SCCG 2007 conference, ISBN 978-80-223-2292-8, pp.216-223,2007

Iso-contouring in time-varying meshes

Slavomír Petrík*
University of West Bohemia, Pilsen Czech Republic

Václav Skala†
University of West Bohemia, Pilsen Czech Republic

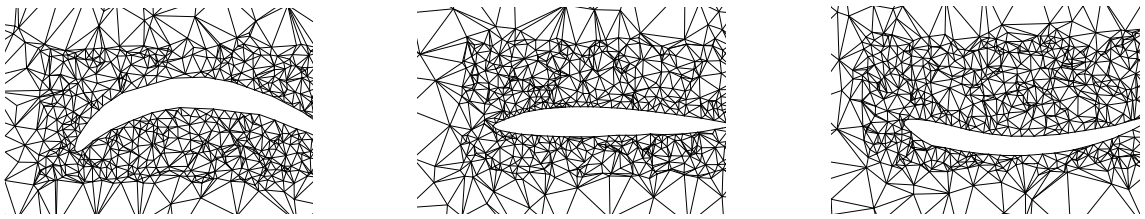


Figure 1: Dynamic mesh adapting around the changing airfoil during simulation.

Abstract

Dynamic meshing techniques are widely used in the Computational Fluid Dynamic (CFD) and Computational Mechanics (CM) simulations. Simulated moving parts or dynamic boundaries of simulation domain force a simulation mesh to change from one time step to another. So, geometry of the mesh cells and their number vary as the simulation time proceeds. This paper presents a novel approach for smooth iso-contours extraction from 2D dynamic mesh at and in between defined time slices. No assumption is made about mesh cells correspondence between successive time slices. The key idea of the proposed method is to establish an edge-edge correspondence between meshes at adjacent time steps. Then the data structure supporting an easy iso-contours interpolation from intermediate time slices is built. We provide the proof that our method produces correct results under defined conditions and demonstrate our approach on the data sets from the CFD simulations.

CR Categories: I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling;

Keywords: iso-contour, time-varying mesh, interactive visualization

1 Introduction and problem definition

Visualization of the iso-surfaces [Lorensen and Cline 1987] evolution over time is a powerful tool for understanding dynamic data behavior. Most of the research done in this area was focused on the case when a simulation mesh remains static and the scalar values in its vertices vary over time. Constantly growing computational power enables the scientists and engineers to perform CFD simulations which include moving parts and dynamic shape of simulation domain. So, the underlying simulation mesh is required to change from one time step to another.

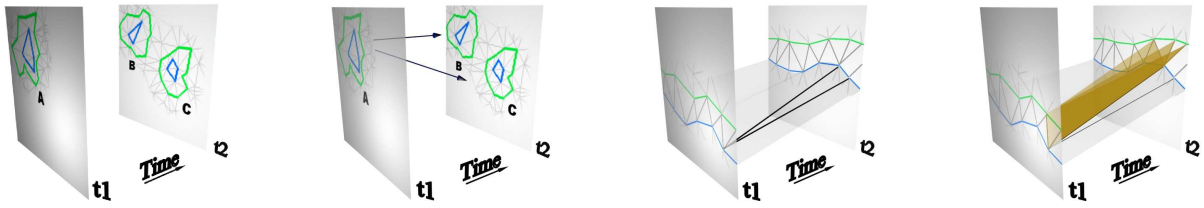
*e-mail: spetrik@kiv.zcu.cz
†e-mail:skala@kiv.zcu.cz

The concept of dynamically moving and changing simulation mesh has evolved into group of techniques commonly known as Arbitrary Lagrangian-Eulerian (ALE) methods [Donea et al. 2004]. In pure Eulerian approach a simulation mesh remains static and a fluid movement is reflected in the changing scalar values associated with mesh vertices. In the Lagrangian approach the movement of a simulation mesh is driven by the fluid flow (mesh vertices remains attached to the same points flowing in a fluid). ALE methods combine the best features from both Eulerian and Lagrangian approach. A mesh generated by the ALE methods conforms to the changing boundaries of a simulation domain while preserving criteria of mesh quality. Examples of dynamic meshes comes from various industrial applications: [Amsden 1997; Marshall 2002; Cavallo et al. 2005; Doleisch et al. 2005].

Up to now, very little has been done in the area of interactive iso-surfaces extraction from dynamic meshes. This problem has been mentioned by the range of authors [Ma 2003; Doleisch et al. 2005; Bernardon et al. 2006]. Most of the visualization techniques for time-varying iso-surface extraction assume static simulation mesh (geometry and number of mesh cells remains constant during the whole simulation). So, they are ill-suited or completely unable to handle dynamic mesh.

The approach proposed in this article focuses on extraction of evolving iso-contours from dynamic mesh at and in between time slices defined in a data set. Formally this visualization problem can be formulated as: Given the input scalar data organized in a sequence of time slices, each of which is represented by a different unstructured mesh with the scalar values associated to its vertices, find a method for smooth continuous iso-surface extraction at and in between the time slices. In this work we restrict to the simpler case of 2D dynamic mesh, however extension of the proposed approach into 3D is also discussed. The key idea of our solution to the problem of iso-surface extraction from dynamic mesh is to establish an edge-edge correspondence between meshes at successive time slices and to build a data structure to support a fast iso-contour extraction for arbitrary time and iso-value. This is very useful for interactive visualization of smoothly evolving iso-contours. With the proposed method, there is no need to re-sample the time-varying meshes for the visualization purposes.

