Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Extrakce iso-ploch z časově proměnných dat

Ing. Slavomír Petrík

disertační práce
k získání akademického titulu doktor
v oboru Informatika a výpočetní technika

Školitel: Prof. Ing. Václav Skala, CSc.
Katedra: Katedra informatiky a výpočetní techniky

Plzeň 2011

University of West Bohemia
Faculty of Applied Sciences

Isosurface extraction from time-varying data

Ing. Slavomír Petrík

doctoral thesis
submitted in partial fulfillment of the requirements
for a degree of Doctor of Philosophy
in Computer Science and Engineering

Supervisor: Prof. Ing. Václav Skala, CSc.
Department: Department of Computer Science and Engineering

Plzeň 2011

Prohlášení

Prědkládám k posouzení a obhajobě disertační práci zpracovanou na závěr doktorského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Prohlašuji, že tuto práci jsem vypracoval samostatně, s použitím odborné literatury a dostupných pramenů uvedených v seznamu, jenž je součástí této práce.

V Plzni dne 4. 3. 2011

Ing. Slavomír Petrík

Abstract

Isosurface visualization is a powerful tool for exploration of numerical data-sets. The topic has been a subject of intensive research for more than two decades. Starting from the simple techniques for static regular datasets in the late '80s, the research has advanced to the interactive exploration of evolving isosurfaces in time-varying datasets.

This work contributes to the field of Isosurface Extraction by the introduction of three new methods. The first method focuses on the static, regular and unstructured datasets. By careful study of the existing methods, a room for their space and time optimization is identified. Our method is based on a newly proposed transformation of the input data into an alternative space. The method decreases space requirements and processing time considerably when compared to existing state-of-the-art methods.

The second part of this work deals with a relatively new topic of Isosurface Extraction from datasets with dynamic meshes. Such datasets usually originate from Computational Fluid Dynamic (CFD) simulations, in which moving boundaries of a simulation domain force the simulation mesh to change itself with each discrete time step. While the techniques for generating dynamic meshes have been intensively studied and optimized, there is a lack of suitable visualization methods.

This work introduces two new methods for Isosurface Extraction from the datasets with dynamic meshes. The first method tries to approach the problem from the geometric point of view. The correspondence of cells between adjacent time steps is calculated back, based on the cells' positions and values. The method focuses on the simpler case of the 2D triangular dynamic meshes. The second method simplifies data preprocessing compared to the first approach and offers the ability to extract isosurfaces from a 3D dynamic mesh.

All proposed methods were applied to real-world datasets. The focus has been set on usability of the proposed methods in practice. The results achieved during tests along with formal complexity comparisons clearly show advantages of the methods proposed.

keywords: isosurface, efficiency, scalar field, dynamic mesh

Abstrakt

Zobrazování iso-ploch je častě používaným nástrojem pro vyšetřování datových množin. Tato tématika je intenzívně studovaná už téměř 20 let. Výzkum pokročil od jednoduchých metod používaných na počátku osmdesátých let až k interaktivnímu zobrazováni iso-ploch z časově proměnných dat.

Tato práce přispívá k výzkumu extrakce iso-ploch třema novými metodami. První prezentovaná metoda je zaměřena na statické datové množiny. Podrobným studiem již existujících metod byl odhalen prostor pro jejich časovou a prostorovou optimalizaci. Prezentovaná metoda je založená na transformaci vstupních dat do nového alternativního prostoru. V porovnání s nejmodernějšími existujícími metodami snižuje naše metoda prostorové nároky a výpočetní čas.

Druhá část této práce je zaměřená na relativně novou tematiku extrakce iso-ploch z datových množin s časově proměnnou sítí. Taková data převážně pochází z numerických simulací proudění kapalin, ve kterých si pohybující se části vyžadují změny sítě s každým dalším časovým krokem. Zatímco techniky pro generování časově proměnných sítí byly intenzívně studovány, vhodných zobrazovacích technik je nedostatek.

První ze dvou prezentovaných metod se zaměřuje na výpočet korespondence buněk sítě ve dvou po sobě následujících časových krocích za pomoci geometrické podobnosti a vzájemné pozice buněk. Metoda je vhodná pro dvourozměrné trojúhelníkové sítě. Druhá metoda zjednodušuje a urychluje předzpracování dat a poskytuje možnost extrakce iso-ploch ze třírozměrných časově proměnných sítí.

