Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Visual Languages and Computing
14 (2003) 233–254
Journal of Visual Languages & Computing
www.elsevier.com/locate/jvlc

A differential volume rendering method with second-order difference for time-varying volume data

Shih-Kuan Liao, Chin-Feng Lin, Yeh-Ching Chung*, Jim Z.C. Lai

Department of Information Engineering, Feng Chia University, Taichung, Taiwan 407, ROC

Received 13 June 2002; received in revised form 9 January 2003; accepted 19 January 2003

Abstract

The differential volume rendering method is a ray casting based method for time-varying volume data. In the differential volume rendering method, the changed voxels between consecutive time steps are extracted to form differential files in advance. When the dataset is to be rendered, changed voxels are projected onto the image plane to determine the positions of changed pixels. Only the changed pixels, instead of all pixels on the image, are updated by casting new rays in each time step. The main overhead of the differential volume rendering method is the determination of changed pixels. In this paper, we propose a two-level differential volume rendering method, in which the determination of changed pixels is accelerated by the aid of the second-order difference. Since changed voxels in two consecutive differential files may partially overlap in the space, the projection computation spent on the overlapped area is redundant. We use this property to extract the difference of changed voxels between consecutive differential files to form the second-order difference. Based on the second-order difference, the changed pixels can be determined more efficiently. The experimental results show that the proposed method outperforms the comparative methods for all test datasets in most cases. In addition, the rendering time can be predicted once the data files are loaded in each time step.

Keywords: Ray casting; Differential volume rendering; Second-order difference; Time-varying volume data; Flow animation; CFD

*Corresponding author. Tel.: +886-424517250x3765; fax: +886-424516101.
E-mail address: ychung@fcu.edu.tw (Y.-C. Chung).

1045-926X/03/$ - see front matter © 2003 Elsevier Science Ltd. All rights reserved.
doi:10.1016/S1045-926X(03)00020-X

1. Introduction

Volume rendering, which projects a 3D volume data into a 2D image to reveal the internal structure of the object, is a computation-demanding task [1–4]. A time-varying volume data (TVVD, also referred as 4D volume data [5]) is a sequence of subsequent volume datasets during a period of time steps. In comparison with a single 3D volume dataset that contains the static internal structure, TVVD contains the dynamic evolution of the structure. For example, CFD simulations can be rendered into a flow animation. However, the computation demand of rendering TVVD is much more than that of rendering a single 3D volume data. In order to reduce the computation amount, many techniques have been proposed to exploit the spatial coherency and temporal coherency in TVVD. The basic idea of these techniques is to reuse the steady portion to avoid redundant storage or computation.

Shen et al. [6] proposed the differential volume rendering method to render TVVD. We refer this method as DVRM in this paper. In this method, the changed voxels between consecutive time steps are extracted to form differential files. These files are used to determine the positions of changed pixels on the image plane. Only the changed pixels, instead of all pixels in the image, are updated by casting new rays at the positions in each time step. Ma et al. [7] encoded each single volume data into an octree for the sake of spatial coherency. For coherency, consecutive octrees are merged. Pointers are used to replace identical subtrees in the consecutive octrees. In [8], a time-space partitioning (TSP) tree was proposed. A TSP tree is a time-supplemented octree that utilizes both the spatial and temporal coherency effectively and allows flexible tradeoff between image quality and rendering speed. Based on the shear warp factorization, Anagnostou et al. [5] proposed a 4D volume rendering technique that detects and renders the changed areas in every volume to exploit the temporal coherency. Besides the above coherency-based techniques, some other techniques are developed based on parallel processing [9], hardware-assistance [10], or transformation [11,12].

Without sacrificing image quality for rendering speed, the image generated by DVRM is the same as that generated by rendering all pixels in the image. When the amount of changed voxels in the differential file is small, DVRM is quite efficient. However, DVRM is not efficient when the amount of changed voxels is large. If the number of changed voxels is large, determining the changed pixels may make the rendering time longer than the time to do ray casting for all the pixels. We are motivated to alleviate the overhead to render TVVD more efficiently. Therefore, we propose a two-level differential volume rendering method in this paper. When TVVD evolves gradually, it is possible that some of the changed voxel positions appear in consecutive differential files. These changed voxels are projected to the same point on the image plane in consecutive time steps. Therefore, the projection computation for these changed voxels can be performed in the first of the consecutive time steps and can be omitted in the following time steps. Based on this property, the proposed method filters out the overlapped voxels and extracts the difference of changed voxel positions between consecutive differential files. The extracted difference information is referred as the second-order difference (SOD). The

differential files store difference information between volume data. We refer this information stored in differential files as the first-order difference (FOD). The proposed method uses FOD to update volume data. Based on SOD, the proposed method can determine the changed pixel positions more efficiently. Since we used FOD and SOD in the proposed method, the name "two-level differential" is used. To evaluate the proposed method, DVRM and a regular ray casting method are used as comparative methods. Three CFD datasets are used as test datasets. The experimental results show that the proposed method outperforms the comparative methods for all test datasets in most cases. In addition, the rendering time of the methods can be predicted once the data files are loaded in each time step.

This paper is organized as follows. In Section 2, the regular ray casting method and the differential volume rendering method are described. In Section 3, the proposed method is described in detail. In Section 4, we present the theoretical analysis and the experimental results. In Section 5, we describe a method of predicting the rendering time and present the prediction results.

2. The ray casting and the differential volume rendering methods

The ray casting method is a common backward-projection algorithm that can generate good quality images [13,14]. The method fires a ray from each pixel position into the volume data. At constant sampling interval, sampling points along the ray are given their interpolated gradients and sample values from the neighboring voxels. Given a transfer function, the gradients and sample values at sampling points are mapped to color and opacity values. Finally, the color and opacity of sampling points along the ray are accumulated to the pixel color [15]. Rendering TVVD by applying the ray casting method on the volume data in each time step without taking advantages of the TVVD coherency is referred as the regular ray casting method (RRCM) in this paper.

Since rays are processed independently from one another in the ray casting method, part of the rays can be updated while the rest of the rays are left unchanged. With this property, DVRM was proposed to render TVVD [6]. The method consists of two phases: the static phase and the dynamic phase. In the static phase, the changed voxels between consecutive time steps are extracted. The positions and values of changed voxels are stored as differential files. In the dynamic phase, the data are rendered. During the dynamic phase, the rendering parameters such as view direction and transfer functions are fixed. The first image is rendered in a regular method. For each following time step, the differential file is loaded to update the volume data. The positions of changed pixels are then determined. At the end of the time step, changed pixels are updated by firing new rays. The determination of changed pixels is based on the changed voxels and given parameters, for example, view direction and sampling method. When discrete rays and zero-order interpolation are used, a changed voxel is projected onto the image plane and the four pixels surrounding the projected point must be updated (shown in Fig. 1). When continuous rays and trilinear interpolation are used, the interpolation space that

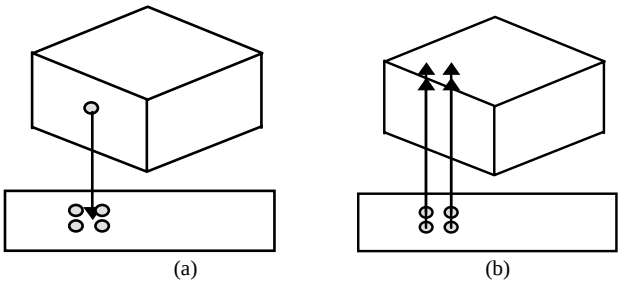

Fig. 1. (a) A changed voxel is projected onto the image plane. (b) The pixels surrounding the projected point must be updated.

encompasses a changed voxel is projected onto the image plane. The pixels located inside the projected region must be updated. If the number of changed pixels is smaller than the number of all pixels, the ray casting time are reduced. For each TVVD, the static phase is performed once. However, in order to explore the TVVD, the dynamic phase may be repeated many times, with different parameters in each time. Therefore, the total saved time become remarkable if the dynamic phase is repeated many times. Up to 90% of saving of the rendering time in the dynamic phase is reported by using DVRM on some datasets. For the above datasets, the amount of the changed voxels ranges from 0.005% to 4.77% of the total voxels between consecutive time steps. When the amount of changed voxels exceeds a threshold, the expensive projection computation makes the rendering time of DVRM longer than that of RRCM.

