An efficient clustering method for fast rendering of time-varying volumetric medical data

Zhenlan Wang · Binh P. Nguyen · Chee-Kong Chui · Jing Qin · Chuan-Heng Ang · Sim-Heng Ong

Abstract Visualization and exploration of time-varying volumetric medical data help clinicians for better diagnosis and treatment. However, it is a challenge to render these data in an interactive manner because of their complexity and large size. We propose an efficient clustering method for fast compression and rendering of these large dynamic data. We divide the volumes into a set of blocks and use a BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) based algorithm to cluster them, which can usually find a high quality clustering with a single scan of the blocks. In addition, the granularity of clusters can be adaptively adjusted by dynamically configuring threshold values. In each cluster of blocks, a *KeyBlock* is generated to represent the cluster, and therefore the storage space of the volumes is reduced greatly. In addition, we assign a lifetime to every *KeyBlock* and implement a dynamic memory management scheme to further reduce the storage space. During the rendering, each *KeyBlock* is rendered as a *KeyImage*, which can be reused if the view transformation and transfer function are not changed. This reuse can help to increase the rendering speed significantly. Experimental results showed that the proposed method can achieve good performance in terms of both speed optimization and space reduction.

Z. Wang · C.-K. Chui
Department of Mechanical Engineering, Faculty of Engineering, National University of Singapore, Singapore, Singapore

Z. Wang
e-mail: mpewzl@nus.edu.sg

C.-K. Chui
e-mail: mpecck@nus.edu.sg

B.P. Nguyen (✉) · S.-H. Ong
Department of Electrical and Computer Engineering, Faculty of Engineering, National University of Singapore, Singapore, Singapore
e-mail: phubinh@nus.edu.sg

S.-H. Ong
e-mail: eleongsh@nus.edu.sg

J. Qin
Department of Diagnostic Radiology, Yong Loo Lin School of Medicine, National University of Singapore, Singapore, Singapore
e-mail: dnrqj@nus.edu.sg

C.-H. Ang
Department of Computer Science, School of Computing, National University of Singapore, Singapore, Singapore
e-mail: dcsach@nus.edu.sg

Keywords Time-varying volume rendering · BIRCH · Clustering · 4-D medical images

1 Introduction

With the development of three-dimensional (3-D) medical imaging techniques, high resolution volumetric data are captured to allow the assessment of the morphology of anatomical and pathologic structures. However, many aspects relevant for diagnostic decisions and treatment planning cannot be evaluated by means of static images. For example, physiological parameters such as blood flow, vessel permeability and tumor perfusion, which are essential to the diagnostic judgement of the damage area of an ischemic stroke, cardiac infarction and the malignancy of tumor, can only be acquired by time-varying (dynamic) image data [1, 2]. Furthermore, physicians can track disease progress or its response to therapy over a short time or over several months or years based on dynamic data. To this end, visualization and exploration of these time-varying volume data are usually crucial for clinicians to make more accurate diagnosis and treatment.

With the advancement in radiological science, dynamic scanning is increasingly popular in clinical applications. Fast rendering of these time-varying data, however, is a challenging task. With marked improvement of spatial and temporal resolution, these data usually has hundreds of megabytes or even gigabytes. This makes time and storage requirement for visualization dramatically increase. Preprocessing such as encoding and compression are usually necessary. Traditional encoding and compression algorithms based on spatial features applied on 3-D data are no longer affordable for dynamic data, as they should be analyzed and processed in the joint feature-temporal space. On the other hand, we need to keep a balance between the compression ratio and the quality of the rendered results to ensure that the clinicians can obtain enough indications for diagnostic decisions.

We propose an efficient clustering method for fast rendering of these time-varying volumetric medical data. We divide the volumes into a set of blocks and cluster them considering both spatial and temporal coherence by employing a modified BIRCH algorithm [3]. In each cluster of blocks, a *KeyBlock* is generated to represent the cluster by considering the contributions of all blocks. Thus the storage space of the volumes is reduced greatly. In addition, we assign a lifetime to every *KeyBlock* and implement a dynamic memory management scheme to further reduce the storage space. During rendering, each *KeyBlock* is rendered as a *KeyImage*, which can be reused if the view transformation and transfer function are not changed. Extensive experiments have been conducted to evaluate the feasibility of the proposed method, in terms of compression speed, compression ratio and rendering speedup. Regression testing is also employed to analyze the impact of the compression scheme on the visual quality of rendering.

The rest of the paper is organized as follows. Section 2 discusses related work. The implementation details of the proposed method are presented in Sect. 3. The experiments are described and discussed in Sect. 4. The paper concludes in Sect. 5.

2 Related work

The size of a typical time-varying volumetric medical image can range from hundreds of megabytes to hundreds of gigabytes. The visualization of such huge datasets is a challenging problem due to overwhelming data size, insufficient memory and I/O bandwidth, and heavy computational requirements. In addition to using high performance hardware, compression-based visualization methods which combine efficient decompression closely with rendering would be extremely useful in such situations.