Všechny navržené metody byly aplikovány na skutečné datové množiny. V průběhu výzkumu byl kladen důraz na použitelnost navržených metod v praxi. Výsledky dosažené v průběhu testu jasně poukazují na výhody navrhovaných metod oproti již existujícím metodám.

klíčová slova: iso-plocha, efektivnost, časově proměnná síť

Copyright ©2011 University of West Bohemia, Czech Republic

Acknowledgements

I would like to thank to his supervisor Prof. Václav Skala, and to my colleagues Martin Janda, Libor Váša, Petr Vaněček, Ivana Kolingerová for numerous consultations and comments and to Randy P. Hessel from Engine Research Center, University of Wisconsin-Madison, USA and to Jindřich Kňourek from CIV, University of West Bohemia, Czech Republic for providing datasets with dynamic simulation mesh. I would also like to thank to my family and to Katka who had been my constant support and source of encouragement.

This work was supported by the projects *VIRTUAL* - Virtual Research-Educational Center of Computer Graphics and Visualization, MSMT Czech Rep. No: 2C 06002, http://virtual.zcu.cz, *3DTV* - Integrated Three-Dimensional Television - Capture, Transmission and Display, FP6-2003-IST-2, Network of Excellence, No: 511568, http://3DTV.zcu.cz, *INTUITION* - Network of Excellence on VIrtual Reality aNd VirTUal Environments ApplIcaTIONs for Future Workspaces, FP6-2003-IST-2, Network of Excellence No: 507248-2.

Contents

Chapter 1

Introduction

1.1 Definitions

Simulation domain is an area interest of a simulation. The choice of a simulation domain depends on the problem under investigation. Definition of a simulation domain in this thesis includes geometric borders of the area where the simulation is going to take place as well as its physical properties.

Simulation mesh is a set of n-dimensional elements (*cells*) which discretizes simulation domain. A numerical simulation produces results over these discreted elements rather than over continuum of a simulation domain. These elements are called *cells*.

Dataset is a set of input data. The two basic components of a dataset (in this thesis) are:

- *Geometry.* Under the term *dataset geometry* we will understand spatial position of the discrete elements (e.g. vertex, cell) defined in the dataset.
- *Variables.* Under the term *dataset variable* we will understand scalar values or vectors defined in the dataset.

Isosurface is a surface represented by the points of constant value (e.g. pressure, temperature, velocity), constructed within a volume of data. Mathematically, the isosurface S for isovalue q is a surface:

$$S(q) = \{\mathrm{x} \mid F(\mathrm{x}) = q\}$$

where $\mathrm{x} \in S$ represents a point in an n-dimensional space; and $F(\mathrm{x})$ is a scalar function F: $R^N \rightarrow R$ defined over input data. 2D version of 3D isosurface is called isocontour.

1.2 Data Types

The purpose of this section is to categorize the types of input data mentioned in this thesis. It is not meant to be an exhaustive list of all possible data types that are used by the scientists and engineers. Our description limits only to the selected types of the datasets suitable for isosurface extraction.

The first categorization of the datasets divides them into two groups with respect to their relation to time.

1. *Static datasets.* A static dataset describes the state of simulation domain in one given moment in time. Neither geometry of the cells nor data values associated with discrete points within dataset are changing.

