A Two-Level Differential Volume Rendering Method for Time-Varying-Volume-Data

Shih-Kuan Liao, Yeh-Ching Chung[1], Jim Z.C. Lai

Department of Information Engineering
Feng Chia University, Taichung, Taiwan 407, ROC
Tel: 886-4-24517250 x3765
Fax: 886-4-24516101
Email: {skliao, ychung, lai}@fcu.edu.tw

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 fractions of volume data between consecutive time steps are extracted to form differential files. Based on the differential files, only the changed pixels, instead of all the pixels in the image, are updated by casting new rays at the positions in each time step. The main overhead of the differential volume rendering method is to determine the changed pixel positions before casting new rays for the changed pixels. In this paper, we propose a two-level differential volume rendering method, which is a modified differential volume rendering method with faster determination of the changed pixel positions. In the proposed method, the determination of the changed pixel positions is accelerated by the aid of second-order-difference. Since voxels in two consecutive differential files may partially overlap in the space, the computation spent on determining the changed pixel positions due to the overlapped area is redundant. We use this property to extract the difference of changed voxel positions between consecutive differential files to form the second-order-difference. Based on the second-order-difference, the changed pixel positions can be determined efficiently. The experimental results show that the proposed method outperforms the differential volume rendering method for all test datasets.

Keywords: Ray casting, differential volume rendering, time-varying-volume-data, flow animation, CFD.

1. INTRODUCTION

Volume rendering, which projects a 3D volume data into a 2D image, revealing the internal structure of the object, is a computation-demanding task [M.Lev90] [W.M.H93] [P.Lac96] [P.F.F98]. A time-varying-volume-data (TVVD) 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 provides dynamics of the phenomenon under study. For example, CFD simulations can be rendered into a flow animation. However, the computation demand in rendering TVVD is much more than that in rendering a single 3D volume data. In order to reduce the computation amount, many

[1] The corresponding author.

techniques have been proposed to exploit the spatial coherency and the 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.* [H.W.S94] proposed the differential volume rendering method to render TVVD. In this method, the changed fractions of volume data between consecutive time steps are extracted to form differential files. These files are used to determine the changed pixel positions on the image plane. Only the changed pixels, instead of all the pixels in the image, are updated by casting new rays at the positions in each time step. Ma *et al.* [K.L.M00] encoded each single volume data into an octree for the sake of the spatial coherency. For the sake of temporal coherency, consecutive octrees are merged. Pointers are used to represent identical subtrees in the consecutive octrees. In [H.W.S99], a time-space partitioning (TSP) tree was proposed. A TSP tree is a time-supplemented octree that utilizes both the spatial and the temporal coherency effectively and allows flexible tradeoff between the image quality and rendering speed. Based on the shear warp factorization, Anagnostou *et al.* [Anagn00] 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 technique [T.C.C97] or transformation technique [R.Wes95] [Y.Dob98].

The differential volume rendering method generates images of the same quality with those generated by a regular ray casting method. However, the overhead of the changed pixel positions determination may limit the method for TVVD. If the number of changed voxels is large, the time to determine the changed pixel positions may make the rendering time greater than that of a regular ray casting method in which all pixels of an image are rendered. To alleviate the overhead, in this paper, we propose a two-level differential volume rendering method. When TVVD evolves gradually, it is possible that some of the changed voxels appear in consecutive differential files. They are projected to the same pixel positions in consecutive time steps. Therefore, the pixel position determination for these voxels can be performed in the first time step of the consecutive time steps and can be omitted in the following steps. Based on this property, the proposed method filters out the overlapped voxel positions 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 the difference information stored in differential files as the first-order-difference (FOD). The proposed method uses the FOD to update volume data. Based on the SOD, the proposed method can determines the changed pixel positions more efficiently. Since we used the FOD and the SOD in the proposed method, the name two-level differential is used. Three CFD datasets are used to evaluate the proposed method. The experimental results show that the proposed method outperforms the differential volume rendering method for all test datasets.

This paper is organized as follows. In Section 2, the differential volume rendering method will be described briefly. In Section 3, the proposed method will be described in detail. In Section 4, the experimental results of a regular ray casting method, the differential volume rendering method, and the proposed method will be given.

