Compression for Large-Scale Time-Varying Volume Data Using Spatio-temporal Features

Kun Zhao[1], Naohisa Sakamoto[2], and Koji Koyamada[2]

[1] Graduate School of Engineering, Kyoto University, Japan
[2] Institute for the Promotion of Excellence in Higher Education, Kyoto University, Japan
{zhao.kun,naohisas}@viz.media.kyoto-u.ac.jp,
koyamada.koji.3w@kyoto-u.ac.jp

Abstract. Data compression is always needed in large-scale time-varying volume visualization. In some recent application cases, the compression method is also required to provide a low-cost decompression process. In the present paper, we propose a compression scheme for large-scale time-varying volume data using the spatio-temporal features. With this compression scheme, we are able to provide a proper compression ratio to satisfy many system environments (even a low-spec environment) by setting proper compression parameters. After the compression, we can also provide a low-cost and fast decompression process for the compressed data. Furthermore, we implement a specialized particle-based volume rendering (PBVR) [2] to achieve an accelerated rendering process for the decompressed data. As a result, we confirm the effectiveness of our compression scheme by applying it to the large-scale time-varying turbulent combustion data.

Keywords: compression, time-varying, volume data, visualization.

1 Introduction

Time-dependent simulations can be found in many scientific fields (e.g. computational fluid dynamics, electromagnetic field simulation or ocean prediction). Good visualization for these time-dependent simulations is very important to clearly show changes and variations over time. Since the time-varying volume data from these simulations always have a very large data size, compression for the volume data is always needed to achieve an efficient visualization.

On the other hand, in some cases a high compression ratio is of course needed, low-cost and fast decompression for the compressed data is also required. For example, as an application of the visualization for ocean prediction dataset, “smart fishing" [7] is one of the cases that require a high compression ratio and also a low-cost and fast decompression. “Smart fishing” means efficient fishing (e.g. rapid search for fishery grounds) by using ocean prediction dataset. This dataset is the large-scale time-varying volume data, and needs to be transferred to fisherman to provide a detailed and interactive analysis for fishery grounds. Since the wireless communication speed on the ocean (mainly through the satellite) is slow, the

G. Tan et al. (Eds.): AsiaSim 2013, CCIS 402, pp. 136–148, 2013.

large-scale time-varying ocean data need to be compressed into a small data size to transfer it to the fisherman. Furthermore, the decompression process should be very low-cost and fast to satisfy low-spec hardware such as a tablet, which the fisherman may hold.

In the previous researches, there are many volume compression methods proposed [3, 4, 5, 6, 8, 9] to achieve the time-varying volume compression. However, these compression approaches always need a costly decompression process or would generate a large decompression size, which would not fit for the application requiring low-cost decompression (e.g. “smart fishing”).

To solve this problem, in the present paper we propose a volume compression scheme based on the spatio-temporal features of the time-varying volume data. In our proposed compression scheme, there are two compression processes performed to separately utilize the spatial features and temporal features of the large-scale time-varying data. In our previous work [1], we have proposed a compression scheme to utilize spatial features. In the present work, we add temporal domain compression to the previous method to achieve higher compression ratio. In detail, to utilize the temporal feature, we first calculate the temporal coherence between every two consecutive time steps, and then delete the vertices sharing coherence with the previous time step to compress the volume into a smaller data size. The compressed volume data is stored with a vertex table, which records the coherence information for each vertex, and a compressed values array. This simple structure could provide a fast decompression process (see details in section 3.2). Additionally, we provide two parameters: block size and tolerance rate, to control the precise of the compression precise and compression ratio. As a result, our compression scheme could provide a proper compression ratio and a low-cost decompression process. Moreover, we also implement a special method into the particle-based volume rendering (PBVR) [2] to render the decompressed volume data. As a result, the temporal features could not only be used to compress the volume data but also to accelerate the rendering process of PBVR.