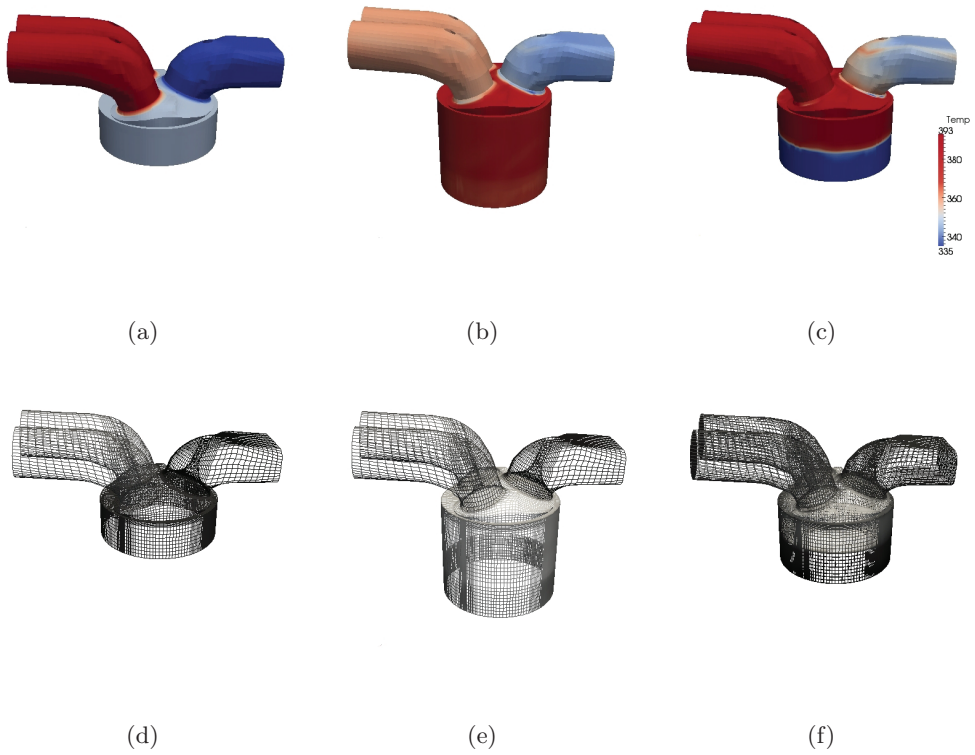
   There are two basic types of static datasets with respect to the shape of cells:

   - Regular static datasets. This is special type of the datasets for which the assumption is made about regularity of the cells shape. Usually, the cells are cubes with common length of sides. Advantage of this kind of dataset is that the cells geometry is implicitly given.
   - Unstructured static datasets. In this kind of dataset the cells can take the shape of tetrahedron, hexahedron or any other N-dimensional element or mix of elements. Geometry of the cells have to be stored along with data values defined within those cells.

2. *Time-varying datasets.* A time-varying dataset describes states of the mesh geometry and values defined within the geometry at defined discrete points in time.

   Two major groups of time-varying datasets are distinguished based on whether geometry of the cell varies over time or not.

   - Static geometry. Time-varying datasets with static geometry usually comes out of the numerical simulation where pure Lagrangian approach is used (i.e. no displacement of the points / cells within a simulation domain). The advantage is that geometry of the cells is independent of time and thus is saved only once.
   - Dynamic geometry. This is the last and most challenging type of datasets discussed in this thesis; challenging from both storage space and computational point-of-view. Whatever calculation is done over such datasets, a large data volume must be taken into account, requiring highly optimized algorithms to be used (figure 1.1).



(a) (b) (c)

(d) (e) (f)

Figure 1.1: Example of a dataset from combustion simulation with time-varying scalar values and dynamic geometry. Figures (a)-(c) show changing boundaries of the simulation domain at the time steps 1, 60, 88, while (d)-(f) show changing mesh.

1.3 Principles and methods

There are two basic phases of the methods for isosurface construction:

1. *Isosurface extraction.* During the extraction phase, the cells intersected by the isosurface are identified.

2. *Isosurface visualization.* Once the extraction is done, various techniques can be employed to calculate the isosurface geometry within intersected cells and to render its shape.

This thesis focuses on the first phase. Based on the data given, different algorithms for isosurface extraction were developed. The focus of these algorithms ranges from low extraction complexity, through space optimization, to I/O effectiveness while handling large volumes of data.

Before the existing methods will be described, an overview of the basic principles is provided in the following paragraphs.

Extremal values. Most of the existing techniques for isosurface extraction are based on manipulation with extremal values of the cells. Under the term *extremal values of a cell* we understand a pair made of *minimum* and *maximum* of all values defined within the cell (typically the values assigned to the points at cell's corners). In the rest of this thesis, we will call it a *min-max* pair of a cell.

Active cells. Traversing through the min-max pairs of the cells in the dataset allows for identification of the *active cells*. An active cell c is a cell intersected by the isosurface for the given *isovalue*; mathematically:

$$c_{min} \leq isovalue \leq c_{max} \tag{1.1}$$

Isosurface extraction. Existing isosurface extraction techniques are based on the organization of the min-max pairs of the cells in some kind of data structure. Transversing such data structure in a defined way allows for identification of the active cells.