3. The two-level differential volume rendering method

In this section, the proposed two-level differential volume rendering method (TLDVRM) is described in detail. Similar to DVRM, TLDVRM also consists of two phases: the static and the dynamic phases. In the static phase, the proposed method extracts FOD and SOD. In the dynamic phase, the proposed method updates volume data by using FOD; determines the positions of changed pixels by using either FOD or SOD; and updates those changed pixels by casting new rays for them. We now define SOD and explain how to determine the changed pixels. The following are the notations used in this section.

- V_t: The volume data in time step t,
- (x, y, z): A voxel,
- (x, y, z, d_t): A changed voxel (x, y, z) and its voxel value d_t in time step t,
- DF_t: The differential file consisting of (x, y, z, d_t), changed voxels and their voxel values, in time step t,
- $P(DF_t)$: The set of changed voxels in DF_t,
- (r, s): A pixel.

3.1. The second-order difference

Given DF_{t-1} and DF_t, changed voxels in DF_{t-1} and DF_t can be classified according to their positions into three categories:

Category 1. Changed voxels are in DF_{t-1} but are not in DF_t.
Category 2. Changed voxels are not in DF_{t-1} but are in DF_t.
Category 3. Changed voxels are in DF_{t-1} and DF_t.

Changed voxels belonging to Categories 1 and 2 are the difference information between DF_{t-1} and DF_t. Changed voxels belonging to Category 3 are overlapped changed voxels of DF_{t-1} and DF_t. Hereby we have the following definitions.

Definition 1. Given DF_{t-1} and DF_t, let $OVP_t = P(DF_{t-1}) \cap P(DF_t)$ denote the overlapped changed voxels that appear in both $P(DF_{t-1})$ and $P(DF_t)$.

Definition 2. Given DF_{t-1} and DF_t, let $MSOD_t = P(DF_{t-1}) - P(DF_t)$, $PSOD_t = P(DF_t) - P(DF_{t-1})$, and $TSOD_t = MSOD_t \cup PSOD_t$, where $MSOD_t$, $PSOD_t$, and $TSOD_t$ denote the *minus* SOD, the *plus* SOD, and the *total* SOD in time step t, respectively.

In Definition 2, $MSOD_t$ is the set of changed voxels that appear in $P(DF_{t-1})$ but not in $P(DF_t)$. $PSOD_t$ is the set of changed voxels that appear in $P(DF_t)$ but not in $P(DF_{t-1})$. $TSOD_t$ is the extracted SOD in time step t. According to Definitions 1 and 2, we have the following intuitive lemmas.

Lemma 1. $MSOD_t \cap OVP_t = \varnothing$ *and* $P(DF_{t-1}) = MSOD_t \cup OVP_t$.

Lemma 2. $PSOD_t \cap OVP_t = \varnothing$ *and* $P(DF_t) = PSOD_t \cup OVP_t$.

An example is given in Fig. 2 to explain the meanings of Definitions 1 and 2.

In Fig. 3, we show how to extract SOD from differential files. For convenience, the three-dimensional volume data, differential files, and SOD are represented with two-dimensional matrices. For volume data and differential files, data stored in the matrices represent voxel values. For SOD, the signs "−" and "+" stored in matrices indicate that the indicated voxels are in the *minus* SOD and in the *plus* SOD, respectively. The voxels whose values are changed in time steps 1 and 2 are extracted to form DF_1 and DF_2, respectively. In time step 2, $TSOD_2$ is extracted by comparing DF_1 and DF_2. In $TSOD_2$, the voxel position with "−" is obtained because the corresponding changed voxel appears in DF_1 but not in DF_2. On the other hand, the voxel position with "+" value is obtained because the corresponding changed voxel appears in DF_2 but not in DF_1. Note that the values of the four inner voxels are changed in both time steps 1 and 2. These inner voxels are overlapped in DF_1 and DF_2, i.e., they form OVP_2.

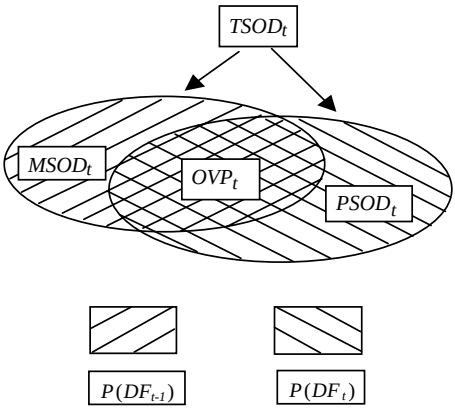

Fig. 2. An example of Definitions 1 and 2.

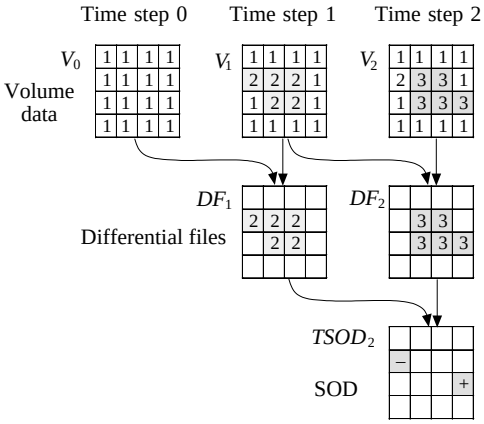

Fig. 3. An example of extracting the differential files and SOD. Grey parts represent changed voxels.

3.2. Determine the positions of changed pixels

We now describe how to determine the positions of changed pixel from FOD and SOD. A changed voxel may cause several changed pixels. On the other hand, a changed pixel may be due to several changed voxels. To clarify their relationships, we give the following definitions.

Definition 3. A voxel (x, y, z) is said to *influence* a pixel (r, s) if the pixel value at (r, s) needs to be updated due to the changed voxel value at (x, y, z). We use $J(x, y, z)$ to denote the set of pixels influenced by voxel (x, y, z).

Definition 4. The number of changed voxels that influence pixel (r, s) in time step t is defined as $I_t(r, s) = |W|$, where $W = \{(x, y, z) | (r, s) \in J(x, y, z)$ and $(x, y, z) \in P(DF_t)\}$. We called $I_t(r, s)$ the *influenced number* of pixel (r, s) in time step t.

$I_t(r,s)$ indicates how many changed voxels influence pixel (r,s) in time step t. For example, assume that there are ten elements, v_1, v_2,... and v_{10} in $P(DF_t)$. Among them, only v_1 and v_6 influence (r,s). Then $I_t((r,s))$ equals 2. From Definition 4, the following corollary is intuitive.

Corollary 1. *Pixel* (r,s) *is a changed pixel in time step* t *if and only if* $I_t(r,s) > 0$.

Since our purpose is to determine the positions of changed pixels, we can calculate $I_t(r,s)$ for each pixel (r,s) and use Corollary 1 to judge whether pixel (r,s) is a changed pixel. $I_t(r,s)$ can be calculated from either FOD or SOD.

To calculate $I_t(r,s)$ from FOD, we can check all changed voxels in DF_t and count how many changed voxels influence (r,s). The algorithm is given as follows.

*Algorithm calculate_*$I_t(r,s)$*_from_FOD*
/* Given DF_t*/
1. For each (r,s), $I_t(r,s) = 0$
2. For each (x,y,z,d_t) in DF_t\{
3. Compute J(x,y,z);
4. For each $(r,s) \in J(x,y,z)$, $I_t(r,s)++$;\}
*End_of_calculate_*$I_t(r,s)$*_from_FOD*

Another way to calculate $I_t(r,s)$ is based on SOD. According to Definitions 1 and 2, voxels in $MSOD_t$ are in $P(DF_{t-1})$ but not in $P(DF_t)$ while those in $PSOD_t$ are in $P(DF_t)$ but not in $P(DF_{t-1})$. Voxel positions in OVP_t are in both $P(DF_{t-1})$ and $P(DF_t)$. Given $I_{t-1}(r,s)$, if (r,s) is influenced by a voxel (x,y,z) in $MSOD_t$,$I_t(r,s) = I_{t-1}(r,s) - 1$ because (r,s) is no longer influenced by (x,y,z) in time step t. If (r,s) is influenced by a voxel (x,y,z) in $PSOD_t$, $I_t(r,s) = I_{t-1}(r,s) + 1$ because (r,s) becomes influenced by (x,y,z) in time step t. If (r,s) is influenced by a voxel in OVP_t, $I_t(r,s) = I_{t-1}(r,s)$. Therefore, OVP_t can be omitted in calculating $I_t(r,s)$. Hereby we have the following algorithm.