To verify the efficiency of our proposed compression scheme, we apply our system to the large-scale time-varying turbulent combustion volume data. The experimental results show the efficiency of our proposed method.

2 Related Work

The large-scale time-varying volume data makes visualization a challenging problem. Due to the large data size, many volume compression techniques have been proposed to decrease the data size so that the rendering process could be done more easily.

Jens Schneider et al. [4] proposed a vector-quantization-based compression scheme for both static and time-varying volume data. With vector-quantization, this compression scheme could achieve a high compression ratio. However, it requires a costly decompression process, which could lead to a low frame-rate rendering result. Chaoli Wang et al. [9] utilized a wavelet-based time-space partitioning (WTSP) tree to compress the time-varying volume data in spatio-temporal domain to achieve an

efficient rendering. Wavelet is a very powerful compression technique. However, the decompression algorithm would cost much time to perform the inverse wavelet transform. Moreover, the preprocess for compression of wavelet is also costly. Sohn et al. [8] described a compression scheme for encoding time-varying volumetric features to support isosurface rendering. It is also based on a wavelet transform with temporal encoding so that the decompression process would need a costly inverse wavelet transform. Weiler et al. [6] proposed a hierarchical wavelet functional representation approach for interactive volume rendering, which also requires a inverse wavelet transform for decompression. Jorg Mensmann et al. [5] proposed a GPU-supported compression scheme for time-varying volume data. It needs to convert simulation data from 32-bit float into 16-bit integration as a preprocess to make sure the data size could be stored in GPU memory. However, the convert may lead to a large data loss because the the precise is decreased from 32-bit to 16-bit. Moreover, they use Lempel-Ziv-Oberhumer (LZO) as the compression algorithm, which could provide a fast decompression speed. But the decompressed data will have a same size with the original data. As for some time-varying data with a very high spatial resolution, the GPU memory still runs a risk of insufficiency especially for a low-spec hardware. The same problem could be found in the work of Yun Jang et al. [3], who used a functional representations to visualize the time-varying data. The decoded frame would need a 3D texture size depends on the original input data. This also would lead to an insufficiency of graphics memory for a low-spec hardware.

In conclusion, even though these compression approaches could achieve a high compression ratio for the time-varying volume data, they always need a costly decompression process or a large decompression size or a costly rendering process, which are not suitable for low-spec hardware.

3 Spatio-temporal Compression

In this paper, we propose a volume compression scheme based on the spatio-temporal features of the time-varying volume data. This compression scheme contains two compression processes: spatial domain compression and temporal domain compression, which to utilize the spatial features and temporal features. Here, the spatial features represent the properties in the volume structure for each independent time step. The temporal features mean the properties the volume data shares by consecutive time steps. These features could be utilized to decrease the volume size.

3.1 Spatial Domain Compression

For the first process, it is the spatial domain compression. We have proposed this compression process last year [1]. Here we make a brief review on it.

With the structured volume data, we use the two-level division to reduce the vertex number firstly, and fast cubic b-spline to reconstruct the divided volume (Figure 1). The two-level division first divides the volume into many blocks with the same cell number on each side (block size), and then subdivides every block into 24 tetrahedra.

After that, the value of newly generated vertex (black dots in Figure 1) is evaluated by fast cubic b-spline. We hereinafter call this structure as "24-tetrahedra" mesh.

After the compression, we implement a special compression data structured to store the "24-tetrahedra" mesh. The compressed data does not need the coordinate and connection information for the compressed tetrahedral mesh. Generally, the tetrahedral volume data needs the coordinate and connection data for the tetrahedral structure. However, in our system, since the division is done uniformly for the whole volume, a proper order of the values array is enough for the "24-tetrahedra" mesh. As a result, the compressed data contains: the block size used in the block division, the original volume resolution and the ordered values array for the divided mesh. Since the vertex number is decreased in the division process, the compressed data can have a very smaller size than the original data. Note that, different block size can generate different vertex number. Hence, we can provide a different compression ratio with a different block size.

