Hardware-Assisted High-Efficiency Ray Casting of Unstructured Time-Varying Flows Using Temporal Coherence

Qianli Ma*, Liang Zeng, Huaxun Xu, Wenke Wang, Sikun Li

***Abstract*—Advances in computational power are enabling high-precision numerical simulations of unsteady flows using unstructured grids. The dynamic ray casting technique with the aid of texture hardware can achieve high-accuracy volume rendering of unstructured time-varying data from these simulations. However, the existing approach does not pay enough attention to temporal coherence, which depresses the rendering rate. Besides this, the texture structure used to store the mesh data results in a waste of GPU memory, which limits the mesh scale of the rendering data. This paper presents a high-efficiency dynamic ray casting algorithm for rendering unstructured time-varying fields using temporal coherence. Meanwhile, the pressure of GPU memory is effectually reduced by a well-designed texture structure. The analysis and experiments demonstrate that our approach gains a much lower cost of both time and space than the existing method and allows rendering time-varying data on a larger mesh scale in real time.**

***Index Terms*—Temporal coherence, unstructured grids, time-varying flows, GPU, ray casting**

I. Introduction

In the field of CFD, unstructured grids are widely applied to solve 3D flows for a high-precision numerical simulation. Advances in computational power enable the simulation of unsteady flows that produces time-varying data with hundreds of time steps. Visualization of these unstructured time-varying data offers the scientists powerful insight into the characteristic of unsteady flows and the reliability of simulation results.

Volume rendering, which is taken as the leading and preferred method to visualize 3D scalar fields, has many applications in flow visualization[1,2,7,9,11,16]. However, it is a challenge to render the unstructured time-varying volume data in real time by reason that: (1) volume rendering of even static unstructured-grid data is expensive due to the large mesh scale and the complicated topology, and (2) the dynamic (time-varying) volume data with a large amount of time steps (see Table 2) increase the difficulty in performing real-time rendering. The availability of texture hardware support for volume rendering enables real-time visualization of static unstructured-grid data. The GPU-based ray casting (HRC)[1] and the Hardware-Assisted Visibility Sorting (HAVS)[2] are two of the fastest volume rendering techniques using texture hardware for static unstructured-grid data. Recently, Bernardon et al.[3] proposed an approach that coupled a compression scheme[4] with these two techniques to render dynamic unstructured-grid volume data (we call them the dynamic HRC and the dynamic HAVS). Then they improved the dynamic HAVS with the aid of multiple processors[5]. However, these approaches do not pay enough attention to temporal coherence that plays an important role in visualizing time-varying data[7,8,9,10,11], which depresses the performance.

HAVS can render data on a larger mesh scale (main memory scale) than HRC (GPU memory scale), while HRC can lead to an image with higher accuracy[2,3] which is especially important for scientists to analyze the high-precision numerical resolutions. However, both the static and the dynamic HRC algorithms use a cell-based texture structure to store the whole mesh data. Each cell texture includes all its vertex data although a cell vertex is usually shared by a group of point-neighboring cells. This cell-based layout results in an inefficient storage since many redundant vertex data are stored in GPU memory. Moreover, the number of the cells is much larger than that of the vertices for most 3D unstructured-grid data from CFD simulations[1,2,3,5,6], so the cell-based texture structure increases the pressure of GPU memory even further.

This paper presents a novel dynamic ray casting algorithm to perform high-efficiency rendering of unstructured time-varying data using temporal coherence with the aid of texture hardware. Besides this, the pressure of GPU memory is effectually reduced by a well-designed texture structure. The analysis and experiments demonstrate that our approach gains a much lower cost of both time and GPU memory than the existing method and achieves a real-time performance even for time-varying data on a large mesh scale. To summarize, the major contributions of this paper are:

- We provide a method to qualitatively analyze temporal coherence of both the cell and the vertex data on unstructured grids. Then the cell and the vertex temporal tables are built based on the analysis result to achieve a lower time cost during ray traversal.
- Taking the characteristic of CFD unstructured grids into account, we design a novel texture structure that separates the vertex data from the cell data to reduce the pressure of GPU

Manuscript received December 8, 2010; revised January 8, 2011. This work is supported by the National Basic Research Program (No. 2009CB723803) and the National Science Foundation Program (No. 60873120) of China.

All the authors are with the College of Computer Science and Technology, National University of Defense Technology, China. (e-mail: {maqianliemail, liangzeng, huaxunxu, wenkewang, sikunli}@gmail.com).

*Corresponding author: phn: +8613974936415, address: Team 7, College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China, e-mail: maqianliemail@gmail.com.

ISBN: 978-988-19251-2-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

memory, allowing the storage of a larger-scale data set than the dynamic HRC.

- We propose to use 16 steps as a basic unit for data compression which enables a smarter codebook than the dynamic HRC, so that the codebook can be loaded faster to avoid rendering stalls while switching codebooks. Moreover, since there are two codebooks (corresponding to consecutive 32-step data) in GPU memory at a moment, they require 32-bit temporal tables which can be nicely laid out inside the textures leading to a compact and efficient storage (detailed in Sec. V).

