i.e., to perform *space leaping*. Several spatial data structures have been used for these purposes: pyramids and octrees [Lev90], multidimensional trees [WG94], kd-trees [SF90], shells [UO93], *extreme vertices encoding* (EVM) [RAA04], [RAG05], distance transforms [ZKV92] and run-length encoding [Lac95b]. Octrees, shells and EVM have also been been used in IVR to quickly identify the cells that contain an isosurface, i.e., *isosurface cells*. We next describe those structures that have been extended to time-varying volume data.

In order to improve the efficiency of volume ray-casting, Levoy [Lev90], propose to construct a pyramid such that for a dataset of N voxels on a side with $N = 2^M + 1$, for some M, the pyramid is composed of $M+1$ binary volumes. A cell i at a pyramid level m contains a zero value if all eight cells at level $m-1$ that form its octants contain a zero value and a value of one otherwise. Therefore, when a ray intersects a zero-valued cell at a level m, the lower levels of the pyramid within the cell don't need to be sampled. This pyramid can be implemented by condensation as a linear octree [Gar82].

Wilhems and Van Gelder [WG92] designed the *Branch-On-Need Octree* (BONO) that partitions the cells efficiently when the dimensions of the volume are not powers of two. The subdivision criterion is that the lower subdivision in each direction covers the largest possible amount of two cells. When used in IVR, the BONO codifies the min-max bounds of each cell. Thus, during the traversal of the octree for the extraction of an isosurface, only cells whose min-max bounds encloses the isosurface value are traversed. A similar scheme is applied in [SH94] but codifying Lipschitz bounds instead of min-max bounds.

Finally, the *Run-Length-Encoding* (RLE) consists of a series of *runs*, composed of a voxel value and the number of consecutive voxels that share this value. The RLE used in the shear-warp rendering proposed by Lacroute [Lac95b] encodes only the property of transparency or emptiness of the voxels. Therefore, it reduces the runs to the number of voxels since the value (transparent or non-transparent) can be deduced from the alternation of the codes.

Value-based compression

Compression techniques based on the value space include the span-space [LSJ96], the interval tree [LSJ96] and the usual compression scheme of signal processing such as the *Discrete Cosine Transform* (DCT), the *Fast Fourier Transform* (FFT) and wavelets [Mur93]. The decomposition of the value space instead of the geometric space has the advantage that it can be applied to structured as well as unstructured data.

The *span-space* was introduced by Livnat et al [LSJ96]. Let C be the set of data cells, to each cell $c_i \in C$, we associate a point $p_i = (min_i, max_i)$ such that $min_i = min_j(v_j)_i$ and $max_i = max_j(v_j)_i$ ($(v_j)_i$ are the values of vertices of the cell c_i). P is the set of points p_i. Then, the isosurface corresponding to a value v will consist of the set of cells, whose corresponding points follow $min_i \leq v < max_i$. As all the interval methods, the intervals have to be ordered either by the maximum or by the minimum value. The authors of this method propose the use of a kd-tree, structure due to Bentley [J.L75], to simultaneously order by maximum and minimum values. The kd-tree is the multidimensional extension of a binary search tree. This method improves the complexity of isosurface extraction. Let n be the number of cells and k the number of cells intersected by the isosurface. While most of the existing algorithms have a worst case complexity of $O(n)$, using octrees [WG92] is $O(klg(n/k)$ and using the span space is $O(\sqrt{(}n)+k)$, taking into account that the construction of the kd-tree is $O(nlgn)$.

The approach [CMM+97] is also based on the span space and to deal with the set of intervals the authors propose an interval tree, structure due to Edelsbrunner [Ede80], instead of a kd-tree and the search complexity is logarithmic. Moreover, for structured datasets in which a marching cubes-like algorithm [LC87] is to be applied only intervals of a subset of cells are stored. This subset are the black cells of a 3D chess-board interpretation of the volume (1/4 of the total number of cells) because the vertices of the generated triangular patches would lay on the edges of the volume dataset and the mentioned subset of cells hold all of them (1 edge is shared by 4 cells).

The *DCT* [GW92][Jai89][Say00] has been used by Ma et al. [LMC02] join to a quantization and encoding to perform a lossy compression or time-varying data. The technique is used to mapping scalar sequences into single scalar indices to exploit the temporal coherence.

The *DCT* is defined by:

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{(2x+1)u\pi}{2N}\right]$$

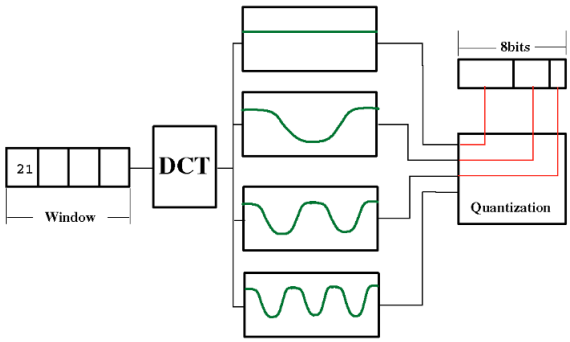

Figure 1: DCT-based encoding example, using a window size of 4 and compressing the first three coefficients into an 8-bit value.

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for u=0} \\ \sqrt{\frac{2}{N}} & \text{for u=1..N-1} \end{cases}$$

where $C(u)$ are the transformed coefficients, N is the number of input samples and $f(x)$ are the input samples. Each coefficient, representing the component of a certain frequency, is quantized in an adaptive scheme that leaves more bits to the high variance features (see Figure 1).

The resulting values are combined into a single number used as an index entry to a 2D palette texture (see Section 4.2.3). As the encoding is performed for each slice, the same scalar value can be encoded into a different compressed value depending on the slice. The authors mention this as a minor problem because the possible artifacts are soften when computing the whole volume rendering integral. Results show that compression can reduce the amount of texture storage required more than three times.

S. Guthe and W. Strasser [GS01] used *wavelets* as the basis to their lossy compression method to perform hardware-driven texture mapping rendering (see Section 4.2.3). The wavelet transform itself is lossless, consisting in analyzing the original signal by applying a wavelet Ψ and scaling Φ filter and down-sampling the resulting signals by a factor of 2 (see Figure 2). The resulting signals are discretized, obtaining what are called the *wavelet coefficients*: H for the high frequencies, and L for the low ones. This transformation can be generalized to 3D, applying the 1D wavelet transform in all three dimensions separately, resulting in a 3D tensor product. Then we apply a 3D wavelet transform to the low sub band recursively (see Figure 3).

Guthe and Strasser compared higher order wavelets, obtaining better results as the order increases. They encoded the individual volume datasets using a 3D wavelet transform. The range of coefficients that would be mapped to zero are discarded, and the rest scaled and quantized. These range and scale parameters depend on human vision sensibility. Compression is performed with a combination of RLE, and the LZW algorithm [ZA77] or an arithmetic encoding, achieving a ratio 100:1 for single volumes. To obtain longer zero runs, a depth first traversal through the wavelet coefficients octree is

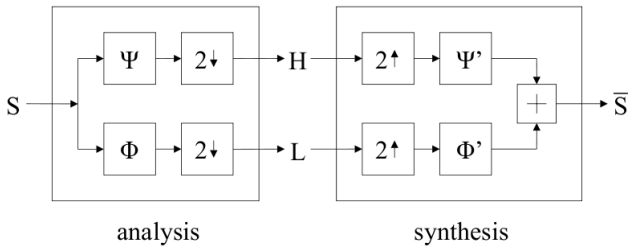

Figure 2: Wavelet transform of a 1D signal S.

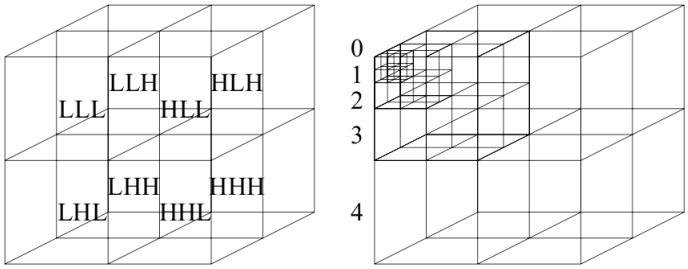

Figure 3: Left: a single step (L=low pass filtered, H=high pass filtered). Right: recursive decomposition (4 steps)

performed. However the non-linear memory access results in a loss of decoding speed. The resulting RLE are compressed with the LZW or an arithmetic encoding. The LZW encodes the sequences by storing the last appearance of the same sub string, while the arithmetic encoding used -an adaptive model- adapts the number of bits employed to encode each value.