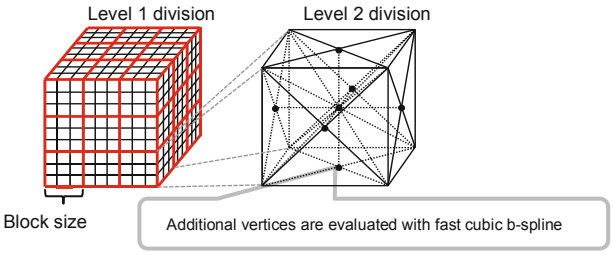

Fig. 1. Spatial domain compression with block size = 3. The left figure shows the first-level division and the right figure shows the second-level division. The black knot means the newly generated vertices by the second-level division.

3.2 Temporal Domain Compression

In many cases, time-varying volume data always keep an invariant mesh. We call such a kind of data as the mesh-invariant time-varying volume data. Since this kind of data often contains much temporal coherence, some parts of the volume may have a strong similarity for the consecutive time steps. Hence, we can achieve volume compression by deleting these similarity parts.

In our proposed technique, we explore the temporal coherence by calculating the difference of the vertex value between every two consecutive time steps. With the temporal coherence, we could compress the volume data by deleting the coherency vertices. At the meanwhile, we keep a very simple structure - vertex table (VT) to manage the coherency property for each vertex, which is referred to reconstruct the volume during the decompression. This simple structure is able to provide a low-cost decompression process. In general, a tree structure such as time-space partitioning (TSP) tree [10] is usually used to manage the time-varying volume to explore the temporal coherence [8, 9]. However, the tree traverse process of the TSP tree during the decompression process is very costly.

(A) Algorithm

Assume two consecutive time steps as the previous time step and current time step; we calculate the value of vertex table for the *i*th vertex as:

$$VT_i = \begin{cases} 1, & \left|V_i^{prev} - V_i^{curr}\right| < tolerance \\ 0, & else \end{cases} \quad (1)$$

Where V_i^{prev} represents the *i*th vertex value of the previous step, and V_i^{curr} represents the *i*th vertex value of the current step. Obviously, "1" means the vertex is coherency, and "0" means the vertex is not coherency. User can set a tolerance rate ϵ, and our system will calculate the tolerance as:

$$tolerance = \epsilon \times (V_{max}^{prev} - V_{min}^{prev}) \quad (2)$$

Here, V_{max}^{prev} and V_{min}^{prev} respectively means the maximum value and minimum value of the previous step. With the user assigned tolerance rate ϵ, the compression algorithm is shown as follows:

1. Calculate VT_i for the *i*th vertex with equation 4.2.
2. If VT_i = 1, delete this vertex from the current time step. If VT_i = 0, store this vertex into the compressed data. Move to the $i + 1$th vertex.
3. f i < N, move to step 1. If not, set i = 0 and perform step 1, 2, 3 to the next consecutive time steps.

(B) Compressed file

With the above compression algorithm, the coherency vertices are deleted. At the meanwhile, the vertex table is also stored with the compressed volume. We use a special structure to store the compressed volume (Figure 2), which contains vertices number, vertex table and compressed values array.

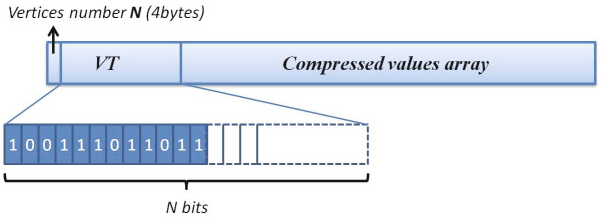

Fig. 2. The structure of temporal compressed volume file. *N* is the vertex number of the spatially compressed volume.