2. THE DIFFERENTIAL VOLUME RENDERING METHOD

The differential volume rendering method proposed in [H.W.S94] consists of two phases. In the static phase, the changed fractions of volume data between consecutive time steps are extracted. The positions and the values of the changed voxels are stored as differential files. In the dynamic phase, the first image is rendered in a regular method. For each followed time step, the differential file is used to update the previous image. Based on the changed voxel positions in differential files, the pixel positions to be updated are determined according to the given parameters such as the view direction, and the sampling method. For example, when discrete rays and zero-order interpolation are used, a changed voxels position is projected onto the image plane. The four surrounding pixels of the projected point must be updated. When continuous rays and trilinear interpolation are used, the interpolation space that encompasses a changed voxel position is projected onto the image plane. The pixels located inside the projected region must be updated. The changed pixels are then updated by firing new rays in the pixel positions. If the number of changed pixels is relative few to the total pixel number, the ray casting time can be greatly reduced. For each TVVD, the static phase is performed once. However, in order to explore the data, 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 is reported by using the differential volume rendering on some datasets [H.W.S94]. However, the dataset does not change dramatically. The changed voxel ratio ranges from 0.005% to 4.77%. When the number of changed voxels is over a threshold, the rendering time become greater than that of a regular ray casting method.

3. THE TWO-LEVEL DIFFERENTIAL VOLUME RENDERING METHOD

We now describe the proposed method in detail. The following are the notations used in this paper.

- V_t: Volume data in time step t
- (x, y, z): A voxel position
- (x, y, z, d_t): A changed voxel in time step t, with value d_t in voxel position (x, y, z)
- DF_t: The differential file consisting of changed voxels in time step t
- $P(DF_t)$: The set of the positions of the changed voxels in DF_t
- (r, s): A pixel position

Similar to the differential volume rendering method, the two-level differential volume rendering method also consists of two phases, static and dynamic phases. In the static phase, the proposed method extracts the FOD and the SOD. In the dynamic phase, the proposed method updates volume data according to the FOD, determines the changed pixel positions by using either the FOD or the SOD, and updates those changed pixels by casting new rays for them. We now define the SOD and explain how to determine the changed pixel positions.

3.1 THE SECOND-ORDER-DIFFERENCE

Given $DF_{t\text{-}1}$ and DF_t, voxels in $DF_{t\text{-}1}$ and DF_t can be classified into three categories:

Category 1. Voxels are in $DF_{t\text{-}1}$ but are not in DF_t.

Category 2. Voxels are not in $DF_{t\text{-}1}$ but are in DF_t.

Category 3. Voxels are in $DF_{t\text{-}1}$and DF_t.

Voxels belong to Categories 1 and 2 are the difference information between $DF_{t\text{-}1}$ and DF_t. Voxels belong to Category 3 are overlapped voxels of $DF_{t\text{-}1}$ and DF_t. We have the following definitions.

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

Definition 2: Given $DF_{t\text{-}1}$ and DF_t, let $MSOD_t = P(DF_{t\text{-}1}) - P(DF_t)$, $PSOD_t = P(DF_t) - P(DF_{t\text{-}1})$, and $TSOD_t = MSOD_t \cup PSOD_t$ 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 voxel positions that appear in $P(DF_{t\text{-}1})$ but not in $P(DF_t)$. $PSOD_t$ is the set of changed voxel positions that appear in $P(DF_t)$ but not in $P(DF_{t\text{-}1})$. $TSOD_t$ is the extracted SOD in time step t. An example is given in Fig. 1 to explain the meanings of Definitions 1 and 2.

According to Definitions 1 and 2, we have the following intuitive lemmas.

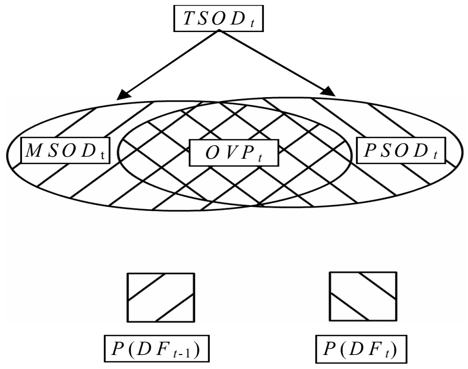


Fig. 1: An example of Definitions 1 and 2.

