Fast insight into time varying datasets with dynamic mesh, CSCC 2014 conference, Advances in Information Science and Applications, Vol.I, pp. 104-109, series: Recent Advances in Computer Engineering Series, ISSN 1790-5109, ISBN 978-1-61804-236-1, Santorini, Greece, 2014

Fast insight into time varying datasets with dynamic mesh

Vaclav Skala, Slavomir Petrik

***Abstract*—**Advances in computational power over the last two decades allowed fluid dynamic simulations that involve moving parts. Various mesh update methods are employed in such simulations to adapt cells around the moving parts, resulting in a separate new definition of the mesh geometry and associated values for each discrete simulation step. Common practice is to visualize every timestep of the simulation as a single static dataset. We present a novel method for interactive visualization of the evolving isosurfaces from the datasets with dynamic mesh. The effectiveness of the method is demonstrated on the real-life datasets from combustion simulation.

***Keywords* —** Iso-contour; time-varying mesh; interactive visualization.

I. Introduction

STEEP increase in computational power over the last 20 years allowed fluid dynamic simulations that involve moving parts. Examples are combustion simulations with moving piston inside an engine cylinder [7, 16], see Fig.1, or falling payload from under an aircraft wing [13]. Various mesh update methods are involved in such simulations. The goal of the mesh update procedure is to re-calculate position and shape of the cells surrounding moving parts. Resulting datasets then have separate definition of the mesh geometry and values of the simulated quantities for each discrete simulation step.

The biggest challenge in visualization of such a data is their large size due to the changing geometry of the mesh at successive timesteps. The inter-timestep correspondence of the mesh Machine cells is usually lost. Another factor that contributes heavily to the overall size of such datasets is time-varying values associated with cells.

Isosurfaces are a standard tool for the investigation of fluid dynamic datasets. Most of the existing methods for efficient isosurface extraction from time-varying datasets assume static mesh geometry over the course of simulation with time-varying values. Only a few method exist that are able to handle datasets with dynamic mesh geometry.

The method presented in this paper allows for interactive playback of the evolving isosurfaces extracted from the datasets with dynamic mesh. We do not make any assumptions about the way the simulation mesh changes between adjacent timesteps. The major contribution of our work is a novel metrics for evaluation of temporal variation in a cell's shape, which is used to re-establish lost inter-timestep cells correspondence. Those cells with similar position, geometry and value are efficiently re-used for multiple timesteps instead of being re-loaded from disk. The method proposed is particularly suitable for the engineering software, in which quick insight into investigated data is required rather than high accuracy of the computed isosurfaces.

II. Related work

The idea of dynamic simulation mesh is not new. Arbitrary Lagrangian-Eulerian (ALE) methods were developed for this purpose. Donea et al. [14] provides a good survey of the field.

Most of the existing techniques for a efficient isosurface extraction from time-varying datasets assume static mesh. In static mesh the number of cells and their geometry do not change during the course of simulation. For such datasets a family of method has been developed based on the notion of Span-Space [5]. In Span-Space each cell of the dataset is

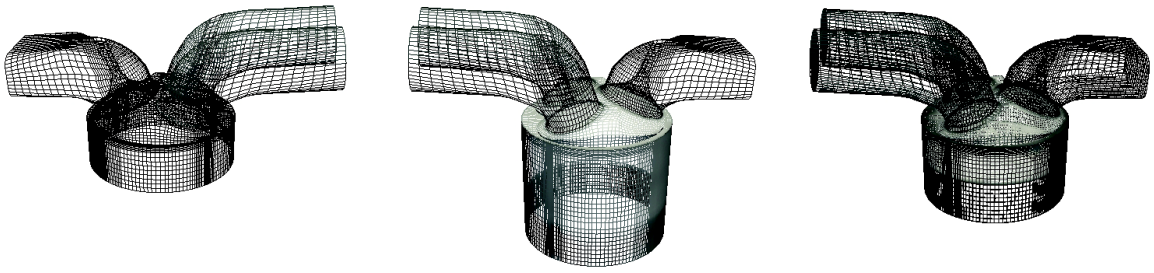

Fig.1: Dynamic mesh updated during the course of a combustion simulation. The mesh adapts total number of cells and their shape around moving parts - piston and fuel intakes.