Obviously, if the time-varying volume data contains much temporal coherence, many coherence vertices could be deleted and a high compression ratio can be provided. We also allow user to change the tolerance rate ϵ to control the compression precise. In general, a high tolerance rate can provide a high compression ratio. On the contrary, a low tolerance rate would provide a relatively low compression ratio while keep a high compression precise.

4 Decompression and Visualization

4.1 Decompression

With compressed data, the decompression process only needs to copy the coherency vertices from the previous time step. Note that our decompression is not to reconstruct the compressed data into the original mesh but to reconstruct it into the "24-tetrahedra" mesh. That is because the original mesh has a large data size, which is very hard to handle. If we decompress the compressed data into the original size, the compression would be meaningless.

In detail, for each time step, we check the VT to determine whether to copy the vertex from the previous time step or not. VT is a simple array so that we do not need the costly tree traverse process, which are often used in other compression method to manage the compressed data structure [8, 9]. For each vertex, if the VT value for the vertex is "1", then copy the vertex from the previous time step; if not, use the vertex stored in the compressed data for this time step. This process is very fast since we only need to conduct the simple vertex copying process.

4.2 Visualization with Specialized PBVR

After the decompression process, we use PBVR to render the decompressed "24-tetrahedra" mesh. PBVR [2] is a stochastic rendering method, which first generates particles for the volume data and then projects particles to get the rendering image. The particle generation is performed in cell-by-cell manner and is done multiple times (repetition level) by using different random number. Generally, a higher repetition level can provide a better image quality. Because PBVR is very suitable for the large-scale irregular volume data [2], we use it to render the "24-tetrahedra" mesh. However, the particle generation always cost much time if the mesh contains many cells. In fact, since the time-varying volume data often contains much temporal coherence, these coherence is also be able to accelerate the particle generation process.

(A) Calculation for CT

Since PBVR generate particles in cell-by-cell manner, we utilize a temporal coherency cells table (CT) to record the coherence property for every cell. This temporal coherency CT could be simply calculated from VT. Basically, we use the following criterion to calculate the *i*th cells table value CT_i:

- CT_i = 1, if and only if the vertex values belonging to this cell are all calculated as "1" in VT.
- CT_i = 0, for any other case.

Figure 3 shows an example to calculate the cells table for the decompressed volume data. Since the decompressed volume data have a tetrahedral mesh, we need to check four vertex values, which belong to some cell, to calculate the CT value for this cell. As a result, the coherency cells are marked with "1" and none-coherency cells are marked with "0".

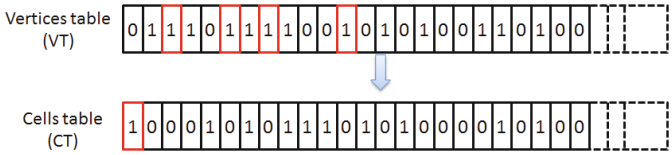

Fig. 3. The process of calculating the cells table. Since a tetrahedral cell has four vertices, every value in the CT needs to check four values on VT, which belongs to this cell.

(B) Generate Particles with CT

After the calculation of CT, the next step is to generate particles with CT. It is obvious that, we only need to generate particles for the none-coherency cell to save the generation time. Assume the cell number of a certain time step as N_c, the generation process is listed as following:

1. For the ith cell, check the value of CT_i.
2. If $CT_i = 0$, generate particles for this cell, and store the generated particles number into a particles table (PT). If $CT_i = 1$ skip this cell.
3. If $i \leq N_c + 1$, do $i = i + 1$ back to step 1. If not, move to the next time step, back to the step 1.

Figure 4 shows an example of generating particles with CT. Since the first time step does not have a CT, our system would first generate particles for all cells of this time step. For the following steps, our system only generates particles for the cell with the CT value equal to 0 and skips the cell with the CT value equal to 1. Moreover, to manage the particles, we also keep a particles table (PT) that stores the number of particles for each cell. During the rendering process, PT would be referred to copy the particles from the previous time step for the coherency cell.