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

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

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

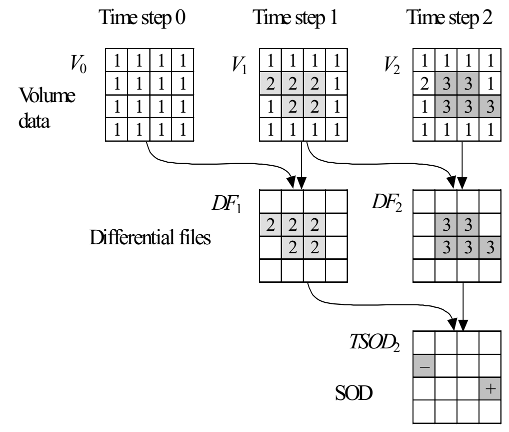


Fig. 2: An example of extracting the SOD from differential files. Grey parts represent changed voxels or changed voxel positions.

3.2 DETERMINE THE CHANGED PIXEL POSITIONS

We now describe how to determine the changed pixel positions from the FOD and the SOD.

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

As mentioned in Section 2, $J(x, y, z)$ depends on the rendering parameters such as view direction and the sampling method. For example, when discrete rays and zero-order interpolation are used, $J(x, y, z)$ consists of the four surrounding pixel positions that surround the projected point of (x, y, z) onto the image plane. When continuous rays and trilinear interpolation are used, the interpolation space that encompasses (x, y, z) is projected onto the image plane and $J(x, y, z)$ consists of the pixel positions that locate inside the projected region.

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

$I_t(r, s)$ indicates how many changed voxels influence (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 $I_t((r, s))$ is equal to 2. From Definition 4, the following corollary is intuitive.

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

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

To calculate $I_t(r, s)$ from the FOD, we can check all the changed voxels in DF_t and count how many voxel positions influence (r, s) to get $I_t(r, s)$. 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.       Calculate $J(x, y, z)$;
   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 the SOD. According to Definitions 1-2, voxels positions 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 $I_{t-1}(r, s)$, by Lemmas 1-2 and Definition 4, if (r, s) is influenced by a voxel position (x, y, z) in $MSOD_t$, $I_t(r, s) = I_{t-1}(r, s) - 1$ since (r, s) is no longer influenced by (x, y, z) in time step t. If (r, s) is influenced by a voxel position (x, y, z) in $PSOD_t$, $I_t(r, s) = I_{t-1}(r, s) + 1$ since (r, s) becomes influenced by (x, y, z) in time step t. If (r, s) is influenced by a voxel position in OVP_t, $I_t(r, s) = I_{t-1}(r, s)$. Therefore, OVP_t can be omitted in calculating $I_t(r, s)$. 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.       Calculate $J(x, y, z)$;
   4.       For each $(r, s) \in J(x, y, z)$, $I_t(r, s)$—;}
   5.  For each $(x, y, z)$ in $PSOD_t$ {
   6.       Calculate $J(x, y, z)$
   7.       For each $(r, s) \in J(x, y, z)$, $I_t(r, s)$++;}
End_of_calculate_$I_t(r,s)$_from_SOD
```

Fig. 3 illustrates the calculation of the *influenced numbers* using the example shown in Fig. 2. In Fig. 3, the two-dimensional pixel positions are represented by one-dimensional arrays. The values stored in the arrays represent the *influenced number*s of the pixel positions. A dotted line indicates that a pixel position is influenced by a voxel position. Each voxel position influences two pixel positions and the voxel positions in the same column influence the same set of pixel positions in this example. In time step 1, the *influenced number*s of the pixels are calculated based on the FOD. The *influence number*s are obtained via the five voxel positions of DF_1 directly. In time step 2, the *influenced number*s are calculated based on the SOD. In this case, pixel positions p_0 and p_1 are influenced by the voxel positions in $MSOD_2$, the *influenced number*s of p_0 and p_1 are subtracted by 1. Pixel positions p_3 and p_4 are influenced by the voxel positions in $PSOD_2$, the *influenced number*s of p_3 and p_4 are added by 1. We can obtain the same *influenced number*s from DF_2 directly, but the computation on $TSOD_2$ is less than that on DF_2 by three voxel positions.

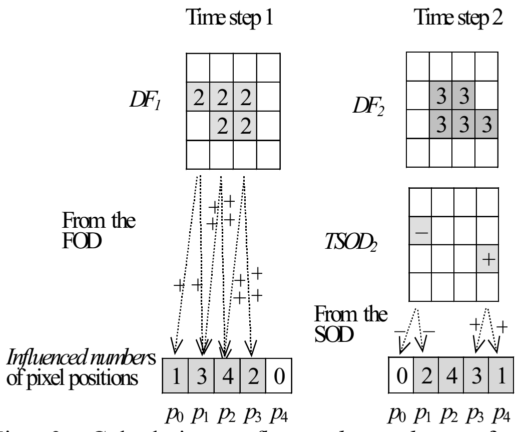


Fig. 3: Calculating *influenced number*s of pixel positions. Grey pixel positions are changed pixel positions.

Since the pixel positions can be determined from the FOD or the SOD, in the following, we discuss under what circumstance the use of the SOD has better performance than that of the 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 $O(|TSOD_t|)$. We have the following remarks.

Remark 1. If $|P(DF_t)| < |TSOD_t|$, the use of the FOD has better performance than that of the SOD.

Remark 2. If $|P(DF_t)| > |TSOD_t|$, the use of the SOD has better performance than that of the FOD.

3.3 ALGORITHM OF THE TWO-LEVEL DIFFERENTIAL VOLUME RENDERING METHOD

The two-level differential volume rendering method is given as follows. In algorithm *TLDVRM*, for the dynamic phase, time steps 0 and 1 are initialization time steps. In time step 0, the first image is rendered. In time step 1, *influenced number*s have to be calculated from the FOD. In the rest time steps, *influenced number*s are calculated from the FOD or the SOD according to Remarks 1 and 2 (lines 14-16). The changed pixels, whose *influenced number*s are greater than zero, are updated (lines 9 and 17).