This work was supported by MŠMT Czech Republic, projects No.LG13047 and LH12181. V,Skala and S.Petrik are with the University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering, Plzen, Czech Republic (http://www.VaclavSkala.eu)

Fast insight into time varying datasets with dynamic mesh, CSCC 2014 conference, Advances in Information Science and Applications,
Vol.I, pp. 104-109, series: Recent Advances in Computer Engineering Series, ISSN 1790-5109, ISBN 978-1-61804-236-1, Santorini, Greece, 2014

represented as a point in plane with coordinates x, y equal to the minimum and maximum value associated with the cell's vertices. Shen et al. use lattice subdivision of the Span-Space in their ISSUE algorithm [6]. Waters et al. [19] organizes the cells into the structure of fixed-sized buckets according to their min-max value.

Originally proposed for the isosurface extraction from static datasets, the idea of Space-Space has soon appeared in the methods for time-varying datasets. The Temporal Hierarchical Index Tree (THIT) [8] assigns the cells of a simulation mesh to its nodes according to a temporal variation of their minimum and maximum values. Weigle and Banks [9] introduced method which treats 3D time-varying dataset as the static 4D data. T-BON technique [11] extends BONO tree [3] for the time-varying datasets. A common BONO tree structure is saved only once for the entire dataset, while minimum and maximum values of the cells are stored separately for each timestep. Time-space Partitioning Tree (TSP) [10] is a standard full octree. Each node of TSP tree has a binary time tree associated. The partial rendered sub-volumes are cached are re-used for faster visualization. The approach of Gregorski [15] builds a hierarchy of diamonds from the original mesh cells. The mesh refinement process (sequence of split and merge operations) ruled by the min, max and error values of the active diamonds is initiated for each iso-surface query, starting from either current refinement or from a root diamond of hierarchy. Recently the Difference Intervals method [21] has been introduced, encoding change of a cell's status by either Add or Remove operation (a cell becomes active/inactive) similarly to the encoding of the video frames.

All of the methods described above assume static simulation mesh. They exploit the fact that the number of mesh cell and their geometry do not change during a simulation. In a dynamic simulation mesh both the number of cells and their geometry change under the deformation of the simulation domain boundaries. Therefore none of the methods above is suitable for the dynamic mesh datasets.

Only a few methods able to handle dynamic simulation mesh were introduced. A system for interactive visualization of the datasets with dynamic simulation mesh is introduced in [18], which assumes certain time intervals in a dataset (topology zones). Within a topology zone the number of cells and their correspondence between timesteps remains constant. Mesh cells are not matched or tracked over topology zone borders (rezone points). The method [22] tries to re-establish inter-timestep cell correspondence based on the geometric and positional similarity of the 2D cells. The method is based on the previous research on the shape reconstruction from planar cross-sections [4]. Later [23] introduced a method that pre-processes all cells into a list of diamonds. Each created diamond is composed of two cells sharing common face. Diamonds are stored in a data structure that allows fast identification of the diamonds intersected by the isosurface for user-defined isovalue.

The method presented in this paper pre-processes the data from each simulation timestep sequentially. First the active cells (cells intersected by the isosurface) are identified. For each active cell we try to find similar cell in the previous timestep and efficiently re-use its geometry information. Similarity of the compared cells is evaluated using our newly proposed metric (Sect. 3). Processed cells are stored in the tree-based structure, which accommodates geometry and scalar values of the cells for all timesteps (Sect. 4).

III. METRICS

The core of our proposed method is a metrics δ for evaluation of temporal changes in a cell's shape. Let's consider two N-dimensional cells C_1 and C_2. The metrics δ compares the shapes of C_1 and C_2 by evaluating of how much the vertices of C_1 have to move in order to *morph* C_1 to C_2:

$$\delta: C_1 \times C_2 \rightarrow \vec{D} \quad (1)$$

where: $\vec{D} \in R^N$, and $|D| \in \langle 0, \infty \rangle$.

For the rest of this article, let's define L to be the length of the longest edge of C_1. The resulting distances between the shapes of C_1 and C_2 have the following meanings:

$|D| = 0$, shape and position of C_1 is equal to C_2
$|D| < 1$, vertices of C_1 moved in total by less than $N * L$
$|D| = 1$, the vertices of C_1 moved by $N * L$ between C_1 and C_2
$|D| > 1$, vertices of C_1 moved in total by more than $N*L$

In other words, if $|D| = 0$ then positions of vertices of C_1 are exactly the same as the positions of vertices of C_2. As the value of $|D|$ grows the two compared cells become more and more different. Note that the metrics δ is not translation, rotation and scale invariant.