II. Related Work

Research so far in time-varying volume data visualization has primarily utilized temporal coherence for fast rendering data on structured grids[3,7,16]. To improve the rendering performance, Shen[8] qualitatively analyzed temporal coherence of each voxel on structured grids and devised a temporal hierarchical index tree for fast isosurface extraction in time-varying fields. However, the tree does not maintain the spatial locality of the voxels and can not be readily adopted for volume rendering. Shen[9] and Ellsworth[10] proposed a time-space partition (TSP) tree for a better use of temporal and spatial coherence to achieve volume rendering of time-varying scalar fields on structured grids. They quantitatively analyze temporal coherence of the subvolumes on each spatial level and only use the mean values of the subvolumes that satisfy the temporal and spatial error tolerance to perform rendering. As a result, the amount of data required to be loaded into the main memory is reduced. This enables the algorithm to render a large-scale time-varying data in real time. Ma[11] organized the structured time-varying volume data with a group of octrees and used temporal coherence to prune the branches for each octree. Thus the demanding storage space is reduced, making it possible to render time-varying data.

Bernardon[3] compressed unstructured time-varying volume data into several codebooks with the vector quantization(VQ) approach[4]. Temporal coherence is used to gain a fast generation of the codebooks. Because the compression is done in a preprocessing stage, temporal coherence is not employed to save the time and space cost of the algorithm. In addition, an important difference between the static and the dynamic HRC algorithms is the representation of the cell gradient for reconstruction purpose during sampling. To reduce the usage of GPU memory, the dynamic HRC stores a gradient matrix [12] to compute the gradient of the scalar field in a cell (cell gradient) on line instead of the pre-computed cell gradient.

III. Temporal Coherence of Unstructured Time-Varying Flows

Sampling is the major part of the ray casting algorithm. During sampling, HRC reconstructs the field at a sample with the cell gradient and a vertex data value[1,3,6,16]. Thus temporal coherence of the cell and the vertex data can be used to reduce the cost of sampling for high-efficiency ray casting. To utilize temporal coherence, a method is presented to qualitatively analyze the temporal coherence of the cell and the vertex data on unstructured grids. In the preprocessing stage, the cell and the vertex temporal tables are built with the aid of the analysis result. Then these temporal tables are used to reduce the time cost for sampling during ray traversal.

A. The span space

The variation of the cell extreme values over time can help to analyze temporal coherence of a cell[8]. The cell extreme values combined with the maximum and the minimum among the whole vertex data values of the cell can be characterized by the span space[13]. Since tetrahedral meshes are the most common forms of unstructured grids, and other types of unstructured-grid cells can be effectually divided into tetrahedra. Therefore we only consider tetrahedral meshes in the following discussion. For a tetrahedral cell t, let $S^i_{t,0}$, $S^i_{t,1}$, $S^i_{t,2}$ and $S^i_{t,3}$ be its four vertex data values at the i th time step. Then the maximum value (denoted by $S^i_{t,\max}$) and the minimum value (denoted by $S^i_{t,\min}$) of cell t at the i th step are obtained by $S^i_{t,\max} = Max(S^i_{t,0}, S^i_{t,1}, S^i_{t,2}, S^i_{t,3})$ and $S^i_{t,\min} = Min(S^i_{t,0}, S^i_{t,1}, S^i_{t,2}, S^i_{t,3})$ respectively. In the span space, each cell is represented by a point whose x coordinate represents its minimum value and whose y coordinate represents its maximum value. For a time-varying field, a cell has multiple corresponding points in the span space, and each point represents the two extreme values of the cell at one time step. Fig.1 shows an example of the span space of cell t in the time interval $[0,15]$.

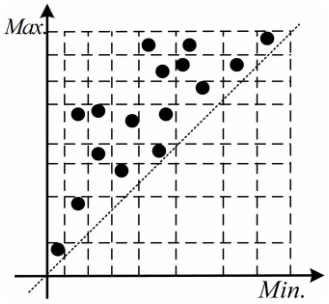


Fig. 1 The span space of cell t in a time interval [0,15]

B. Cell temporal coherence and cell temporal table

Given a time interval $[i, j]$ ($i, j \in \{0,1,...,n-1\}$ and $i \le j$), a cell's temporal coherence is determined by the spread of the cell's $j-i+1$ corresponding points in the span space. The narrower the spread is, the lower temporal variation and the stronger temporal coherence that the cell has. To quantify the spread, the lattice subdivision scheme[14] is applied to the span space. The scheme subdivides the span space into $N \times N$ non-uniformly spaced rectangular elements. The subdivision should ensure that the points are evenly distributed among the elements. Fig.1 is an example of the lattice subdivision of 8×8 lattice elements.

With the aid of the lattice subdivision, we can quantify the spread with $K \times K$ lattice elements ($K \in \{1,2,...,N\}$). A cell has strong temporal coherence in the time interval $[i, j]$ if its corresponding points in this interval are located within a spread of 2×2 lattice elements. Using this strong temporal coherence condition, we can build the cell's temporal (CT) table in the whole time interval of a time-varying field. For a time-varying field with n time steps, each cell has an n-bit CT table with binary entries whose values are decided by the