```
Algorithm TLDVRM
    /* Static phase */
        1.  Obtain the FOD;
        2.  Obtain the 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.      calculate_ $I_t(r,s)$_from_FOD;
        9.      Perform ray casting for each pixel
                position with $I_1(r,s) > 0$; }
        10. For time step $t > 1$ {
        11.     Read $DF_t$;
        12.     Update volume data;
        13.     Read $TSOD_t$;
        14.     If $|TSOD_t| < |P(DF_t)|$, then
        15.         calculate_$I_t(r,s)$_from_SOD;
        16.     else
                    calculate_$I_t(r,s)$_from_FOD;
        17.     Perform ray casting for each pixel
                position with $I_t(r,s) > 0$; }
End_of_TLDVRM
```

4. EXPERIMENTAL COMPARISONS

To evaluate the proposed method, we compare the proposed method (referred as TLDVRM) with the regular ray casting method (referred as REGULAR) and the differential volume rendering method (referred as DVRM). Three CFD datasets are used as test TVVD. The datasets are numerical simulations of various arrangements of jets issuing vertically into a horizontal crossflow [C.B.L01]. In the first dataset, only a jet issues into a crossflow. The dataset is in 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 in 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 in size of 81*65*65 voxels and has 100 time steps. Snapshots of the first, the second, and the third dataset are illustrated in Fig. 4(a), 4(b) and 4(c), respectively. The experiments are performed on a Linux system running on a Pentium III 800 MHz platform with 512 MB RAM.

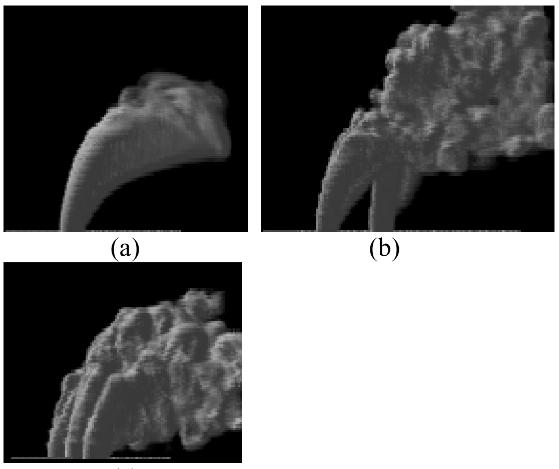
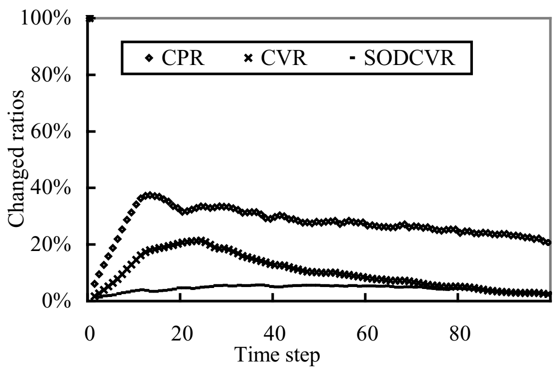


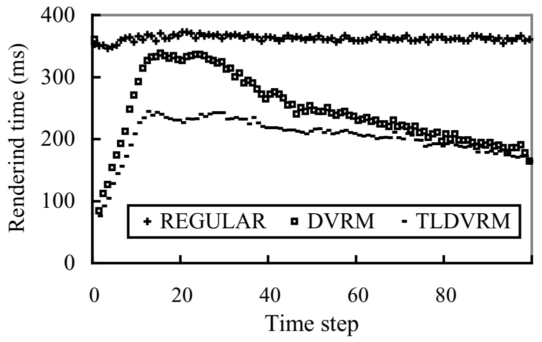
Fig. 4: Snapshots of the three test datasets.

Let CPR (changed pixel ratio) be the ratio of the changed pixels number to the total pixels number, CVR (changed voxel ratio) be the ratio of the changed voxels number ($|P(DF_t)|$) to the total voxels number, and SODCVR (SOD changed voxel ratio) be the ratio of the number of the SOD ($|TSOD_t|$) to the total voxels number in a time step. For each dataset, we show CPR, CVR, SODCVR, and the rendering time of the three methods in each time step. In a time step, the rendering time of each method includes the time to load all necessary files, to determine the changed pixel positions if necessary, and to perform ray casting for the necessary pixels. According to the experimental results, the time to load files is very small in comparison with the rendering time, ranging from 0.7% to 3.9%. Therefore, we will mainly discuss the time to determine the changed pixel positions and to perform ray casting.