To exploit the temporal coherence, the differences between frames are used to compress the volume sequence applying a method similar to the MPEG motion compensation [[MP93] [[MP96]. The technique is called *windowed motion compensation* because it expands the blocks used to predict motion, so they overlap each other. Later on, the overlapped areas are filtered to compensate. The authors obtain a whole volume compress ratio up to 200:1 using this method.

Wavelet transform has been used also by R.Westermann in [Wes95] during the compression process, but in this case there is no decompression process because the render is performed in the transformed space (see Section 4.4).

3.5 Architecture

Another classification criterion that we have used in this survey is the type of architecture used in volume rendering: general purpose architecture or dedicated ones. General purpose architectures provide maximum flexibility and they have recently improved their performance with the development of new programmable GPUs. However, they still fail at providing real-time frame rates for rendering of high -resolution 3D and 4D

datasets. Therefore, many efforts have been put on the development of special-purpose volume rendering architectures. As an example, the Visualization Lab's team leaded by professor Kaufman, has been working since 1982 on this topic and has designed the Cube-1 [KB88], Cube-2 [BKX90], Cube-3 [PKC94] and Cube-4 [PK96] architectures.

3.6 Use of parallelism

The improvement of the efficiency of volume rendering and the fact that some visualization installations are inherently concurrent, because various researchers interact simultaneously with the same datasets, have also lead to the design of parallel solutions in general-purpose architecture as well as in special purpose ones. The parallelization of ray-casting [MPS92] and shear-warp [Lac95a] [SL02] are particularly hot topics that have yield to numerous publications.

Ma et al. [LMC02] have applied a parallel scheme to a *2D textures* based rendering process obtaining an speedup closer to six with nine nodes. They use eight nodes to render volume slabs and the ninth node to control the process and display the final image. The volume slabs reflect an object-space task partitioning [MPHK93][Neu93] that allow a *back-to-front* (BTF) visibility ordering. As the 2D texture hardware requires to slice the volume orthogonal to each grid axis (object-aligned slices), three slabs -one aligned to each axis- are distributed to each node. The slab more orthogonal to the viewing angle is the one used. The partial images present on each graphic card frame buffer are transfered to the control node which composite them into the final image using a binary-swap method [MPHK93]. This software approach gives higher precision arithmetic to blend the slabs than the one that could be obtained using the graphics chip.

Three possible approaches to manage P-processors for rendering time-varying data sets can be performed:

- *intra-volume parallelism*: The P-processor machine is dedicated to render a single frame volume.
- *inter-volume parallelism*: P data volume are processed simultaneously. Only the processor main memory limits this approach.
- *hybrid parallelism*: It balances the two previous methods. P processors are grouped in L groups, with $L < P$. Each group renders one frame volume data. The choice of L basically depends on the size of the data set and on the type and the scale of parallel machine.

Parallel rendering of time-varying data can be analyzed by three performance metrics:

- start-up latency: time until the rendered image of the first volume appears
- overall execution time: time until the rendered image of the last volume appears

- inter-frame delay: the average time between the appearance of two consecutive rendered frames.

3.7 Temporal coherence

As mentioned in Section 1, the continuity throughout time that makes similar consecutive frames is called *temporal coherence*. According to [Sud93], two different types of temporal coherence can be defined:

- **Image-space temporal coherence** is the similarity between consecutive frame instants.
- **Object-space temporal coherence** is the similarity between the scene model states at consecutive frame instants

In the bibliography, *Image-space temporal coherence* is also called *Frame-to-frame coherence*. The distinction between these two types of coherence is key to differentiate contributions. Table 4 shows the different types of coherence used in the existing papers.

Method	Image-space based	Object-space based
[Wes95]		x
[GS01]		x
[LMC02]		x
[SCM99]	x	x
[CD99]		x
[AAW00]		x
[She98]		x
[BPRS98]		x
[NM02]		x
[SB05]		x
[WWS03]		x
[SH99]		x
[BAS02]		x
[WS03]	x	
[SJ94]		
[LCL02]		
[YS93]		x
[WSK02]		x

Table 4: Image-space temporal coherence versus object-space temporal coherence.

4 Survey

4.1 IVR methods

4.1.1 3D + time models

We next describe several methods that essentially differ in the data structure used to expedite the isosurface cell search process (search indices).

The Temporal Branch-On-Need Octree

The *Temporal Branch-On-Need Octree* (T-BON) is an extension to time-varying volume data of the BONO described in Section 3.4. It was first proposed by Sutton at al. [SH99], [SH00] in order to accelerate isosurface extraction of large dynamic sets by making an efficient use of I/O and memory. Given a voxel model with properties varying throughout time, the authors propose to construct in a pre-processing step as many BONO as key instants. These BONO are stored into disk separating the global information of the trees such as branching factors and pointers to children from the extreme values of the nodes. The global information is stored only once whereas there is a different set of extreme values for every key-instant. The basic search algorithm first loads the tree infrastructure into memory. Then, given a key-instant and an isovalue, it fetches recursively from disk the nodes of the octree corresponding to that key-instant and whose extreme values span the isovalue. The algorithm then computes the disk blocks containing data points of the selected leaf nodes into a list. Once the block list is complete, it is traversed in order to read the data blocks sequentially from disk. All the required information resides then in memory. The algorithm next proceeds as usually, by traversing the tree in order to construct the isosurface. In addition, the authors propose to use the *node bricking* strategy proposed by Chiang at al [CSS98] that consists of packing nodes into disk blocks in order to read several nodes at once and to achieve better I/O performance. The T-BON can be extended to curvilinear and non-regular grids. In the original papers, it is tested on computational fluid dynamics simulation data: regular Cartesian grids of 512^3(Rayleigh-Taylor hydrodynamic instability), 256^3 (jet shockwave), curvilinear grids of less than 100^3 (Impinging Jet) and an irregular grid composed of 10^6 tetrahedral cells (electrical simulation in a human torso). With these datasets, the T-BON with node and data bricking produces factors of improvement up to 23 in the average case. As signaled by the authors, this algorithm does not make use of temporal coherence, except from the fact that the tree structure is stored only once.

The Temporal Hierarchical Index Tree

The method presented in [She98] uses a *Temporal Hierarchical Index Tree* (THIT) which is an extension of the span-space structure, described in Section 3.4 and, therefore, falls also into the category of value partition methods. The decomposition of the value space instead of the geometric space has the advantages that it can be applied both to structured and to unstructured data and that the search space dimension is two.

The THIT is a natural extension of the span space. Given a time interval (a, b), to each cell, c_i, define its minimum and maximum values at the time instant t, min_i^t and

max_i^t, and the minimum and maximum values over time as:

$$min_a^b = MIN(min_i^t), t = a..b$$
$$max_a^b = MAX(max_i^t), t = a..b$$

Then, each cell of the dataset is associated with as much points $p_i^t = (min_i^t, max_i^t)$ as time instants. A cell is characterized by its variation over time and to quantify this variation the 2D region containing these points is subdivided into a determined number of non-uniformly spaced rectangles (lattice subdivision), see Figure 4. After this, we say that a cell has low temporal variation if all the points associated with the cell fall within an area of 2×2 lattice elements. Finally, a THIT is a binary tree data structure with nodes of the form N_i^j. The root is the node N_a^b corresponding to the initial time interval and we associate to this node all the cells that follow the mentioned low temporal variation criterion. If there are cells that cannot be associated to this node, the time interval is split in half and the tree is constructed recursively (see Figure 5).

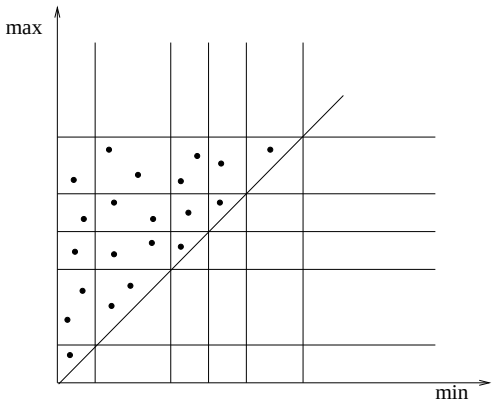

Figure 4: Example of a span-space subdivided into 5×5 lattice elements

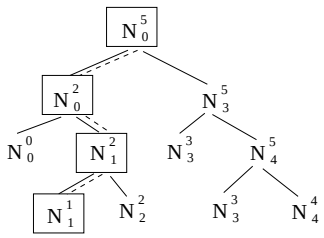

Figure 5: A temporal hierarchical index tree for a time interval [0, 5] and the traversal path corresponding to the isosurface query at time step 1

To extract an isosurface at time instant t, we first locate the nodes in the THIT that contains this time value (all the nodes N_a^b such that $a \leq t < b$).