*Algorithm calculate_*$I_t(r,s)$*_from_SOD*
/*Given $I_{t-1}(r,s)$ and $TSOD_t$*/
1. For each (r,s), $I_t(r,s) = I_{t-1}(r,s)$;
2. For each (x,y,z) in $MSOD_t$ {
3. Compute $J(x,y,z)$;
4. For each $(r,s) \in J(x,y,z)$, $I_t(r,s)--$;\}
5. For each (x,y,z) in $PSOD_t$ {
6. Compute $J(x,y,z)$
7. For each $(r,s) \in J(x,y,z)$, $I_t(r,s)++$;\}
*End_of_calculate_*$I_t(r,s)$*_from_SOD*

Fig. 4 illustrates the process of calculating *influenced numbers* using the example shown in Fig. 3. In Fig. 4, the two-dimensional pixels are represented by one-dimensional arrays. The values stored in the arrays represent the *influenced numbers*

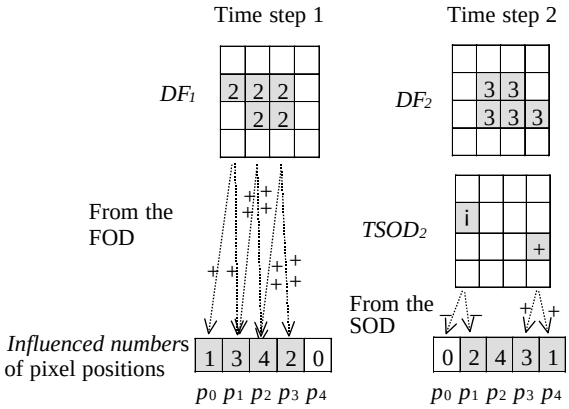

Fig. 4. Calculating influenced numbers of pixels. Grey pixels denote changed pixels.

of the pixels. A dotted line indicates that a pixel is influenced by a voxel. Each voxel influences two pixels and the voxels in the same column influence the same set of pixels in this example. In time step 1, the *influenced numbers* of the pixels are calculated based on FOD, from the voxels in DF_1 directly. In time step 2, the *influenced numbers* are calculated based on SOD. Pixels p_0 and p_1 are influenced by the voxels in $MSOD_2$. Therefore, the *influenced numbers* of p_0 and p_1 are subtracted by 1. Pixels p_3 and p_4 are influenced by the voxels in $PSOD_2$. Therefore, the *influenced numbers* of p_3 and p_4 are added by 1. We can also obtain the same *influenced numbers* from DF_2 directly, but the computation based on $TSOD_2$ is less than that based on DF_2 by three voxels.

The changed pixels can be determined from either FOD or SOD. In the following, we discuss under what circumstance using SOD is more efficient than using FOD and vice versa. In time step t, the time complexity of algorithm *calculate_*$I_t(r,s)$*_from_FOD* is $O(|P(DF_t)|)$ and the time complexity of algorithm *calculate_*$I_t(r,s)$*_from_SOD* is $O(|TSOD_t|)$. Now we may present the following remarks.

Remark 1. If $|P(DF_t)| < |TSOD_t|$, using FOD is more efficient than using SOD.

Remark 2. If $|P(DF_t)| < |TSOD_t|$, using SOD is more efficient than using FOD.

3.3. Algorithm of the two-level differential volume rendering method

The two-level differential volume rendering method is given as follows. In the static phase, FOD and SOD are obtained. In the dynamic phase, for time step 0, the volume data is read, the gradient on each voxel is computed, and the ray casting is performed for each pixel. The ray casting for each pixel includes the following operations on sampling points: interpolations of gradients and voxel values from the neighboring voxels, mappings of the gradients and voxel values to color and opacity values, and composition of the color and opacity values to the final pixel color. For

time step 1, the differential file is read to update the volume data and the gradient on each voxel is computed. Then the *influenced number*s are calculated from FOD. The pixels with *influenced number*s that are greater than zero are updated. For the time step $t > 1$, the differential file is read to update the volume data, the SOD file is read, and the gradient on each voxel is computed. The *influenced numbers* are calculated from FOD or SOD according to Remarks 1 and 2. The pixels with *influenced number*s that are greater than zero are updated. The algorithm is given as follows.

Algorithm TLDVRM
/*Static phase*/
1. Obtain FOD;
2. Obtain SOD;
/*Dynamic phase*/
3. For time step 0 {
4. Read volume data and perform ray casting for each pixel position;}
5. For time step 1 {
6. Read DF_1;
7. Update volume data;
8. Compute gradients;
9. *calculate_*$I_t(r,s)$*_from_FOD*;
10. Perform ray casting for each pixel position with $I_1(r,s) > 0$;}
11. For time step $t > 1${
12. Read DF_t,
13. Read $TSOD_t$;
14. Update volume data;
15. Compute gradients;
16. If $|TSOD_t| < |P(DF_t)|$, then
17. *calculate_*$I_t(r,s)$*_from_SOD*;
18. else *calculate_*$I_t(r,s)$*_from_FOD*;
19. Perform ray casting for each pixel position with $I_t(r,s) > 0$;}
End_of_TLDVRM

4. Analysis and experimental results

In this section, we first analyze the theoretical rendering time of RRCM, DVRM, and TLDVRM. Then, we compare their experimental performance on three CFD datasets.

4.1. Theoretical analysis

Each rendering process for the following time step is a pipeline composed of several stages: any time step of RRCM, the time step $t > 0$ in the dynamic phase of DVRM, and the time step $t > 1$ in the dynamic phase of TLDVRM. We divide the pipeline into the following stages: F, the stage to load files and update volume data;

G, the stage to compute the gradients; P, the stage to determine the changed pixels if necessary; C, the stage to do ray casting. The rendering time of a method in a time step is therefore the sum of the processing time of all stages and is expressed as the following equation:

$$T_t(M) = T_t(M,F) + T_t(M,G) + T_t(M,P) + T_t(M,C) \tag{1}$$

where t is a time step, M is a method and $M \in \{RRCM, DVRM, TLDVRM\}$, $T_t(M)$ denotes the rendering time of method M in time step t, $T_t(M,F)$, $T_t(M,G)$, $T_t(M,P)$, and $T_t(M,C)$ denote the processing time of the stages F, G, P, and C in $T_t(M)$, respectively.

Stage F is an I/O process and $T_t(M,F)$ depends on the size of the loaded data in each time step. Let $|V_t|$ be the number of voxels in the volume data $|V_t|$. For a given dataset, $|V_t|$ is the same in all time steps. In RRCM, the loaded data is a volume data file and the loaded data size is $|V_t|$ because each voxel value is one byte. In DVRM, the loaded data is a differential file and the loaded data size is $4|P(DF_t)|$ because each changed voxel (x,y,z,d_t) is four bytes. In TLDVRM, the loaded data includes a differential file and a SOD file. The loaded data size is $4|P(DF_t)| + 3|TSOD_t|$ because each voxel (x,y,z) in the SOD file is three bytes. Since the disk I/O is done sector by sector, $T_t(M,F)$ is proportional to the number of read sectors. $T_t(RRCM,F)$, $T_t(DVRM,F)$ and $T_t(TLDVRM,F)$ are expressed as the follows.

$$T_t(RRCM,F) = \lceil |V_t|/B \rceil \times C_f \tag{2a}$$

$$T_t(DVRM,F) = \lceil 4|P(DF_t)|/B \rceil \times C_f \tag{2b}$$

$$T_t(TLDVRM,F) = \lceil 4|P(DF_t)|/B \rceil \times C_f, + \lceil 3|TSOD_t|/B \rceil \times C_f \tag{2c}$$

where B is the size of a sector and C_f is the time to read a sector.