Methods. The basic method for identification of active cells is simple sequential traversal of all cells in the dataset. The test is made for each cell, whether a min-max interval of the cell covers given isovalue. This method called *Marching Cubes* has been introduced by Lorensen and Cline in 1987 [34]. In their original paper Lorensen and Cline introduced 15 ways of how a cubical cell can be intersected by an isosurface. Using the scheme provided, geometry of the isosurface can be easily constructed within the active cells and rendered out.

Many researchers inspired by this seminal work tried to improve and extend it in many ways. The main drawback of the Marching Cubes is its high computational cost. All cells in a datasets have to be visited and tested. Number of methods have been introduced to avoid this costly computation and to decrease the time needed for extraction (chapter 2).

Advantage of the min-max based techniques for isosurface extraction is that they are applicable on both regular and unstructured static datasets, regardless of the cells shape. Only min-max pair of each each cell is considered during the extraction phase. Geometry of cells is taken into account just in the visualization phase.

The next step in the isosurface extraction is represented by the techniques able to handle time-varying data. Most of the research in this area focuses on the datasets with static geometry.

The simplest method for time varying data is to build auxiliary data structure separately for data at each time step. This approach, though working, is time and space inefficient.

In order to optimize the isosurface extraction existing techniques incorporate the time dimension into their supporting data structures. The focus of the available techniques ranges from space optimized data structures to I/O efficient mechanisms designed for large data volumes. Section 3.4 provides overview of these techniques.

Recently the computational power and data storage space increased up to the level that can be considered as reasonably sufficient for the numerical simulations in the domains with moving boundaries and inner parts. These simulations closely reflect reality in many applications in the automobile and the aerospace industry.

Despite growing size and complexity of this kind of simulations there is still lack of the suitable methods for isosurface extraction. The techniques which count with static geometry of the dataset along its time dimension fail for the dynamic geometry data.

The second half of this thesis is dedicated to the overview of the dynamic meshing (chapter 3). The techniques for construction of dynamic mesh are studied as a base for further research. Overview of the few existing methods for isosurface extraction from such kind of datasets is provided. Then our contribution to this field is described and analyzed.

1.4 Goal and contribution

Our main aim is to study properties of the time-varying datasets with static and dynamic mesh geometry and to find a transformation of such data into a space, which allows for isosurface extraction in the space and time optimized way.

The author's contribution to the field of isosurface extraction is threefold:

1. *Space and time optimized isosurface extraction from static datasets*

   The main drawback of the existing methods for isosurface extraction from static datasets are their high space requirements and long construction time of the data structure that aids in the process of the extraction of active cells. Some of the existing methods require the size of memory as much as three times larger than the size of input dataset.

   In the section 2.6 a novel method is introduced which decreases the memory space required down to around 1.1 times of the size input dataset (in the average case). Considerable improvement are also achieved in shortening the time needed to build auxiliary data structure.

2. *Isosurface extraction from dynamic datasets - geometric approach*

   Lack of the suitable visualization methods for the time-varying datasets with dynamic geometry was the main motivator for the second half of the research presented in this thesis.

   First of our two proposed approaches to the isosurface extraction from the datasets with dynamic geometry is based on the technique to find back geometric correspondence between the mesh cells in the neighboring time steps. The advantages and disadvantages of this approach are discussed in the section 3.6 of this thesis.

3. *Isosurface extraction from dynamic datasets - min/max approach*

   High complexity of our first (geometry-based) solution to the problem led us to the proposal of the min/max-based solution. The cells are represented by their min-max values and organized in the couples in order to achieve lower memory consumption. The technique decreases complexity of the supporting data structure construction. The active cells can be identified at interactive frame rates (section 3.7).

Chapter 2

Static datasets

2.1 Overview

This chapter deals with the optimized ways of isosurface extraction from the static regular and unstructured datasets, a subject of research for more than two decades. The research in the field of isosurface extraction for static datasets has been driven by the effort to minimize the number steps necessary to identify the parts of the input volume intersected by the isosurface. Various techniques have been proposed, many of which are output optimal (meaning, they only need M steps to identify M active cells). However, the results of our research will show that there is still a room for space and time optimization of the supporting data structures as well as for the isosurface extraction process itself.