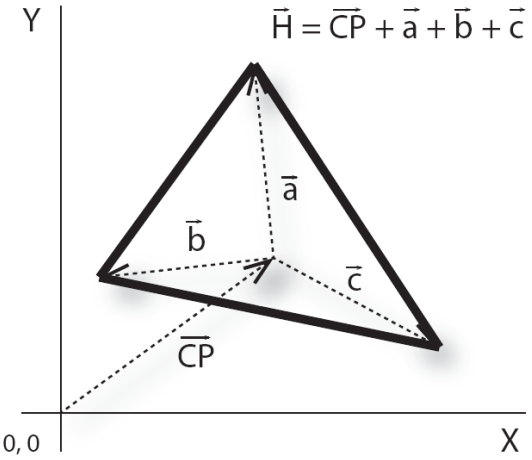

Fig.2: Computation of vector $\vec{H}$ for a 2D triangular cell C. $\vec{H}$ is equal to the vector sum of average center CP of cell C and all vectors from the center CP to the cell's vertices. Computation of $\vec{H}$ scales well to the other dimensions, e.g. tetrahedral cells in 3D. Calculated vector $\vec{H}$ is kept as a description of a cell's position and shape and is used later in the algorithm presented.

The metrics (or distance function) δ between the shape and position of C_1 and C_2 is defined as:

Fast insight into time varying datasets with dynamic mesh, CSCC 2014 conference, Advances in Information Science and Applications, Vol.I, pp. 104-109, series: Recent Advances in Computer Engineering Series, ISSN 1790-5109, ISBN 978-1-61804-236-1, Santorini, Greece, 2014

$$\delta(C_1, C_2) = \vec{H}_1 - \vec{H}_2 \quad (2)$$

$$\vec{H}_i = center(\overrightarrow{CP_i}) + \sum_{j=1}^{N} (\vec{V}_{ij} - \vec{C}P_i), i \in \{1,2\} \quad (3)$$

where: $\vec{H}_i$ represents the center point of the i-th cell computed as the average sum of the positions of all cell's vertices; N is dimensionality of the cell and $\vec{V}_{ij}$ represents position of j-th vertex of the i-th cell.

IV. ALGORITHM

Input parameters of our algorithm are start and finish timestep T_S, finish timestep T_F and selected isovalue q. The algorithm presented is thus able to identify active cells for one selected isovalue for any timestep between T_S and T_F of the dataset.

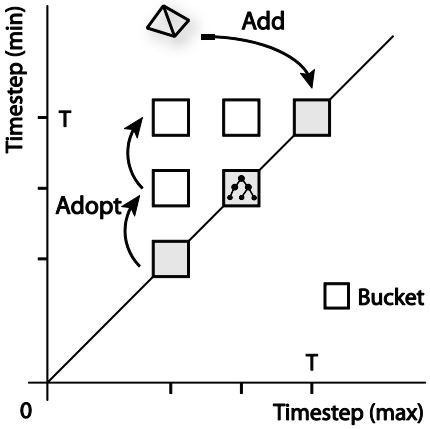

Fig.3: Buckets of the proposed data structure are organized in the 2D grid. Each bucket contains cells organized in a Red-Black tree. The operation Add for timestep T adds a cell into the bucket on position (T, T) of the grid of buckets. The operation Adopt removes the cell from its previous bucket and adds it to the bucket over the original one. In this way, similar cells are re-used between timesteps. For each cell of the dataset either *Add* or *Adopt* is performed.

Next we describe our data structure that keeps all active cells for entire time window between user-selected start and finish timestep. The data structure consists of 2-dimensional lattice of $M \times M$ buckets, where M is equal to the total number of simulation steps. Only half of the buckets above the diagonal in lattice is used. The other half of the buckets under the diagonal plays no role in our algorithm and does not need to be initialized. Along each axis, one bucket represents one timestep. X and Y axis of the lattice have meaning of minimum and maximum timestep similarly to the min-max isovalues in Span-Space [5].