ISBN: 978-988-19251-2-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

following principle. First, we find a series of consecutive subintervals (denoted by $[0, n_0 - 1]$, $[n_0, n_1 - 1]$,…, $[n_{m-1}, n_m - 1]$, $[n_m, n-1]$) that divide the time interval $[0, n-1]$ into several parts. The division should make each subinterval include as many points as possible as long as they satisfy the strong temporal coherence condition. It guarantees that the cell has strong temporal coherence within each subinterval, and has weak temporal coherence between two consecutive subintervals. Then the cell's CT table can be created as shown in Fig. 2. Here, if the i th bit is filled with "0", it means that the cell has strong temporal coherence between the $i-1$ th and the i th time steps. Otherwise, it means that the cell has weak temporal coherence between the two steps.

$$\{\underbrace{1\,0\,0\ldots0}_{n_0\,\text{bits}},\ \underbrace{1\,0\,0\ldots0}_{(n_1-n_0)\text{bits}},\ \ldots,\ \underbrace{1\,0\,0\ldots0}_{(n_m-n_{m-1})\text{bits}},\ \underbrace{1\,0\,0\ldots0}_{(n-n_m)\text{bits}}\}$$

Fig. 2 A CT table in the time interval $[0, n-1]$

C. *Vertex temporal coherence and vertex temporal table*

The vertex temporal (VT) tables can be created from the CT tables. As mentioned above, temporal coherence of a cell is characterized by the variation of the cell extreme values which are the maximum and the minimum of the cell's vertex data values. It means that if the cell has strong temporal coherence in the given time interval $[i, j]$, each of its vertices also has strong temporal coherence. In most cases, it comes to the conclusion that a vertex has the same temporal table as the cell it belongs to. However, a vertex is usually shared by several cells. Consequently, temporal coherence may be different in strength among these point-neighboring cells.

In fact, there is usually strong spatial coherence among the neighboring cells by reason of the generation scheme for 3D unstructured grids[15,18]. It results in the similar VT tables among the point-neighboring cells. However, when there are discontinuity phenomena (e.g., shock waves) in flows, the state of the fluid as described by the density, pressure and other primitive variables can change radically across the discontinuity boundary. This also means that spatial coherence will be locally broken when a discontinuity arises during the development of an unsteady flow, which results in weak spatial coherence among the point-neighboring cells near the discontinuity boundary. To solve this conflict, we stipulate that when there are two or more point-neighboring cells with different temporal coherence at the i th time step (corresponding to the i th bit of a CT table), the shared vertex has weak temporal coherence with a VT table whose i th bit is "1". Suppose cell t_1 and cell t_2 are point-neighbors sharing vertex v. Given their CT tables {1000 0110 0011 1100} and {1000 0110 0010 0000}, the VT table of vertex v is {1000 0110 0011 1100}.

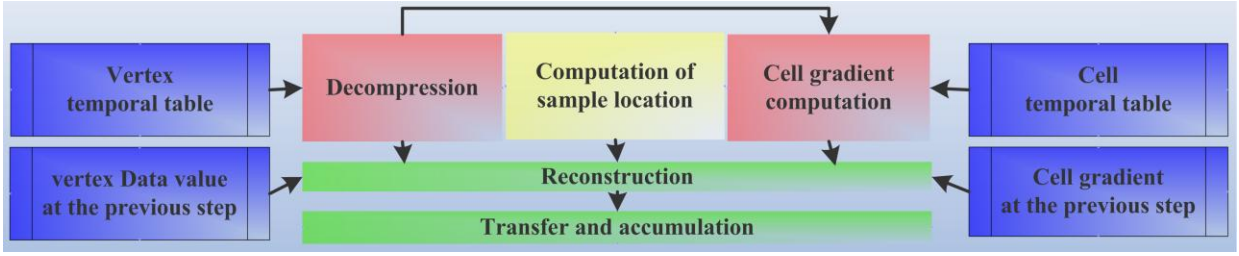


Fig. 3 Algorithm overview (sampling for one cell)

IV. Temporal Coherence Based Dynamic HRC Algorithm

We devise a high-efficiency dynamic ray casting algorithm for rendering unstructured time-varying data using temporal coherence. On each viewing ray, the algorithm does sampling once a cell during ray traversal (the sample is a ray-cell intersection) and transfers the reconstruction result (the field value at the sample) into color (RGBA) which is accumulated to the relevant pixel to form the image. Here, we focus on using temporal coherence to reduce the time cost of reconstruction which is the kernel part of sampling. The overview of our algorithm is illustrated in Fig. 3. Given the current time step i and cell t , it basically carries out the following steps:

Step 1: Compute the location of a new sample.

Step 2: Decompress the vertex data value(s) and compute the cell-gradient.

- Step 2.1: Evaluate the necessity of gradient computation using the CT table. If necessary, jump to Step 2.3.
- Step 2.2: Evaluate the necessity of data decompression for the reference vertex of cell t using the relevant TV table. If necessary, do decompression for the reference vertex, otherwise jump to Step 3.
- Step 2.3: Evaluate the necessity of data decompression for all the vertices of cell t using the VT tables and decompress the vertex data value(s). Then compute the cell-gradient with the gradient matrix[12] and the decompressed vertex data.

Step 3: Reconstruct the field at the sample.

- Do reconstruction using the computed cell gradient (or the cell gradient at the previous time step) and the decompressed vertex data value (or the vertex data value at the previous time step) by the linear gradient reconstruction method.

Step 4: Do color transfer and accumulation.