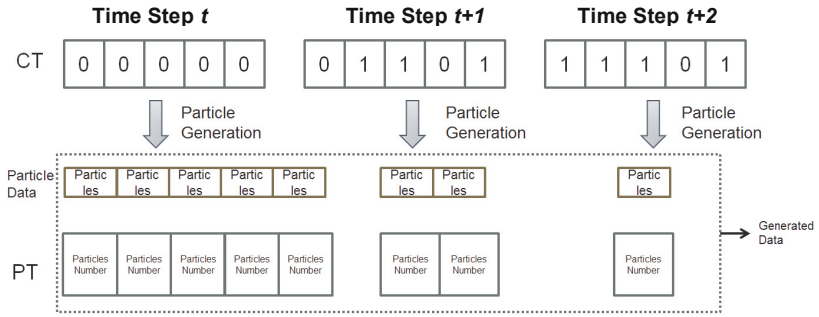

Fig. 4. The particle generation process with CT

(C) Rendering with CT and PT

With the partly generated particle data, we need to reconstruct it into the complete data before we render it into image plane. Here, the reconstruction process of the particle data mainly contains the copy process for the coherency cell. This copy process is performed with referring CT and PT simultaneously (Figure 5). In detail, the reconstruction process for the particle data is shown as follows:

1. For the *i*th cell, check the value of CT_i.
2. If $CT_i = 0$, do $PT_i^{curr} = PT_j^{part\_curr}$. And copy the particles from the partly generated particle data by referring the value of PT_i^{curr} to get the particle index of the particles needed to be copied. Then, perform $i = i+1$; $j = j+1$, move to step 4.
3. If $CTi = 1$, do $PT_i^{curr} = PT_i^{prev}$. And copy the particles from the reconstructed particle data of the previous time step by referring the value of PT_i^{curr} to get the particle index of the particles needed to be copied. Then, perform $i = i + 1$, move to step 4.
4. If $i \leq N_c + 1$, back to step 1. If not, move to next time step, back to step 1.

Here, PT_i^{curr} represents the PT value for the current time step of cell i. $PT_j^{part\_curr}$ represents the PT value for the partly generated particle of the current time step of the cell i (PT_0 in Figure 5). PT_i^{prev} represents the PT value for the reconstructed particle data of the previous time step of the cell i (Reconstructed PT in Figure 5). Figure 5 shows an example for the particle reconstruction process. After the reconstruction process, the particle data is projected to the image plane so that we can obtain the visualization result for the time-varying volume data.

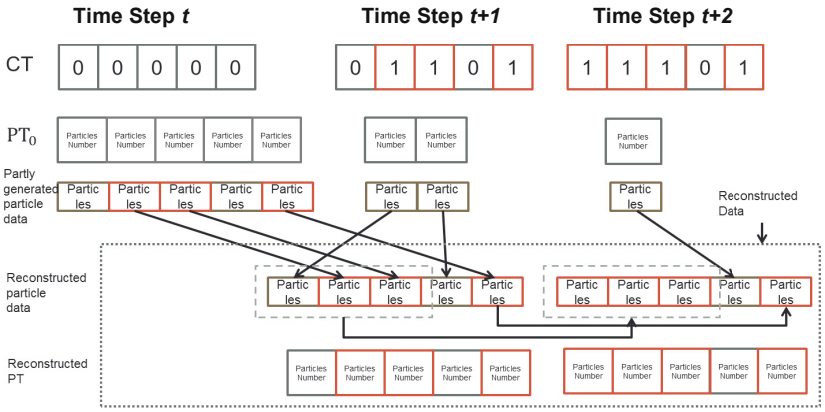

Fig. 5. The particle reconstruction process with CT

5 Experiments and Results

In this paper, we apply our method to the large-scale time-varying dataset obtained from a turbulent combustion simulation. The grid structure is Cartesian with uniform spacing and it is mesh-invariant time-varying volume data. There are 480×720×120 voxels composed in float type, and a total of 122 time steps. At each time step, there are 5 variables and about 158.2MB for each variable. In our experiment, we use the Y modality data and it reaches 18.9GB for all time steps. The experiment is conducted with an Intel Core i7-2820QM CPU (2.3 GHz), an NVidia GeForce GTX580M 2GB GPU, and 16GB system memory. The operation system is Ubuntu 12.04 LTS.