The content of this chapter is divided into three principal parts. In the first part, the existing methods are studied and evaluated (sections 2.2, 2.3 and 2.4). According to their base principle the methods are divide into three groups:

1. Marching methods
2. Surface growing methods
3. Value-space methods

The second part deals with comparison of the existing methods (section 2.5). The comparison focuses on the space and time complexity of supporting data structures, as well as on the search process for active cells identification. Out of this comparison the main motivation for our research in this field is formulated (section 2.5.4).

In the third part of this chapter, our approach for optimized isosurface extraction is introduced (section 2.6). Formal analysis and practical tests of our method clearly show advantages and improvements over the existing solutions.

2.2 Marching methods

Marching methods described in this section take into account geometric location of the examined cells. There are two ways how this geometric information is treated in order to find active cells. The simplest approach is to visit the cells one-by-one check whether a cell is intersected by isosurface for the supplied isovalue [34]. More advanced approached is to first divide the volume of the input datasets into subvolumes and examine (or skip) each subvolume based on the maximum range of values within it [63].

In 1987 Lorensen and Cline introduced the famous *Marching Cubes* algorithm [34]. Their method is considered as a simples basic method to identify the active cells and extract the isosurface geometry.

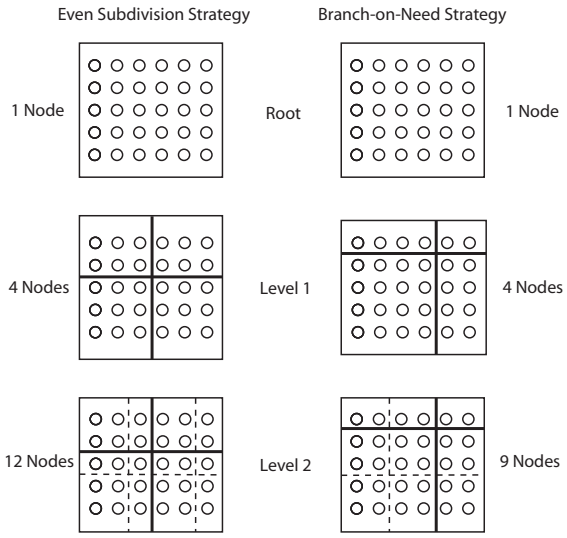
The basic principle of the Marching Cubes is traverse all the cells in the dataset one-by-one. For each cell c a simple test is made whether the min-max interval of the cell covers the supplied isovalue q. In the case that the test $c_{min} \leq q \leq c_{max}$ is positive, the cell e is evaluated as active and added into the list of active cells.

Given a particular isovalue, only a portion of all cells in the dataset is intersected by the isosurface. For this reason, traversal of all cells is unnecessary, making the the Marching Cubes algorithm computationally inefficient.

Wilhelm and van Gelder introduced a *Branch-On-Need Octree* (BONO) [63]. The technique is inspired by the traditional octree subdivision of the volume. BONO technique is more efficient in the terms of number of subdivision nodes. Space of the BONO node is divided by assigning the-largest-possible-power-of-two rows and columns to a lower branch and the rest of the subspace is divided among the other three "quadrants" (in 2D). Figure 2.1 a shows the 2D example of BONO subdivision of regularly placed cells. *Min/max* values of the cells are stored in the nodes of a BONO tree, so only those subtrees satisfying condition min $<$ isovalue $<$ max are traversed during active cells extraction. The time complexity of this method is *O(K + log N/K)*, where K is the number of isosurface cells, and N is the total number of cells.

Majority of the datasets that appear in practice have *smooth* gradients of the contained scalar values without frequent cases of sudden value change in a local neighborhood of the cells. BONO method is well suited for this kind of regular dataset.

The biggest disadvantage of the BONO method appears when it is applied to the high frequency noise data. In this case the subdivision become inefficient because for any isovalue supplied most of the BONO's subdivision nodes have to be visited. Consequently, the search for active cells may be computationally nearly as expensive as a simple marching method of Lorensen and Cline [34].