Each bucket in our data structure accommodates certain amount of cells organized in a Red-Black tree. The Red-Black trees introduced by Guibas and Sedgewick in 1978 [1] are self-balancing binary trees, which guarantee worst-case running time $O(n \log n)$, for n accommodated items, for insert, delete and search operations. The data in Red-Black tree are kept in the non-leaf nodes (one data item per node), thus their space complexity is $O(n)$. Space complexity $O(n)$ makes the Red-Black trees suitable data structure for large number of cells that we need to accommodate due to fact that the geometry of simulation mesh is changing with each discrete timestep of a simulation.

Before we describe the algorithm itself, let us define two operations for a cell C: *Add* and *Adopt*, see Fig.3. Operation *Add*] for the cell C and the timestep T, adds C into the bucket (T, T), which lies on the diagonal of the lattice. On the other hand, operation *Adopt* for cell C and timestep T removes C from the bucket $(x, T-1)$ and adds C into the bucket (x, T), where $x \in \langle 0, T-1 \rangle$. By adding or removing a cell from a bucket we mean inserting/deleting the cell out of/into the tree inside the bucket.

The algorithm to populate our data structure is governed by the outer loop in which all timesteps from the user-defined time span T_S, T_F are prepossessed one-by-one, starting from earliest one. For the currently processed timestep, only active cells (e.g. cells intersected by the isosurface) are filtered out and further processed.

For each active cell we keep geometry (position of its vertices), values associated with its vertices and hash key H. The hash key H for a cell is computed using equation Eg.3 from the previous section [24]. H is used later in our algorithm for accommodation of a cell into the buckets of the data structure described above as well as for the comparison of the cell's shape with a candidate cell from successive timestep.

The algorithm starts at the row equal to the first timestep T of the lattice of buckets, see Fig.4. For each timestep all its active cells are processed one-by-one. For each active cell C the algorithm runs the inner loop that traverses buckets of the row $T-1$ in the columns 0 to $T-1$. If a candidate cell C_X is found the algorithm performs the *Adopt* operation for the cell C and the timestep T, otherwise the *Add* operation for the timestep T is performed.

The metrics δ described in the previous section is used for assessment of temporal variations of a cell's shape. C_X (from the timestep $T-1$) is pronounced to be a *predecessor* of C (from the timestep T) only if the distance $|D|$, Eq.1., between C and C_X is smaller than the user defined threshold Delta. In such case the operation *Adopt* is performed, removing C_X from its bucket B and inserting C as an approximate substitution of C_X into the bucket above B. If a candidate C_X for C is not

found, the operation *Add* for the cell C is performed. By increasing the treshold Delta for $|D|$ the user may store active cells more space-efficiently (i.e. more *Adopt* operations will be performed); however, the higher the Delta the higher approximation of the actual isosurface is extracted and rendered. In this way the user can balance between the algorithm's space demands and isosurface accuracy.

During the isosurface rendering step for a timestep T, the active cells are collected from all buckets in and above the grid row corresponding to T and left of and of the column T, see Fig.4. Once the active cells are collected, the isosurface

geometry can be computed and rendered out.

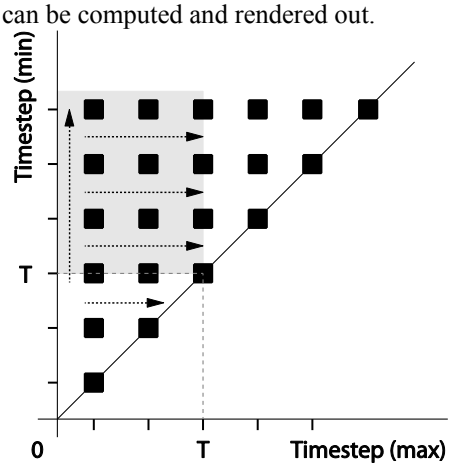

Fig.4: Active cells extraction.

The active cells for the timestep T are collected from the buckets that lie inside or on the borders of the greyed-out area. Dotted arrows show the way in which the buckets are traversed during active cells extraction. For each traversed bucket all its cells are collected by traversing its internal tree.

V. RESULTS

In order to test performance of our method, we ran the series of tests on two datasets produced during Computational Fluid Dynamic simulations. Both datasets are from simulations of the combustion process inside a valve of a diesel engine. Multiple scalar and vector variables like pressure or temperature were computed. The specificity of both datasets is that the geometry of the mesh is not fixed. The total number and shape of the cells changes with each discrete simulation step. The goal of the mesh update procedure is to re-build the mesh around moving parts inside valve. The moving parts are piston and fuel inlets on the top of cylinder.