5.1 Compression Ratio

In our experiment, the compression ratio has been calculated as original data size over compressed data size. The compression ratios with different block size and tolerance rates are shown in Table 1. From this table we can see, a larger block size and a higher tolerance rate can provide a higher compression ratio. Of course, a smaller block size could provide a finer mesh and a lower tolerance rate could also generate a lower compression error.

Table 1. Compression ratio with different block size and tolerance rate

Tolerance Rate \ Block Size	2	3	4	5	6	7
10^{-7}	3.95:1	13.40:1	31.65:1	61.35:1	105.26:1	166.67:1
10^{-6}	4.14:1	14.06:1	33.11:1	64.52:1	111.11:1	175.44:1
10^{-5}	4.39:1	14.88:1	35.09:1	68.03:1	117.65:1	185.19:1
10^{-4}	4.92:1	16.67:1	39.37:1	76.34:1	131.58:1	208.33:1
10^{-3}	6.15:1	20.83:1	49.26:1	95.24:1	163.93:1	256.41:1

5.2 Compression Error

Except for compression ratio, a quantitative metric to evaluate the error is also needed. We use the following expression to evaluate the compression error (*CE*):

$$CE = \frac{\sum_{i=1}^{N} \left| V_{ori}^{i} - V_{com}^{i} \right|}{N} \tag{6}$$

Here N is the vertex number in the original volume. V_{ori}^{i} is the normalized value for vertex i of the original volume. Since the compressed volume is in "24-tetrahedra" structure, we linearly interpolate this mesh into the uniform mesh that is same with the original volume. V_{com}^{i} is the normalized value for vertex i of the interpolated volume. Table 2 shows compression error generated for the time step 35.

Table 2. Compression error with different block size and tolerance rate (time step 35)

Tolerance Rate \ Block Size	2	3	4	5	6	7
10^{-7}~10^{-3}	0.24%	0.46%	0.65%	0.99%	1.24%	1.65%

In fact, in our experiment we found that the compression error is mainly affected by spatial domain compression process. For example, Table 3 shows the compression error generated by the temporal domain compression separately. As we can see this compression error is much smaller than the results shown in Table 2. That is why we have the same total compression error for different tolerance rate (Table 2).

Table 3. Separate compression error for temporal domain compression (block size = 2, time step 35)

Tolerance Rate	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}
Compression Error	8.11×10^{-10}	8.03×10^{-9}	8.03×10^{-9}	8.03×10^{-9}	8.03×10^{-9}

5.3 Decompression Speed

With our proposed compression algorithm, the decompression time are measured with different block sizes and different tolerance rates. Here, the decompression time for all the time steps is shown in Table 4. From this table we could see that, since we do not need any value extraction process during the decompression, it is performed as fast as about 6 second or 7 seconds for all time steps.

Table 4. Decompression time with different block size for all time steps

Tolerance Rate \ Block Size	2	3	4	5	6	7
10^{-7}	6.63s	6.07s	5.92s	5.86s	5.84s	5.83s
10^{-6}	6.78s	6.17s	6.00s	5.96s	5.95s	5.92s
10^{-5}	6.88s	6.16s	6.01s	5.97s	5.93s	5.87s
10^{-4}	7.02s	6.37s	6.24s	6.18s	6.16s	6.13s
10^{-3}	6.92s	6.30s	6.19s	6.09s	6.07s	6.06s

5.4 Rendering with PBVR

After the decompression for the volume data, we render the decompressed data with our specialized PBVR process. Table 5 shows the speed of the particle generation with the decompressed volume data. Here, the block size is 2 and repetition level is 144. Since different tolerance rate would also generate different temporal coherence for the compressed volume data, we show the particle generation time for different tolerance rate.