In the Section 2 we provide a list of existing related techniques and also brute-force solution to this problem is discussed. Sections 3 and 4 describe our proposed solution, followed by the proof of correctness in the Section 5. Numerical stability and other aspects of the proposed method are discussed in the Section 6. Extension of the proposed method into 3D is discussed in the Section 7. Finally the test results are provided in the Section 8.



(a) (b) (c) (d)

Figure 2: Four stages of the proposed method: (a) *iso-components* extraction from the original data set, (b) matching and tracking of iso-components between successive time slices, (c) mapping of inner envelopes of iso-components, (d) final edge-to-edge mapping.

2 Previous work

"Time-varying unstructured mesh data sets have been either rendered in a brute-force fashion or just re-sampled and down-sampled onto a regular grid for further visualization calculations." [Ma 2003]. When working over static structured or unstructured mesh, vertices connectivity in between time slices is implicit. A lot of research has been devoted to a fast extraction of time-varying iso-surfaces [Shen 1998; Weigle and Banks 1998; Sutton and Hansen 1999; Shen et al. 1999]). All of these methods are well designed for the static regular simulation mesh; however they are not able to handle dynamic unstructured mesh.

Situation changes dramatically when trying to visualize data organized in a dynamic mesh. The correspondence between mesh vertices at successive time slices may not be one-to-one, and thus one value may be interpolated to many [Weigle and Banks 1998]. This may results in many possible iso-surfaces evolution scenarios. Moreover, due to the moving boundaries of simulation domain, some mesh vertices may not have their direct equivalent in the successive time slice.

Closely related to our work is the approach of Szymczak [Szymczak 2005], investigating the iso-contour evolution in between time slices. Szymczak assumes a static regular mesh. Doleisch et al. [Doleisch et al. 2005] presented method for datasets with dynamic simulation mesh. Their technique divides the entire time frame of a dataset into so-called *topology zones*, within which connectivity of mesh vertices is implicit. However, mesh cells are not matched or tracked over topology zone borders called *rezone points*. Fig. 3 shows the principle of *brute-force* solution to the extraction of iso-contours from the dynamic meshes. A hyper-surface ruled at both ends by the iso-contours extracted at successive time slices is built and cut at the desired time. This solution is on an account of limited interactivity, because a hyper-surface has to be recomputed every time a user changes desired iso-value.

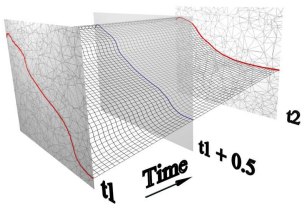


Figure 3: The *brute-force* approach to iso-contours extraction from the dynamic meshes. Hyper-surface is build between iso-contours in successive time slices and cut at the desired time.

3 Proposed method

As stated in the Introduction the goal is to build an edge-edge correspondence between adjacent time slices (Fig. 2(d)). Then we build a data structure to achieve fast and smooth visualization of iso-contours evolution for arbitrary time and iso-value. Proposed method works upon *iso-components*. Iso-components are strips of neighboring triangles extracted from simulation mesh at particular time step. Each iso-component covers certain range of iso-values and has its inner and outer envelope composed of the lists of mesh vertices (Fig. 4).

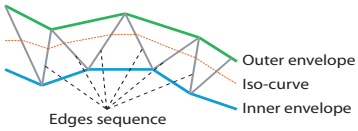


Figure 4: *Iso-component* is a strip of neighboring triangles with inner (IE) and outer (OE) envelope.

Instead of tracking mesh vertices over time and trying to correlate them, a set of iso-components is extracted from each time slice (Fig. 2(a)). Iso-components are then matched and tracked between the pairs of adjacent time slices (Fig. 2(b)). Finally an edge-edge mapping is established between edges (Fig. 2(c), (d)) of corresponding iso-components. This edge-edge correspondence is used during final interpolation of the iso-contours in between defined time slices. Preprocessing phase as described by the following list has to be done for each pair of consecutive time slices. In the visualization phase the preprocessed data are used for a fast iso-contours extraction.

1. Preprocessing:
   (a) *Iso-components extraction* from the defined time slices, forming stand-alone entities with inner and outer envelope (Fig. 2(a)).
   (b) *Iso-components tracking* to determine evolutionary events [Samtaney et al. 1994] which happen between two adjacent time slices (Fig. 2(b)).
   (c) *Mapping of iso-components' inner envelopes* vertices (Fig. 2(c)) to gain a guide for the final edge-edge mapping.
   (d) *Edge-edge mapping*, to make out the final sets of edge pairs for all tracked iso-components pairs (Fig. 2(d)).
2. Visualization based on established edge-edge mapping.

