Z-DIAMONDS: A FAST ISO-SURFACE EXTRACTION ALGORITHM FOR DYNAMIC MESHES

Slavomír Petrík
University of West Bohemia
Univerzitní 8, Pilsen, Czech Republic

Václav Skala
University of West Bohemia
Univerzitní 8, Pilsen, Czech Republic
skala@kiv.zcu.cz

ABSTRACT

Dynamic simulation meshes are successfully used in many scientific and engineering fields. We present simple and efficient algorithm for a fast extraction of the iso-surfaces from the data sets with dynamic mesh. Simulation mesh at each time step is preprocessed into a list of diamonds composed of the original mesh cells. The *min* / *max* values of all diamonds are stored in a Time-space Partitioning Tree (TSP), which is used to quickly locate the diamonds intersected by a queried iso-surface. To overcome the large memory requirements due to the dynamic changes of a simulation mesh, an out-of-core approach is used to dynamically load geometry of the active diamonds during visualization. We demonstrate our approach on the data sets from the Computational Fluid Dynamics (CFD) simulations.

KEYWORDS

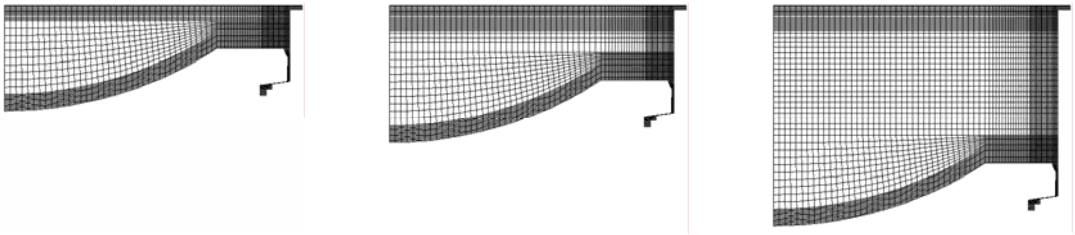
Iso-surface, dynamic, interactive visualization

1. INTRODUCTION

Researchers in many scientific and engineering fields use iso-surfaces to investigate data sets. Most of the current techniques for iso-surface construction assume static simulation mesh (e.g. [Shen H.-W., 1998, Weigle Ch. and Banks D.C., 1998, Waters K.W. and Co Ch.S., 2006]). Simulation domain can change from time step to time step because of deforming boundaries, moving parts or flexible bodies. In order to maintain the mesh quality criteria, the dynamic re-meshing techniques are often employed. Examples of dynamic meshing come from various industrial applications [Amsden A.A., 1997, Marshall L., 2002, Doleisch et al., 2005, Cavallo P. et al., 2005]. Figure 1 shows the example of the dynamic simulation mesh from a simulation of combustion process in an engine.

While the methods for generating dynamic meshes are being rapidly developed, there is a lack of suitable visualization techniques. We propose simple and efficient algorithm for extraction of the iso-surfaces from the data sets with dynamic mesh. Proposed Z-Diamonds method works over triangular or tetrahedral meshes. We do not make any assumption about the way a simulation mesh changes between adjacent time steps. We also do not attempt to reconstruct the evolving iso-surfaces in between time steps defined in a data set. To overcome the problem of dynamic simulation mesh, we first preprocess a simulation mesh at each time step into a list of *diamonds*. The *min* / *max* values of all diamonds are stored in a common TSP tree [Shen H.-W., 1999], which is used to quickly locate the diamonds intersected by a queried iso-surface (active diamonds). Geometry of the active diamonds is then read from disk in an out-of-core fashion [Wang et al., 2007].

In the following sections, we first provide the list of the related techniques for visualization of time-varying data (section 2). Then we present our novel approach to this visualization problem (section 3) and demonstrate its performance on the real CFD data sets (section 4). Finally we draw conclusions and suggest possible areas for future work.



Time step: 400 Time step: 420 Time step: 440

Figure 1. The example of the dynamic mesh from a simulation of combustion process in an engine. The simulation mesh changes its layout according to the actual vertical position of a moving piston

2. PREVIOUS WORK

The idea of dynamic simulation meshes is not a new one. Arbitrary Lagrangian-Eulerian (ALE) methods were developed just for this purpose. While an overview of the ALE methods is outside the scope of this paper, [Donea J. et al, 2004] provides a good survey of the field.