Table 5. The result of particle generation for all time steps (block size)

Tolerance Rate	N/A	10^{-7}	10^{-6}	10^{-5}	10^{-4}	10^{-3}
Particle Generation Time	3,947.9s	3,256.8s	3,214.5s	3,149.1s	3,149.1s	2,786.1s

To make a comparison, the particle generation time without using the coherence cells table is shown in "N/A" row. This result shows our specialized PBVR can save much particle generation time (especially for a tolerance rate of 10^{-3}), which verifies the utilizing of temporal coherence into the PBVR process is very efficient.

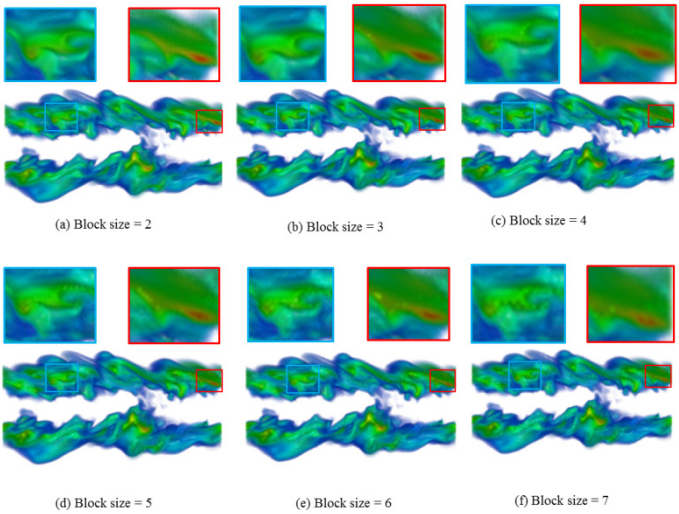

Fig. 6. Rendering result for the spatio-temporally compressed volume data with different block size and a tolerance rate of 10^{-3} (PBVR with phong shading and repetition level of 144). The part in the frame shows the detailed change with different block sizes.

Moreover, the rendering result for different block size with a tolerance rate of 10^{-3} is also shown in Figure 6. Here, PBVR is done with phong shading and a repetition level of 144. As mentioned in section 5.2, compression error is mainly affected by the block size, so that the image quality is also mainly affected by block size. It could be observed that finer details of the data are kept better when we have a low block size (see the blue frame of Figure 6). But we could also see that, even a larger block size such as 7, the important features (such as the red part in the red frame of Figure 6) could still be kept.

6 Discussion

From the experimental results we could see, our compression scheme could compress the time-varying data into different size with different compression parameters - block size and tolerance rate, which are used to control the calculation precise for spatial features and temporal features. Thus, we could provide a proper compression ratio to satisfy many system environments (even a low-spec environment) by setting proper compression parameters. This is a main feature of our compression scheme. Moreover, another main feature is that a fast decompression speed could be provided for the spatio-temporally compressed data. This could be observed in Table 4, where the decompression process cost only about 6 seconds or 7 seconds for all 122 time steps of the time-varying combustion data. After decompression, our specialized PBVR to utilize the temporal coherence also shows efficiency that the particle generation time could be accelerated remarkably (see Table 5). This accelerated rendering process is another feature of our compression scheme.

On the other hand, we also made a study for compression error. The block size and tolerance rate also affect the compression error. Obviously, a smaller block size with a lower tolerance rate could provide a compression result with a higher precise. In the experiment for the combustion data, we could see that when the tolerance rate is set smaller or equal to 10^{-3} the compression error is mainly affected by block size. But generally, we could also see that, even a larger block size such as 7, the important features (see Figure 6) could still be kept. This result shows that our compression scheme could provide a high compression ratio while keep a good visual quality.