Preprocessing phase begins by building a list L of all different scalar values $(v_1, v_2, ..., v_i, v_{i+1}, ..., v_n)$ associated with the mesh vertices of two adjacent time slices. This list of iso-value ranges is then sorted in ascending order. Next a set of the iso-components is extracted from each time slice using Continuation method [Wyvill et al. 1986]. Continuation method tracks the particular iso-value into the neighboring triangles, creating single connected iso-components with inner and outer envelopes (Fig. 2(a)). Iso-components are extracted from each time slice for each iso-value $\frac{v_i+v_{i+1}}{2}$ of the list L. To find the initial cell for Continuation method and particular iso-value the Path Seeds technique [Carr and Snoeyink 2003] is used, which builds up upon the work on Contour trees [Carr et al. 2000].

Iso-components matching and tracking step (Fig. 2(b)) is basically the *feature tracking problem* first described by the Samtaney et al. [Samtaney et al. 1994]. Their approach has been later improved in the various ways by a range of other authors [Silver and Wang 1998; Reinders et al. 1999; de Leeuw and van Liere 2001]. To determine evolutionary events (birth, death, split, merge) which happen to the iso-components at or in between time slices we use iso-components' area overlapping test. This test marks the successors / predecessors of iso-components if their relative overlapping area exceeds user defined threshold α.

Once we have iso-components and their predecessors / successors defined, a mapping of their inner envelope vertices is done (Fig. 2(c)). This mapping provides a guide for the final edge-edge mapping between iso-components from a pair of adjacent time slices. This problem is essentially a problem of linear morphing of two polygons and is described in detail in Section 4. Our solution to this "polygon morphing" problem is inspired by the approach of Bajaj et al. [Bajaj et al. 1996]. They assumed only closed polygons and use a complex set of orientation rules for mapped polygons, which is not possible in our case, because we also have to deal with open polygons (iso-component may be open, so its inner envelope is also an open polyline).

Mapping of inner envelope vertices provides a guide to the edge-edge mapping for each pair of successive iso-components (Fig. 2(d)). In this step successive iso-components' edges emanating from the vertices joined by the inner envelopes mapping procedure are paired. Such edge-edge pairs are then stored into a file as a result of the preprocessing phase.

During the visualization, queries of the form *query(iso-value, time)* are accepted and processed. First the pair of adjacent time slices covering the queried *time* is selected. Next, the iso-component covering the queried *iso-value* is selected from the earlier time-slice. As depicted by the Fig. 5, a list of selected iso-component's edges is traversed and points *C* and *E* are interpolated from point-pairs *A*-*B* and *D*-*F* according to the desired iso-value. Point *R* is then interpolated out of *C*-*E* according to the desired *time*. Extracted *R* points are then connected by line, approximating resulting iso-contour.

When the *time* value is changed by user during interactive exploration only the new set of *R* points has to be interpolated out and a resulting iso-contour is assembled from them. In the worse case when the new *time* value does not fit into the time interval spanned by the currently selected pair of time slices a new proper pair of time slices is selected.

When the *iso-value* is changed and fits into the range of iso-values covered by the currently selected iso-component, an interpolation scheme described in the previous paragraph is done, otherwise a new proper iso-component is selected. Results of such interactive exploration of CFD data sets are described in Section 8.

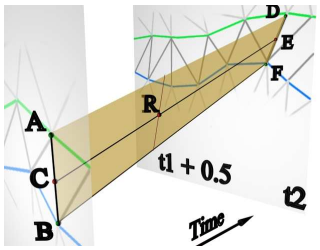


Figure 5: Points on the mapped edges used during visualization.

4 Mapping of iso-components' Inner Envelope Vertices

Iso-component may be open or closed, depending on the processed data. Therefore, let's define the common rule for an iso-component's inner envelope orientation.

A gradient of the form $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$ computed at any of iso-component's inner envelope vertices can be used to unambiguously determine the area *above* (in the direction of gradient) and *below* (opposite the direction of gradient) an inner envelope. Thus, we define that an inner envelope is oriented in the direction such that area "above" it is on its right side (Fig. 6(a)). The case when $\nabla f = 0$ is left unsolved by our method and the user should be notified about its occurrence.

It has to be said that the iso-component may have one or more successors (because an iso-component may split into two or more others in the successive time slice).

Let's denote the set of iso-component's inner envelope vertices as IE_{t1} and a set of all vertices of its successors' inner envelopes as IE_{t2}. In the following text the term *slicechord* represents an imaginary line connecting two vertices from adjacent time slices - investigated vertex q and its *candidate vertex* $C(q)$ (Fig. 6(b)).

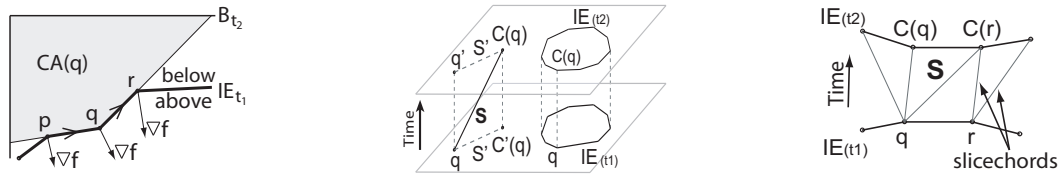
Algorithm for mapping IE_{t1} onto vertices of IE_{t2} visits each vertex $q \in IE_{t1}$ and looks for a candidate vertex $C(q) \in IE_{t2}$. For each q a *candidate area* (clarified in the Def. 1) is determined. Candidate area restricts the set of possible candidate vertices from IE_{t2} and the closest one of them is connected by a slicechord with q. If the closest one doesn't meet all the criteria for candidate vertex (Def. 3), then the second closest is taken and so on. Special care is taken to the so-called *multivertices* discussed later in Section 4.1.