Among the relatively small number of published papers on time-varying volumetric medical images compression, methods which treat the data as a four-dimensional (4-D) field are dominant. In [4], a 4-D tree, which is an extension of the octree, is used for encoding time-varying data. Other methods often use the discrete wavelet transform (DWT), followed by quantization and/or a coefficient partitioning technique such as the embedded zerotree wavelet (EZW) [5] and set partitioning in hierarchical trees (SPIHT) [6], and finally a symbol coding method. Zeng et al. used 4-D DWT and extended EZW to 4-D to encode the echocardiographic data [7]. SPIHT is extended to 4-D and used with 4-D DWT in [8] to compress fMRI and 4-D ultrasound images. Liu and Pearlman extended subband block hierarchical partitioning (SBHP), another coefficient partitioning technique originally described in [9], to 4-D and used it with 4-D wavelet decomposition for enabling progressive fidelity and resolution decompression of 4-D images [10]. Generally, a method that relies on the 4-D wavelet transform can offer relatively high compression ratios with reasonable fidelity. However, it is not easy to achieve a fast decompression process due to the high complexity of the 4-D wavelet transform. In addition, a number of time steps (i.e., frames) may need to be decoded even if only one of them is to be manipulated or rendered.

The other approaches separate time dimension from spatial dimensions. In [11], a 4-D volume rendering algorithm based on time-space partitioning (TSP) tree is proposed, and the algorithm is improved by using new color-based error metrics [12]. The TSP tree is effective in capturing spatial and temporal coherence of data and rendering performance is thus improved. However, the TSP tree is built as a supplementary data structure, and consequently, results in extra memory overhead and it cannot reduce the space or I/O bandwidth effectively. Shen and Johnson proposes the differential volume rendering method, in which only the changed pixels are re-rendered in each time step [13]. However, the process of determining the changed pixels may be long especially when the amount of changed pixels is large. In [14] and [15], this process is improved by using a two-level differential volume rendering method. The rendering performance of this method is improved since the time for determining the positions of changed pixels is reduced. However, this method cannot completely take advantage of the data coherence to further accelerate rendering.

Wang et al. proposes the dynamic linear level octree (DLLO) data structure for 4-D volume rendering [16]. This method effectively resolves the I/O bandwidth problem and exhibits distinguished rendering speedup, but the employment of octree restricts its flexibility to exploit more extensive data coherence while decomposing the volume data. In [17], high-pass coefficients from the Laplacian decomposition are encoded using vector quantization. During the rendering, the volume data is decompressed on-the-fly and rendered using hardware texture mapping. This method can be applied to time-varying data set since each volume frame can be encoded separately, producing an index texture and local codebooks for every time step. Although the decompression speed is considerably fast due to the simple decoding, this approach does not exploit the dependency among voxels in different volumes. Furthermore, for a given resolution, the compression ratio is fixed and does not depend on the content of the volume. Several methods consider a 4-D image as 3-D video and extend the motion estimation and compensation techniques in video coding to 3-D for exploiting redundancies in all four dimensions [18, 19]. However, it is difficult for these methods to achieve a good rendering performance since a number of prior frames need to be decoded before processing and rendering an intermediate frame.

3 The clustering-based volume rendering method

3.1 Clustering

A time-varying volumetric medical dataset usually contains a sequence of 3-D volumes, which are collections of voxels with density values. We first divide the dataset into a set of blocks (cubes). Given dataset Ψ including v volumes with n voxels in each dimension, we can uniformly divide these volumes into blocks with m voxels in each dimension. Thus, Ψ can be divided into $v \times (n/m)^3$ blocks. A good choice of voxel number m can improve the quality of the clustering results. It usually depends on the voxel number n and the characteristics of the dataset. The estimation of it is beyond the scope of this paper. We adopt the BIRCH method to cluster these blocks for two reasons: (1) the I/O cost of the BIRCH algorithm is linear with the size of the dataset; and (2) the granularity of clusters can be adaptively adjusted by dynamically configuring threshold values.

3.1.1 BIRCH-based clustering

BIRCH-based clustering technique is used to exploit the homogeneousness of time-varying volumetric data. The blocks from all volumes are grouped into different clusters, and each cluster is represented by its centroid and radius. We denote block B_i as a vector: $\mathbf{B}_i = [x_1^i, x_2^i, \dots, x_M^i]$, where x_j^i is the intensity value of jth voxel and M is the number of voxels in block B_i, which equals to m^3. The distance between two blocks B_i and B_j is defined as follows:

$$D(\mathbf{B}_i, \mathbf{B}_j) = \frac{\|\mathbf{B}_i - \mathbf{B}_j\|}{M} = \frac{\sqrt{\sum_{k=1}^{M}(x_k^i - x_k^j)^2}}{M} \tag{1}$$

which is the *normalized Euclidean distance* of the two vectors representing the two blocks. We define the centroid and the radius of a cluster containing N blocks as follows:

$$\mathbf{C} = \frac{\sum_{i=1}^{N}\mathbf{B}_i}{N}, \tag{2}$$

$$R = \max_i\bigl(D(\mathbf{B}_i, \mathbf{C})\bigr). \tag{3}$$

In our BIRCH-based implementation, all blocks are organized as a height-balanced tree, named *CF* tree, with two parameters, the branching factor F and the distance threshold D_{thres}. One of the advantages of BIRCH algorithm is that it can usually find a high quality clustering with a single scan of the blocks. This is important for large time-varying volumetric medical data. Here we simply describe our implementation with some adaptations and improvements of the original BRICH algorithm. More details of BIRCH can be found in [3].