The Move3D dataset consists of 149 timesteps, capturing a complete combustion cycle inside cylinder. The total number of cells (as well as their shape) varies between 40k and 115k. The dataset occupies 8.31 GB of space. The second dataset Valve consists of 960 timesteps and occupies 21.2 GB of storage. Total number of cells for the second dataset changes between 60k and 90k.

All tests were done on Intel Core2 Duo T5750 2GHz workstation with 4 GB of RAM. The method has been implemented in C language using Standard Template Library and compiled with GCC compiler.

A preprocessing time has been measured for both datasets. Fig.4. shows that the preprocessing time depends on the overall size of the dataset (total number of cells for given isovalue) rather than on the ratio of *Add* and *Adopt* operations performed. This is due to the fact that the time required for *Adopt* operation (and searching for predecessor cell) is only slightly different than the time required for *Add*.

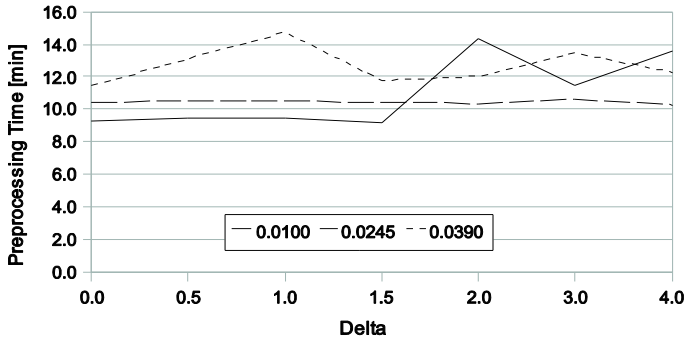

Fig.5: Preprocessing times for Move3D dataset. Relationship between preprocessing time and the Delta parameter of the proposed method. The preprocessing time depends on the total number of cells for given isovalue rather than on the number of Add and Adopt operations performed.

The saving of runtime memory depends on the value of the parameter Delta, Eq. 2. The higher the value of Delta the higher space savings are achieved and vice versa. Higher threshold for Delta means that higher number of cells will be re-used between the timesteps, i.e. operation *Adopt* will be performed instead of *Add*, which requires new memory allocation. Thus, the overall space saving depends on the similarity of the changing simulation mesh in the adjacent timesteps.

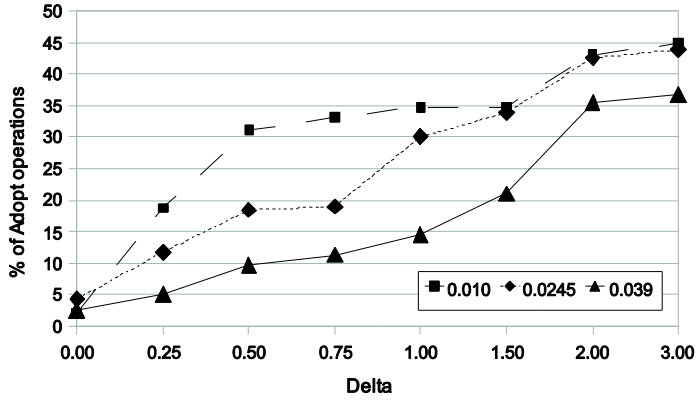

Fig.6: Move3D dataset. Percentage of the Adopt operations vs. parameter Delta of the proposed method for three different isovalues. For high values of Delta the method achieves around 50% of the Adopt operations, which is proportional to the space savings for the given value of Delta.

Fig.5 shows the relationship between the increasing value of the parameter Delta and the percentage of the *Adopt* operations performed over the cells in the Move3D dataset for three different isovalues. We have only chosen parameter Delta in the range 0.0 to 3.0; higher values of Delta show higher degree of the damage of the produced isosurface, Fig.8.