This traversal and slicechords building process obviously leaves some of the IE_{t2} vertices uncovered by the mapping, and thus the mapping procedure is done for uncovered IE_{t2} vertices once again backward onto the IE_{t1} vertices. Mapping the IE vertices in both directions (IE_{t1} onto IE_{t2} and IE_{t2} onto IE_{t1}) cause that some vertex may be mapped onto more than one vertex in the adjacent time slice.

Set of produced slicechords defines mapping of IE_{t1} vertices onto IE_{t2}. The last step in this mapping algorithm is to remove *redundant slicechords* which are defined by the Def. 4, so we have a simple mapping of IE_{t1} to IE_{t2}.

Now let's define the precise rules for mapping the inner envelopes vertices.

Definition 1. *Candidate area* of vertex q (denoted $CA(q)$) is an area which lies below the inner envelope the q belongs to. Borders of candidate area are defined by two line segments **qp** and **qr** and the borders of mesh at successive time slice (Fig. 6(a)).



(a) (b) (c)

Figure 6: (a) Candidate area CA(q) of vertex q (shadowed region). Borders of CA(q) are defined by the segments **qp** and **qr** and the borders of mesh at successive time slice B_{t_2}. (b) Projection S' of slicechord S. (c) Redundant slicechord S.

Definition 2. *Projection* [Bajaj et al. 1996] of a slicechord S spanned between time slices onto those 2D time slices, is denoted by appending a prime sign (') to S (Fig. 6(b)).

Definition 3. This definition contains rules for candidate vertex of q (denoted $C(q)$) and possible slicechord spanned between q and $C(q)$.

1. $C(q) \in \{v | v \in IE_{t2} \cap CA(q)\}$
2. $q \notin CA(C(q))$
3. Projection S' of a slicechord S spanned between q and $C(q)$ does not cross projections of:
   (a) any other slicechord S'_n, except their joining in the slicechords' end-points,
   (b) any line segment of iso-component's inner envelope the q belongs to, nor the line segment of the iso-component's successor / predecessor inner envelope.
4. If q is multivertex (defined in Section 4.1), then $C(q)$ has to lie inside candidate areas of all vertices the q is composed of. And vice versa, if C'(q) is multivertex then q can't lie inside candidate area of any of the vertices the multivertex C'(q) is composed of.

Definition 4. *Redundant slicechord S* is a slicechord, which has both endpoints, incident with slicechords other than S (Fig. 6(c)).

Because inner envelopes mapping procedure is very sensitive to their shapes, we now define the criterion that has to be fulfilled.

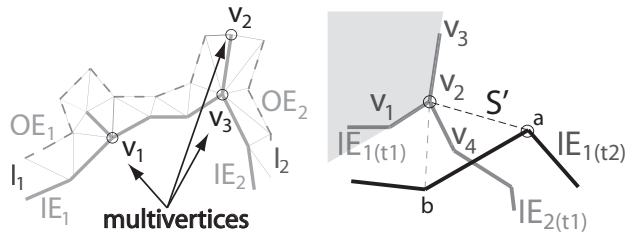
Criterion 1. Continuing along the inner envelope A in the direction of its orientation, the sequence of crossing points with the mapped inner envelope B from adjacent time slice have to follow the orientation of A.

Criterion 1 essentially states an assumption that the data are not changing rapidly, so, the shapes of mapped inner envelopes don't differ significantly from time slice to time slice. In the case of sudden significant data change between time steps we are able to detect it by the violation of criterion 1 as well as in the iso-components tracking phase. In this case the mapping algorithm will still finish, but it does not guarantee the correct shape of the resulting iso-contour and user should be notified of such "critical" areas. Experiments (Section 8) show that violation of criterion 1 is very rare in the data sets available for our tests.

Another question is how this inner envelope mapping procedure handles the cases when one inner envelope splits into two or more on the successive time slice. Note, that this is an open problem in polygon mapping and its correct solution is hard to find. Although, described mapping algorithm is unable to handle very complicated splits or merges of the inner envelopes, it provides reasonable results in the most cases of inner envelope splits and merges. This statement is supported by the proof of correctness (Section 5) and also has been verified experimentally on the testing data sets.

4.1 Handling multivertices

Definition 5. *Multivertex* is a vertex of an iso-component's inner envelope IE shared multiple times by the IE or shared by two inner envelopes of different iso-components from the same time slice. Fig. 7(a) shows two iso-components I_1, I_2 with inner envelopes IE_1, IE_2 sharing multivertices v_2, v_3. Multivertex v_1 is used two times by the IE_1.