Yet current techniques for a fast iso-surface extraction assume static simulation mesh (i.e. the number of mesh cells and their geometry do not change during a simulation). Even if the ability of these methods to extract an iso-surface from the data sets with dynamic mesh is very limited or none at all, we provide their brief overview, because they are useful at the certain step of our method.

Wilhelms and van Gelder introduced Branch-on-Need-Octree [Wilhelms J. and van Gelder A., 1992]; the space efficient version of the standard octree. Livnat et al. [Livnat Y. et al, 1994] introduced notion of *span space*, in which cells are represented as the points with x (*minimum*) and y (*maximum*) coordinates. Shen et al. use lattice subdivision of the span space in their ISSUE algorithm [Shen H.-W. et al, 1996].

This initial research on fast iso-surface extraction from the static data sets has been also extended onto the data sets which cover certain period of time. Weigle and Banks [Weigle Ch. and Banks D.C., 1998] introduced method which treats 3D time-varying data set as the static 4D data. Temporal Hierarchical Index Tree (THIT) [Shen H.-W., 1998] assigns the cells of a simulation mesh to its nodes according to a temporal variation of their values. T-BON technique [Sutton P. and Hansen Ch., 1999] extends BONO tree [Wilhelms J. and van Gelder A., 1992] for the time-varying data sets. A common BONO tree structure is saved only once for the entire data set, while minimum and maximum iso-values for its nodes are stored separately for each time step. Time-space Partitioning Tree (TSP) [Shen H.-W., 1999] is a standard full octree. Each node of TSP tree has a binary time tree associated. The partial rendered sub-volumes are cached for selected TSP nodes and used to speed up time-varying data visualization. The approach of Gregorski [Gregorski B., 2004] 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 a root diamond of a hierarchy. Recently the Difference Intervals [Waters K.W. and Co Ch.S., 2006] technique has been introduced, encoding change of a cell's status between adjacent time steps by either *add* or *remove* (cell become active / inactive) operation which is used for fast extraction of the active cells during interactive iso-surface visualization.

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. This fact can not be in general exploited for dynamic simulation meshes, since the number of mesh cells and their geometry may change under the deformation of the simulation domain boundaries.

Doleisch et al. [Doleisch et al., 2005] introduced system for interactive visualization of the data sets with dynamic simulation mesh. They assume certain time intervals in a data set (*topology zones*), within which the number of cells and their correspondence between time steps remains constant. Mesh cells are not matched or tracked over topology zone borders (*rezone points*). In our approach, each time step represents a different topology zone. Moreover, we do not make any assumption about the way a simulation mesh changes (or remains static) between adjacent time steps.

3. METHOD

The key contribution of our method is its ability to handle dynamic simulation mesh. We make no assumption about the way a simulation mesh changes between the adjacent time steps. Presented Z-Diamonds method works with triangular or tetrahedral meshes. Since a simulation mesh dynamically changes, we do not try to match the original mesh cells between adjacent time steps. Instead we preprocess the mesh at each time step into a list of *diamonds*. Each diamond is composed of two neighboring simplicial cells, sharing a common face (i.e. two triangles in 2D or two tetrahedra in 3D, Figure 2). Diamonds are not matched or tracked between adjacent time steps. In this way the problem of changing simulation mesh is overcome. In the rest of this article we will assume tetrahedral mesh as an input of our method.

Pairing of the tetrahedral mesh cells into the diamonds has several advantages over the simple tetrahedral representation of a mesh. Firstly, a single diamond is represented by five vertices and five scalar values, in comparison to the eight vertices with associated scalar values necessary to represent two separate tetrahedra. Since the active diamonds are dynamically loaded for each iso-surface query, such "five-vertex" representation of the diamonds reduces the I/O traffic during interactive visualization. Secondly, it allows use of the Marching Diamonds algorithm [Anderson et al, 2005] to produce smoother resulting iso-surfaces.