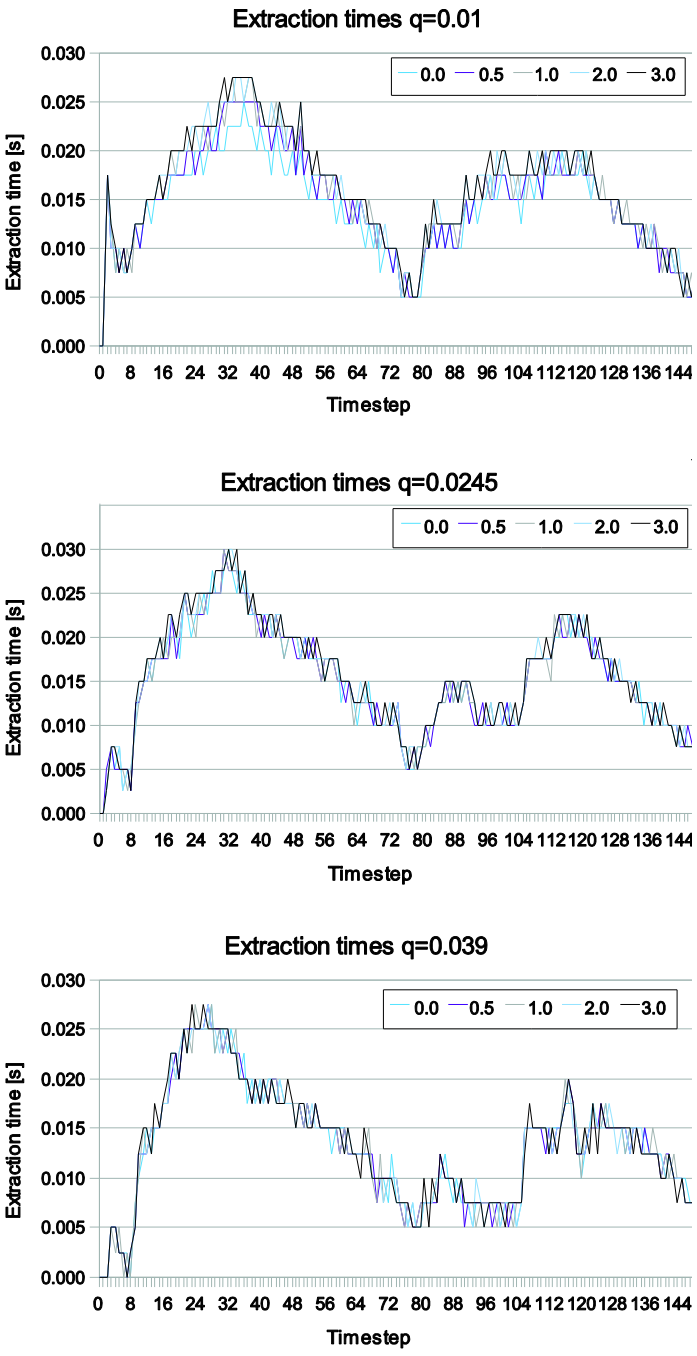

Fig.7: Extraction times for the Move3D dataset for three different isovalues q: 0.01, 0.0245 and 0.039 and different values of the parameter Delta between 0.0 and 3.0.

Finally, we measured the extraction times of the sets of active cells for three different isovalues and different values of parameter Delta of the Move3D dataset. Due to the common difficulties in measuring code execution time under 1 ms, for each pair isovalue-Delta a total of 400 extractions were made and the average time was computed. The overall extraction time of the active cells show no dependency on the parameter Delta used, but is proportional to the number of buckets visited during the extraction step (i.e. total number of active cells extracted). Fig.7 shows extraction times for the Move3D datasets for three different isovalues and parameter Delta between 0.0 and 3.0.

For visual assessment of the damage fragments of the produced isosurface with growing parameter Delta, Fig.8 shows isosufaces for the isovalue 0.0245 of the quantity AMU (total viscosity) of the Move3D dataset for three different values of Delta: 0.0, 1.0 and 3.0. For the tested datasets Delta > 3.0 produced considerable damage of the isosurface. However, the degree of the damages in higher Deltas is dataset-depended and thus left to be a choice of the user.

VI. Conclusion

An efficient method has been described, capable of interactive visualization of the evolving isosurfaces from the time-varying datasets with dynamic mesh. The method consists of the computationally inexpensive preprocessing step with logarithmic space and time complexity and the extraction step, during which the active cells are idenfied and collected. Two basic stones of the method are the metrics for assessment of the temporal change of a cell's shape (Sec. 3) and the buckets-based data structure (Sec.4) that facilitates space-efficient storage of the similar cells.