Stage G computes the gradient on each voxel. The computation is essentially the same for each voxel. Therefore, $T_t(M,G)$ is proportional to $|V_t|$. $T_t(M,G)$ is expressed as

$$T_t(M,G) = |V_t| \times C_g \tag{3}$$

where C_g is the time to compute the gradient of a voxel.

Stage P determines the positions of changed pixels. In RRCM, there is no determination of the changed pixels. Therefore, $T_t(RRCM,P) = 0$. In DVRM and TLDVRM, the computation in stage P is mainly the projections of changed voxels onto the image plane and the calculation of *influenced numbers* based on the projected point. Therefore, $T_t(DVRM,P)$ is proportional to $|P(DF_t)|$ and $T_t(TLDVRM,P)$ is proportional to the smaller of $|P(DF_t)|$ and $|TSOD_t|$. $T_t(RRCM,P)$, $T_t(DVRM,P)$ and $T_t(TLDVRM,P)$ are expressed as the following equations.

$$T_t(RRCM,P) = 0 \tag{4a}$$

$$T_t(DVRM,P) = |P(DF_t)| \times C_p \tag{4b}$$

$$T_t(TLDVRM,P) = \mathrm{Min}(|P(DF_t)|, |TSOD_t|) \times C_p \tag{4c}$$

where C_p is the time spent for a changed voxel position in stage P.

Before discussing stage C and $T_t(M, C)$, we have to discuss the numbers of sampling points along rays. A given data size, view direction and sampling interval determine the numbers of sampling points of rays. Usually the sampling interval is set to 1.0. Sometimes the sampling interval is shorter for the sake of higher resolution and better image quality. The shorter the sampling interval is, the more the sampling points are in a fixed length of ray. The volume data size and view direction determines the lengths of intersections of the cast rays and the volume data. The length of an intersection may be different from that of another intersection. Only insides the intersections, sampling points are taken. As a result, the number of sampling points of a ray may be different from that of another ray as shown in Fig. 5. In RRCM, all of the rays are cast in each time step. Let TS_t denotes the total number of sampling points in time step t. TS_t is the sum of the numbers of sampling points of all rays in time step t. Different view directions make TS_t different. However, if the view direction is given, TS_t is the same in all of the time steps. In DVRM and TLDVRM, only rays corresponding to changed pixels are cast. Let CS_t denote the number of cast sampling points in time step t. CS_t is the sum of the numbers of sampling points of the rays corresponding to changed pixels in time step t. Even if the view direction is fixed in each time step, CS_t may be different in each time step. However, in a given time step, CS_t are the same in DVRM and TLDVRM. The computation in stage C is mainly the interpolations, mappings and compositions of sampling points. For each sampling point, the computation is essentially the same. Therefore, $T_t(RRCM, C)$ is proportional to TS_t, while $T_t(DVRM, C)$ and $T_t(TLDVRM, C)$ are proportional to CS_t. $T_t(RRCM, C)$, $T_t(DVRM, C)$ and $T_t(TLDVRM, C)$ are given below.

$$T_t(RRCM, C) = TS_t \times C_c \tag{5a}$$

$$T_t(DVRM, C) = CS_t \times C_C \tag{5b}$$

$$T_t(TLDVRM, C) = CS_t \times C_C \tag{5C}$$

where C_C is the interpolation, mapping, and composition time of a sampling point.

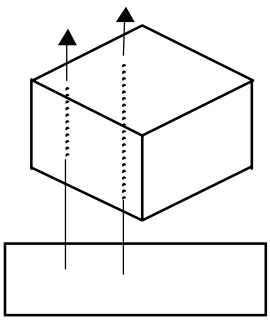

Fig. 5. The numbers of sampling points along different rays may be different.

Based on the above analysis, the rendering time of the three methods are given by Eqs. (6)–(8), respectively.

$$T_t(RRCM) = \lceil |V_t|/B \rceil \times C_f + |V_t| \times C_g + TS_t \times C_C \tag{6}$$

$$T_t(DVRM) = \lceil 4|P(DF_t)|/B \rceil \times C_f + |V_t| \times C_g + |P(DF_t)| \times C_p + CS_t \times C_C \tag{7}$$

$$T_t(TLDVRM) = \lceil 4|P(DF_t)|/B \rceil \times C_f + \lceil 3|TSOD_t|/B \rceil \times C_f + |V_t| \times C_g + \mathrm{Min}(|P(DF_t)|, |TSOD_t|) \times C_p + CS_t \times C_C \tag{8}$$

Based on Eqs. (6)–(8), the relative performance comparisons of pairs between RRCM and DVRM, RRCM and TLDVRM, and DVRM and TLDVRM are given as follows.

Remark 3. When Inequality (9) is true, DVRM is superior to RRCM.

$$\lceil |V_t|/B \rceil \times C_f + TS_t \times C_C > \lceil 4|P(DF_t)|/B \rceil \times C_f + |P(DF_t)| \times C_p + CS_t \times C_C \tag{9}$$

Remark 4. When $|P(DF_t)| < |TSOD_t|$ and Inequality (10) is true, TLDVRM is superior to RRCM.

$$\lceil |V_t|/B \rceil \times C_f + TS_t \times C_C > \lceil 4|P(DF_t)|/B \rceil \times C_f + \lceil 3|TSOD_t|/B \rceil \times C_f + |P(DF_t)| \times C_p + CS_t \times C_C \tag{10}$$

Remark 5. When $|P(DF_t)| < |TSOD_t|$ and Inequality (11) is true, TLDVRM is superior to RRCM.

$$\lceil |V_t|/B \rceil \times C_f + TS_t \times C_C > \lceil 4|P(DF_t)|/B \rceil \times C_f + |TSOD_t| \times C_p + CS_t \times C_C \tag{11}$$

Remark 6. When $|P(DF_t)| < |TSOD_t|$, DVRM is superior to TLDVRM by $\lceil 3|TSOD_t|/B \rceil \times C_f$.

Remark 7. When $|P(DF_t)| > |TSOD_t|$ and Inequality (12) is true, TLDVRM is superior to DVRM.

$$|P(DF_t)| \times C_p > \lceil 3|TSOD_t|/B \rceil \times C_f + |TSOD_t| \times C_p \tag{12}$$

4.2. Experimental results

Three CFD datasets are used as test TVVD. The datasets are numerical simulations of various arrangements of jets issuing vertically into a horizontal crossflow [16]. In the first dataset, only a jet issues into a crossflow. The dataset is of the size of $81 * 49 * 65$ voxels and has 100 time steps. In the second dataset, two jets located along the crossflow direction issue normally into a crossflow. The dataset is

of the size of $101*49*81$ voxels and has 100 time steps. In the third dataset, three jets located perpendicular to the crossflow direction issue normally into a crossflow. The dataset is of the size of $81*65*65$ voxels and has 100 time steps. In all dataset, each voxel value is 1 byte. The snapshots of the first dataset in time steps 20, 50, and 80 are shown in Figs 6(a-1, a-2, a-3), respectively. The snapshots of the second dataset in time steps 20, 50, and 80 are shown in Figs 6(b-1, b-2, b-3), respectively. The snapshots of the third dataset in time steps 20, 50, and 80 are shown in Figs 6(c-1, c-2, c-3), respectively. The experiments are performed on a Pentium III 800 MHz platform with 256 MB RAM.

Fig. 7(a) shows $|V_t|$, $|P(DF_t)|$, $|TSOD_t|$, TS_t and CS_t of the first dataset. Fig. 7(b) shows $T_t(M,F)$, $T_t(M,G)$, $T_t(M,P)$, and $T_t(M,C)$ of RRCM, DVRM, and TLDVRM for the first dataset. In Fig. 7(a), $|V_t|$ and TS_t are fixed in all time steps. $|P(DF_t)|$, $|TSOD_t|$and CS_t are varied in each time step. There is an apparent gap between the curves of $|P(DF_t)|$ and $|TSOD_t|$. For all time steps, the sum of $|TSOD_t|$ is about 42% of the sum of $|P(DF_t)|$. This gap makes TLDVRM save 58% of the time to determine changed pixel positions than DVRM. There is also an apparent gap between $|P(DF_t)|$ and CS_t. By comparing $(|P(DF_t)|/|V_t|)$ with (CS_t/TS_t), we can see that the former is usually far less than the latter. This means that a small ratio of changed voxels may lead to a larger ratio of sampling points to be processed. The above phenomenon is due to that a changed pixel is updated by processing all the sampling points, no matter how many of them are changed, along the ray.