(a) (b)

Figure 2. Each diamond is composed of two neighboring simplicial mesh cells, sharing a common face. Reference diamonds are depicted for 2D (a) and 3D (b) simulation mesh

Once we have a list of diamonds for each time step, we need a data structure to support fast extraction of the active diamonds. Because of the large amount of data needed for representation of dynamic simulation mesh, we keep only the *min* / *max* values of the diamonds in the main memory. The *min* / *max* values of all diamonds from all time steps are organized in a common TSP tree [Shen H.-W., 1999]. Geometry of the active diamonds is dynamically loaded from a disk in an out-of-core fashion. Since we extract a queried iso-surface only at the discrete time steps, specified in a data set, we do not deal with the topological changes of the iso-surfaces in between adjacent time steps. We assume that the data sets processed by our method are sampled sufficiently along the time dimension.

During the preprocessing phase of the Z-Diamonds method a list of diamonds is created for each single time step. In the visualization phase a common TSP tree is built and used for interactive extraction and visualization of the queried iso-surfaces. The following sections detail about the diamonds-building phase, the TSP tree and its usage for fast extraction and visualization of the iso-surfaces from the data sets with dynamic meshes.

3.1 Diamonds Building Process

In the preprocessing phase a simulation mesh at each discrete time step is treated separately. The process of building diamonds is initiated, starting form an arbitrary initial tetrahedron T_{init}. One of the tetrahedra sharing a common face with T_{init} is chosen, to form a new 3D diamond D_0 with T_{init} (Figure 3). We choose that neighbor of T_{init}, whose additional vertex has scalar value closest to those of T_{init}, so as to ensure that the resulting diamond D_0 will cover as small a range of iso-values as possible. This increases the probability that a queried iso-surface will intersect both tetrahedra in a diamond. The diamond-building process continues recursively for each of six tetrahedra sharing faces with D_0. During the diamond building process, each tetrahedron is assigned to at most one diamond. If all of the neighboring tetrahedra have already been assigned to some diamond, a new diamond composed of only one tetrahedron and one empty vertex is created. A list of diamonds for each time step is sorted according to their *minimum* values and stored in a separate file as a result of the preprocessing phase.

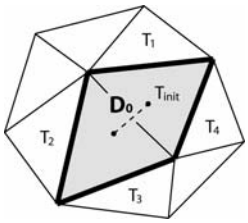


Figure 3. 2D example of the recursive diamond-building process. Two neighboring triangle cells are paired into the 2D diamond D_0. The process continues recursively for each of four triangles sharing edges with D_0

The algorithm of pairing tetrahedra into the diamonds is related to the *perfect matching* problem [Karpinski M. and Rytter W., 1998]. In our case, the goal of *perfect matching* would be to build as few as possible diamonds composed of only one tetrahedron and one empty vertex. As will be shown in the section 4, a relatively small amount of such one-tetrahedron diamonds does not significantly influences the overall performance of the method. Therefore, we do not employ advanced algorithms for pairing tetrahedra into the diamonds, because they could potentially raise the time and space complexity of the preprocessing phase with little or no benefit.

3.2 TSP Tree

The concept of TSP tree was introduced in [Shen H.-W., 1999]. The TSP tree is designed for fast extraction of the active cells for queried *iso-value* and *time-step*. TSP tree is a standard full octree, which recursively subdivides a data volume until a predefined minimum size of sub-volume is reached. To store temporal information into the TSP tree, each TSP node N_i has assigned a binary time tree B_i. B_i recursively bisects the entire time span of the data set [0,t], until a unit time step is reached. So, each leaf node of B_i represents a single time step. Figure 4 depicts the TSP tree and one of its nodes in the form of binary time tree. Each node of B_i represents a time span $[t_a,t_b]$ and keeps the *min* / *max* iso-values of the cells within the subspace of the TSP node N_i during $[t_a,t_b]$.

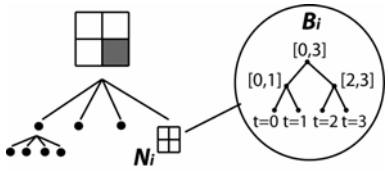


Figure 4. TSP tree is a standard full octree. Each node of TSP tree has assigned a binary time tree. In this case, the data set consists of four time steps (picture from [Shen H.-W., 1999])

A list of diamonds from each single time step is first sorted according to the diamonds' minimum values and then organized row-by-row in a 2D diamond table. So, each time step has its own 2D diamond table. Diamond tables of all time step have a common size equal to $\left\lceil \sqrt{|L_x|} \right\rceil$ in each direction, where $|L_x|$ is the size of the longest diamond list L_x.