(a) (b)

Figure 7: (a) Example of multivertices v_1, v_2, v_3. (b) Proposed slicechord S' violates Point 4 of the Def. 3.

Multivertices, if not treated properly may become a source of problems for correct inner envelopes mapping as described in Section 4. For multivertex, Point 4 of Def. 3 has to be fulfilled when searching for its candidate vertex. Omitting this rule may result in improper inner envelopes mapping, thus the edges of iso-components will not be mapped properly. Consequently an iso-contour interpolated out of such corrupted edge-edge mapping may be misshaped.

Fig. 7(b) depicts an example of what Point 4 of Def. 3 means in practice. Multivertices v_2, v_3 are shared by the $IE_{1(t1)}$ and $IE_{2(t1)}$. When looking for a candidate vertex $C(v_2)$ of v_2, then possible slicechord S' has to be refused because $a \in IE_{1(t2)}$ doesn't lie inside all candidate areas of v_2. Vertex a does lie inside candidate area $CA(v_2)$ made of v_2v_3, v_2v_4, but as can be seen the gray shaded candidate area $CA(v_2)$ made of v_2v_1, v_2v_3 violates Point 4 of Def. 3 because a does not lie inside it. Fig. 7(b) suggests to handle this situation by spanning slicechord between vertex b and $v_2 = C(b)$.

5 Proof of correctness

In order to secure smooth resulting iso-curve without self-crossing or other disturbing artifacts, we have to prove, that our inner envelope mapping algorithm will always produce an ordered set of slicechords, such that sequence of indices at their both sides is monotonically increasing (or decreasing) (i.e. slicechords are not crossing each other):

$$\begin{aligned} &\forall p,s \in IE_{(t1)}, p \neq s: \; p < s \;\Leftrightarrow\; C(p) \leq C(s), or \\ &\forall p,s \in IE_{(t1)}, p \neq s: \; p < s \;\Leftrightarrow\; C(p) \geq C(s) \end{aligned} \tag{1}$$

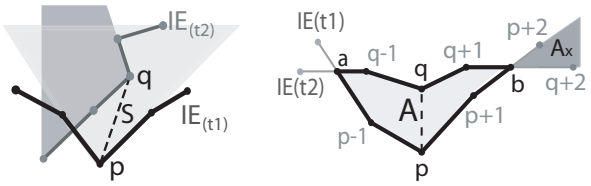
Let's consider slicechord S_p^q spanned between vertex $p \in IE_{t1}$ and $q = C(p) \in IE_{t2}$. Vertex q has to fulfill all the requirements from the Def. 3. Therefore, S_p^q can't lie inside both $CA(p)$ and $CA(q)$ at the same time (Fig. 8(a)). Otherwise, it will violate Point 2 of the Def. 3:

$$S_p^q \notin (CA(p) \cap CA(q)) \tag{2}$$

In example given by the Fig. 8(b) the possible area (denoted A) for any slicechord emanating from vertex p, is bordered by the sequence of edges between pa, pb and qa, qb (because of Point 3b of the Def. 3). By accepting S_p^q the area A is divided into two subareas. So, any future slicechord emanating from $p-1$ can be placed only into one of the subareas of A, because two slicechords can't cross each other (Point 3a of the Def. 4).

This ensures, that the condition 1 always holds within A. Point 3b of the Def. 3, together with Def. 1 of *candidate area* guarantee, that no slicechord will be accepted between the vertices which belong to A and vertices of any other such area A_x along mapped inner envelopes.

If the Criterion 1 (Section 3) holds for mapped $IE_{(t1)}$ and $IE_{(t2)}$, then the resulting set of slicechords spanned between $IE_{(t1)}$ and $IE_{(t2)}$ always satisfies condition 1.



(a) (b)

Figure 8: (a) Shadowed regions show CA(p) and CA(q) and their product. (b) Area A bordered by two adjacent intersections of $IE_{(t1)}$ and $IE_{(t2)}$.

6 Numerically stable implementation

The two main issues that may influence numerical stability of the implementation are very close scalar values associated with adjacent mesh vertices and very close position of the two adjacent vertices.

The first one can be avoided by rounding all scalar values associated with mesh vertices to the n-1 digits, where n is guaranteed accuracy of the data type of scalar values associated with mesh vertices. The last digit is saved for the first phase of the method, when the middle values of iso-intervals are determined (Section 2). Otherwise, the iso-interval may collapse into a value thinner than precision of the used data type and the iso-value (equal to the half of an iso-interval limit) used during iso-component extraction won't be determined properly. Consequently the extracted iso-components will be corrupted.

The second one - the very close position of two adjacent vertices - may result in extremely sharp triangles, causing numerical instability while working with position of those vertices. By the term "very close" we mean, the distance of two adjacent vertices which is lower than guaranteed accuracy of data type used for representation of their position. This issue has to be watched while generating the input mesh, before processing the data by our method. This is because remeshing of input mesh is very sensitive operation which depends on the particular data and may consequently influence the result of visualization. Therefore, we assume that input meshes of our method are free of such "very close vertices".

7 3D case

In the case of 3D time-varying simulation mesh, the overall principle of this method can be used; however some of its aspects have to be treated in a slightly different manner.