When a new block B is ready, we recursively descend the *CF* tree to compute the distance between the block and the centroids of existing clusters and find the cluster X_i having the smallest distance to B. This recursive operation can also be performed iteratively. Then we compute the value of new radius R_i' of X_i under the assumption that B is inserted into it. If $R_i' \le D_{\text{thres}}$, we add the block to X_i and recompute its centroid C_i'. When there is no space for B, i.e. the number of block in X_i is greater than F, we must split the leaf node by choosing the farthest pair of blocks as seeds and redistributing the remaining blocks. If $R_i' > D_{\text{thres}}$, a new cluster is created containing only the block B. The pseudo code of our BIRCH-based clustering algorithm is presented in Algorithm 1.

In Algorithm 1, when a new block B is inserted to into the cluster X_i on trial, the centroid and radius of the cluster must be recomputed. The conventional method for computing the new centroid and radius of the cluster is to apply (2) and (3). However, this method is computationally expensive since all of the existing blocks in the cluster needed to be accessed repeatedly during the block insertion. To reduce the cost of computation, an approximate method is proposed for the cluster radius estimation. As illustrated in Fig. 1, an existing cluster X_i has N_i blocks, the centroid C_i and the radius R_i. When a new block B is inserted into X_i, the new centroid C_i' is computed as

$$C_i' = \frac{N_i \times C_i + B}{(N_i + 1)} \tag{4}$$

and the new radius R_i' is estimated based on the following formulas:

$$\begin{aligned} r_1 &= R_i + \frac{d}{N_i + 1}, \\ r_2 &= \frac{d \times N_i}{N_i + 1}, \\ R_i' &= \max(r_1, r_2), \end{aligned} \tag{5}$$

Algorithm 1 BIRCH-based clustering algorithm

```
1:  /* $RSet$: the set of blocks, initially including all the blocks.
    $CF$: the $CF$ tree which is initially empty. */
2:  while ($RSet \neq empty$) do
3:    select $B$ from $RSet$
4:    if ($CF \neq empty$) then
5:      /* Find the cluster with minimum distance to $B$ */
6:      select $X_i$: min(distance($B$, $X_i.centroid$))
7:      /* Try to insert the block into the cluster */
8:      compute $R'_i$
9:      /* Judge if the insertion is appropriate */
10:     if ($R'_i \leq D_{\text{thres}}$) then
11:       $X_i.radius \Leftarrow R'_i$
12:       compute $X_i.centroid$
13:       $X_i.size \Leftarrow X_i.size + 1$
14:       if ($X_i.size > F$) then
15:         split $X_i$
16:       end if
17:       continue
18:     end if
19:   end if
20:   create $cluster$
21:   $cluster \Leftarrow B$
22:   $CF \Leftarrow CF + cluster$
23:   $RSet \Leftarrow RSet - B$
24: end while
25: return $CF$
```

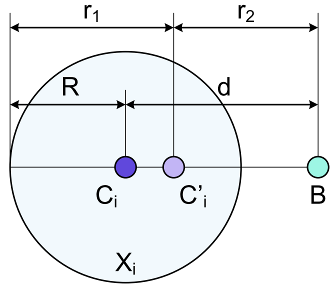


Fig. 1 Estimation of the center and radius of a cluster for a trial insertion of a block

where d is the distance between the trial block B and the centroid R_i. r_1 and r_2 are the two candidate radii, and the greater one will be chosen as the radius of the updated cluster. This mathematical model mimics the linear interpolation between two weighted points in the M-dimensional space. If a block is inserted into a cluster, the new center of the cluster will be pulled towards the inserted block, and the displacement is inversely proportional to the weights of the two M-dimensional points, which are the number of blocks represented respectively.

It is easy to prove that (2) and (4) produce the same results, meaning that there is no error introduced in the computation of the new centroid. However, (5) tends to produce a value greater than that of (3), i.e. the radius could be over estimated. The cluster is actually denser than that implied by the estimated radius. Therefore, this method is effective in producing clusters strictly under the pre-defined error-tolerance D_{thres}. Section 3.1.2 will present one technique to adjust D_{thres} for improving the quality of the clustering. The proposed method is also computationally efficient as it avoids accessing blocks that are already in the cluster. Furthermore, in contrast to (3), the computation of the radius in (5) is independent of the centroid. Thus, only the radius is evaluated when trying to insert a block into a cluster, and the centroid is updated only when the radius satisfies the cluster criterion. This implementation significantly improves the performance of the clustering process.

3.1.2 Clustering granularity

It is obvious that the threshold value D_{thres} greatly affects the size and quality of the clustering. For example, cluster centroids can be too close or cross each other if D_{thres} is too small. The number of clusters n_c also increases when D_{thres} decreases. On the other hand, if D_{thres} is too large, the intensity changes in the data may be lost during the rendering. Fortunately, BIRCH provides mechanisms to dynamically increase the threshold value when building the *CF* tree. Thus, we can set small initial value for D_{thres} and adjust it based on the *CF* tree that has been built. In the default implementation of BIRCH, D_{thres} is set as 0. In our implementation, we usually set the initial value based on the nature of the volumetric data, e.g. the intensity distribution of the data.