We build one TSP tree which holds *min*/*max* values of all diamonds from all time steps. Each leaf node of a TSP tree corresponds to a position (x, y) in the 2D diamond tables. Binary time tree assigned to a leaf node of TSP tree holds the *min*/*max* values of the diamonds from all time steps placed at position (x, y) in the 2D diamond tables. Binary tree assigned to an internal node of TSP tree holds the *min*/*max* values of the diamonds within a subspace of 2D diamond tables. So, we build a quad-tree-based TSP tree with binary time tree associated with each TSP node.

Sorting a list of diamonds before it is organized in a 2D diamond table for particular time step increases probability that the diamonds placed at corresponding positions in the diamond tables of two adjacent time steps will have the same or at least very similar *min* / *max* values. This effect is exploited to achieve better memory footprint of a common TSP tree. To be more specific, we stop recursive subdivision of each binary time tree B_i (Figure 4) once the child nodes n_1 and n_2 of the current node n have the same *min* / *max* values or when the difference between their *min* and *max* values is lower than the user specified constant α. In such

case we only update the time span and *min* / *max* values of the node *n*. Constant α provides a tool to manage the tradeoff between the memory footprint of a TSP tree (for large data sets) and fast and accurate extraction of a set of active diamonds (for smaller data sets). The lower the constant α, the more precise and smaller set of active diamonds is extracted by traversing a common TSP tree, but the memory requirements of TSP tree grow. On the other hand, for the higher values of the constant α, a better memory footprint is achieved (the binary time trees will contain less nodes), but the percentage of the extracted active diamonds which do not cover the specified *iso-value* will grow.

3.3 Iso-surface Extraction

Prior to the visualization itself, a common TSP tree is built (section 3.2), holding the *min* / *max* values of the all diamonds from all time steps. We do not store geometry of the diamonds in a TSP tree. Only the extreme *min* / *max* values of the diamonds are stored in the binary time trees at the nodes of TSP tree.

For each query (*iso-value*, *time step*) a common TSP tree is traversed. At each TSP node N_i, an associated binary time tree B_i is traversed. If a leaf node of B_i is reached and its iso-value range covers queried *iso-value*, then we continue recursively to the child nodes of N_i. Otherwise, we skip traversing the whole sub-tree of the TSP node N_i. If N_i is a leaf node of TSP tree, then a corresponding diamond D is read from the file which keeps the diamond list for queried *time step*. Since the records for all diamonds have the same size, the position of the record of diamond D can be easily calculated.

Once all the active diamonds are read from disk, a resulting iso-surface is rendered using techniques like *Marching Tetrahedra* [Treece G. M. et al, 1998] or *Marching Diamonds* [Anderson et al, 2005]. Figure 5 shows the example of a part of an iso-surface interpolated out of the active diamonds extracted by the Z-Diamonds method.

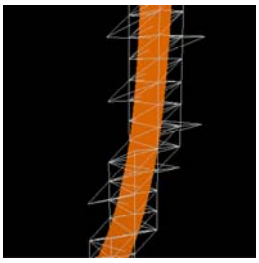


Figure 5. A part of rendered iso-surface with highlighted active diamonds

4. RESULTS

Presented Z-Diamonds method was implemented in C#. Tests were done on Intel Pentium 3.2GHz workstation with 2GB of RAM and ATI FireGL V5200 graphics adapter. Table 1 provides information about the data sets used for tests. *Motor* data set is a simulation of combustion process in an engine. Simulation mesh changes its layout according to the vertical position of a piston inside a valve. *Wind tunnel* data set is from a simulation of an air flow around solid body, moving inside a wind tunnel. *Airfoil* data set is from a 2D simulation of a low-speed air wave, flowing around the leading edge of a flexible airfoil. Table 1 also provides the preprocessing times and size of the preprocessed data. For each data set we preprocessed the simulation meshes at all time steps and one scalar quantity.