Fig. 5(a) shows CPR, CVR, and SODCVR in each time step for the first dataset. Both CPR and CVR rise fast in the early time steps, reach the peak at 37.4% and 21.3%, respectively, and decay gradually to 20.6% and 2.4%, respectively. Note that even when the CVR is at its peak value (21.3%), the SODCVR is confined in 4.8%. The accumulated SODCVR is about 42% of the accumulated CVR for the 100 time steps. These differences between CVR and SODCVR are very helpful in reducing the rendering time. Fig. 5(b) shows the rendering times of the three methods on the first dataset. Note that the gap between DVRM and TLDVRM is similar to the gap between CVR and SODCVR. Since DVRM and TLDVRM use the same method to update changed pixels, the gap is mainly caused by the difference in determining the changed pixel positions. As we have expected, the benefit of the SOD is remarkable when CVR is high and SODCVR is low. For example, in time step 20, the rendering time of DVRM is 87.6% of that of REGULAR. However, in the same time step, the rendering time of TLDVRM is reduced to only 60.7% of that of REGULAR. In the later time steps, the rendering time of DVRM approaches to that of TLDVRM gradually. The reason is that CVR reduces gradually and finally the time to perform ray casting becomes dominating. For the 100 time steps, the rendering time of DVRM and TLDVRM is 68.1% and 56.6% of that of REGULAR, respectively. In TLDVRM, except in the initialization time step, only in one time step the *influenced numbers* are determined from the FOD rather than from the SOD.



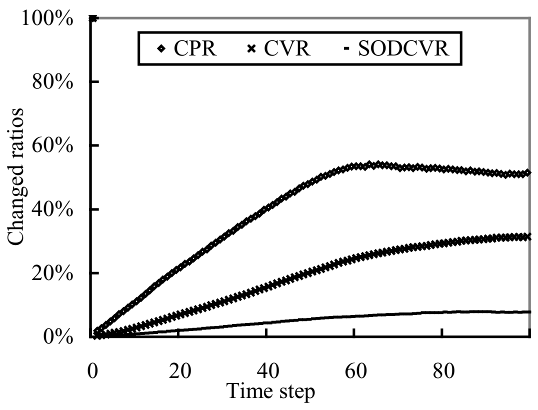
(a)



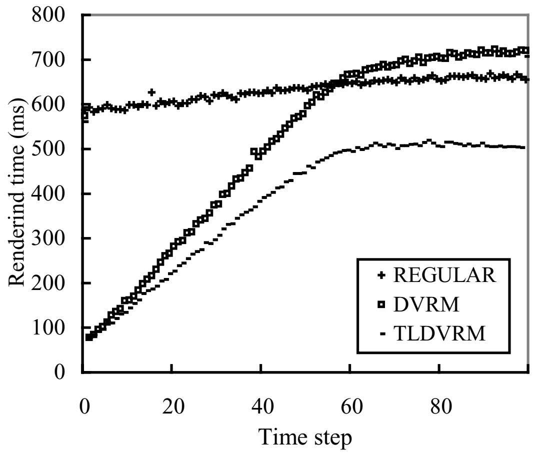
(b)

Fig. 5: Experimental results for the first dataset. (a) CPR, CVR, and SODCVR. (b) The rendering time of the three methods in each time step.

Fig. 6(a) shows CPR, CVR, and SODCVR in each time step for the second dataset. Both CPR and CVR rise gradually in the early time steps and reach about 52% and 31% in the later time steps, respectively. Again, SODCVR is confined in 7.8% when CVR is as high as 31.2% and the accumulated SODCVR is about 26.4% of the accumulated CVR for the 100 time steps. Fig. 6(b) sketches the rendering time of the three methods on the second dataset. Again, the gap between DVRM and TLDVRM is similar to the gap between CVR and SODCVR. Before time step 56, DVRM is more efficient than REGULAR, and TLDVRM reduces the rendering time further. After time step 56, the rendering time of DVRM exceeds that of REGULAR. However, TLDVRM is still more efficient than REGULAR. For the 100 time steps, the total rendering time of DVRM and TLDVRM is 80.8% and 61.6% of that of REGULAR, respectively. In all time steps except the initialization time step, the *influenced numbers* are determined from the SOD in TLDVRM.