Hybrid approach

Bajaj et al [BAS02] propose a progressive isosurface tracking of time-varying fields that divides the search in two phases, the range-space and the geometry, in order to exploit the spatial coherency as well as the value space coherency. The key idea of the algorithm is to perform a contour propagation of each connected component of an isosurface. At each key-instant, a connected component may be split, may disappear and may change shape. According to the simulation results observed average time is linear with the number of triangles.

Topological changes over time

In [SB05] the authors describe an algorithm to use temporal coherence for the topology representation of an isosurface. The authors use the *contour tree* (CT) structure [BR63], [HC03] to represent the set of contours of an isosurface and define a contour correspondence between sets of contours of two consecutive time steps based on overlapping relations between contours. The contour correspondence is computed in a preprocess and a *topology change graph* (TCG) is maintained over all time steps. The method allows segment, track, visualize and quantify the evolution of any contour and considers all the possible cases as creation and vanishing of contours as well as contour split and merge. It also allows to compute quantitative information as surface area and volume of contours. Two related works based also on the CT structure are the contour spectrum [BPS97] that manages geometric and topological information as area, volume and gradient integral and the contour plane [KRS03] that allows to manage the number of contours (as a measure of the topological changes) of a given isosurface over time.

Multiresolution representation

Gregorski et al [GSDJ04] propose a adaptively isosurface extraction based on compressed time-varying data. The authors work with large volumes, data size of about 2 TB, and so with large isosurfaces, several million triangles. Then the adaptive isocontouring is the strategy used. The refinement of a tetrahedral mesh by longest-edge bisection is used and algorithms for isosurface extraction at different levels of detail are presented.

4.1.2 4D models

Algorithms that obtain isosurfaces with a 3D+time strategy typically ignore features that may be present when data is examined in different cross-sections, and obtain poor animations through the four dimension. To resolve these problems two strategies are presented that work directly on 4D models.

Weigle and Banks [WB98] [WB96] present a recursive contour meshing to extract iso-valued features in time-varying volume data. The algorithm wants to triangulate contours in arbitrary dimensions. The idea consists in splitting the cells into simplexes, each n-cell can be transformed into different n-simplexes and then the contour from the simplexes is obtained. A square is a 2-cell, a cube is a 3-cell and a hypercube is a 4-cell and they can split into triangles that are 2-simplex, tetrahedron that are 3-simplex and hypertetrahedron that are 4-simplex, respectively. Next, the algorithm construct

the isosurface in each simplex using a recursive triangulating process applied to the various dimensional faces of the polyhedra that compose the isosurface.

Bhaniramka et al. [BWC00] provide an algorithm based on the *Marching Cubes* (MC) strategy to extract isosurfaces in four dimensions. The authors have developed a process that automatically generate the tables for any dimension and triangulating them obtains the 4D isosurfaces, which can be sliced over any time to provide smooth animation or sliced through oblique hyper-planes to examine time-evolving features in another way. A hypercube in four dimensions has sixteen vertices and using the MC strategy 2^{16} possible vertex labelings. Even considering the symmetries, there are 222 different cases. The manual analysis of all the cases is impractical and can produce errors. In addition, the generalization to higher dimensions are even more problematic. Therefore, the authors proposed an algorithm for automatically generating a lookup table that contains all the possible 2^{2^d} cases of the labelled hypercube in a d-dimensional regular grid and its triangulation for each case.

4.2 DVR methods

4.2.1 Ray-casting

Reprojection techniques

Yagel and Shi [YS93] propose a frame-to-frame coherent ray-casting that exploits the capability of skip empty space (space-leaping). It stores in a *C-Buffer* the coordinates of the first non-transparent voxel encountered by the ray emitted at each pixel. This *C-Buffer* is initialized in the first frame. The next frames can re-use this coordinates to traverse the voxel model, in the case that the camera is static. Moreover, the *C-Buffer* can be re-used if the model rotates by reprojecting the intersection points of a pixel to the new pixel. When the reprojection is finished, there are pixels that are empty, in this case, we need to start sampling to the volume boundary. Other drawback is that several coordinates can be mapped on the same new pixel, but the authors propose one solution. Algorithm 1 illustrates the pseudo code of this strategy. This approach speeds up ray casting computations when the camera or the transfer function change.

Recently, Wan et al. [WSK02] 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 [MRS+03].

Incremental ray-casting

Shen and Johnson [SJ94] 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

Algorithm 1 Yagel and Shi pseudo code for space-leaping based ray-casting algorithm

```
for all pixel (i,j) in the screen do
   [u,v,w]:=ray_enter_vol([i,j]);
   C_buffer[i,j]:=shoot_ray([u,v,w],[i,j]);
end for
for all change that requires re-rendering do
   if change is rotation then
      for all pixel(i,j) in the screen do
         [i',j']:=Transform(C_buffer[i,j]);
         add_coord(C_buffer[i,j],T_Buffer[i',j']);
      end for
      for all pixel(i,j) in the screen do
         C_buffer[i,j]:=elim_hidd(T_buffer[i,j]);
         if C_buffer[i,j] is empty then
            C_buffer[i,j]:=ray_enter_vol([i,j]);
         end if
      end for
   end if
   for all pixel(i,j) in the screen do
      C_buffer[i,j]:=shoot_ray(C_buffer[i,j],[i,j]);
   end for
end for
```

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 (see algorithm 2). 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.

Algorithm 2 Shen and Johnson algorithm based on ray-casting that exploits the temporal coherence to re-cast only those pixels that intersect with changed voxels.

```
for all time step t do
  for all changed element[x,y,z] do
    pixel_list:=corresponding_pixels(x,y,z);
    update_volume(x,y,z);
    insert_pixels(ray_cast_list,pixel_list);
  end for
  for all pixel [u,v] in ray_cast_list do
    value:=cast_ray(u,v);
    update_image(u,v,value);
  end for
  display_image();
end for
```

Liao at al. [LCL02] shares the idea of Shen and Johnson [SJ94] of recasting, at each frame, only those pixels that intersect with changed voxels. This paper proposes an improvement of [SJ94] technique to reduce the cost of calculate which pixels need to be recasted. This technique consists of computing (in a preprocessing stage) two additional differential files for each frame, called *First Order Differential file* (FOD) and *Second Order Differential file* (SOD). FOD stores the changed voxels at one frame, and SOD the exclusive-OR between two consecutive FOD (the voxels that are in the first FOD but are not in the second one, and the voxels that are not in the first one but they are in the second one). The authors propose a method to calculate the changed pixels using FOD or SOD files. The computing cost using one of both files depends of their size, so at each frame the method can choose one strategy or the other to minimize it. This technique can avoid the cost of projection the modified voxels to calculate the recasted pixels.

In [TGFP05], the time-varying dataset is encoded as a Temporal Run-Length (TRL) structure that stores for every voxel a sequence of codes composed of the property value and the number of succesive frames in which this value remains constant. The rendering algorithm based on this object-space structure is an incremental ray-casting algorithm that additionally uses a temporal image buffer storing for each pixel the next

instant of time in which the pixel must be recomputed. At every frame, the algorithm casts only rays through pixels that may change. The TRL provides then the needed information to actualize the next instant of change of the pixels corresponding to re-casted rays. The algorithm uses three different space-leaping strategies to avoid having to sample empty regions of the volume. A simultaneous version of the algorithm that computes all the images at a time, in batch, is also described. This method can handle simultaneously several data modalities because an on-purpose out-of-core strategy is used to handle large datasets.

Ray-casting isosurfaces

Reinhard et al. [RHP02] 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. During the visualization, only one file containing the given iso-surface value and time step needs to be loaded into memory. Each file contains a set of voxels represented with the coordinates (x,y,z) and its value. The creation of these files is performed in a preprocessing stage. 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.