Suppose that D^i_{thres} is the current threshold value and turns out to be too small. A heuristic approach is used to estimate next value D^{i+1}_{thres}. First, the total number of clusters n_c increases with the number of inserted blocks n_b. By preserving a relationship table of n_c and n_b during the *CF* tree building, we can estimate n^{i+1}_c using a least squares linear regression. Thus, we can approximately set

$$D^{i+1}_{\text{thres}} = D^i_{\text{thres}} \times \frac{n^{i+1}_c}{n^i_c}. \tag{6}$$

Second, if we want to decrease the number of clusters in current *CF* tree, it is reasonable that we increase D^i_{thres} by adding the distance of two closest clusters to it so that at least these two clusters can be merged. We set the new threshold D_{thres} based on the following equation:

$$D^{i+1}_{\text{thres}} = \max\left(D^i_{\text{thres}} \times \frac{n^{i+1}_c}{n^i_c}, D_{\min}\right). \tag{7}$$

3.1.3 Output data

In the above descriptions, for simplicity, we assume that each volume has n voxels in each dimension and each block has m voxels in each dimension. In practice, the number of voxels in each dimension of a volume can be different. For an adaptation to the size of the volume, each dimension of blocks can have different number of voxels. In addition, a dataset with a large number of time steps can be divided into multiple groups of frames in time order in which the proposed clustering algorithm is applied.

In our implementation, the centroid of a cluster is termed a *KeyBlock*. The output of the clustering step is a binary file containing three following sections:

1. *Header information* stores the resolution of the 3-D volumes, number of time steps, voxel format, data description and pointers to other sections.
2. *Volume-KeyBlock table* is a collection of lookup tables corresponding to all the 3-D volumes, one table for each volume at one time step. Each table can be considered as a 3-D array in which each element is a number representing the link between the corresponding block in the volume and the *KeyBlock* of the cluster it belongs to. This number actually is the index of the *KeyBlock* in the *KeyBlock data* section.
3. *KeyBlock data* contains all *KeyBlock* generated.

For efficient memory management, each *KeyBlock* is associated with a *last-volume number* (*LVN*), which is the index number of the last volume which contains blocks belonging to the cluster represented by the *KeyBlock*. The *LVN* indicates the life period during which a *KeyBlock* is used to reconstruct volume(s) from time to time and should reside in the memory. It also indicates the expiring time after which the *KeyBlock* should be released. The *KeyBlocks*, therefore, are not released one by one as the order they are loaded in. A dynamic memory management scheme should be employed during the implementation. In this way, *KeyBlocks* are stored so that they can be properly loaded and released as the sequence of volume being processed.

3.2 Rendering

In the rendering stage, each 3-D volume is reconstructed and rendered using any of various existing volume rendering techniques directly or with some optimizations. For instance, a ray casting-based rendering method using the proposed clustering algorithm can be described as below.

Denote the index of the working volume as q. Initially, the volume at the first time step is used as the working volume ($q = 1$) and the following steps are executed:

1. *KeyBlocks* whose *LVNs* are less than q are released together with their associated partial-image buffers. The final image of the current time step is initialized.
2. *KeyBlocks* are read from the binary file in turn. Each *KeyBlock* is associated with a partial-image buffer, and *KeyImage*, the rendering result of each *KeyBlock*, is saved into the partial-image buffer. After all the *KeyBlocks* in volume q are loaded, they are rendered according to the following two rules:
   - *Rule 1*. If current volume is the first volume, all the *KeyBlocks* are rendered.
   - *Rule 2*. If the current model-view transformation or transfer functions are changed as compared to that in the previous time step, all *KeyBlocks* are re-rendered; otherwise, only *KeyBlocks* that are newly loaded are rendered.
3. The *KeyImages* of the *KeyBlocks* are composed in 2-D space according to the *Volume-KeyBlock table* of volume q and the final image is constructed by the following rules:
   - *Rule 1*. According to the current viewing direction, blocks in volume q are accessed in front-to-back order. Using the information in the *Volume-KeyBlock table*, *KeyBlocks* can be easily located.
   - *Rule 2*. The *KeyImages* of the *KeyBlocks* are composed into the final image at the corresponding projection area.

   After all blocks of volume q are processed, the final image is produced and displayed.
4. To proceed the volume at the next time step, q is increased by one ($q = q + 1$).

The above steps are repeated until the whole sequence is processed. In this algorithm, once the *KeyImages* are produced, the final image is generated by composing their colors and opacities in front-to-back order based on the theory of partial ray composing. The final image can be composed from the *KeyImages* by using the *over* operator [20]. 2-D resampling of the *KeyImages* may be required if the sampling rate of the *KeyImage* is different from that of the final image or when they are not sampled along the same set of rays. The early-ray-termination [21] is still possible for both *KeyBlock* rendering and *KeyImage* composition, where samples in *KeyImages* can be safely skipped when pixels of the final image are already opaque.

In the rendering methods using *over* operator directly (e.g. ray casting), *KeyImages* are used as the intermediate results for fastening the composition step; thus, improving the rendering speed. In other methods (e.g. texture mapping), the *KeyImage* may not be used since no *over* operator is directly performed. However, the rendering speed still improves in this clustering-based rendering algorithm since the use of *KeyBlocks* helps reduce the I/O bandwidth effectively.

4 Experimental results and discussion

In our experiments, three time-varying volume datasets named HAND, HEART, and ABDOMEN in DICOM format were used to evaluate the performance of the proposed algorithm (Table 1). They were all acquired at the National University Hospital, Singapore. The computing platform was a 2.52 GHz Intel Pentium IV desktop PC equipped with 1 GB RAM and a NVIDIA Quadro 4 700 XGL graphics card with 64 MB onboard memory.