A. *Linear gradient reconstruction method*

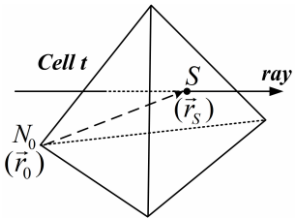


Fig. 4 Principle of the linear gradient reconstruction

The location of the sample (ray-cell intersection) can be obtained by using radial-polyhedron intersection [17]. Then the field at the sample is reconstructed by the linear gradient reconstruction method [6] (illustrated in Fig. 4) which is employed by the static and the dynamic HRC algorithms.

ISBN: 978-988-19251-2-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Suppose the intersection S is the sample of the current cell t. Given the sample location $\vec{r}_S$, the field (denoted by Q_S) at the sample can be reconstructed by the following linear gradient reconstruction equation:

$$Q_S = Q_0 + \nabla Q_t \bullet (\vec{r}_S - \vec{r}_0) \quad (1),$$

where the vector ∇Q_t is the cell gradient with the three components $\nabla Q_{t,x}$, $\nabla Q_{t,y}$ and $\nabla Q_{t,z}$, $\vec{r}_0$ and Q_0 are respectively the location and the data value of a cell vertex (called the reference vertex).

B. Temporal coherence based time-varying data reconstruction

Since the static HRC algorithm already uses texture memory to store the data, adding the time-varying data consume even more GPU memory. To reduce the memory consumption, the dynamic HRC algorithm uses the compressed Q_0 and the on-line computed ∇Q_t instead of the original Q_0 and the pre-computed ∇Q_t to perform reconstruction. This does assist in reducing the pressure of GPU memory. However, the on-line gradient computation and data decompression make reconstruction cost more time, which depresses the rendering rate. To maximize the rate, we use the CT and VT tables to accelerate gradient computation and data decompression during reconstruction. With the CT table, we can evaluate the necessity of gradient computation for reconstruction of a new sample. This helps to reduce the times of both gradient computation and data decompression. Given the time step i and cell t, if the i th bit of cell t 's CT table is "0", the gradient of cell t at the i th step (denoted by ∇Q_t^i) is approximately equal to the one at the $i-1$ th step (denoted by ∇Q_t^{i-1}). Thus the gradient ∇Q_t^{i-1} can be reused to perform reconstruction at the i th step instead of on-line gradient computation. With the VT table, we can evaluate the necessity of data decompression for the relevant vertices. This can also help to accelerate the gradient computation requiring the decompressed vertex data. Similarly, if the i th bit of vertex v_k 's VT table is "0", the data value of vertex v_k at the i th step (denoted by Q_k^i) is approximately equal to the one at the $i-1$ th step (denoted by Q_k^{i-1}). Thus the data value Q_k^{i-1} can be reused to perform reconstruction or gradient computation at the i th step instead of on-line data decompression.

V. Hardware-Assisted Implementation

The mesh scale of the data that HRC can render is limited by the capacity of GPU memory. This also means that special care must be taken when choosing how to layout the data inside textures. Moreover, a remarkable difference between the structured-grid and the unstructured-grid data is that the number of the cells is much larger than that of the vertices for most unstructured-grid data from CFD simulations [1,2,3,5,6,15]. Keeping this in mind, we design a novel data structure so that the time-varying fields can be nicely laid out and fit in the textures to save GPU memory space, allowing the storage of a larger mesh scale data set. Our data structure separates the vertex data from the cell data in a different manner from both the static and the dynamic HRC algorithms that merge the vertex data with the cell data inside the textures. This is very important for reducing the pressure of GPU memory. In addition, time-varying data with a large amount of steps make data loading (from the hard disk to GPU memory) the bottleneck of the volume rendering pipeline. We employ the same VQ approach[4] as the dynamic HRC does to compress the unstructured time-varying fields. An important difference is the scheme of data loading. We propose 32 steps as a basic unit (different from the dynamic HRC using 64 steps) for data loading which need the temporal tables 32 bits in length and thus leads to a compact and efficient texture structure (detailed in Sec. V)

A. Data compression and management

In the preprocessing stage, the VQ approach is employed to do the compression. It divides the time-varying data into several groups, each of which includes data within m consecutive time steps (where m is considered to be a square number for simplicity). Then the data in each group are compressed into a codebook (packed with 2D textures). During rendering, the codebook is loaded into GPU memory and accessed by its two indices for data decompression. To avoid rendering stalls while loading the codebook, the first two codebooks (corresponding to the first two groups of data) and the 32-bit temporal tables are loaded into GPU memory at the beginning of rendering. After the last time step data of the first codebook are accessed, the texture references are swapped to the second one which is already in GPU memory. The rendering process continues, while the next codebook and temporal tables are loaded in place of the first ones, so that the texture data of the next time step can be prepared before it is required.

The dynamic HRC algorithm uses 64 steps ($m=64$) per group as a basic unit for data loading. Each codebook uses $72KB = 256\times64\times4\text{B}+256\times8\times4\text{B}$ [3]. Instead, we propose 16 steps per group ($m=16$) whose codebook uses $20KB = 256\times16\times4\text{B}+256\times4\times4\text{B}$. Fig. 5 shows the layout of the codebook texture. This important change brings three main advantages. First, it reduces usage of GPU memory since the time of rendering 16-step time-varying data is enough to perform loading of the next group data. Second, there are always 32-step data be in GPU memory at a moment (there are two codebooks corresponding to two consecutive groups in GPU memory at a moment) that need a 32-bit temporal tables and thus leads to a compact and efficient texture structure (detailed in Sec. V.*B*). Third, compared to the 64-step data per group, the 16-step data can be compressed into a smarter codebook leading to faster data loading which can help to avoid rendering stalls while switching codebooks.