Use of a T-BON

Ma et al. [MSSS98] explore the use of a BONO [WG94] for time-varying regular data in a 3D+time model, the T-BON, as described in Section 4.1. The construction of the tree consists of three steps: quantization of the volume, construction of a BONO for every instant of time and merging of the subtrees that are identical in successive BONOs, based on the object-space temporal coherence of the data. Quantization is the simplest lossy compression method that uses a limited number of bits to represent a larger number of raw data values. According to the data characteristics, they use three lossy quantizers: uniform, non-uniform and adaptive. The variability of the data importance is measured with an importance function based on the opacity transfer function provided by the user. In this sense, the BONO depends directly on the transfer function. If the classification parameters vary, the building process should be performed again. Next, this data structure is rendered with ray-casting by processing the first BONO completely, and only the modified subtrees of the following BONOs. To do so, an auxiliary octree, called the *compositing tree*, is constructed, similar to the BONO, that stores at each node the partial image corresponding to the subtree taking into account the image-space temporal coherence. At successive frames, when a subtree changes, its sub-image is recomputed and composited at its parent level in the hierarchy. This rendering optimization is based on a fixed viewing position. If the viewing position changes, a new compositing tree should be created. They propose to construct a complete tree at regular intervals of the possible viewing positions in order to allow random points of view. Also, they use some optimizations as *front-to-back* FTB early-termination and space-leaping with the octree codification.

In addition, in order to accelerate the visualization process, [MC00] uses supercomputers to perform rendering calculations and displays the resulting images on end

desktop computers connected over a wide-area network. They combine efficient processor management and compression to achieve near-interactive remote visualization on parallel computers. They minimize the interframe delay and analyze how processors can be partitioned in order to obtain interactivity. They evaluate several compression methods and transform mechanisms to compress the data into the supercomputer and decompress it into remote user. The rendering is obtained using a parallel ray-casting volume renderer and they analyze the optimal number of processor partitions to use into the supercomputer.

Use of a TSP

Shen et al. [SCM99] introduced a spatial octree that stores data volume capturing both temporal and spatial coherence for time-varying data. That spatial octree is called *Time-Space Partitioning tree* (TSP). To store the temporal information, each node of the TSP is a binary tree. Each node in the binary tree represents the same subvolume in the spatial domain but in a different time. Some other information is stored in those binary tree to help in the tree traversal during the volume rendering process: the mean value of voxels within the subvolume at each time and measurements of spatial and temporal errors and coefficient of variation. The TSP can be obtained in the preprocess. To perform volume rendering at run time, the TSP is first traversed to identify the subvolumes that satisfy the user query. Then the located subvolumes are rendered in the correct order to construct the final image. In fact, the authors propose a divide-and-conquer strategy. In the traversal, the nodes in the TSP are recursively visited in the FTB visibility order according to the viewing direction. Then the subvolumes that are selected are rendered independently and the final image is then constructed by compositing the partial ones. To accelerate the time-varying volume rendering, the authors propose to store the partial images of the subvolumes in each node of the TSP structure.

A 4D approach

In [WWS03] the data is treated as a 4D data field. A 4D hyperplane is defined as $ax + by + cz + dt + e = 0$ and a 4D projection can be interpreted taking into account three families of hyperplanes:

- $d = 0$. In this case the hyperplane intersects the volume at the same location for all the time steps. It shows the time evolution along a particular viewing vector.
- $d \neq 0$, $a = b = c = 0 \rightarrow dt + e = 0$. It shows the time-varying sequence at time instant $t = -e/d$
- $d \neq 0 \wedge (a \neq 0 \vee b \neq 0 \vee c \neq 0)$. In this case space is shifted by time. At each time step, we select a different slice of the volume. It shows the evolution of a feature during time.

The authors analyze several integration operators and transfer functions (alpha composition, first hit, additive projection, etc.) for the three families and implement a ray casting strategy.

4.2.2 Splatting

The work presented in [NM02] is focussed on the compression of 4D datasets. It is achieved using the more efficient sampling grid 4D BCC. This compression is almost lossless when the reconstruction is based in a Gaussian kernel and, therefore, suitable for splatting. BCC grids are a generalization of hexagonal grids and the compression rate grows with the number of dimensions (3D: 0.71, 4D: 0.5). A BCC grid can be interpreted as two cubic Cartesian grids interleaved with spacing $\sqrt{2}T$ and offset in all directions $T/\sqrt{2}$, T being the sampling distance of the cubic Cartesian grid. The data is further reduced by using a RLE for the relevant data points. The 4D data is visualized by obtaining a 3D hyperslice (3D BCC), as a list of 3D voxels, by interpolating an arbitrary hyperplane. Then shading, visibility-ordering and depth compositing is performed for any new 3D viewpoint and transfer function. The strategy applied is an image-aligned splatting [KMC99]. This approach works well for all kinds of slices, although the maximum rendering speedups (up to 30 %) due to the reduction in the dataset are mainly for slices aligned with three of the four major axes.

In [BPRS98] a 4D approach is also followed. The work is focussed on highlighting topological features (holes, cavities, extrema, etc.) and does not consider rendering techniques (occlusion, realism, etc.). It uses an octree representation of the hypervolume and follows a splatting strategy which is a generalization of that presented in [LH91]. The authors present also a graphical user interface for interacting with parallel projections of the hypervolume. The splatting strategy uses a transfer function to highlight the desired features, can deal with a multiresolution representation (octree) and takes advantage of texture mapping graphics hardware. The splatting technique used is very simple because they perform parallel projection and do not consider the ordering of voxels since they do not perform occlusion. Moreover, as the value in each voxel is assumed constant, each splat is the projection of a n-cube and the luminosity distribution can be computed exactly by using a bivariate box spline.

4.2.3 Hardware-driven texture mapping

The use of *multiple 2D textures* to exploit hardware possibilities to render a volume as slices has been used by Ma et al. [LMC02] in combination with a lossy compression in a parallel implementation. The method extracts the temporal coherence using the DCT coefficients and compressing them as described in Section 3.4. The resulting values are combined into a single number used as an index entry to a 2D palette texture. This way the value stored in each texel represents an approximation of a sequence of scalar values. The sequence is reproduced by updating the color palette at each frame, changing the entries to the color found in the transfer function for the scalar encoded by its index (see Algorithm 3).

The number of time-varying values collapsed into a single one is called *window*. Each window stores the average of the sequence values, which produces image artifacts when changing from a window to the next one. To avoid all transition artifacts occur at once in the whole volume, the starting times of the windows for each slice

Algorithm 3 Time-varying encoding changing color palette

```
Color transf_func[256]; // LUT from the transfer function
Color palette[256]; // computed color palette for each time step
int decoder[N][256]; // N time varying scalars encoded by each
// of 256 possible texel values created by the compression process
for all timestep t (0..N-1) do
   for all palette entry i (0..255) do
      palette[i]=transf_func[decoder[t][i]];
   end for
   setCurrentFramePalette(palette);
   renderTexture();
end for
```

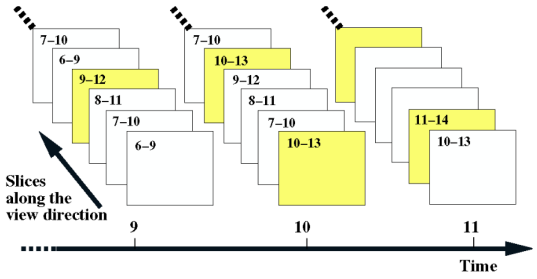

Figure 6: Window interleaving: Every time step only some of the volume slices textures are replaced to avoid general transition artifacts. The numbers on each slice indicate which time steps the texture stores.

are interleaved as shown in the Figure 6. This technique is also useful to avoid loading a whole volume at certain time steps, which would require storing a copy of the next volume in texture memory. Using window interleaving, only some textures are flushed to be replaced by new ones at each time step. The process is applied to each volume axis-aligned slice independently. Thus, the selected slices are those ones more orthogonal to the viewing direction.

A sequential and parallel implementation (see Section 3.6) has been tested on high variability simulation data sets.

S. Guthe and W. Strasser [GS01] used 2D and 3D textures, both of them exploiting temporal coherence in a lossy compression context on real low variable data sets. The compressed encoding is performed using the wavelet transform as explained in Section 3.4. The 3D texture visualization is performed directly by hardware in an ATI Radeon graphics card that supports this feature, in combination with shells. The shells are small sphere subsections useful to better approximate the different distances between texture slices as seen in Figure 7. On the other hand, as the Nvidia GeForce used by the authors didn't support 3D textures, they performed a tri-linear interpolation between any two slices as described by Rezk-Salama et al. [RSEB$^+$00] to approximate these

distance differences. They use 2D textures to perform a shear-warp rendering schema as can be seen in the next section.