3D version of 2D iso-components are iso-volumes extracted by the continuation method (Section 3) tracking the iso-values into the neighboring tetrahedron. Such iso-volume has inner and outer envelope in the form of mesh composed of 2D triangles. Iso-volumes tracking and matching has been well studied in the past decades. Inner envelope mapping algorithm as described in Section 4 then can be used to search candidate vertices in the 3D space. Candidate area as defined by the Def. 1 is a 3D subspace with borders governed by the incident triangles (from the inner envelope) of the investigated vertex. In the last phase the tetrahedra of the mapped iso-volumes are mapped, guided by the slicechords spanned between iso-volumes' inner envelopes. From this basic point of view no principal restrictions are known for the 3D extension of the proposed method.

Various aspects of this 3D extension need to be further investigated before its application onto data sets. Also, the preprocessing phase of such 3D version of the method needs to be optimized because 3D computations are in generally more expensive than their 2D versions.

8 Tests

The method has been implemented in C# and tests run on Intel 3.2GHz workstation with 2GB of RAM. Two 2D data sets were used for testing.

Airfoil data set is the result of simulation of low-speed air wave hitting the leading edge of the flexible airfoil. Scalar values associated with the mesh vertices are the velocity magnitudes of advancing air wave. The data set consists of 200 time steps each represented by the triangular adaptive mesh. Number of mesh vertices vary between 16 000 and 17 000. Simulation mesh adapts to the changing shape of flexible airfoil at each simulation step. For the test we took every 10th time slice and computed the iso-contour evolution in between them by our method.

Fig. 9 (a) and (b) show the original data at time steps 10 and 100. Fig. 9 (c) shows the adapted simulation mesh at time step 100. Iso-contour evolution computed by our method between time steps 30 and 50 is depicted by the Fig. 9 (d). Finally a set of iso-contours extracted by our method at inter-time slice times is depicted by the Fig. 9 (e) and (f). Because the number of mesh cells is different for each time step (dynamic mesh) a classic point-wise linear interpolation known from static regular simulation grids can not be used.

Payload data set originate from CFD simulation of payload release from under an aircraft wing. Data sets consists of 500 time steps from which every 10th time step has been saved and used as an input of our method. Number of samples at each time step vary between 60 000 and 62 000. Simulation mesh adapts itself around falling payload as the simulation time proceeds. Fig. 11 depicts the set of iso-contours extracted by our method. The application interface containing sliders for interactive set up of iso-value and time of desired iso-contours is also depicted.

The third test is the accuracy comparison of our method and linear interpolation. In order to be able to do such comparison our method was applied onto the data set with static simulation mesh. During the simulation a low speed air wave hits the leading edge of the static airfoil. So, the simulation mesh remains static during the whole simulation. Again every 10th time step was used and iso-contour evolution between them was calculated by the both our method and linear interpolation. Results of calculations were compared against the real data produced during the simulation.

Graph on the Fig. 10 shows the average deviation of the calculated iso-contours against the real data. Average deviation of our method is almost the same or better (time steps 45 to 195) as that of linear interpolation. Slightly worse deviation of our method before time step 35 reflects the fact that the scalar data are changing fast at the beginning of the simulation when the overall speed of air wave is high. Thus, we assume that our method achieves approximately the same accuracy in the case of dynamic mesh (where the linear interpolation can not be used) as that of linear interpolation achieves in the cases of static simulation mesh.

For both ”Airfoil” and ”Payload” data sets, once we preprocessed the input meshes as described in this article, we were able to interactively explore the smoothly evolving iso-contours without any noticeable delays in visualization. Tab. 1 provides preprocessing times for both data sets as well as extraction times for various iso-values and simulation times. This has been achieved without expensive re-sampling of the original data sets onto regular static grid, which introduces higher error and allows the case when a vertex of regular steady grid interpolated in time dimension hits the moving object in the successive time slice (thus, no scalar value suitable for interpolation is at that place) or runs completely out of the simulation mesh in the successive time slice because of the time-varying boundaries of a simulation domain.

		Flexible airfoil	**Payload release**
Preprocessing time [minutes]		13.3	42.6
iso-value	**timestep**	**Extraction time [ms]**	
0.1	10	0.2133	0.5668
0.5	50	0.1158	0.7885
0.6	180	0.1288	0.6841
0.85	230	-	0.6382
1.2	240	-	0.8833

Table 1: Performance results for both CFD data set. Extraction times were measured for various iso-values and time steps.