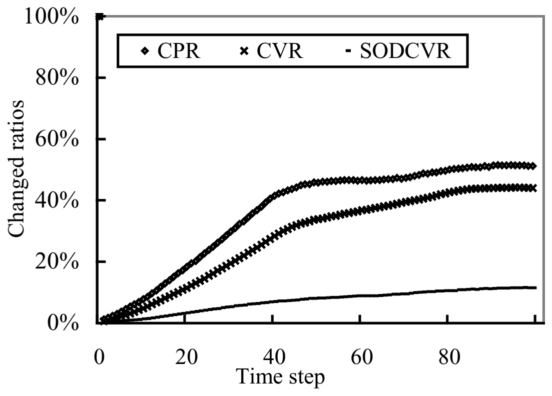
(a)



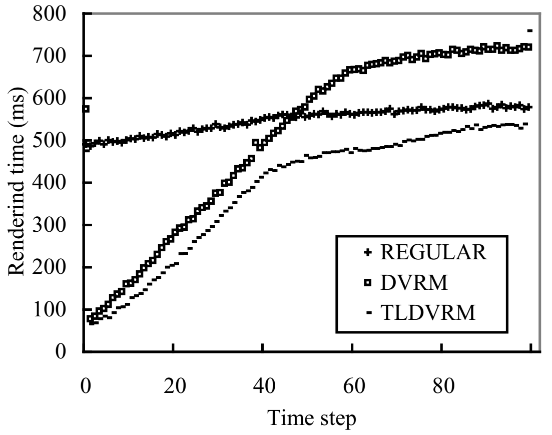
(b)

Fig. 6: Experimental results for the second dataset. (a) CPR, CVR, and SODCVR. (b) The rendering time of the three methods in each time step.

Experimental results on the third dataset are very similar to those of the second dataset. Fig. 7(a) shows CPR, CVR, and SODCVR in each time step for the third dataset. Both CPR and CVR rise gradually in the early time steps and reach about 52% and 44% in the later time steps, respectively. Again, SODCVR is confined in 11.5% when CVR is as high as 44.1% and the accumulated SODCVR is about 25.2% of the accumulated CVR for the 100 time steps. Fig. 7(b) sketches the rendering time of the three methods on the third dataset. Similarly, the gap between DVRM and TLDVRM is similar to the gap between CVR and SODCVR. Before time step 46, DVRM is more efficient than REGULAR, and TLDVRM reduces the rendering time further. After time step 46, the rendering time of DVRM exceeds that of REGULAR, but TLDVRM is still more efficient than REGULAR. For the 100 time steps, the total rendering time of DVRM and TLDVRM is 100.4% and 71.1% of that of REGULAR, respectively. The total rendering time of DVRM exceeds that of REGULAR while TLDVRM still outperforms REGULAR. In all time steps except the initialization time step, the *influenced number*s are determined from the SOD in TLDVRM.



(a)



(b)

Fig. 7: Experimental results for the third dataset.
(a) CPR, CVR, and SODCVR. (b) The rendering time of the three methods in each time step.

5. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a two-level differential volume rendering method to render TVVD. The core concept of the proposed method is to determine the changed pixel positions by using the SOD. By using the SOD, the time of changed pixel position determination can be reduced. If a large number of voxels change their values in consecutive time steps, the saved time in changed pixel position determination become remarkable. Experimental results on three CFD datasets show that the proposed method can greatly reduce the time to determine the changed pixel positions.