The most expensive part of the preprocessing step is a sorting of the diamonds according to their minimum values (section 3.1). Sorting step has usually logarithmic time complexity, depending on the sorting algorithm used, thus the overall time complexity of the preprocessing of one time step t is $O(n * \log(n))$, where n is the number of mesh cells at the time step t.

Table 1. The data sets used for tests of Z-Diamonds method

Data set	# of time steps	Cells per time step	Size of data set	Preprocessing time	Size of the preprocessed data
Motor (3D)	148	40,000 to 115, 000	3.2 GB	86 mins 32s	1.2 GB
Wind tunnel (3D)	700	400 k to 430 k	7.5 GB	120 mins 8s	3.4 GB
Airfoil (2D)	500	30,000 to 32,000	781 MB	9 mins 22s	181 MB

As stated in the section 3.1, the process of pairing original mesh cells into the diamonds may produce the diamonds composed of only one mesh cell and one empty vertex. Figure 6 shows the number of created diamonds composed of just one mesh cell for the *Motor* and *Wind tunnel* data sets. The percentage of created one-cell-diamonds is lower than 14% of all created diamonds. We have accepted this price in exchange for low time complexity of the preprocessing phase.

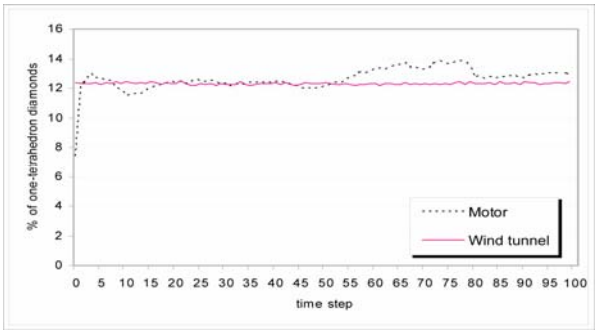


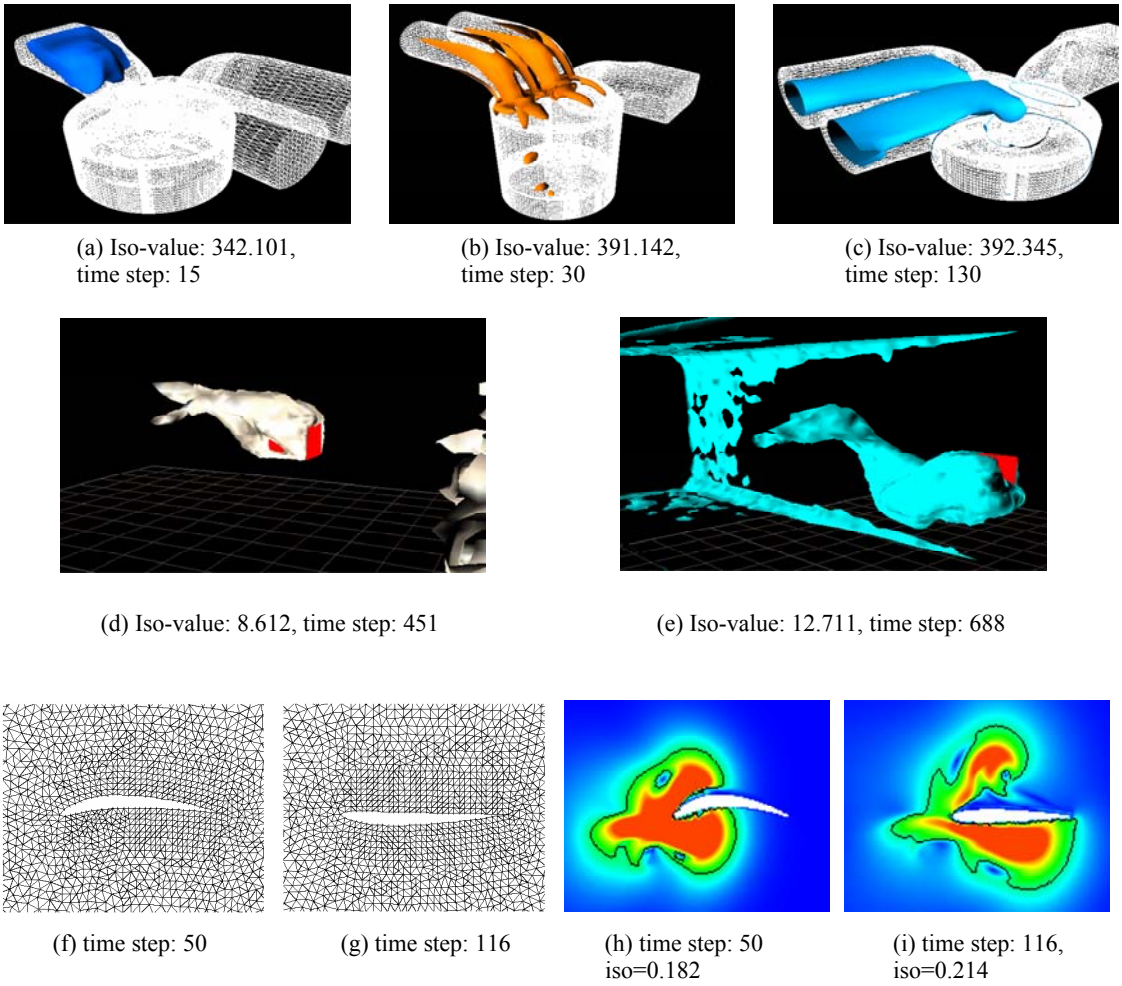
Figure 6. Percentage of the diamonds composed of only one tetrahedron and one empty vertex for the *Motor* and *Wind tunnel* data sets