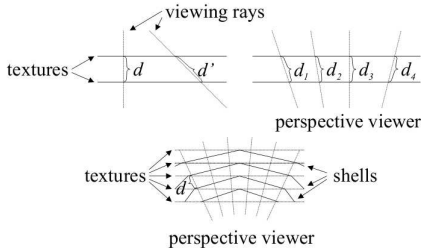

Figure 7: Different distances *d* between textures introduced by different viewing points or perspective projection.

The TSP tree, explained in Section 4.2.2, has also been used to speed-up texture-based rendering [ECS00]. 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 fast volume rendering is achieved by rendering a combination of flat-shaded and solid-textured polygons. Flat-shaded polygons represent regions that have a high spatial coherence compacted on the TSP tree codification. Solid-textured polygons represent regions having high variation, both in spatial and temporal domains. The overhead of generating additional slicing planes in the subvolumes for the texture generation is reduced by an incremental slicing algorithm similar to the standard scan-line polygon rasterization algorithm. The rendering time is also reduced loading and retaining in memory the textures that have to be renderered in next steps.

4.2.4 Shear-warp

Anagnostou et al. [AAW00] proposes an approach for rendering time-varying data based on the Shear-warp technique. The work is applied to volume datasets where only a relatively small part of the volume changes, that is, volume data with a high coherence in time, and treats separately the time dimension from the spatial one. In fact, the work wants to exploit the existence of spatial and temporal coherence using an adapted RLE of the volume. There is a preprocessing stage that detects, encodes and saves to disk the changes over time. Next, the rendering stage loads the changed areas from the disk, updates the volume and renders only the changed area. The pre-processing stage is viewpoint independent, while the rendering stage is not.

To exploit the spatial coherence the volume data is compressed using a RLE. This data structure performs well in the case of a single volume, but becomes inefficient with multiple time-varying data. The authors propose an additional structure that expands the RLE encoded volume to its previous dimensions in space, preserving the run-length information. When a change is detected over time, the expanded RLE is updated by properly inserting the modified runs in the volume scan-line.

To exploit the temporal coherence the authors based their work on the property called partial ray compositing, that means that the final intensity obtained for each ray can be derived compositing different partial intensities. In volume data when a great percentage of the volume remains unchanged, the size of the changed area is small compared to the size of the column that needs re-rendering. The authors propose the notion of a *slab*, that is a thick slice. According to the partial ray paradigm, they split the ray into a number of rays each of which composites voxels belonging to a slab. The partial image produced in this way, that is the intensities corresponding to each slab, are stored in another data structure. Then only slabs that are occupied by the changed area need to be recomputed and compositing them with the unchanged ones.

Clyne and Dennis [CD99] present a volume rendering system based on a parallel implementation of Shear-warp algorithm. They extend an existing serial implementation of the algorithm in the VolPack library. This package operates on preprocessed RLE data volumes, that computes the view-independent opacity and gradient information of the data samples. Shading is performed using a shade tree that can be implemented as a lookup table simulating the Phong lighting equations. The cost of the rendering time is the cost of the projection of the volume into the baseplane image and the cost of warping the baseplane image into the final image. The authors propose an strategy for parallelize each of these computational phases. The baseplane image is partitioned into small groups of contiguous scanlines. Each processor computes the final image for a group of scanlines until the whole final image is obtained. To render time-varying data uses a double-buffer strategy that keeps two copies of data in memory, one which is rendered in fact and the other one which is computed.

As mentioned in previous section, S. Guthe and W. Strasser [GS01] used 2D textures to implement a Shear-warp algorithm, exploiting temporal coherence in a lossy compression context. The compressed encoding is performed using the wavelet transform as explained in Section 3.4. The basis is to render an axis-aligned 2D texture stack, the one best aligned with the view direction. The angle correction is performed using the register combiner feature that lets to modify the opacity depending on the angle between the normal and vision vectors. This technique produces artifacts when switching from one stack to another. So, they render the three axis-aligned texture stacks and combine the results. The problem is that it consumes more time. An alternative using directly 3D textures is presented (see Section 4.2.3).

4.3 Other uses

4.3.1 Tranfer functions for Time-Varying Data

Designing transfer functions for time-varying data sets is a new problem in volume visualization that is often ignored. In an early work, called the Contour Spectrum ([BPS97]), the user can display the changes of the underlying contour functions over time. The contour spectrum is a statistical signature consisting of a variety of scalar data and contour attributes, computed over the range of scalar values. Isovalues of in-

terest can be chosen manually from different times and contour functions values. The result is a set of opacity maps over time. [JKM01] studies different semi-automatic approaches for transfer function generation in order to obtain a single or minimal set of informative opacity transfer functions for a time-varying data-sets. They propose two types of methods that obtain opacity functions: *summary-functions* and *summary-volume*. The first one consists of algorithms which analyze each frame separately, create a transfer function, and then try to combine these transfer functions into a summary function (in a similar way as 3D+time). To combine the transfer functions, they use four methods based on the opacity values: single representative, average, union (maximum opacity) and coherency-based. This summary function is used during rendering of all frames.

Single representative methods select a single frame and apply it to the others. Coherency-based methods utilize the coefficient of variation (COV_{st}) metric, previously explained in the section 2.3. If the COV value of a time interval is less than a certain threshold value, it means that the values do not change significatively over time and then, the opacity values average is used over the interval. If the COV value is high, the interval is subdivided and the COV of the subintervals is computed. The final value is computed from values of the subinterval of lowest COV value.

The other class of techniques, *summary-volume based* techniques, does not ignore the temporal dimension and operates upon the entire set of volumes to generate a transfer function. It creates a single volume that combines all frames and then they extract a transfer function from this volume. To create the summary volumes based on the properties values, the methods applied are: averaging and coherency. Again, this function is used during the rendering stage.

As in data sets with periodic motion with non-regular boundaries, these techniques do not obtain satisfactory images, they propose a method which generates a set of transfer functions based on the coherency-based methods of summary-functions (*multiple transfer functions*). For each value data in the transfer function, a traversal of time intervals is performed in order to return a set of time intervals with the COV value less than a specified threshold with its mean opacity. Then, for each obtained time interval, the midpoint time is calculated and used to create a transfer function. Each value of these transfer functions of the midpoint time is the average value calculated previously in the corresponding value time interval.

4.3.2 Volume animation based on skeletons

Some volumetric applications such as volume morphing, object simplification, physically-based deformations and animations automatic path navigations and volume modelling can be improved using volume based skeletons. [NKHS98] proposes a technique to animate volumes using a Skeleton Tree. In order to animate a volume model, they extract a skeleton that captures the essential topology of the volume object to be animated. This skeleton is a medial axis as it is defined as the central points of the object and the distance of these points to the surface. They use the approximated distance based

on the (3,4,5)-weighted metric. Moreover, they can obtain different kinds of skeletons varying the values of a thinness parameter, that graduates the density of the skeleton in order to achieve finer control of the deformation. In any case, this parameter value doesn't guarantee the skeleton connectivity.

Once the skeleton is extracted, an automatic connection process is performed based on the spatial coherence and the distance value coherence of the skeleton points. The skeleton voxels define the set of nodes of a weighted undirected graph and they are fully connected. Edge weights are based on the degree of spatial and distance value coherences. Then, the minimum spanning tree of the graph defines the Skeleton Tree. The next step is to define, over the skeleton tree, constraints for deformation (varying the distances) and motion (changing the point locations). In [NS01], the authors propose the use of Alias Wavefront to interact with the Skeleton Tree.

The final step is to reconstruct the final deformated and animated volume object from the skeleton information. Using the distance metric and using bounding boxes from the distance transform, they can approximate the final surface of the volume object.

4.4 Rendering in the transformed space

In [Wes95] R.Westermann computes the volume rendering integral [DCH88] [KH84] [Lev88] directly over the transformed space instead of decompressing data before performing the render. First, at each time step, the volume is transformed and compressed using the wavelet transform (see Section 3.4). Second, the data is analyzed to detect the low and high frequency regions, to distinct those which remain constant over time from the high fluctuating parts called *focus of interest*. Third, low frequency regions are reconstructed with larger error tolerance than high ones. The rendering process is performed tracing rays on the multiscale representations of the original three dimensional signal as had been done before in [GLDK95][Mur93][Wes94], but taking profit of temporal coherence. They successfully tested this method over high variable simulation data sets.

5 Conclusions

In this report, we have presented a state-of-the-art on time-varying volume rendering. Since most of the papers focus on speeding up rendering, it is actually difficult to compare them and to be able to determine if one is more efficient than another, because the datasets used are different, the computers also and the efficiency strongly depends on implementation details.