A potential future work is to determine the condition of switching between the proposed method and a regular method. We have determined the condition to switch between the FOD and the SOD from which the *influenced number*s are calculated. However, we saw in the datasets that a few changed voxels may lead to a lot of changed pixels in some cases. The proposed method may be less efficient than a regular ray casting method. Maybe we can perform some statistics on the volume data in the static phase. Based on the statistical data, we can switch to a more efficient method in the dynamic phase.

REFERENCES

[Anagn00] Anagnostou, T. J. Atherton and A. E. Waterfall: 4D Volume Rendering with Shear Warp Factorization, Volume Visualization and Graphics Symposium 2000, 2000.

[C.B.L01] C. B. Liao, T. C. Lu, M. F. Wu and T. Y. Feng: Numerical Simulation of Multiple Vertical Jets Issuing into an Incompressible Horizontal Crossflow, The 8th National Computational Fluid Dynamics Conference, Taiwan, 2001.

[H.W.S94] H. W. Shen and C. R. Johnson: Differential Volume Rendering: A Fast Volume Visualization Technique for Flow Animation, Proceeding of the Visualization ’94 Conference Pages 180-187, 1994.

[H.W.S99] H. W. Shen, L. J. Chiang and 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, Page 371–377, 1999.

[K.L.M00] K. L. Ma and 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.

[M.Lev90] M. Levoy: Efficient Ray Tracing of Volume Data, ACM Transactions on Graphics, 9(3): 245-261, 1990.

[P.F.F98] P. F. Fung and P. A. Heng: Efficient Volume Rendering by IsoRegion Leaping Acceleration, The Sixth International Conference in Central Europe on Computer Graphics and Visualization'98, 1998.

[P.Lac96] P. Lacroute: Analysis of a Parallel Volume Rendering System Based on the Shear-Warp Factorization, IEEE Transactions on Visualization and Computer Graphics, vol. 2, no. 3, pp. 218-231, 1996.

[R.Wes95] R. Westermann: Compression Domain Rendering of Time-Resolved Volume Data, Proceeding of The Visualization ’95 Conference Pages 168-175, 1995.

[T.C.C97] T. C. Chiueh and K. L. Ma: A Parallel Pipelined Renderer for Time-Varying-Volume-Data, NASA/CR-97-206275, ICASE Report No. 97-70, 1997.

[W.M.H93] W.M. Hsu: Segmented Ray Casting for Data Parallel Volume Rendering, Proceedings of 1993 Parallel Rendering Symposium (PRS’93), pp. 7-14, San Jose, 1993.

[Y.Dob98] Y. Dobashi, V. Cingoski, K. Kaneda, And H. Yamashita: A Fast Volume Rendering Mothod for Time-Varying 3D Scalar Field Visualization Using Orthonormal Wavelets, IEEE Transactions On Magnetics, 34(5), 1998.