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

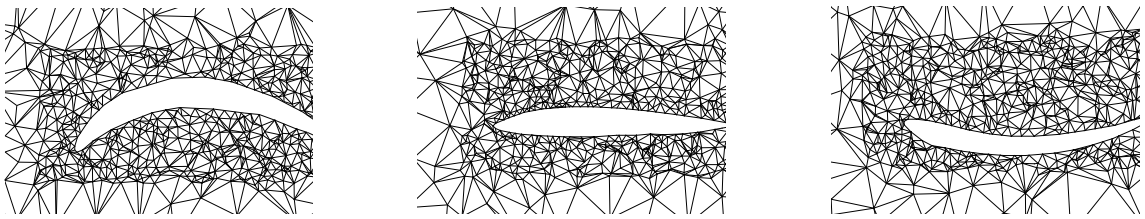


Figure 1: Dynamic mesh adapting around the changing airfoil during simulation.

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.

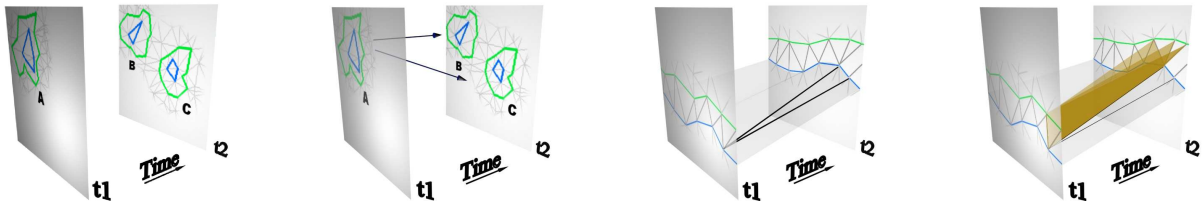
*e-mail: spetrik@kiv.zcu.cz
†e-mail:skala@kiv.zcu.cz

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.

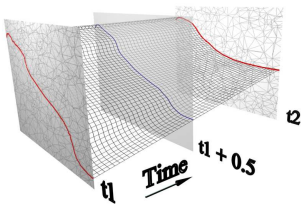


Figure 3: The *brute-force* approach to iso-contours extraction from the dynamic meshes. Hyper-surface is build between iso-contours in successive time slices and cut at the desired time.

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).

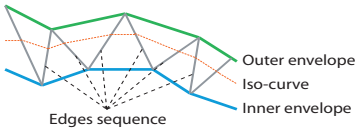


Figure 4: *Iso-component* is a strip of neighboring triangles with inner (IE) and outer (OE) envelope.

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.

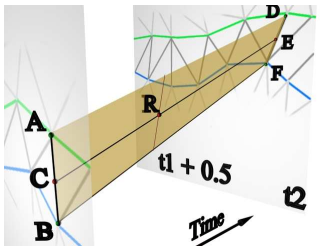


Figure 5: Points on the mapped edges used during visualization.

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)).

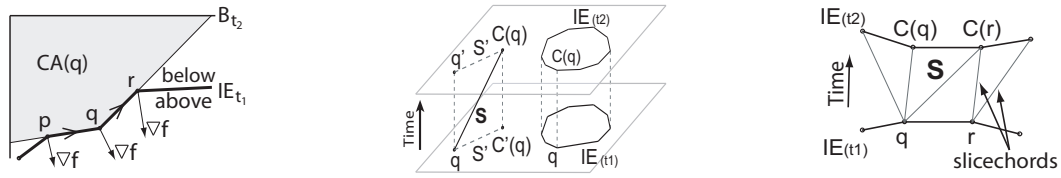
Algorithm for mapping IE_{t1} onto vertices of IE_{t2} visits each vertex $q \in IE_{t1}$ and looks for a candidate vertex $C(q) \in IE_{t2}$. For each q a *candidate area* (clarified in the Def. 1) is determined. Candidate area restricts the set of possible candidate vertices from IE_{t2} and the closest 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.

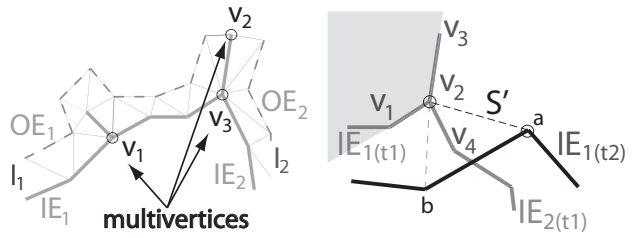
Criterion 1. Continuing along the inner envelope A in the direction of its orientation, the sequence of crossing points with the mapped inner envelope B from adjacent time slice have to follow the orientation of A.

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 $v_2 = C(b)$.