Table 2 draws the iso-surfaces extraction times during interactive visualization. Query execution times stated in the table 2 include time for active cells extraction by traversing a TSP tree, loading of active diamonds geometry from disk and extraction of iso-surface geometry from loaded active diamonds. Extracted iso-surfaces for iso-values and time steps from the table 2 are depicted by the figure 7. All measurements were done with constant $\alpha = 0$ (section 3.2).

Table 2. The extraction times, numbers of active diamonds and numbers of triangles on the resulting iso-surfaces for selected iso-values and time steps for *Motor* and *Wind tunnel* data sets

Data set / iso-value / time step	Extraction time	# of active diamonds	# of triangles on the iso-surfaces
Motor / 342.101 / 15	386 ms	3,262	4,354
Motor / 391.124 / 30	612 ms	15,900	36,768
Motor / 392.345 / 130	514 ms	6,741	15,398
Wind tunnel / 8.612 / 451	289 ms	3,133	4,932
Wind tunnel / 12.711 / 688	602 ms	16,324	38,202

We do not provide any comparison with the previous approaches. This is because we have not found any published technique for iso-surfaces extraction from the data sets with dynamic simulation mesh. The method of Doleisch et al. [Doleisch et al., 2005] assumes continuous time spans in a data set, within which the number of mesh cells and their correspondence between adjacent time step remains constant (*topology zone*). In our data sets the simulation mesh changes with each discrete time step. So, each time step represents one topology zone. Other approaches for a fast extraction of iso-surfaces from time-varying data sets (section 2) are unable to handle dynamic simulation mesh.



(a) Iso-value: 342.101, time step: 15

(b) Iso-value: 391.142, time step: 30

(c) Iso-value: 392.345, time step: 130

(d) Iso-value: 8.612, time step: 451

(e) Iso-value: 12.711, time step: 688

(f) time step: 50

(g) time step: 116

(h) time step: 50 iso=0.182

(i) time step: 116, iso=0.214

Figure 7. Iso-surfaces extracted and rendered from: (a) - (c) *Motor* data set - temperature, (d), (e) *Wind tunnel* data set – total pressure, and (f) - (i) 2D *Airfoil* data set – speed magnitude

5. CONCLUSIONS

Visualization of the evolving iso-surfaces is the powerful tool for understanding of the dynamic data behavior. We presented the Z-Diamonds method for fast extraction of the iso-surfaces from the data sets with dynamic simulation mesh. Proposed method allows interactive extraction and visualization of the iso-surfaces for any desired iso-value and discrete time step defined in a data set. Extraction of the iso-surface is done at interactive frame-rates and was demonstrated on the data sets from CFD simulation.