The raw image data and their descriptions are extracted from the datasets to form the input data of the experiments. In the encoding phase, we used the proposed clustering algorithm with different initial distance threshold values D_{thres} to compress each dataset. The output of this phase are binary files in the format described in Sect. 3.1.3; each file corresponds to a specific value of D_{thres}. The computing times and the compression ratios were measured during the compression. The parameters used and results of this phase are shown in Tables 2 to 4. In these tables, the compression ratio (CR) is defined as follows:

$$\text{CR} = \left(1 - \frac{\text{Compressed file size}}{\text{Uncompressed file size}}\right) \times 100\%. \tag{8}$$

The proposed cluster-based time-varying volume rendering algorithm was implemented using two underlying rendering techniques: 2-D texture mapping and 3-D texture mapping. If 3-D texture mapping is supported by the graphics card, the rendering speed can be improved due to the fast hardware accelerated 3-D interpolation. Otherwise, 2-D texture mapping can be used; however, software sampling may be required to create the texture images for the three major orientations. The experiments evaluate the improvement in term of rendering speed of our algorithm compared to that of the 2-D texture mapping and 3-D texture mapping techniques.

After an encoded dataset is loaded into the system, it is rendered repeatedly 20 times with different preset viewing angles while the rendering timing of each time step is recorded. The execution times of the last 10 running times are then averaged and reported as the performance result of the dataset. The design of the experiment ensures that the recorded execution times are obtained when the system is stable and renderers can benefit from the I/O cache if possible. The speedup ratios obtained are presented in Table 5.

As seen from Table 5, depending on the specific dataset and the underlying rendering techniques used, the proposed algorithm yielded the rendering speedup of 1.5 to 9.4 compared to the corresponding regular algorithm. This is due

Table 1 Dataset specifications

Dataset	HAND	HEART	ABDOMEN
Bits allocated	8	8	8
Rows × Columns	512 × 512	192 × 156	256 × 256
Slices	136	27	12
Time steps	5	20	39
Pixel size (mm)	0.39 × 0.39	1.67 × 1.67	1.02 × 1.02
Inter-slice spacing (mm)	4.0	8.0	5.0
Size (MB)	171.25	15.42	29.25
Modality	MRA	MRI	MRU

Table 2 Results of encoding the HAND dataset using different distance threshold values

Test name	Block size	D_{thres}	Time (s)	Size (MB)	CR (%)
HAND A	16 × 16 × 17	0.10	3953	36.27	78.7
HAND B	16 × 16 × 17	0.15	3044	26.29	84.5
HAND C	16 × 16 × 17	0.20	2525	20.94	87.7

Table 3 Results of encoding the HEART dataset using different distance threshold values

Test name	Block size	D_{thres}	Time (s)	Size (MB)	CR (%)
HEART A	12 × 13 × 27	0.10	22	3.53	77.1
HEART B	12 × 13 × 27	0.15	16	2.27	85.3
HEART C	12 × 13 × 27	0.20	13	1.61	89.6

Table 4 Results of encoding the ABDOMEN dataset using different distance threshold values

Test name	Block size	D_{thres}	Time (s)	Size (MB)	CR (%)
ABDOMEN A	16 × 16 × 12	0.10	305	20.76	29.0
ABDOMEN B	16 × 16 × 12	0.15	264	17.46	40.3
ABDOMEN C	16 × 16 × 12	0.20	236	14.93	49.0

Table 5 Speedup ratios over the regular rendering techniques obtained when applying our method on different datasets

Test name	2-D texture mapping	3-D texture mapping
HAND A	7.14	1.56
HAND B	8.72	1.64
HAND C	9.44	1.69
HEART A	3.62	2.23
HEART B	4.71	2.39
HEART C	5.56	2.47
ABDOMEN A	3.98	1.59
ABDOMEN B	4.59	1.63
ABDOMEN C	5.14	1.74

to the fact that the data coherence in time-varying volume datasets is extensively exploited in both space and time dimensions through the employment of our clustering method. A number of works analyzed in Sect. 2 also considered both spatial and temporal coherence. However, they mostly were based on adjacent regions in two dimensions. In our method, blocks that are grouped into a cluster may come from any portion of the dataset in both space and time dimensions, leading to an efficient saving of storage space and I/O bandwidth. Another advantage of the method is the simple decoding mechanism which is essential for fast volume rendering algorithms. Furthermore, similar regions can be represented by the same *KeyBlock* and are rendered only once if rendering parameters are unchanged. This also significantly reduces the time cost for rendering.

The proposed algorithm performs a lossy compression of the time-varying volume data. It is necessary to analyze the impact of the compression scheme on the visual quality of rendering. A regression testing method is employed for this purpose. The regression testing compares a test image that is produced with an algorithm being evaluated with a reference image that is assumed to be indeed correct. The comparison takes into account dithering and anti-aliasing effects, and creates an output image representing the difference between the test image and the reference image [22]. The difference of the two images is also quantified in terms of *absolute error* (E_A) and *thresholded error* (E_T), which are calculated based on the following equations:

$$D_i = \frac{|r_i^v - r_i^t| + |g_i^v - g_i^t| + |b_i^v - b_i^t|}{3}, \qquad E_A = \frac{\sum_i D_i}{L_c - 1}, \tag{9}$$

$$A_i = \begin{cases} D_i - T & \text{if } D_i - T > 0 \\ 0 & \text{otherwise,} \end{cases} \qquad E_T = \frac{\sum_i A_i}{L_c - 1}, \tag{10}$$