The method is particularly suitable for the applications where fast insight into the dataset is more important than high accuracy of the produced isosurfaces. The relative simplicity of the proposed method allows its easy implementation.

Acknowledgment

The authors thank to colleagues at the University of West Bohemia for fruitful discussions and to anonymous reviewers for their comments which helped to improve this manuscript.

References

[1] Guibas L. J., Sedgewick R.: *A dichromatic framework for balanced trees*. Proceedings of the 19th Annual Symposium on Foundations of Computer Science, 8-21.

[2] Lorensen W. E., Cline H. E.: Marching cubes: *A high resolution 3D surface construction algorithm*. Proceedings of ACM SIGGRAPH '87, 163-169.

[3] Wilhelms J., van Gelder A.: *Octrees for faster isosurface generation*. ACM Trans. Graph., 11(3), 201-227.

[4] Bajaj Ch. L., Coyle E. J., Lin K.-N.: *Arbitrary topology shape reconstruction from planar cross sections*. Graphical Models and Image Processing, 58(6), 524-543.

[5] Livnat Y., Shen H.-W., Johnson Ch. R.: *A Near Optimal Isosurface Extraction Algorithm Using the Span Space*. IEEE Transactions on Visualization and Computer Graphics, 2(1), 73-84.

[6] Shen H.-W., Hansen Ch. D., Livnat Y., Johnson Ch. R.: *Isosurfacing in Span Space with Utmost Efficiency*, IEEE Visualization '96, 287-294.

[7] Amsden A.A.: KIVA-3V: *A block-structured KIVA program for engines with vertical or canted valves*. Los Alamos NATIONAL LABORATORY, Technical Report LA-13313-MS.

[8] Shen H.-W.: *Iso-surface extraction in time-varying fields using a temporal hierarchical index tree*. Proceedings of Visualization '98, 159-166.

[9] Weigle Ch., Banks D. C.: *Extracting iso-valued features in 4-dimensional scalar fields*. Proceedings of IEEE Symposium on Volume Visualization '98, 103-110.

[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.* Proceedings of Visualization '99, 371-377.

[11] Sutton P., Hansen Ch. D.: *Isosurface extraction in time-varying fields using a temporal branch-on-need tree*. Proceedings of Visualization '99, 147-153.

[12] de Leeuw W., van Liere R.: *Chromatin decondensation: a case study of tracking features in confocal data*. Proceedings of Visualization '01, 441-444.

[13] Fluent news: *Dynamic Mesh*, Volume: XI, editor: Liz Marshall, Fluent Inc.

[14] Donea J., Huerta A., Ponthot J.-Ph., Rodriguez-Ferran A.: *Encyclopedia of Computational Mechanics*, Volume 1. John Wiley \& Sons.

[15] Gregorski B.: *Adaptive Extraction of Time-Varying Isosurfaces*. IEEE Transactions on Visualization and Computer Graphics 10(6), 683-694.

[16] Cavallo P., Hosangadi A., Ahuja V.: *Transient simulations of valve motion in cryogenic systems*. Proceeding of 35th AIAA Fluid Dynamics Conference and Exhibit.

[17] Szymczak A.: *Subdomain-aware contour trees and contour tree evolution in time-dependent scalar fields*. Proceedings of Shape Modeling International '05, 136-144.

[18] Doleisch H., Mayer M., Gasser M., Priesching P., Hauser H.: *Interactive Feature Specification for Simulation Data on Time-Varying Grids*. SimVis'05, 291-304.

[19] Waters K. W., Co Ch. S., Joy K. I.: *Isosurface Extraction Using Fixed-Sized Buckets*. IEEE VGTC Symposium on Visualization, 207-214.

[20] Bernardon F., Callahan S., Comba J., Silva C.: *Interactive volume rendering of unstructured grids with time-varying scalar fields*. Proceedings of Eurographics Symposium on Parallel Graphics and Visualization '06, 51-58.

[21] Waters K. W., Co Ch. S.: *Using Difference Intervals for Time-Varying Isosurface Visualization*. IEEE Transactions on Visualization and Computer Graphics, 12(5), 1275-1282.

[22] Petrik S., Skala V.: *Iso-contouring in Time-varying Meshes*. SCCG 2007 Proceedings, 216-223.

[23] Petrik S., Skala V.: *Z-Diamonds: A Fast Iso-surface Extraction Algorithm for Dynamic Meshes*. IADIS Computer Graphics and Visualization proceedings 2007.