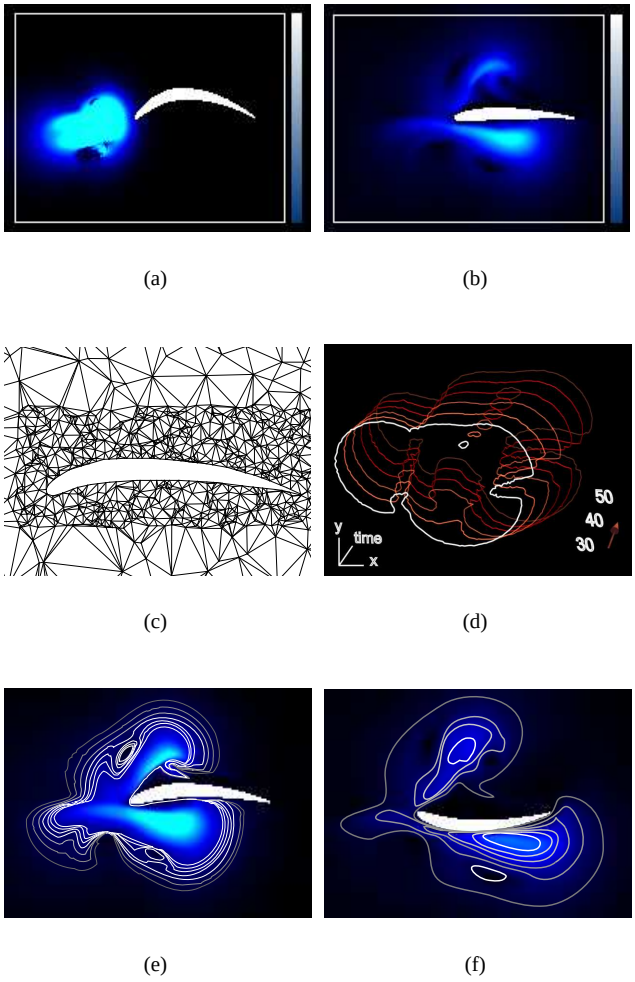


Figure 9: Dataset from CFD simulation of low-speed air wave hitting the leading edge of experimental dynamic airfoil. (a)-(b) input data (speed magnitude) at the time steps 10 and 100, (c) dynamic adaptive mesh (time step 80), (d) iso-curve evolution computed by the proposed method between the time steps 30 and 50, (e)-(f) extracted iso-curves at time steps 74 and 186.

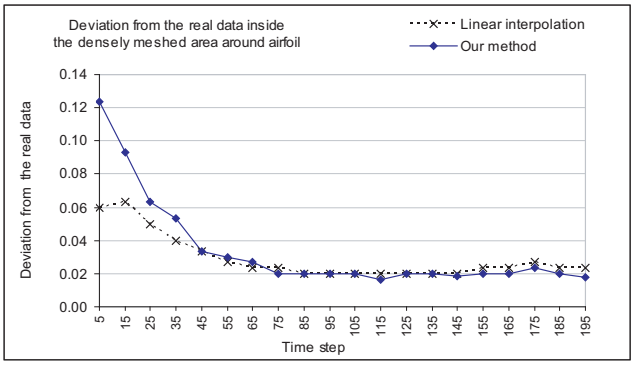


Figure 10: Accuracy comparison: Our method vs. linear interpolation. Data from CFD simulation of low-speed airwave hitting the leading edge of static airfoil (static simulation mesh).

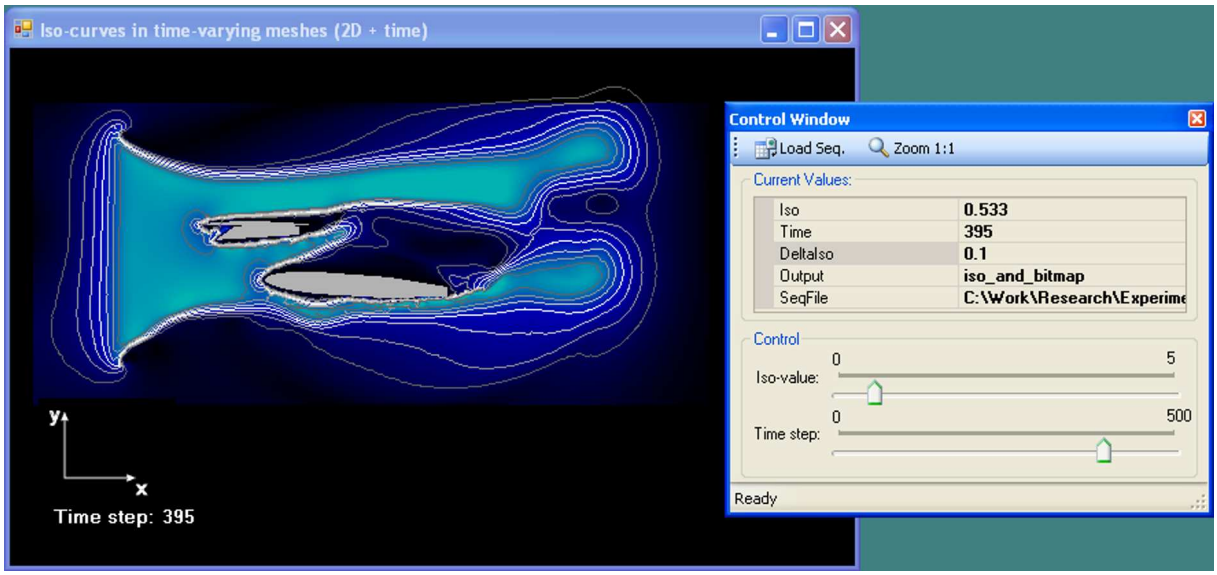


Figure 11: Iso-curves extracted from payload-release simulation (iso-value=0.533, time step: 395). Blue rendered original data at the background (speed magnitude values) are just for illustration and comparison purposes.