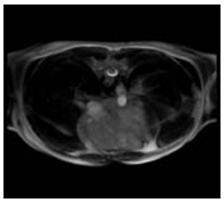
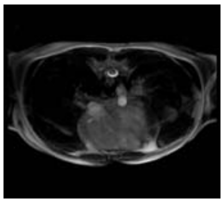
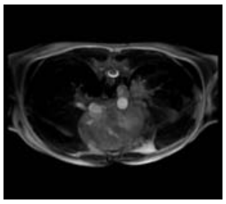
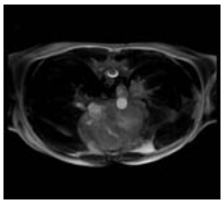
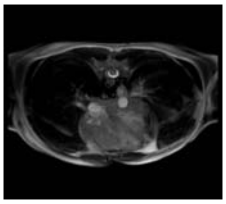
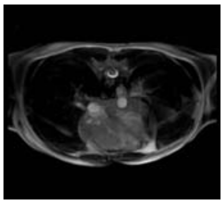
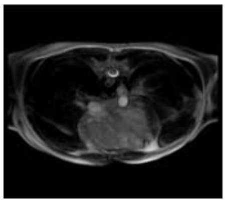
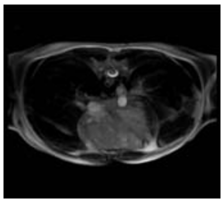
where (r_i^v, g_i^v, b_i^v) and (r_i^t, g_i^t, b_i^t) are the ith color pixel value (red, green and blue) of the reference image and the test image respectively; D_i is the difference between the ith pixel of the two images; L_c is the number of color levels of each channel; T is a threshold-tolerance for pixel differences. Thus, the *absolute error* is the total error in comparing the two images, and the *thresholded error* is the error for a given pixel minus the threshold and clamped at a minimum of zero. The latter will be more effective in representing the noticeable differences between two images.

In our implementation, all the images are generated in color with red, green and blue channels, and each channel has 8 bits, i.e., $L_c = 2^8 = 256$ levels. In order to avoid misunderstanding of images with the introduction of pseudo-colors, pixels are assigned with the same value for all three channels and the images thus appear in gray. A threshold-tolerance of 5 is used in the analysis of the image quality. It is less than 2% of the maximum pixel difference and normally is not noticeable by human eyes. The images gener-

Table 6 Error analysis of cluster-based rendering algorithm

Test name	E_A	E_T
Inter-step	547.113	104.500
HAND A	15.796	0.009
HAND B	17.938	0.059
HAND C	30.066	0.541
Inter-step	360.980	52.322
HEART A	39.980	0.321
HEART B	54.321	1.182
HEART C	73.284	7.749
Inter-step	2761.917	1658.556
ABDOMEN A	206.086	17.966
ABDOMEN B	254.729	36.396
ABDOMEN C	343.661	70.943

Table 7 Comparison of the image quality at different time steps between 2-D texture-mapped rendering and cluster-based rendering of HEART dataset ($D_{\text{thres}} = 0.15$)

Reference image	Rendered image	Error
		$E_A = 118.0$ $E_T = 11.50$ Step 1
		$E_A = 46.2$ $E_T = 0.30$ Step 7
		$E_A = 48.5$ $E_T = 0.18$ Step 13
		$E_A = 41.0$ $E_T = 0.04$ Step 20

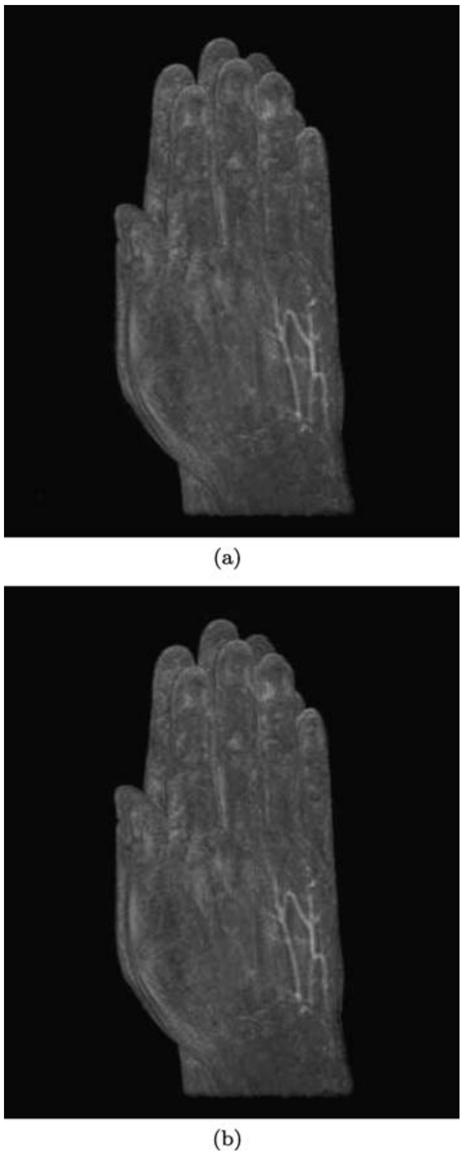


Fig. 2 Comparison of the image quality between (**a**) 2-D texture-mapped rendering and (**b**) cluster-based rendering on the volume at the last time step in HAND dataset ($D_{\text{thres}} = 0.20$)