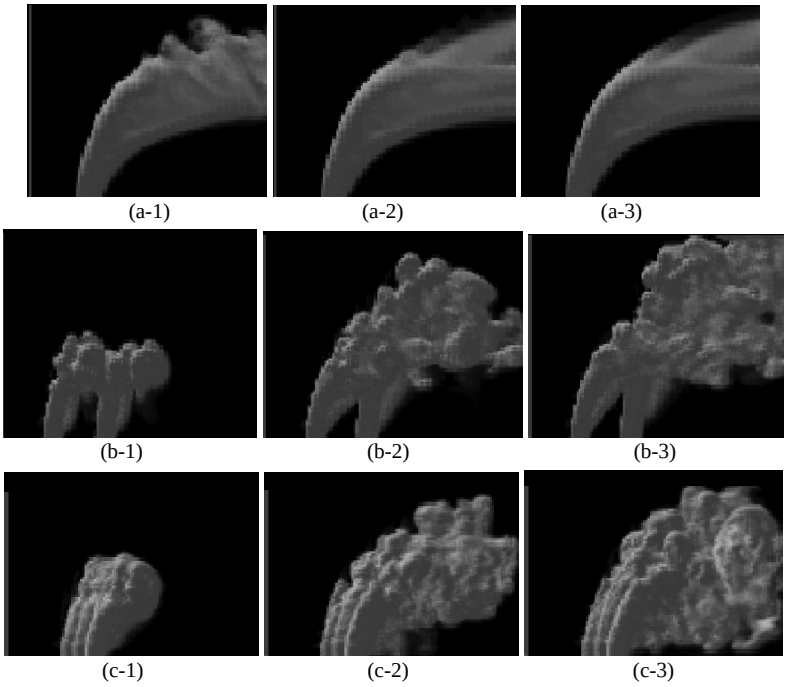

Fig. 6. Snapshots of the three test datasets.

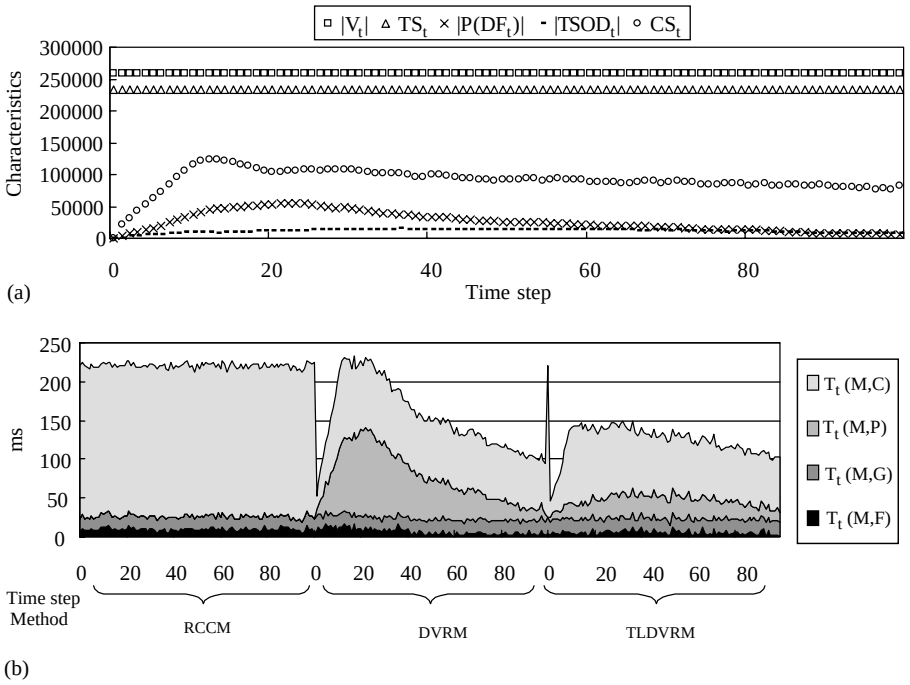

Fig. 7. (a) The characteristics of the first dataset. (b) Rendering time of the three methods for the first dataset.

In Fig. 7(b), $T_t(M,F)$, $T_t(M,G)$, $T_t(M,P)$, and $T_t(M,C)$ are accumulated, and the accumulated area becomes $T_t(M)$. In Fig. 7(b), $T_t(M,F)$ is too small and can be neglected. $T_t(M,G)$ conforms to Eq. (3) and is about 18 milliseconds. $T_t(DVRM,P)$ and $T_t(TLDVRM,P)$ conform to Eqs. (4b) and (4c), and C_p is about 1/500 milliseconds per voxel. Note that the shapes of curves of $T_t(DVRM,P)$ and of $|P(DF_t)|$ are similar. The shapes of curves of $T_t(TLDDVRM,P)$ and of $|TSOD_t|$ are also similar. $T_t(M,C)$ conforms to Eqs. (5a)–(5c), and C_C is about 1/1200 milliseconds per sampling point.

In most time steps except those between 18 and 30, DVRM is superior to RRCM because either $|P(DF_t)|$ or CS_t is small enough to make Inequality (9) true in Remark 3. In time steps 18–30, both $|P(DF_t)|$ and CS_t are large, and they jointly make Inequality (9) false. For TLDVRM and RRCM, TLDVRM is superior to RRCM in all time steps. The superiority is due to Inequality (11) in Remark 5 for most of the time steps. For TLDVRM and DVRM, in most of the time steps except the last few ones, TLDVRM is superior to DVRM because $|P(DF_t)| > |TSOD_t|$ and Inequality (12) in Remark 7 is true. The superiority of TLDVRM to DVRM is remarkable in time steps 10–40, where $|P(DF_t)|$ is much greater than $|TSOD_t|$. In the last few time steps, $|P(DF_t)|$ is very close to $|TSOD_t|$. The performance of DVRM is almost the same as that of TLDVRM. Fig. 8(a) shows $|V_t|$, $|P(DF_t)|$, $|TSOD_t|$, TS_t, and CS_t of the second dataset. Fig. 8(b) shows $T_t(M,F)$, $T_t(M,G)$, $T_t(M,P)$, and $T_t(M,C)$ of

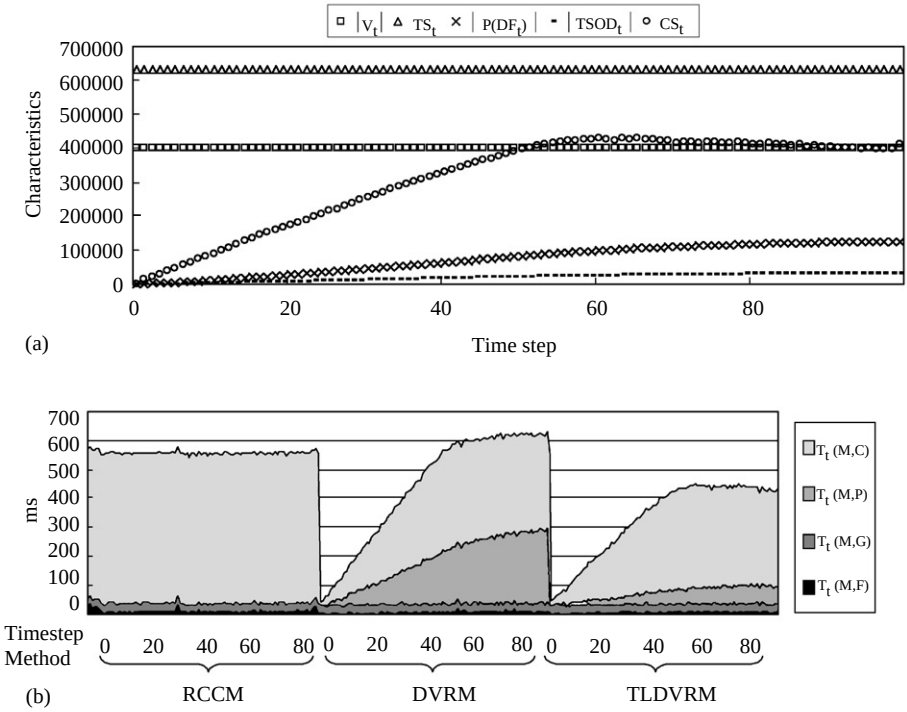

Fig. 8. (a) The characteristics of the second dataset. (b) Rendering time of the three methods for the second dataset.