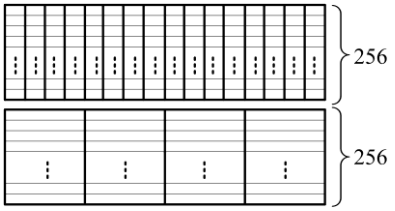


Fig. 5 Layout of the codebook texture for 16-step data

B. GPU texture structure

As mentioned above, the texture structures of both the static and the dynamic HRC algorithms merge the cell data with the

ISBN: 978-988-19251-2-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

vertex data and use the cell as a basic unit to store the fields[1,3]. They store the locations and the field values (or the codebook indices) of the cell's vertices and the cell gradient together in each cell texture. However, this texture structure is extravagant for HRC for a large amount of vertex data redundantly stored in GPU memory. To reduce the memory consumption, a texture structure is designed to separate the vertex data from the cell data as shown in table 1. The cell and the vertex textures respectively include the CT and the VT tables with a length of 32 bits (see the green part).

The CT table is used to evaluate the necessity of gradient computation during reconstruction. If not necessary, the gradient ∇Q_t^{i-1} (cell t 's gradient at the $(i-1)$ th step) can be reused to perform reconstruction at the i th step. So the cell gradient ∇Q_t^{i-1} (12B) should be stored in the cell texture and be updated with the lapse of time (see the red part in Table 1.(a)). It is combined with the 32-bit CT table (4B), nicely fitting in a texture vector (16B), which leads to a compact and efficient texture structure. Besides this, the gradient matrix[12] in the dynamic HRC is employed for on-line computation of the cell gradient. Therefore, the matrix should be stored in the cell texture (64B). In addition, the texture coordinates of the relevant vertices (for building the relationship between a cell and its vertices) and the face-neighboring cells (for ray traversal) should also be stored in the cell texture.

Similarly, we use the VT table to evaluate the necessity of data decompression. So the vertex data value at the previous time step (denoted by Q_N^{i-1}) should be stored in the vertex texture and be updated with the lapse of time (see the red part in Table 1.(b)). Besides this, the vertex texture should store the location of the vertex (12B) to compute the sample location [17] and the cell gradient. We combine it with Q_N^{i-1} (4B) to form a texture vector. To decompress the vertex data value, we use 12B to store the codebook indices which are combined with the 32-bit VT table just to form a texture vector.

Table 1 GPU texture structure used in our algorithm
(a) Cell texture (for one cell)

Cell Data	Texture Coordinates		Texture Data			
	u	v	R	G	B	A
vertex index	t_u	t_v	$N_u(v_{t,0})$	$N_v(v_{t,0})$	$N_v(v_{t,1})$	$N_v(v_{t,1})$
vertex index	t_u	t_v+dv	$N_u(v_{t,2})$	$N_v(v_{t,2})$	$N_v(v_{t,3})$	$N_v(v_{t,3})$
temporal coherence	t_u	t_v+2dv	$\nabla Q_{t,x}^{i-1}$	$\nabla Q_{t,y}^{i-1}$	$\nabla Q_{t,z}^{i-1}$	CT_t
gradient matrix	t_u	t_v	$m_{0,0}^t$	$m_{0,1}^t$	$m_{0,2}^t$	$m_{0,3}^t$
gradient matrix	t_u	t_v+dv	$m_{1,0}^t$	$m_{1,1}^t$	$m_{1,2}^t$	$m_{1,3}^t$
gradient matrix	t_u	t_v+2dv	$m_{2,0}^t$	$m_{2,1}^t$	$m_{2,2}^t$	$m_{2,3}^t$
gradient matrix	t_u	t_v	$m_{3,0}^t$	$m_{3,1}^t$	$m_{3,2}^t$	$m_{3,3}^t$
face adjacency	t_u	t_v+dv	$t_u(a_{t,0})$	$t_v(a_{t,0})$	$t_u(a_{t,1})$	$t_v(a_{t,1})$
face adjacency	t_u	t_v+2dv	$t_u(a_{t,2})$	$t_v(a_{t,2})$	$t_u(a_{t,3})$	$t_v(a_{t,3})$

(b) Vertex texture (for one vertex)

Vertex Data	Texture Coordinates		Texture Data			
	u	v	R	G	B	A
vertex coordinate	N_u	N_v	$\vec{r}_{N,x}$	$\vec{r}_{N,y}$	$\vec{r}_{N,z}$	Q_N^{i-1}
codebook index	N_u	N_v+dv	$mean_N$	$i4_N$	$i16_N$	VT_N

C. Analysis of the space cost

With our texture structure, the data stored per cell use $144\text{B} = 9\times 16\text{ B}$, and the data stored per vertex use $32\text{B} = 2\times 16\text{ B}$. Suppose there are c cells and v vertices in the tetrahedral mesh. Then the storage of the mesh data is given by $c\times 144B + v\times 32B$. For 16-step data per group, the codebook takes up the storage of $20KB$ (mentioned in Sec. V.*A*). At a moment, there are two codebooks corresponding to two consecutive groups (32 steps) in GPU memory. As a result, the space cost of our approach is given by $c\times 144B + v\times 32B + 40KB$.