(a)

Figure 2.1: 2D example of BONO subdivision. The largest possible power of two number of rows and columns is assigned to a lower left branch, and the rest of the space is divided among the other quadrants (picture and description from [63]).

2.3 Surface growing methods

The principle of surface growing methods is to determine starting cell known as the *seed* and trace the isosurface outward to the neighboring cells. These methods rely on knowledge of a sufficient starting set of cells, since growing a surface from a single cell can only capture one component of an isosurface.

Itoh et al. [26, 27, 28] introduced surface growing technique based on the *extrema graph*. The method is based on the following simple rule: 'If there is a closed isosurface, then there exist extremum points both inside and outside of the isosurface. If there is an open isosurface, then the isosurface intersects the boundary of the volume.' According to the above rule, the cells intersected by a closed isosurface are to be found around an inner extremum point, and cells intersected by an open isosurface are to be found on the boundary.

In the pre-process of the extrema graph method of Itoh et al., extremum points are first extracted. Then the extremum points are connected for form a graph. At the same time, boundary cells are registered in a list and sorted according to the minimum and maximum values of their nodes (figure 2.2).

When the isovalue is specified, the cells on the arcs of extrema graph and sorted boundary cells are visited. This step guarantees that at least one cell each isosurface component is found. Having these basic seeds, the isosurface components are constructed using the propagation algorithm.

The total cost of preprocessing is regarded as $O(m^2\ log\ m)$, where m denotes the number of extremum points. The number of cells visited during active cells extraction is regarded as $O(n^{2/3})$. The time required for preprocessing can be significant in the case of high frequency noise in the data, causing high m values.

In 2001 Itoh et al. proposed *Volume Thinning* method [29] for construction of the extrema graph. The method, originally used in the image processing, visits all the cells in the datasets and eliminate those that are unnecessary for creation of the extrema skeleton. The skeleton created, retains topological features in the volume like holes or voids.

Another solution has been found in the work on *Contour trees* [9, 55]. A Contour tree is a structure that summarizes all possible contours on the map (figure 2.3). In [9] Carr et al. provide algorithm for constructing so call *Join tree* and *Split tree* and finally merge these two into a final Contour tree.

In 1997 van Kreveld [56] et al. introduced algorithm for construction of a *Seed set*, which contains the cells intersected by the isosurface components (one seed per component).

Bajaj et al. [4] provide three algorithms, first, for constructing nearly optimal seed set, minimizing the storage overhead for the search structure. Another two algorithms focus on extremely fast Contour tree computation suitable for large datasets that cannot be kept in main memory (out-of-core computation).

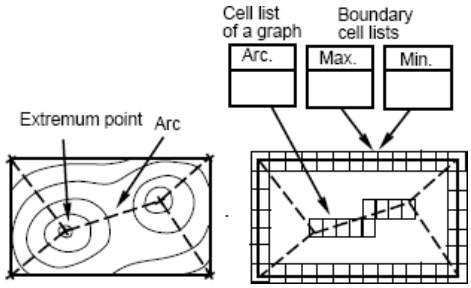


Figure 2.2: Example of the 2D extrema graph and boundary cell list [29].

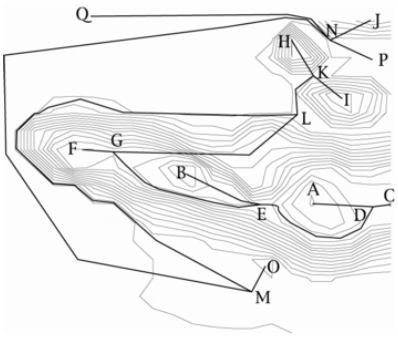


Figure 2.3: Example of contours and contour trees (source: web page of H.Carr at www.csi.ucd.ie).

2.4 Value-space methods

The methods presented in this section differ from the marching methods (section 2.2) in the way how the a single cell is represented. A triple (*ID*, *min*, *max*) is used to represent a single cells. Cell's ID is a unique cell identifier defined when the cell is created, allowing to access cell without any information about the cell's geometry (consecutive cells' IDs don't necessarily implies their geometric adjacency). A couple of scalars *min*, *max* then specifies minimum and maximum scalar values defined within the space of the cell.