RRCM, DVRM, and TLDVRM for the second dataset. In Fig 8(a), $|V_t|$ and TS_t are fixed in all time steps. $|P(DF_t)|$, $|TSOD_t|$, and CS_t increase gradually and then become steady. There is also an apparent gap between the curves of $|P(DF_t)|$ and $|TSOD_t|$. For all time steps, the sum of $|TSOD_t|$ is about 26% of the sum of $|P(DF_t)|$. This gap makes TLDVRM save 74% of the time to determine changed pixels. In this case, we use a shorter sampling interval 0.8333. This makes $TS_t > |V_t|$. There is also an apparent gap between $|P(DF_t)|$ and CS_t. By comparing $(|P(DF_t)|/|V_t|$ with (CS_t/TS_t), we can also see that the former is usually far less than the latter. Again, this means that a small ratio of changed voxels may lead to a larger ratio of sampling points to be processed because of the same reason in the first dataset.

In Fig. 8(b), $T_t(M,F)$, $T_t(M,G)$, $T_t(M,P)$, and $T_t(M,C)$ are accumulated, and the accumulated area becomes $T_t(M)$. In Fig. 8(b), $T_t(M,F)$ is too small and can be neglected. $T_t(M,G)$ conforms to Eq. (3) and is about 28 milliseconds. $T_t(DVRM,P)$ and $T_t(TLDVRM,P)$ also conform to Eqs. (4b)–(4c) and C_p is about 1/500 milliseconds per voxel position. The shapes of curves of $T_t(DVRM,P)$ and of $|P(DF_t)|$ are similar. The shapes of curves of $T_t(TLDVRM,P)$ and of $|TSOD_t|$ are also similar. $T_t(M,C)$ conforms to Eqs. (5a)–(5c) and C_C is about 1/1200 milliseconds per sampling point. Before time step 58, DVRM is superior to RRCM because either $|P(DF_t)|$ or CS_t is small enough to make Inequality (9) in

Remark 3 true. $|P(DF_t)|$ and CS_t become larger and larger, and they make Inequality (9) false after step 58. For TLDVRM and RRCM, TLDVRM is superior to RRCM in all time steps. The superiority is due to Inequality (11) in Remark 5. For TLDVRM and DVRM, TLDVRM is also superior to DVRM in all time steps. The superiority is due to Inequality (12) in Remark 7. No time step fits Remarks 4 and 6 because $|P(DF_t)| > |TSOD_t|$ in all time steps.

Fig. 9(a) shows $|V_t|$, $|P(DF_t)|$,$|TSOD_t|$, TS_t and CS_t of the third dataset. Fig. 9(b) shows, $T_t(M,F)$, $T_t(M,G)$, $T_t(M,P)$, and $T_t(M,C)$ of methods RRCM, DVRM, and TLDVRM for the third dataset. From Fig. 9, we have similar observations as those of Fig. 8.

It is interesting that the I/O time is neglected in the experimental results. The phenomenon seems conflicting to some previous works [9,10,17]. When a TVVD is rendered in parallel, the I/O time is relatively significant because the rendering time is short. When the TVVD size is very large and the physical memory does not fit the dataset, the I/O also becomes a bottleneck. However, our dataset is not very large

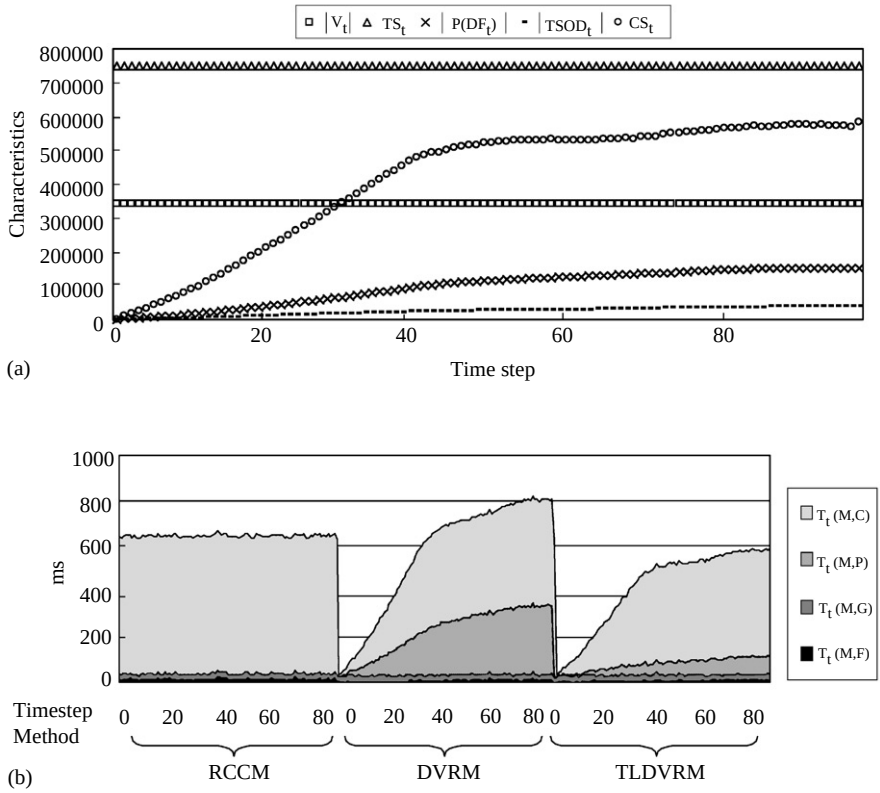

Fig. 9. (a) The characteristics of the third dataset. (b) Rendering time of the three methods for the third dataset.

and the platform is a single processor desktop computer. Hence the I/O time is not so significant in the rendering time.

5. Rendering time predictions

In the above experiments, the rendering time is measured after rendering. It is useful if the rendering time can be predicted in advance. For example, we can switch to the best method in each time step. In practice, the prediction of $T_t(RRCM)$ is rather simple. $T_t(RRCM)$ is almost constant in all time steps. If $T_t(RRCM)$ is measured in the first few time steps, for the rest of time steps, $T_t(RRCM)$ can be set as the previously measured $T_t(RRCM)$.

For DVRM (or TLDVRM), $T_t(DVRM)$ (or $T_t(DVRM)$) can be calculated if all the terms in Eqs. (7) or (8) are known. $|V_t|$ is fixed for a given dataset and TS_t is fixed for given parameters. $|V_t| \times C_g$, C_p and C_C are constant in each time step theoretically. In practice, $|V_t| \times C_g$, C_p and C_C are run time dependent and are known by measurement. Therefore, it takes several time steps to measure reliable values of $|V_t| \times C_g$, C_p and C_C in the first few time steps. In each time step, when stage F is performed, $|P(DF_t)|$, $\lceil 4|P(DF_t)|/Brceil \times C_f$ (for DVRM), $\lceil 4|P(DF_t)|/B \rceil \times C_f + \lceil 3|TSOD_t|/B \rceil \times C_f$ (for TLDVRM), $|TSOD_t|$ (for TLDVRM) can be obtained. As a result, immediately after stage F, the only unknown term is CS_t. If CS_t can be estimated, $T_t(DVRM)$ (or $T_t(DVRM)$) can be predicted after stage F is performed.

The accurate value of CS_t cannot be known until stage P is finished. However, it is possible to estimate an approximate CS_t immediately after stage F. If $|P(DF_t)|$ is large enough, a changed pixel is usually influenced by several changed voxels. Picking up any one of the changed voxels is enough to get the changed pixel. It is possible that a good sampling from $P(DF_t)$ may pick up enough changed voxels to get most of the changed pixels. If so, CS_t can be estimated approximately. Therefore, we use the following estimation technique to estimate an approximate CS_t. First, the technique randomly picks up a small ratio, say 5%, of changed voxels from $P(DF_t)$ as samples. Then the technique uses the picked samples to determine a set of changed pixels. Finally, the technique uses the set of changed pixels to calculate the estimated CS_t. The time for the estimation is very short, say, 5% of $|P(DF_t)| \times C_p$.