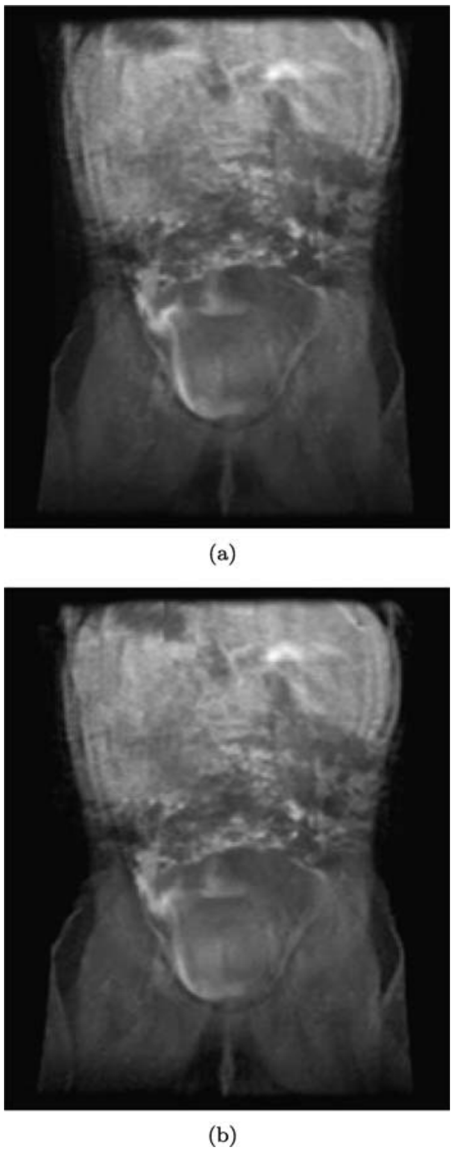


Fig. 3 Comparison of the image quality between (**a**) 2-D texture-mapped rendering and (**b**) cluster-based rendering on the volume at the last time step in ABDOMEN dataset ($D_{\text{thres}} = 0.20$)

ated by the regular texture mapping method are employed as the reference images, and the image quality of cluster-based rendering is evaluated based on the following procedures. For each dataset, the regression testing is applied to each pair of corresponding images at each time step. The *absolute error* (E_A) and *thresholded error* (E_T) of this time step are calculated accordingly. After the regression testing is finished for all time steps, the results are averaged and used to represent the error of the cluster-based rendering of this dataset. Based on the images generated by the regular texture mapping method, the regression testing is also applied between the images at successive time steps. The results are averaged and used to indicate the inter-step differences. It provides us an effective reference to evaluate the rendering quality. The inter-step errors also serve as a good measurement of the coherence of the dataset. Details of regression testing can be found in [22]. Table 6 presents the results from the error analysis. It is clear that the cluster-based rendering algorithm achieves very good rendering quality from the negligible error. Selected images are shown in Table 7, Figs. 2 and 3.

5 Conclusion

In this paper, we introduced a clustering-based volume rendering algorithm, which is a new method for fast visualization of time-varying volumetric medical images. The algorithm takes advantage of the inherent characteristics of time-varying volume data. The data coherence is deeply exploited in both spatial and time dimensions through the employment of the clustering technique so that the rendering performance is enhanced. Since there is no restriction on the underlying type of renderers, the algorithm also provides flexibilities for further extensions. Extensive experiments are performed based on the texture-based implementations of this algorithm. A good performance was achieved in terms of both speed acceleration and space reduction. Results demonstrated the superiority of this method over regular algorithms for time-varying volume rendering. We could obtain over 89% of compression and up to 9 times increase in rendering speed. Based on the analytical results of regression testing, errors introduced due to clustering-tolerance are quantitatively and visually small. Hence, high rendering fidelity can be achieved.

In future, besides conducting more experiments on larger datasets, we will integrate some importance analysis techniques into our clustering method to ensure the dynamic features of the time-varying volumetric medical data can be correctly visualized after compression. The relationship between some parameters, such as the initial threshold values, and the characteristics of the data will also be investigated.