9 Conclusion

We presented a method for iso-contours extraction from dynamic meshes. A method is able to compute iso-contour also in between defined time slices. In this way a smooth continuous iso-contours evolution can be visualized. This is particularly useful in the cases of large simulations when only every n-th time step is saved and the evolution of iso-contour can be interpolated and visualized by the proposed method. The method has been successfully applied onto the data sets from Computational Fluid Dynamics simulations.

The main contributions of presented method are:

- Method supports a sequence of triangular meshes with time-varying geometry / topology as its input, without *a priori* knowledge about vertices connectivity between successive time slices.
- There is no need to re-sample the original data onto the static regular grid, which is required by most of the existing methods for time-varying data visualization.
- Iso-contour visualization itself is done at interactive frame-rates, which makes investigation of dynamic data behavior efficient.
- Steps are incorporated which allow to tune the method up for a particular dataset without changing its overall principle (selection of the best suitable method for iso-components matching between successive time steps).

Future work involves further exploration of the method's 3D extension and also improvements of the method that would lead to its higher accuracy and ability to handle data sets sampled more sparsely along the time dimension.

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. Data sets are courtesy of Centre of Computer Graphics and Data Visualization, University of West Bohemia, Czech Republic.

References

AMSDEN, 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.

BAJAJ, C. L., COYLE, E. J., AND LIN, K.-N. 1996. Arbitrary topology shape reconstruction from planar cross sections. *Graphical Models and Image Processing 58*, 6, 524–543.

BERNARDON, F., CALLAHAN, S., COMBA, J., AND SILVA, C. 2006. Interactive volume rendering of unstructured grids with time-varying scalar fields. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization '06*, 51–58.

CARR, H., AND SNOEYINK, J. 2003. Path seeds and flexible isosurfaces using topology for exploratory visualization. In *Proceedings of Symposium on Data Visualization '03*, Eurographics Association, 49–58.

CARR, H., SNOEYINK, J., AND AXEN, U. 2000. Computing contour trees in all dimensions. In *Proceedings of SODA '00*, Society for Industrial and Applied Mathematics, 918–926.

CAVALLO, P., HOSANGADI, A., AND AHUJA, V. 2005. Transient simulations of valve motion in cryogenic systems. In *Proceeding of 35th AIAA Fluid Dynamics Conference and Exhibit*.

DE LEEUW, W., AND VAN LIERE, R. 2001. Chromatin decondensation: a case study of tracking features in confocal data. In *Proceedings of Visualization '01*, IEEE Computer Society, 441–444.

DOLEISCH, H., MAYER, M., GASSER, M., PRIESCHING, P., AND HAUSER, H. 2005. Interactive feature specification for simulation data on time-varying grids. In *Proceedings of SimVis '05*, 291–304.

DONEA, J., HUERTA, A., PONTHOT, J.-P., AND RODRIGUEZ-FERRAN, A. 2004. *Encyclopedia of Computational Mechanics*, vol. 1. John Wiley & Sons.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of ACM SIGGRAPH '87*, ACM Press, 163–169.

MA, K.-L. 2003. Visualizing time-varying volume data. *Computing in Science and Engineering 5*, 2, 34–42.

MARSHALL, L., Ed. 2002. *Fluent news: Dynamic Mesh*, vol. XI. Fluent Inc.

REINDERS, F., POST, F. H., AND SPOELDER, H. J. W. 1999. Attribute-based feature tracking. In *Data Visualization '99*. Springer-Verlag Wien, 63–72.

SAMTANEY, R., SILVER, D., ZABUSKY, N., AND CAO, J. 1994. Visualizing features and tracking their evolution. *Computer 27*, 7, 20–27.

SHEN, H.-W., CHIANG, L.-J., AND MA, K.-L. 1999. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (TSP) tree. In *Proceedings of Visualization '99*, IEEE Computer Society Press, 371–377.

SHEN, H.-W. 1998. Iso-surface extraction in time-varying fields using a temporal hierarchical index tree. In *Proceedings of Visualization '98*, IEEE Computer Society Press, 159–166.

SILVER, D., AND WANG, X. 1998. Tracking scalar features in unstructured datasets. In *Proceedings of Visualization '98*, IEEE Computer Society Press, 79–86.

SUTTON, P., AND HANSEN, C. D. 1999. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (T-BON). In *Proceedings of Visualization '99*, IEEE Computer Society Press, 147–153.

SZYMCZAK, A. 2005. Subdomain-aware contour trees and contour tree evolution in time-dependent scalar fields. In *Proceedings of Shape Modeling International 05*, IEEE Computer Society, 136–144.

WEIGLE, C., AND BANKS, D. C. 1998. Extracting iso-valued features in 4-dimensional scalar fields. In *Proceedings of IEEE Symposium on Volume Visualization '98*, ACM Press, 103–110.

WYVILL, G., MCPHEETERS, C., AND WYVILL, B. 1986. Data structure for soft objects. *The Visual Computer 2*, 4, 227–234.