The main contributions of the presented method are:

- The method supports the data sets with dynamic simulation meshes. This is an advantage over the existing visualization techniques, which are based on the assumption, that the number, geometry and correspondence of the mesh cells between adjacent time steps remain constant.
- Since the preprocessing of each time step is done independently of the others, the preprocessing phase can be easily parallelized.
- Time and space complexities of the preprocessing and visualization phase are low. This is especially important for the large scientific data sets.

There are numerous possible ways for the future work. With the desktop workstation configuration mentioned in the section 4, we are able to interactively extract iso-surfaces from up to 50 consecutive time steps. Therefore, modification of the method for better memory footprint is expected.

ACKNOWLEDGEMENT

This work has been supported by the project 3DTV NoE FP6 No: 511568 and Ministry of Education, Youth and Sports of the Czech Republic project VIRTUAL No: 2C06002. We would like to thank Randy Hessel from Engine Research Center, University of Wisconsin-Madison, USA for *Motor* data set and New Technologies Research Center for providing the *Wind tunnel* data set.

REFERENCES

Anderson et al, 2005. Marching diamonds for unstructured meshes. *Proceedings of IEEE Visualization '05*. pp. 423–429.

Amsden A.A., 1997. KIVA-3V: A block-structured KIVA program for engines with vertical or canted valves. *Technical Report LA-13313-MS,* Los Alamos NATIONAL LABORATORY, Los Alamos, New Mexico 87545, USA.

Cavallo P. et al, 2005. Transient simulations of valve motion in cryogenic systems. *Proceedings of the 35th AIAA Fluid Dynamics Conference and Exhibit*, Toronto, Canada.

Donea J. et al, 2004. *Arbitrary Lagrangian-Eulerian methods*. Encyclopedia of Computational Mechanics. John Wiley and Sons.

Doleisch et al., 2005. Interactive feature specification for simulation data on time-varying grids. *Proceedings of SimVis'05.* pp. 291–304.

Gregorski B., 2004. Adaptive extraction of time-varying isosurfaces. *In IEEE Transactions on Visualization and Computer Graphics*, Vol. 10, No. 6, pp. 683–694.

Karpinski M. and Rytter W., 1998. *Fast Parallel Algorithms for Graph Matching Problems*. Clarendon Press, Oxford.

Livnat Y. et al, 1994. A near optimal isosurface extraction algorithm using the span space. *In IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, No. 1, pp. 73–84.

Marshall L., 2002. *Fluent news: Dynamic Mesh*, Vol. 9, Fluent Inc.

Shen H.-W., 1999. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. *Proceedings of the conference on Visualization '99*. Los Alamitos, USA, pp. 371–377.

Sutton P. and Hansen Ch., 1999. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (T-BON). *Proceedings of the conference on Visualization '99*. Los Alamitos, CA, USA, pp. 147–153.

Shen H.-W., 1998. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. *Proceedings of the conference on Visualization '98.* Los Alamitos, USA, pp. 159–166.

Shen H.-W. et al, 1998. Isosurfacing in span space with utmost efficiency (ISSUE). *Proceedings of the conference on Visualization '96.* Los Alamitos, USA, pp. 287-294.

Shi Q. and JaJa J., 2006. Isosurface extraction and spatial filtering using persistent octree. *In IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 5, pp. 1283-1290.

Treece G. M. and Prager R. W. and Gee A. H., 1999. Regularized marching tetrahedra: improved iso-surface extraction. *In Computers and Graphics*, Vol. 23, No. 4, pp. 583-598.

Weigle Ch. and Banks D.C., 1998. Extracting iso-valued features in 4-dimensional scalar fields. *Proceedings of the IEEE symposium on Volume visualization '98*, New York, USA, pp. 103–110.

Waters K.W. and Co Ch.S., 2006. Using difference intervals for time-varying isosurface visualization. *In IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 5, pp. 1275–1282.

Wilhelms J. and van Gelder A., 1992. Octrees for faster isosurface generation. *In ACM Transaction on Computer Graphics*, Vol. 11, No. 3, pp. 201–227.

Wang Q. et al., 2007. An efficient and scalable parallel algorithm for out-of-core isosurface extraction and rendering. *In Journal of Parallel and Distributed Computing*, Vol. 67, No. 5, pp. 592-603.