Experimental estimations for CS_t with 10% and 5% of $P(DF_t)$ are tested. The estimated CS_t and the accurate CS_t in each time step for the first, the second, and the third datasets are shown in Figs. 10(a–c), respectively. The estimation results are satisfying in most cases. In most cases, with 10% of $P(DF_t)$ as samples, the estimated CS_t is about 91% of the accurate CS_t in average. In most cases, with 5% of $P(DF_t)$ as samples, the estimated CS_t is about 87% of the accurate CS_t in average. Only when $|P(DF_t)|$ is too small and CS_t is too large, for example, in the latter time steps of the first dataset, the estimation results are not satisfying.

Based on the above experience values, we use 5% of $P(DF_t)$ as samples and compensate the estimated CS_t with a factor 1.1 in the rendering time predictions. The rendering time prediction algorithms are given as follows.

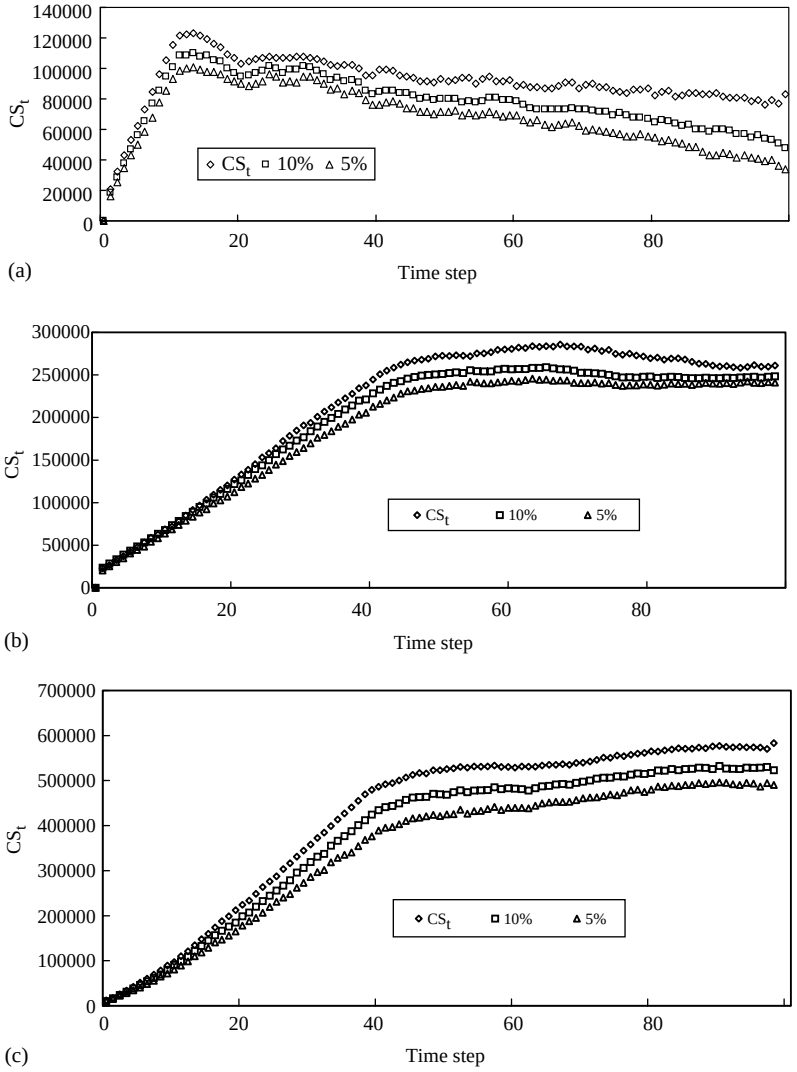

Fig. 10. Taking 10% and 5% of $P(DF_t)$ to predict CS_t for (a) the first, (b) the second, and (c) the third datasets.

Algorithm Predict_T(DVRM)

1. For time step$t<5$\{
2. Measure $|V_t| \times C_g, C_p, C_C$;\}
3. For time step $t >= 5$\{
4. In stage F \{
5. Measure $\lceil 4|P(DF_t)|/B \rceil \times C_f$\}
6. After stage F \{

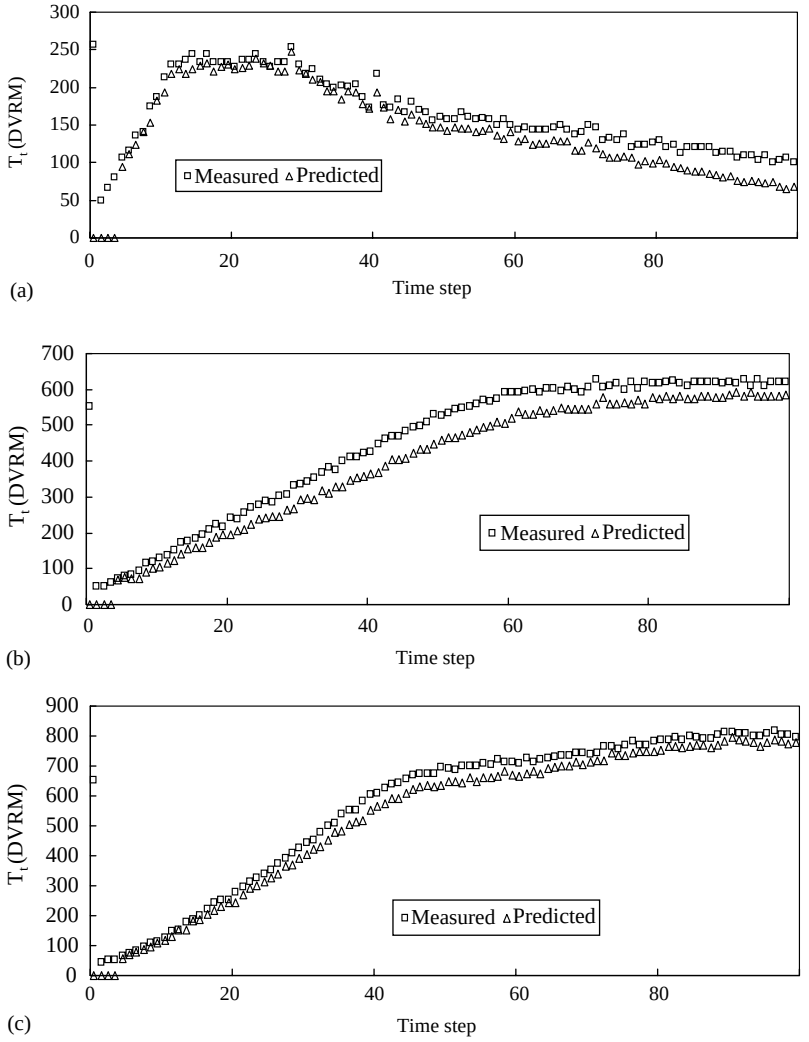

Fig. 11. The predicted $T_t(DVRM)$ and the measured $T_t(DVRM)$ for (a) the first, (b) the second, and (c) the third datasets.