The dynamic HRC[3], which combines the cell data with the vertex data and use the cell as a basic unit to store the mesh data, costs 192B storage per tetrahedron. Since it uses 64-step data per group, at a moment, the codebooks of two groups uses $144KB = 2\times 72KB$. So the space cost of the dynamic HRC is given by $c\times 192B + 144KB$.

As mentioned above, for most 3D unstructured-grid data from CFD simulations, the number of the cells is much larger than that of the vertices. As a result, our approach achieves a lower cost of GPU memory than the dynamic HRC, which allows the storage of dynamic data on a larger mesh scale. Moreover, from the experimental results (Sec. VI), it is easy to find that the rendering rate can be considerably improved by using temporal coherence of time-varying flows.

Table 2 Comparisons of the rendering rates between our approach and the Dynamic HRC

Unstructured-Grid Time-Varying Data	Data Scale			Mean Time (fps)	
	Mesh Scale		Time Steps	Our Approach	Dynamic GRC
	Cells	Vertices			
Forward step shocks	263K	54K	240	35.2	23.4
Pitching NACA 0012 airfoil	530K	101K	700	15.3	8.9
Supersonic aircraft	892K	207K	350	6.8	——

VI. Experiments

Our algorithm is implemented on Red Had Enterprise Linux 5 with an nVIDIA GeForce GTS 250 graphics card (1024MB) and a 2.67GHz Intel® Core™ i7 920 processor (2048MB RAM). To test the validity of our approach, we render the following data from CFD simulations by our algorithm and the dynamic HRC. Table 2 shows the comparisons of the performances between these two algorithms. The experimental results demonstrate that our approach gains a much higher rendering rate and allows rendering time-varying data on a larger mesh scale than the dynamic HRC.

A. Forward step shocks

The flow of forward step shocks is a classic unsteady flow in the wind tunnel experiments. The ultrasonic flow comes from left and form an arched shock before the step (see Fig. 6). The shock is reflected back from top to bottom for its great strength. After three times of reflection, the final state of the unsteady flow forms as shown in Fig. 6.

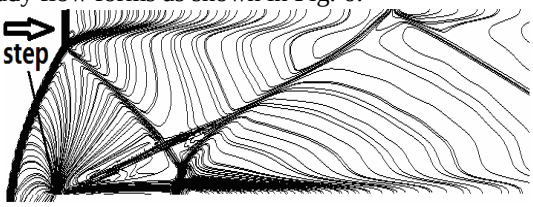


Fig. 6 The final state of the forward step shocks

The time-varying data from the simulation of the forward step shocks are rendered by our approach. The user is allowed

ISBN: 978-988-19251-2-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

to slow down or pause the dynamic rendering for further analysis of the fields. Fig.7 shows the rendering results (pressure fields) of some important steps when the flow pauses.

B. Pitching NACA 0012 airfoil

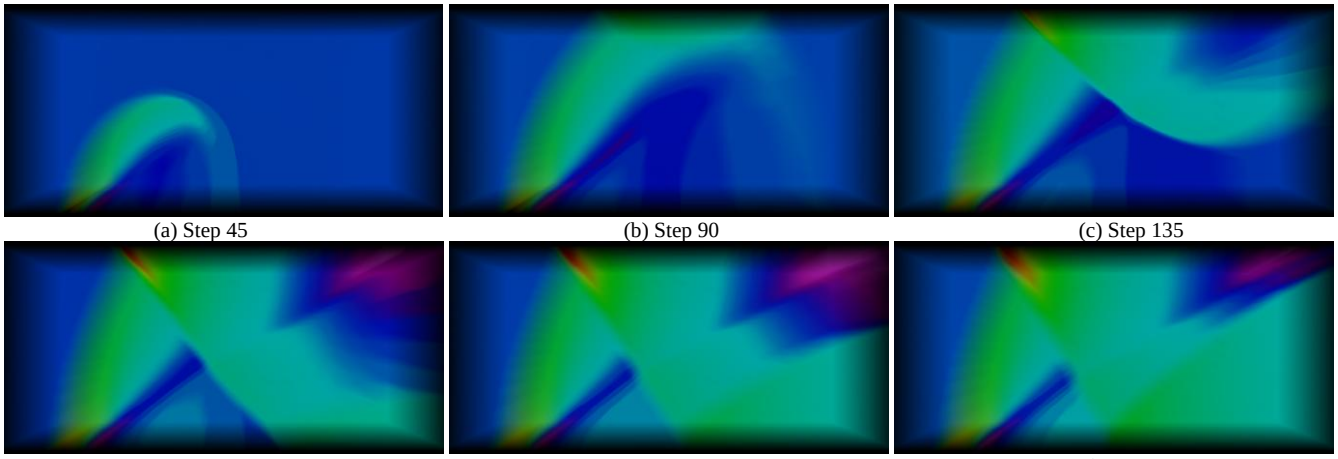
Fig. 8 displays the rendering results of the time-varying density fields from the simulation of the unsteady transonic flow past a pitching NACA 0012 airfoil. This is a benchmark case that includes hundreds of time steps, some of which are shown here.