[24] Hradek,J., Skala,V.: *Hash Function and Triangular Mesh Reconstruction*, Vol.29, No.6., pp.741-751, Computers&Geosciences, Pergamon Press, ISSN 0098-3004, 2003

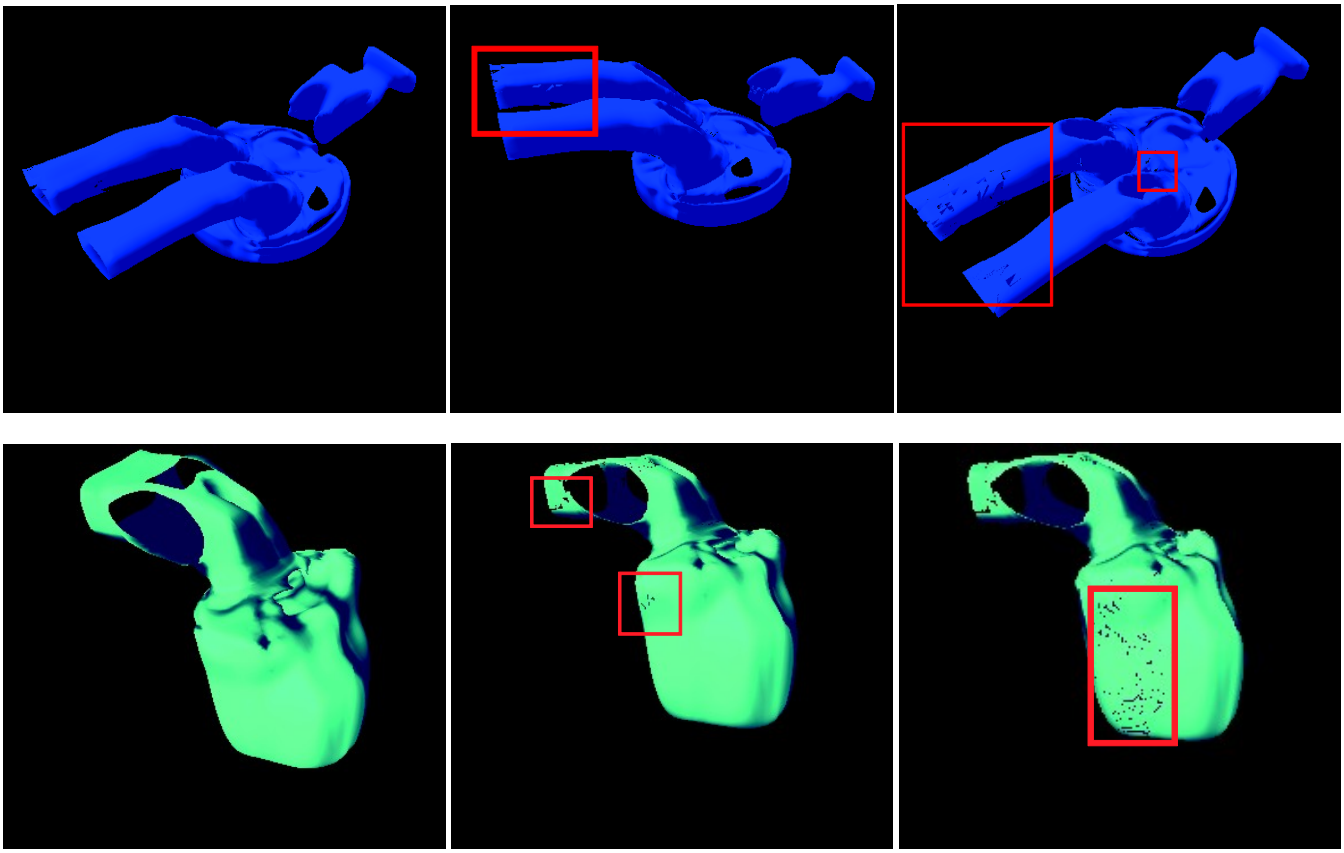

Fig.8: Isosurfaces of the Move3D dataset. Top row: timestep 13, bottom row: timestep 113. Isovalue q=0.0245 of the AMU (total viscosity).
Each isosurface has been generated with different value of Delta; from left to right: Delta = 0.0, 1.0, 3.0.