In order to conclude this report, we next describe open problems in time-varying volume rendering that we believe can be future research lines. Different changes can occur throughout time camera movement, lighting conditions, user preferences, model movement, deformation and property values variation. Some papers focusing

on navigation around volume models [YS93], others on navigation through volumes [BJNN98] and others address *Time-varying volume rendering*, i.e rendering static volumes with varying property values. No specific algorithms for interactive navigation around and inside a time-varying volume model have been yet proposed. In this report, we have focused on time-varying volume rendering strategies.

Existing papers assume that the models are segmented. Whether the segmentation procedure is the same or not for all the models or must be adapted throughout time, this is not addressed in volume rendering bibliography. Moreover, except the paper [JKM01], existing works assume that the classification function is valid for all the time span. Since this assumption may not be true if the variability of the data is high, more research efforts should be put on classification of time-varying data.

There are two main approaches to time-varying volume models [Neo03]: to consider them as particular cases of *n*-dimensional models or to treat them as temporal sequences of 3D models. In the former case, research focuses on selecting as efficiently as possible the relevant information of the 4D model given a user query: for example, given an isosurface value and a key-instant, how to efficiently extract the surface visiting as directly as possible the set of isosurface cells. Papers of the latter category focus on generating a sequence of frames at a set of key-instants, either by extracting an isosurface at a series of instants in order to animate them (IVR) or by rendering the models directly (DVR). The problem that these papers address is how to improve the efficiency of the rendering algorithms. As pointed out by Woodring and Shen [WS03], other types of visualizations than sequences of frames, may provide different ways of analyzing time-varying data. This is an open research field, almost absent in the current bibliography, that would demand imaginative solutions. In order to speed up IVR as well as DVR, data structures already used in static volumes have been extended to time-varying data: TSPs extend octrees [SCM99], T-BONs [MSSS98] extend BONOs, THITs [She98] extend the span-space structure and *temporal run length* (TRLs) [TGFP05] extend the RLE. Some of these data structures have been used to efficiently manage the information without actually exploiting the coherence between successive frames. The performance of the algorithms could benefit from the use of this type of coherence. Other data structures such as the EVM [RAA04], the distance transforms [ZKV92] and the Render Lists [HBH03] may be also extensible to 3D+time. Moreover, new data structures, specifically designed for time-varying data could be explored.

Most of the DVR methods focus on ray-casting, probably because of its simplicity and flexibility. In the future, much efforts should be put in hardware-driven rendering that is nowadays one of the techniques more used to visualize static volumes [MGS02] and splatting which is emerging as a flexible technique to render point-based volume and surface models [CRZP04]. Moreover, the combination of various rendering techniques such as the two-level rendering proposed in [HBH03] may be adapted to time-varying data.

Another important aspect of time-varying volume rendering is the type of application in which it is used. As mentioned by Ma [Ma03], scientific simulations may need

strategies such that the data are visualized while they are being generated without need of storing a 4D or a 3D+time model.

Finally, the existing bibliography addresses unimodal rendering. The combination of multiple modalities, along with the merging of surface and volume models has been treated for static models only. Another open research line is to propose efficient rendering methods able to deal with multiple modality some or all of them varying throughout time and hybrid surface/volume time-varying scenes.

A Acronyms

BCC: Body centered cubic
BONO: Branch-on-need-octree
BTF: Back-to-front
CC: Cubic Cartesian
CFD: Computational Fluid Dynamics
COV: Coefficient of variation
CT: Contour tree
DCT: Discrete cosine transform
DVR: Direct volume rendering
EVM: Extreme vertices model
FFT: Fast Fourier transform—
FTB: Front-to-back
IVR: Indirect volume rendering
LZW: Lempel, Ziv, and Welch (authors)
RLE: Run length encoding
T-BON: Temporal branch-on-need
TCG: Topology change graph
THIT: Temporal Hierarchical Index Tree
TRL: Temporal run length
TSP: Time-Space Partitioning tree
UFAC: Unsteady Flow Advection Convolution

References

[AAW00] 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.

[Bad88] S. Badt. Two algorithms for taking advantage of temporal coherence in ray tracing. *The Visual Computer*, 4:55–64, 1988.

[BAS02] C. Bajaj, A.Shamir, and B. Sohn. Progressive tracking of isosurfaces in time-varying scalar fields. Technical Report Tr02-44, University of Texas, 2002.

[BJNN98] M. L. Brady, K. K. Jung, H. Nguyen, and T. Nguyen. Interactive volume navigation. *IEEE Transactions on Visualization and Computer Graphics*, 4(3):243–256, 1998.

[BKX90] R. Bakalash, A. Kaufman, and Z.-Y. Xu. Cube: Building a full scale VLSI-based volume visualization system. *5th EG Workshop on Graphics Hardware*, September 1990.

[BP01] G. Besuievsky and X. Pueyo. Animating radiosity environments through the multi-frame lighting method. *Journal of Visualization and Computer Animation*, pages 93–106, 2001.

[BPRS98] 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.

[BPS97] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel Schikore. The contour spectrum. In *IEEE Visualization*, pages 167–174, 1997.

[BR63] R.L. Boyell and H. Ruston. Hybrid techniques for real-time radar simulation. In *IEEE Proceedings Fall Joint Computer Conference'63*, pages 445–458, 1963.

[BSG+01] D. Bartz, W. Strasser, O. Gurvit, D. Freudenstein, and M. Skalej. Interactive and multi-modal visualization for neuroendoscopic interventions. *VisSym 2001*, page 9, 2001.

[BWC00] 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.

[CCD90] J. Chapman, T.W. Calvert, and J. Dill. Exploiting temporal coherence in ray tracing. *Proc. Graphics Interface*, 1990.

[CD99] J. Clyne and J. Dennis. Interactive direct volume rendering of time-varying data. *datVisualization'99*, pages 109–120, 1999.

[CFM+94] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution volume dataset modeling and visualization based on simplicial complexes. *ACM Proc. 1994 Symposium on Volume Visualization*, pages 19–26, October 1994.

[CMM+97] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):158–170, 1997.

[CRZP04] Wei Chen, Liu Ren, Matthias Zwicker, and Hanspeter Pfister. Hardware-accelerated adaptive ewa volume splatting. In *VIS '04: Proceedings of the IEEE Visualization 2004 (VIS'04)*, pages 67–74. IEEE Computer Society, 2004.

[CS97] Y. Chiang and C. Silva. I/o optimal isosurface extraction. *Proceedings of IEEE-Visualization'97*, pages 293–300, 1997.

[CSS98] Y. Chiang, C. Silva, and W. Shroeder. Interactive Out-Of-Core isosurface extraction. *Proceedings of Visualization'98*, pages 167–174, 1998.

[CT99] S. Coorg and S. Teller. Temporally coherent conservative visibility. *Comput. Geom. Theory Appl.*, 12(1-2):105–124, 1999.

[DCH88] R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *ACM Computer Graphics*, 22(4):65–74, August 1988.

[DH92] J. Danskin and P. Hanrahan. Fast algorithms for volume ray-tracing. *1992 Workshop on Volume Visualization*, pages 91–106, 1992.

[ECS00] 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.

[Ede80] H. Edelsbrunner. Dynamic data structures for orthogonal intersection queries. Technical Report F59, Tecn. University Graz, 1980.

[Gar82] I. Gargantini. Linear octrees for fast processing of three-dimensional objects. *CVGIP*, 4:363–374, 1982.

[Gla89] A. Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989.

[GLDK95] M. Gross, L. Lippert, A. Dreger, and R. Koch. A new method to approximate the volume rendering equation using wavelets and piecewise polynomials. *Computers and Graphics*, 19(1), 1995.

[GPR+94] T. Guenther, C. Poliwoda, C. Reinhard, J. Hesser, R. Maenner, H.P. Meinzer, and H.J. Baur. Virim: A massively parallel processor for real-time volume visualization in medicine. In *Proceedings of the 9th Eurographics Workshop on Graphics Hardware*, pages 103–108, 1994.

[GS01] 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.

[GSDJ04] B. Gregorski, J. Senecal, M.A. Duchaineau, and K.I. Joy. Adaptive extraction of time-varying isosurfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):683–694, 2004.

[GW92] R.C. Gonzalez and R.E. Woods. Digital image processing. *Addison-Wesley*, 1992.

[GWP91] E. Grller and W-Purgathofer. Using temporal and spatial coherence for accelerating the calculation of animation sequences. *Eurographics'91*, pages 103–113, 1991.