C. Supersonic aircraft

Fig. 9 shows the rendering results (*u* velocity fields) of the flow fields around a supersonic aircraft. The flow rounds the aircraft and develops into complicated swirling vortices at the tail. The time-varying data on a large mesh scale of 892K cells and 207K vertices can not be rendered with the dynamic HRC due to memory limitations of storing the mesh on GPU.

VII. CONCLUSION

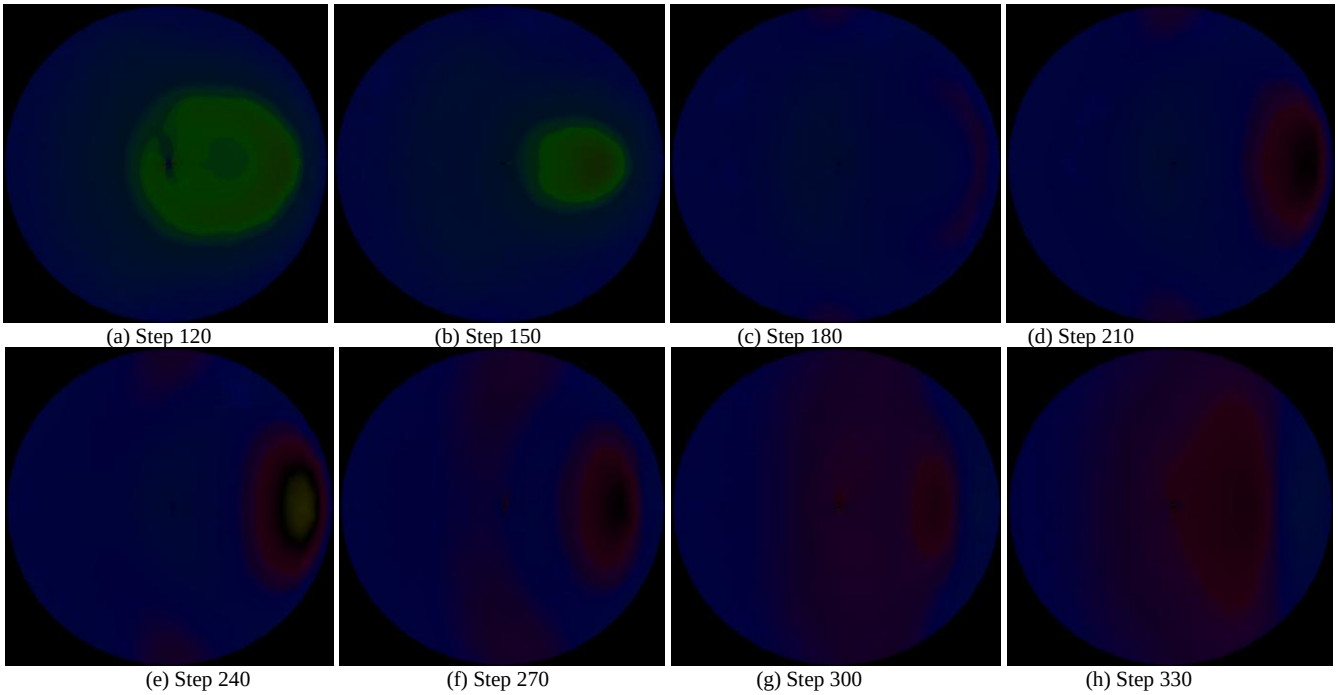
Volume rendering of dynamic unstructured-grid fields is a challenging problem in flow visualization. To maximize the rendering rate, temporal coherence of the time-varying data should be effectively utilized. However, research so far has primarily utilized temporal coherence to render time-varying data on structured grids. In this paper, we devise a scheme for using temporal coherence to achieve high-efficiency volume rendering of dynamic unstructured-grid data. We choose to perform rendering on the framework of the ray casting technique by reason of its high accuracy, which is especially important for flow visualization. Unfortunately, the mesh scale of the data that GPU-based ray casting algorithm can render is limited by the capability of texture memory. To make full use of GPU memory, a texture structure is designed to separate the vertex data from the cell data, which allows rendering time-varying data on a larger mesh scale. The experiments demonstrate that our approach achieves a much higher performance on both time and space, and allows rendering a larger mesh-scale time-varying data than the existing method.



(a) Step 45 (b) Step 90 (c) Step 135

(d) Step 170 (e) Step 210 (f) Step 240

Fig. 7 Rendering results of different time steps using our approach (forward step shocks)