7. Estimate CS_t with 5% of $P(DF_t)$
8. Calculate $T_t(DVRM) = \lceil 4|P(DF_t)|/B \rceil \times C_f + |V_t| \times C_g + |P(DF_t)| \times C_p + 1.1 \times CS_t \times C_C$

End_of_ Predict_T$(DVRM)$

Algorithm Predict_T$(TLDVRM)$

1. For time step t$<5\{$
2. Measure $|V_t| \times C_g, C_p, C_C;\}$

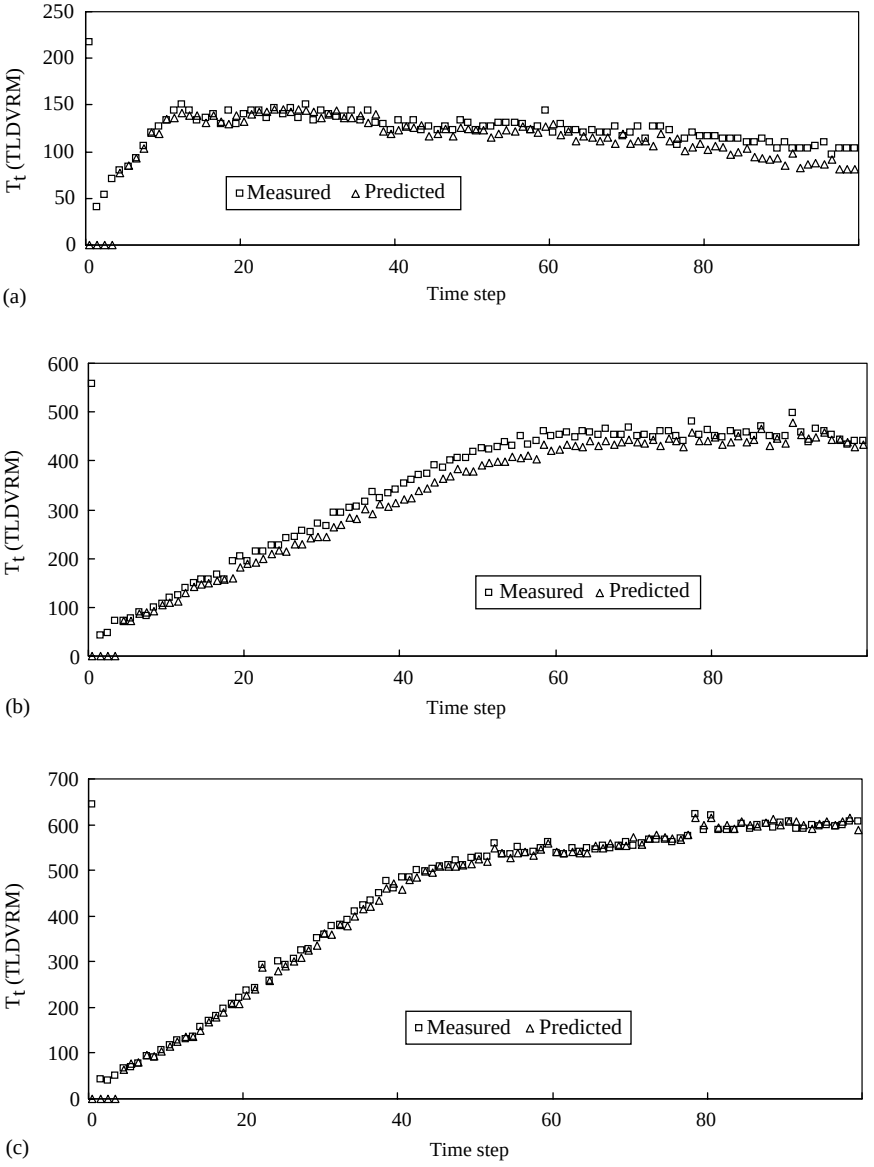

Fig. 12. The predicted $T_t(TLDVRM)$ and the measured $T_t(TLDVRM)$ for (a) the first, (b) the second, and (c) the third datasets.

3. For time step t > = 5{
4. In stage F{
5. Measure $\lceil 4|P(DF_t)|/B \rceil \times C_f + \lceil 3|TSOD_t|/B \rceil \times C_f$
6. After stage F{
7. Estimate CS_t with 5% of $P(DF_t)$

8. Calculate $T_t(TLDVRM) = \lceil 4|P(DF_t|/B\rceil \times C_f + \lceil 3|TSOD_t|/B\rceil \times C_f + |V_t| \times C_g + \mathrm{Min}(|P(DF_t)|,|TSOD_t|) \times C_p + 1.1 \times CS_t \times C_C$
End_of_Predict_T$(TLDVRM)$

Figs. 11(a–c) show the actually measured $T_t(DVRM)$ and predicted $T_t(DVRM)$ in each time step for the first, the second, and the third datasets, respectively. Figs. 12(a–c) show the actually measured $T_t(TLDVRM)$ and predicted $T_t(TLDVRM)$ in each time step for the first, the second, and the third datasets, respectively. The prediction results are excellent in most cases.

6. Conclusions and future work

In this paper, we have proposed TLDVRM to render TVVD. The core concept of the proposed method is to determine the positions of changed pixels by using SOD. From the theoretical analysis and the experimental results, we have the following conclusions. TLDVRM is superior to RRCM and DVRM in most cases. The performance of DVRM and TLDVRM is closely related to the characteristics such as $|P(DF_t)|$, $|TSOD_t|$, and CS_t of the datasets. When $|P(DF_t)|$ is much larger than $|TSOD_t|$, the superiority of TLDVRM to DVRM is remarkable. In addition, we can make excellent predictions of the rendering time.

A potential future work is based on the phenomenon that a small ratio of changed voxels may lead to a large ratio of sampling points to be processed. When a changed pixel is updated, all the sampling points along the ray must be processed even if very few sampling points change along the ray. As a result, a lot of effort is spent on the unchanged sampling points. It is possible to accelerate the rendering by reducing this kind of effort. Another potential future work is based on the rendering time prediction. The rendering time prediction is possible to be applied to the time-critical TVVD rendering. For example, some kind of weightings may be given to each changed pixel. According to their weightings, some changed pixels may be chosen to sacrifice to meet the requested time constraint.

References

[1] M. Levoy, Efficient ray tracing of volume data, ACM Transactions on Graphics 9 (3) (1990) 245–261.
[2] P.F. Fung, P.A. Heng, Efficient volume rendering by IsoRegion leaping acceleration, The Sixth International Conference in Central Europe on Computer Graphics and Visualization '98, 1998.
[3] P. Lacroute, Analysis of a parallel volume rendering system based on the shear-warp factorization, IEEE Transactions on Visualization and Computer Graphics 2 (3) (1996) 218–231.
[4] W.M. Hsu, Segmented ray casting for data parallel volume rendering, Proceedings of 1993 Parallel Rendering Symposium (PRS '93), San Jose, 1993, pp. 7–14.
[5] Anagnostou, T.J. Atherton, A.E. Waterfall, 4D Volume rendering with shear warp factorization, Volume Visualization and Graphics Symposium 2000, 2000.
[6] H.W. Shen, C.R. Johnson, Differential volume rendering: a fast volume visualization technique for flow animation, Proceeding of the Visualization '94 Conference, 1994, pp. 180–187.

[7] K.L. Ma, H.W. Shen, Compression and accelerated rendering of time-varying volume datasets, Workshop on Computer Graphics and Virtual Reality, 2000 International Computer Symposium, Taiwan, 2000.
[8] H.W. Shen, L.J. Chiang, K.L. Ma, A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree, Proceedings of the Visualization '99 Conference, pp. 1999, 371–377.
[9] T.C. Chiueh, K.L. Ma, A Parallel Pipelined Renderer for Time-Varying-Volume-Data, NASA/CR-97-206275, ICASE Report No. 97-70, 1997.
[10] J. Kniss, P. McCormick, A. McPherson, J. Ahrens, J. Painter, A. Keahey, C. Hansen, Interactive texture-based volume rendering for large data sets, IEEE Computer Graphics and Applications 21 (4) (2001) 52–61.
[11] R. Westermann, Compression domain rendering of time-resolved volume data, Proceeding of The Visualization '95 Conference, 1995, pp. 168–175.
[12] Y. Dobashi, V. Cingoski, K. Kaneda, H. Yamashita, A fast volume rendering method for time-varying 3D scalar field visualization using orthonormal wavelets, IEEE Transactions On Magnetics 34 (5) (1998) 3431–3434.
[13] M. Levoy, Display of surfaces from volume data, IEEE Computer Graphics and Applications 5 (3) (1988) 29–37.
[14] H. Ray, H. Pfister, D. Silver, T.A. Cook, Ray casting architectures for volume visualization, IEEE Transactions on Visualization & Computer Graphics 5 (3) (1999) 210–223.
[15] T. Porter, T. Duff, Compositing digital images, Computer Graphics 18(3) (1984).
[16] C.B. Liao, T.C. Lu, M.F. Wu, T.Y. Feng, Numerical simulation of multiple vertical jets issuing into an incompressible horizontal crossflow, The 8th National Computational Fluid Dynamics Conference, Taiwan, 2001.
[17] P.M. Sutton, C. Hansen, Accelerated isosurface extraction in time-varying fields, IEEE Transactions on Visualization & Computer Graphics 6 (2) (2000) 98–107.