[HBH03] M. Hadwiger, C. Berger, and H. Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *VIS '03: Proceedings of the conference on Visualization '03*, pages 40–45. IEEE Computer Society Press, 2003.

[HBS03] 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.

[HC03] U. Axen H. Carr, J. Snoeyink. Computing contour trees in all dimensions. *Computational Geometry: Theory and Applications*, 24(2):75 – 94, 2003.

[HDM03] 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.

[HKW+95] L. Hong, A. Kaufman, Y.C. Wei, A. Viswambharan, M. Wax, and Z. Liang. 3D virtual colonoscopy. *IEEE Computer Graphics & Applications*, pages 26–32, 1995.

[HZ82] 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.

[Jai89] A.K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

[JKM01] T.J. Jankun-Kelly and Kwan-Liu Ma. A study of transfer functions generation for time-varying volume data. In Klaus Mueller and Arie Kaufman, editors, *Proceedings of the Joint IEEE TCVG and Eurographics Workshop on Volume Graphics 2001*, pages 51–68. IEEE TCVG and Eurographics, Springer-Verlag, 2001.

[J.L75] J.L.Bentley. Multidimensional binary search trees used for associative search. *Communications ACM*, 19(9):509–516, 1975.

[Kau90] A. Kaufman. *Volume Visualization*. IEEE Computer Society Press, August 1990.

[KB88] A. Kaufman and R. Bakalash. Memory and processing for 3D voxel-based imagery. *IEEE Computer Graphics & Applications*, 8:10–23, November 1988.

[KH84] J. Kajiya and B. Von Herzen. Ray tracing volume densities. *ACM Computer Graphics*, 18(3):166–174, July 1984.

[KK99] K. Kreeger and A. Kaufman. Mixing translucent polygons with volumes. *Proc. IEEE Visualization*, pages 191–198, 1999.

[KMC99] J. Huang K. Mueller, N. Shareef and R. Crawfis. High-quality splatting on rectilinear grids with efficient culling of occluded voxels. *IEEE Trans. Vis. and Comp. Graph.*, 5(2):116–134, 1999.

[KRS03] Lutz Kettner, Jarek Rossignac, and Jack Snoeyink. The safari interface for visualizing time-dependent volume data using iso-surfaces and contour spectra. *Comput. Geom. Theory Appl.*, 25(1-2):97–116, 2003.

[Lac95a] P. Lacroute. Fast volume rendering using a Shear-Warp factorization of the viewing transformation. Technical report, Departments of Electrical Engineeering and Computer Science, Stanford University, September 1995.

[Lac95b] P. Lacroute. Real-time volume rendering on shared memory multiprocessors using the Shear-Warp factorization. *ACM Symposium on Parallel Rendering*, pages 15–23, 1995.

[LC87] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *ACM Computer Graphics*, 21(4):163–169, July 1987.

[LCL02] 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.

[Lev88] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8:29–37, May 1988.

[Lev90] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[LH91] D. Laur and P. Hanrahan. Hierarchical splatting: A progressive refinement algorithm for volume rendering. *ACM Computer Graphics*, 25(4):285–318, July 1991.

[LL94] 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.

[LMC02] 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.

[LSJ96] Y. Livnat, H.W. Shen, and C.R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1), March 1996.

[Ma03] K.L. Ma. Visualizing time-varying volume data. *Computing in Science and Engg.*, 5(2):34–42, 2003.

[MC00] 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.

[MGS02] M. Meissner, S. Guthe, and W. Strasser. Interactive lighting models and pre-integration for volume rendering on pc graphics accelerators. In *Proceedings of Graphics Interface 2002*, pages 209–218. IEEE Press, 2002.

[MHB+00] M. Meissner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical evaluation of popular volume rendering algorithms. *Proc. Volume Visualization 2000*, pages 81–91, 2000.

[MHT+96] K. Mori, J. Hasegawa, J. Toriwaki, H. Anno, and K. Katada. A fast rendering method using the tree structure of objects in virtualized bronchus endoscope system. *Proc. Visualization in Biomedical Computing*, pages 33–42, September 1996.

[[MP93] Mpeg-1 coding of moving pictures and associated audio for digital storage media at up to about 1,5 mbit/s, 1993.

[[MP96] Mpeg-2 generic coding of moving pictures and associated audio information, 1996.

[MPHK93] Kwan Liu Ma, James S. Painter, Charles D. Hansen, and Michael F. Krogh. A data distributed, parallel algorithm for ray-traced volume rendering. In *PRS '93: Proceedings of the 1993 symposium on Parallel rendering*, pages 15–22, New York, NY, USA, 1993. ACM Press.

[MPS92] C. Montani, R. Perego, and R. Scopigno. Parallel rendering of volumetric dataset on a hypercube architecture. *Proc. 1992 ACM Workshop on Volume Visualization*, pages 9–16, 1992.

[MPT03] 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.

[MRS+03] M. Martin, E. Reinhard, P. Shirley, S. Parker, and W. Thompson. Temporally coherent interactive ray tracing. *Journal of Graphics Tools*, 1(72):41–48, 2003.

[MSSS98] 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.

[Mur93] S. Muraki. Volume data and wavelet transform. *IEEE Computer Graphics & Applications*, 13(4):50–56, July 1993.

[Neo03] N. Neophytou. Hardware-assisted volume rendering of time-varying data. Technical report, Stony Brook University, 2003.

[Neu93] Ulrich Neumann. Parallel volume-rendering algorithm performance on mesh-connected multicomputers. In *PRS '93: Proceedings of the 1993 symposium on Parallel rendering*, pages 97–104, New York, NY, USA, 1993. ACM Press.

[NKHS98] N.Gagvani, D. Kenchammana-Hosekote, and D. Silver. Volume animation using the skeleton tree. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 47–53, New York, NY, USA, 1998. ACM Press.

[NM02] N. Neophytos 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.

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

[PHK+99] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The volumepro real-time ray-casting system. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 251–260, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[PK96] H. Pfister and A. Kaufman. Cube-4 - a scalable architecture for real-time volume rendering. *ACM/IEEE Symposium on Volume Visualization*, pages 47–54, October 1996.

[PKC94] H. Pfister, A. Kaufman, and T. Chiueh. Cube-3: A Real-Time Architecture for High-Resolution Volume Visualization. In In *1994 Workshop on Volume Visualization*, pages 75–83, Washington, DC, October 1994.

[PPL+99] 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.

[PTN97] A. Puig, D. Tost, and M.I. Navazo. An interactive cerebral blood vessel exploration system. *ACM Visualization'97*, 1997.

[RAA04] J. Rodríguez, D. Ayala, and A. Aguilera. *Geometric Modeling for Scientific Visualization*, chapter EVM: A Complete Solid Model for Surface Rendering, pages 259 – 274. Springer, 2004.

[RAG05] J. Rodríguez, D. Ayala, and S. Grau. VolumeEVM: A new approach for surface/volume integration. *Computers & Graphics*, 29(2):217 – 224, april 2005.

[RHP02] 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.

[RSEB+00] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 109–118, New York, NY, USA, 2000. ACM Press.

[Say00] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 2000.

[SB05] B. Sohn and C. Bajaj. Time-varying contour topology. *IEEE Transactions on Visualization and Computer Graphics*, to appear, 2005.

[SCES02] C.T. Silva, Y. Chiang, and J. El-Sana. Out-of-core algorithms for scientific visualization and computer graphics. *IEEE Visualization'02. Course Notes.*, pages 1–30, 2002.

[SCM99] 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.

[SF90] K.R. Subramanian and D.F. Fussell. Applying space subdivision techniques to volume rendering. *Proc. Visualization'90*, pages 150–159, 1990.

[SGS95] G. Sakas, M. Grimm, and A. Savopoulos. Optimized maximum intensity projection (mip). *6th Eurographics Workshop on Rendering*, pages 81–93, June 1995.

[SH94] B. Stander and J. Hart. A lipschitz method for accelerated volume rendering. *Proc. 1994 Symposium on Volume Visualization*, October 1994.

[SH99] P. Sutton and C. Hansen. Isosurface extraction in time-varying field using a temporal branch-on-need tree (t-bon). *Proceedings of Visualization'99*, pages 147–153, 1999.

[SH00] P. Sutton and C. Hansen. Accelerated isosurface extraction in time-varying field. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):98–107, 2000.

[She98] Han-Wei Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In *Proceedings of the conference on Visualization '98*, pages 159–166. IEEE Computer Society Press, 1998.