However, a shortcoming of our system is that the particle generation time is still too long. As we can see from Table 5, even though we set the tolerance rate as 10^{-3}, it could still cost 2,786.1s for all time steps. This is because we have set the repetition level as 144, which means for each time step we need to generate particles with different random number for 144 times. By doing this we can get a good image quality for the time-varying data. Of course, we can get a much faster particle generation speed by setting a lower repetition level, but the image quality would also be affected (see details in T. Kawamura et al.'s work in [2]).

7 Conclusion and Future Work

In this paper, we have proposed a volume compression scheme for the large-scale time-varying volume data using spatio-temporal features. With this compression scheme, we could provide a proper compression ratio to satisfy many system environments (even a low-spec environment) by setting proper compression parameters. Moreover, since our compression scheme does not need special process such as inverse transfer during the decompression, we could provide a low-cost and fast decompression. Furthermore, we also implemented a special method into PBVR to accelerate the particle generation process when we render the spatio-temporally compressed volume. These features are verified in an experiment for the time-varying combustion data. As our future work, we plan to implement other rendering methods into our system to achieve a faster rendering while keep a good visual quality. Furthermore, since our system is suitable for the low-spec hardware for the low-cost decompression and rendering, we also plan to make an application for "smart fishing", to help fishers find rich fishery grounds.

Acknowledgement. This research was partially supported by the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Grant-in-Aid for Research Program on Climate Change Adaptation (RECCA), and by Japan Science and Technology Agency (JST), A-STEP project ("The research and development of fusion visualization technology", AS2415031H).

References

1. Zhao, K., Sakamoto, N., Koyamada, K.: A Volume Compression Scheme Based on Block Division with Fast Cubic B-spline Evaluation. In: Xiao, T., Zhang, L., Fei, M. (eds.) AsiaSim 2012, Part III. CCIS, vol. 325, pp. 373–387. Springer, Heidelberg (2012)

2. Kawamura, T., Sakamoto, N., Koyamada, K.: A Level-of-Detail Rendering of a Large-Scale Irregular Volume Dataset Using Particles. Journal of Computer Science and Technology 25(5), 905–915 (2010)
3. Jang, Y., Ebert, D.S., Gaither, K.: Time-Varying Data Visualization Using Functional Representations. IEEE Transactions on Visualization and Computer Graphics 18(3), 421–433 (2012)
4. Schnerder, J., Westermann, R.: Compression domain volume rendering. Proceedings of IEEE Visualization, 293–300 (2003)
5. Mensmann, J., Ropinski, T., Hinrichs, K.: A GPU-Supported Loss-less Compression Scheme for Rendering Time-Varying Volume Data. Volume Graphics Eurographics Association, pp. 109–116 (2010)
6. Weiler, M., Botchen, R.P., Stegmeier, S., Ertl, T., Huang, J., Jang, Y., Ebert, D.S., Gaither, K.P.: Hardware-assisted feature analysis of procedurally encoded multifield volumetric data. Computer Graphics and Applications 25(5), 72–81 (2005)
7. Aps, R., Fetissov, M., Lassen, H.: Smart management of the Baltic Sea fishery system: Myth or reality? In: Baltic International Symposium (BALTIC) 2010 IEEE/OES US/EU, (2010), pp. 1-9.
8. Sohn, B.-S., Bajaj, C., Siddavanahalli, V.: Volumetric video compression for interactive playback. Computer Vision and Image Understanding 96(3), 435–452 (2004)
9. Wang, C., Gao, J., Li, L., Shen, H.-W.: A Multiresolution Volume Rendering Framework for Large-Scale Time-Varying Data Visualization. In: Proceedings of the International Workshop on Volume Graphics, pp. 11–223 (2005)
10. 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: IEEE Visualization 1999, pp. 371–377 (1999)