References

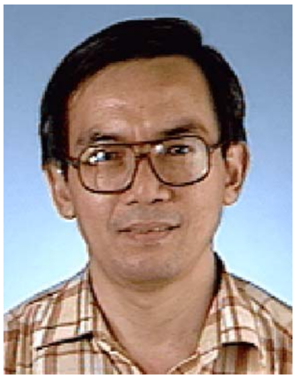
1. Armitage, P., Behrenbruch, C., Brady, M., Moore, N.: Extracting and visualizing physiological parameters using dynamic contrast-enhanced magnetic resonance imaging of the breast. Med. Image Anal. **9**(4), 315–329 (2005)
2. Kamasak, M., Bouman, C., Morris, E., Sauer, K.: Direct reconstruction of kinetic parameter images from dynamic PET data. IEEE Trans. Med. Imaging **24**(5), 636–650 (2005)
3. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An efficient data clustering method for very large databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 103–114. ACM, New York (1996)
4. Wilhelms, J., Gelder, A.V.: Multi-dimensional trees for controlled volume rendering and compression. In: Proceedings of the Symposium on Volume Visualization 1994 (VVS'94), pp. 27–34. ACM, New York (1994)
5. Shapiro, J.M.: Embedded image coding using zerotrees of wavelet coefficients. IEEE Trans. Signal Process. **41**, 3445–3462 (1993)
6. Said, A., Pearlman, W.A.: A new, fast, and efficient image codec based on set partitioning in hierarchical trees. IEEE Trans. Circuits Syst. Video Technol. **6**, 243–250 (1996)
7. Zeng, L., Jansen, C., Marsch, S., Unser, M., Hunziker, P.: Four-dimensional wavelet compression of arbitrarily sized echocardiographic data. IEEE Trans. Med. Imaging **21**(9), 1179–1187 (2002)
8. Lalgudi, H., Bilgin, A., Marcellin, M., Nadar, M.: Compression of fMRI and ultrasound images using 4D SPIHT. In: Proceedings of the IEEE International Conference on Image Processing 2005 (ICIP 2005), vol. 2, pp. 746–749 (2005)
9. Chrysafis, C., Said, A., Drukarev, A., Islam, A., Pearlman, W.: SBHP—A low complexity wavelet coder. In: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing 2000 (ICASSP'00), vol. 4, pp. 2035–2038 (2000)
10. Liu, Y., Pearlman, W.: Four-dimensional wavelet compression of 4-D medical images using scalable 4-D SBHP. In: Proceedings of the Data Compression Conference 2007 (DCC'07), pp. 233–242 (2007)
11. Shen, H.W., Chiang, L.J., Ma, K.L.: A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. In: Proceedings of the IEEE International Conference on Visualization 1999 (VIS'99), pp. 371–545 (1999)
12. Ellsworth, D., Chiang, L.J., Shen, H.W.: Accelerating time-varying hardware volume rendering using tsp trees and color-based error metrics. In: Proceedings of the IEEE Symposium on Volume Visualization 2000 (VVS'00), pp. 119–128. ACM, New York (2000)
13. Shen, H.W., Johnson, C.: Differential volume rendering: A fast volume visualization technique for flow animation. In: Proceedings of the IEEE International Conference on Visualization 1994 (VIS'94), pp. 180–187 (1994)
14. Liao, S.K., Lin, C.F., Chung, Y.C., Lai, J.Z.C.: A differential volume rendering method with second-order difference for time-varying volume data. J. Vis. Lang. Comput. **14**(3), 233–254 (2003). Computer Graphics and Virtual Reality
15. Liao, S.K., La, J.Z.C., Chung, Y.C.: Time-critical rendering for time-varying volume data. Comput. Graph. **28**(2), 279–288 (2004)
16. Wang, Z., Chui, C.K., Cai, Y., Ang, C.H., Teoh, S.H.: Dynamic linear level octree-based volume rendering methods for interactive microsurgical simulation. Int. J. Image Graph. **6**(2), 155–172 (2006)
17. Schneider, J., Westermann, R.: Compression domain volume rendering. In: Proceedings of the IEEE International Conference on Visualization 2003 (VIS'03), pp. 293–300 (2003)
18. Kassim, A., Yan, P., Lee, W.S., Sengupta, K.: Motion compensated lossy-to-lossless compression of 4-D medical images using integer wavelet transforms. IEEE Trans. Inf. Technol. B **9**(1), 132–138 (2005)
19. Sanchez, V., Nasiopoulos, P., Abugharbieh, R.: Efficient lossless compression of 4-D medical images based on the advanced video coding scheme. IEEE Trans. Inf. Technol. B **12**(4), 442–446 (2008)
20. Porter, T., Duff, T.: Compositing digital images. ACM Comput. Graph. **18**(3), 253–259 (1984)
21. Levoy, M.: Efficient ray tracing of volume data. ACM Trans. Graph. **9**(3), 245–261 (1990)
22. Schroeder, W., Martin, K.M., Lorensen, W.E.: The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 2nd edn. Prentice-Hall, Upper Saddle River (1998)

Zhenlan Wang obtained his B.Eng. in Information Science and Technology at Xian Jiaotong University in 1999 and Ph.D. in Computer Science at National University of Singapore in 2006. His research interests are mainly in visualization, computer graphics and virtual reality for medicine. Dr. Zhenlan Wang is currently working at Microsoft in USA.

Binh P. Nguyen received the B.Eng. in Information Technology and M.Sc. in Information Processing and Communications from Hanoi University of Technology, Vietnam in 2002 and 2004, respectively. He is currently a lecturer at the School of Information and Communication Technology, Hanoi University of Technology, Vietnam and working toward the Ph.D. degree in the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. His research interests include visualization of medical images, GPU-based algorithms, multi-core and multi-GPU computing.

Chee-Kong Chui is currently an Assistant Professor in the Control and Mechatronics Group, Department of Mechanical Engineering, National University of Singapore, Singapore. He was the principal investigator of the Biomedical Simulation & Device Design Project at the then Institute of Bioengineering prior to pursing a Ph.D. in Biomedical Precision Engineering Lab, The University of Tokyo, Japan. His research interests include computer integrated and robot-assisted surgery, human-machine interface, medical device design, biomechanical modeling and simulation.

Jing Qin received his B.Eng. and M.Eng. degrees from the Institute of Information Engineering of the University of Science and Technology Beijing, China. He continued his graduate study and received his Ph.D. degree from the Department of Computer Science and Engineering of the Chinese University of Hong Kong in 2009. Now he is a research staff of Department of Diagnostic Radiology, National University of Singapore. His research interest lies in the broad area of medical imaging, modeling and simulation as well as computer-assisted interventions.



Chuan-Heng Ang is currently a senior lecturer in the School of Computing, National University of Singapore. His research interest is the design and implementation of spatial data structures.

Sim-Heng Ong is an associate professor in the Department of Electrical and Computer Engineering and the Division of Bioengineering, National University of Singapore. He received his B.E. (Hons.) from the University of Western Australia and his Ph.D. from the University of Sydney. His major fields of interest are computer vision and biomedical image processing. He has over 250 papers published in international journals and conference proceedings.