[SJ94] 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.

[SL02] J.P. Schulze and U. Lang. The parallelization of the perspective shear-warp volume rendering algorithm. In *EGPGV'02: Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pages 61–69, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[Sud93] O. Sudarsky. Exploiting temporal coherence in animation rendering - a survey. *Technical Report CIS9326*, 1993.

[TGFP05] D. Tost, S. Grau, M. Ferré, and A. Puig. Ray-casting time-varying volume data sets with frame-to-frame coherence. *Report LSI-05-20. Submitted to IEEE Visualization 2005*, 2005.

[TMG01] T. Theussl, T. Möller, and E. Gröller. Optimal regular volume sampling. *Proc. Visualization'01*, pages 91–98, 2001.

[Tos91] D. Tost. An algorithm of hidden surface removal based on frame-to-frame coherence. *Proceedings of Eurographics'91*, pages 261–273, September 1991.

[UO93] J.K. Udupa and D. Odhner. Shell rendering. *IEEE Computer Graphics & Applications*, pages 58–67, November 1993.

[WB96] Chris Weigle and David C. Banks. Complex-valued contour meshing. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 173–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[WB98] C. Weigle and D.C. Banks. Extracting iso-valued features in 4-dimensional scalar fields. *Proc. Symposium on Volume Visualization*, pages 103–110, October 1998.

[WE98] B. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. *Proceedings of SIGGRAPH '98*, pages 169–178, 1998.

[WEE03] D. Weiskopf, K. Erlebacher, and T. Ertl. A texture-based framework for spacetime coherent visualization of time-dependent vector fields. In *IEEE Visualization 2003*, pages 107–114, 2003.

[Wes90] L. Westover. Footprint evaluation for volume rendering. *ACM Computer Graphics*, 24(4):367–376, August 1990.

[Wes94] R. Westermann. A multiresolution framework for volume rendering. In *ACM Workshop on Volume Visualization*, pages 51–58, 1994.

[Wes95] 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.

[WG92] J. Wilhems and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[WG94] 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.

[WKB+02] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek. Interactive global illumination. *Proceedings of the 13th Eurographics Workshop on Rendering*, pages 15–24, 2002.

[WS03] 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.

[WSK02] 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.

[WTK+99] Ming Wan, Qingyu Tang, Arie Kaufman, Zhengrong Liang, and Mark Wax. Volume rendering based interactive navigation within the human colon (case study). In *VIS '99: Proceedings of the conference on Visualization '99*, pages 397–400, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[WWS03] 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.

[YS93] 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.

[ZA77] J. Ziv and A. Lempel. A. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 1977.

[ZKV92] K.J. Zuiderveld, A.H.J. Koning, and M.A. Viergever. Acceleration of ray-casting using 3d distance transforms. *In Proceedings of SPIE1808 Visualization in Biomedical Computing*, pages 324–335, 1992.

Method		[LMC02]	[GS01]	[Wes95]	[WB98]
Data sets	Data type	Simulation	Images	Simulation	Simulation
	Sampling	Regular	Regular	Regular	Regular
	Grid	Uniform	Uniform	Uniform	Uniform
	Variability	High	Low	High	-
Data Model	Type	Voxels	Voxels	Voxels	Voxels
	Domain	Spatial	Spatial	Frequential	Spatial
	Temporal	3D + time	3D + time	3D + time	4D model
	Compression	lossy	lossy	lossy	no
Visual.	Type	Direct	Direct	Direct	Indirect
	Camera	Dynamic	Dynamic	Static	-
	Transfer Function	Dynamic	Dynamic	Dynamic	-
	Illumination	Dynamic	Static	Static	-
	Rendering	Texture based	Tex & Shear	Raycasting	IVR
Main Contribution		DCT&Tex	Wavelets&Tex	Comp.domain	Rec contour mesh
Optimization					
Applications					Isosurf

Table 5: Classification of all the references surveyed.

Method		[GSDJ04]	[She98]	[BPRS98]	[NM02]
Data sets	Data type	Simulation	Simulation	Simulation	Simulation
	Sampling	Regular	Irregular	Regular	Regular
	Grid	Uniform	Non-uniform	Uniform	Uniform
	Variability	High	High	High	High
Data Model	Type	Voxels	Points	Voxels	Voxels
	Domain	Spatial	Spatial	Spatial	Spatial
	Temporal	3D+time	3D+time	nD	4D
	Compression	lossy	-	-	lossless-uniform
Visual.	Type	Indirect	Indirect	Direct	Direct
	Camera	-	Static	Static	Static
	Transfer Function	-	-	Static	Static
	Illumination	-	Static	Static	Static
	Rendering	IVR	IVR	Splatting	Splatting
Main Contribution		Adapt surf	THIT	nD Splat	BCC grid
Optimization			-	-	-
Applications		Isosurf	CFD	5D molecular interact	-

Table 6: Classification of all the references surveyed (more)

Method		[LCL02]	[RHP02]	[SH99]	[ECS00]
Data sets	Data type	Simulation	Simulation	Simulation	Simulation
	Sampling	Regular	Regular	Regular	Regular
	Grid	Uniform	Uniform	Uniform	Structured
	Variability	High	High	High	Static/Slow
Data Model	Type	Voxels	Voxels	Voxels	Voxels
	Domain	Spatial	Spatial	Spatial	Spatial
	Temporal	3D+time	3D+time	3D+time	
	Compression	losless	losless	-	losless
Visual.	Type	Direct	Direct	Indirect	Direct
	Camera	-	-	-	Dynamic
	Transfer Function	-	-	-	
	Illumination	-	-	-	Dynamic
	Rendering	RayCasting	RayCasting	Indirect	Texture based
Main Contribution		pixel coherency	Isosurface rend	Temporal BON	TSP texture-based
Optimization					GPU-based and incremental slicing
Applications		Time-varying vis.		IsoSurface extr	CFD

Table 7: Classification of all the references surveyed (more)

Method		[CD99]	[SCM99]	[AAW00]	[BWC00]
Data sets	Data type	Simulation	Simul&Images	Phantom	Images
	Sampling	Regular	Regular	Regular	Regular
	Grid	Uniform	Uniform	Uniform	Uniform
	Variability	High	High	Low	-
Data Model	Type	Voxels	Voxels	Voxels	Voxels
	Domain	Spatial	Spatial	Spatial	Spatial
	Temporal	3D+time model	3D+time model	3D+time model	4D model
	Compression	no	no	no	no
Visual.	Type	Direct	Direct	Direct	Indirect
	Camera	Dynamic	Static	Static	-
	Transfer Function	Static	-	-	-
	Illumination	Dynamic	-	-	-
	Rendering	Shear-warp	Raycast/all	Shear-warp	IVR
Main Contribution		Parallel implem.	TSP tree	Time-var shear	Time-var MC
Optimization					
Applications		Visualiz	Visualiz	Visualiz	Isosurf

Table 8: Classification of all the references surveyed (more)

Method		[YS93]	[SJ94]	[WSK02]
Data sets	Data type	-	Simulation	Medical
	Regular	Regular	Regular	Regular
	Grid	Uniform	Uniform	Uniform
	Variability	Static	Low	Static
Data Model	Type	Voxels	Voxels	Voxels
	Spatial	Spatial	Spatial	Spatial
	Temporal	3D	3D+time	3D
	Compression	Null	losless	Null
Visual.	Type	Direct	Direct	Direct
	-	Dynamic	Static	Dynamic
	Transfer Function	-	-	-
	Illumination	-	-	-
	Rendering	RayCasting	RayCasting	RayCasting
Main Contribution		Re-use Space-Leaping	Pixel Coherence	Re-use Space-Leaping
Optimization		model traverse	-	model traverse
Applications		Fly-around navigation	Time-varying vis.	Fly-through

Table 9: Classification of all the references surveyed (more)

Method		[SB05]	[WWS03]
Data sets	Data type	Simulation	Simulation
	Sampling	Irregular	Regular
	Grid	Non-uniform	Uniform
	Variability	High	High
Data Model	Type	Tetrahedra	Voxels
	Domain	Spatial	Spatial
	Temporal	3D + time	4D
	Compression	-	-
Visual.	Type	I	D
	Camera	Static	Static
	Transfer Function	Static	Dynamic
	Illumination	Static	Static
	Rendering	IVR	ray-casting
Main Contribution		Time-var contours	4D proj interpret
Optimization		-	-
Applications		Hemoglobin dynam	CFD

Table 10: Classification of all the references surveyed (the end)