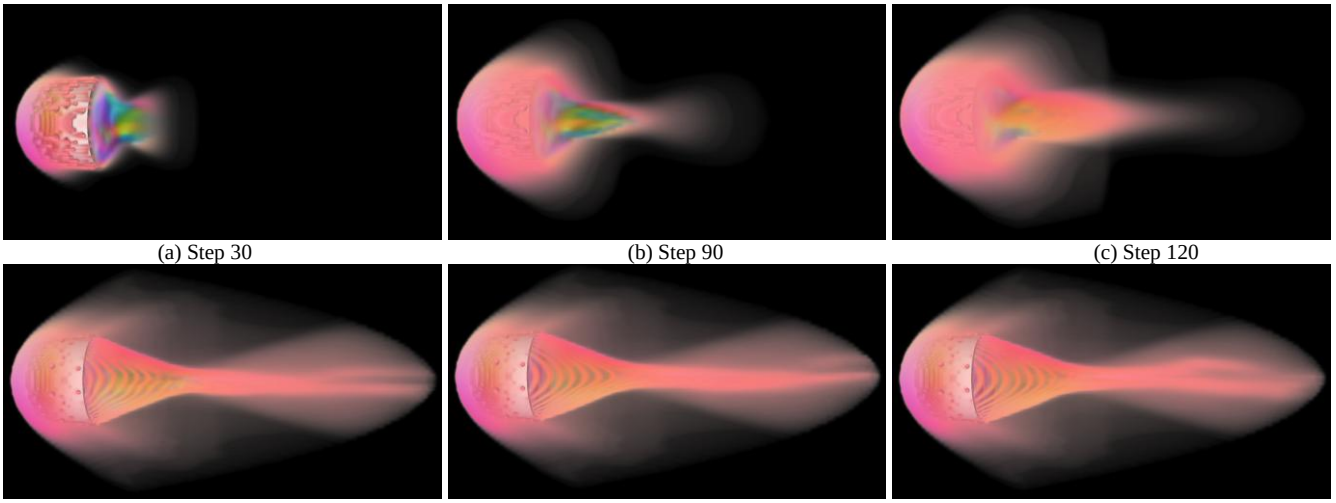


(a) Step 120 (b) Step 150 (c) Step 180 (d) Step 210

(e) Step 240 (f) Step 270 (g) Step 300 (h) Step 330

Fig. 8 Rendering results of different time steps using our approach (pitching NACA 0012 airfoil)

ISBN: 978-988-19251-2-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)



(a) Step 30 (b) Step 90 (c) Step 120

(d) Step 180 (e) Step 230 (f) Step 290

Fig. 9 Rendering results of different time steps using our approach (supersonic aircraft)

REFERENCES

[1] F.F. Bernardon, C.A. Pagot, J.L.D. Comba, C.T. Silva. Gpu-based tiled raycasting using depth peeling. Journal of Graphics Tools, 2006, 11(4): 1–16.

[2] S.P. Callahan, M. Ikits, J.L.D. Comba, C.T. Silva. Hardware-Assisted Visibility Sorting for Unstructured Volume Rendering. IEEE Transactions on Visualization and Computer Graphics, 2005, 11(3): 285–295.

[3] F.F. Bernardon, S.P. Callahan, J.L.D. Comba, C.T. Silva. Volume rendering of time-varying scalar fields on unstructured meshes. Technical Report UUSCI-2005-006, SCI Institute, 2005.

[4] J. Schneider, R. Westermann. Compression domain volume rendering. In Proceedings of IEEE Visualization 2003: 293-300.

[5] F.F. Bernardon, S.P. Callahan, C.T. Silva. An adaptive framework for visualizing unstructured grids with time-varying scalar fields. Parallel Computing 2007, 33(6):391-405.

[6] M. Weiler, M. Kraus, M. Merz, T. Ertl. Hardware-based ray casting for tetrahedral meshes. In Proceedings of IEEE Visualization 2003: 333–340.

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

[8] H.-W. Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In Proceedings of IEEE Visualization 1998: 159–166.

[9] H.-W. Shen, L.-J. Chiang, and K.-L. Ma. A fast volume rendering algorithm for time-varying field using a time-space partitioning (tsp) tree. In Proceedings of IEEE Visualization 1999: 371–377.

[10] D. Ellsworth, L.-J. Chiang, H.-W. Shen. Accelerating time-varying hardware volume rendering using tsp trees and color-based error metrics. In Proceedings of Volume Visualization Symposium 2000: 119–128.

[11] K.-L. Ma, H.-W. Shen, Compression and Accelerated Rendering of Time-Varying Volume Data. International Computer Symposium Workshop on Computer Graphics and Virtual Reality, 2000: 82–89.

[12] C. Lurig, R. Grosso, T. Ertl. Implicit Adaptive Volume Ray Casting. In Proceedings of the International Conference on Computer Graphics and Visualization 1997: 114–120.

[13] Y. Livnat, H.-W. Shen, C.R. Johnson. A near optimal isosurface extraction algorithm using the span space. IEEE Transactions on Visualization and Computer Graphics, 1996, 2(1): 73-84.

[14] H.-W. Shen, C.D. Hansen, Y. Livnat, C.R. Johnson. Isosurfacing in span space with utmost efficiency(ISSUE). In Proceedings of IEEE Visualization 1996: 287–294.

[15] Dimitri J. Mavriplis. Unstructured-mesh discretizations and solvers for computational aerodynamics. AIAA Journal, 2008, 46(6): 1281-1298.

[16] C.T. Silva, J.L.D. Comba, S.P. Callahan, F.F. Bernardon. A survey of GPU-based volume rendering of unstructured grids, Brazilian Journal of Theoretic and Applied Computing, 2005, 12(2): 9–29.

[17] Schneider, P. J., Eberly, D. H.: Geometric Tools for Computer Graphics. Morgan Kaufmann, 2003: 9-16.

[18] Joe F. Thompson, B. K. Soni, N. P. Weatherill. Handbook of grid generation, CRC Press, 1999: 693-701.

ISBN: 978-988-19251-2-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)