The (*ID*, *min*, *max*) representation of cells is well suited for both structured and unstructured datasets, because it is independent of geometry of the cells. There are three kinds of methods that use this representation:

1. Methods which organize the id,min,max triplets in so-called span-space and then use some data structure to divide and index the span-space [32, 46].

2. Methods which sort and group the cells into lists and define a way how to traverse this list in order to extract active cells [19, 47, 11, 57, 60].

3. The method of Bordoloi and Shen [7] doesn't fit into none the two above mentioned categories. Their method first transforms the *min*, *max* values into an alternative 2D space, which is then used to construct data structure that aids during active cells extraction.

The next paragraphs provide description of the major methods from each of the three groups listed above.

The notion of *Span-space* has been originally introduced by Livnat et al [32]. In the Span-space the cells are represented by their minimum (x-coordinate) and maximum values (y-coordinate). Active cells are identified by the simple restriction of Span Space along min and max axis. Figure 2.4 illustrates the process of isosurface extraction using Span Space (the area of active cells for isovalue q is enhanced).

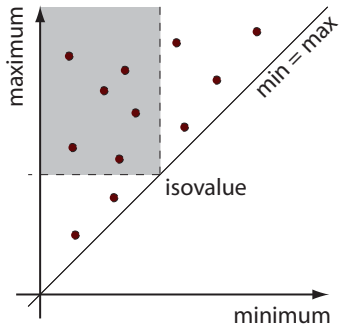


Figure 2.4: Span-space. Each black dot represents one cell.

Livnat et al. originally proposed kd-tree for spatial subdivision of Span-space in the *NOISE* algorithm [32]. Kd-trees designed by Bentley in 1975 [6] are basically multidimensional binary search trees. Each node of the tree holds one of the data values and two subtrees as children. The subtrees are constructed so that all nodes in left/right subtree holds values lower/higher than the parent's node value. The parent's node value which is computed as a median of values in the left and right subtree. The points, representing the active cells can be extracted by simply traversing kd-tree with worst case time-complexity of $O(\sqrt{N} + K)$, where N is total number of the cells in the dataset and K represents the number of isosurface cells. Figure 2.5 shows span-space partitioned by the nodes of Kd-tree.

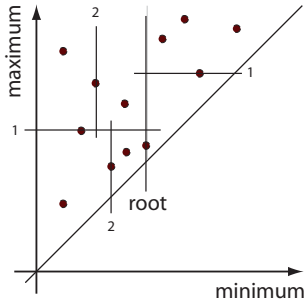


Figure 2.5: Kd-tree. The lines represent the structure of kd-tree. The *root* line represents the first split of along the min coordinate.

Shen et al. proposed lattice subdivision of Span Space in the *ISSUE* methods [46]. A simple two-dimensional regular lattice is used to subivide Span Space into LxL regions. ISSUE lowers the time complexity of the active cells to $O(log(\frac{N}{L}) + \frac{\sqrt{N}}{L} + K))$, where N is total number of the cells in the dataset, L is the user specified parameter defining the number of lattice elements along each axis and K represents the number of isosurface cells. Figure 2.6 shows lattice subdivision of Span Space.

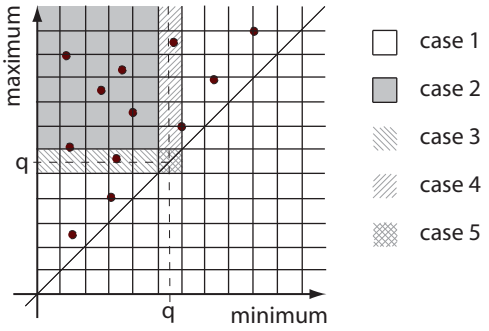


Figure 2.6: Span Space subdivided by LxL regular lattice. Given the isovalue q, lattice elements can be classified into five different cases.

The two previous methods, NOISE and ISSUE belong to the first of three groups mentioned at the beginning of this section. The next 5 methods described below belong to the second group, focusing on sorting of the cells into lists and their efficient traversals for active cells identification.

Gallagher [19] designed a *span filter* to optimize the performance of the isosurface extraction. Initially, the range of the scalar values in the dataset is subdivided into several subranges termed buckets. The number of buckets that a cells scalar values cross is defined as the span length. Cells are then distributed into different span lists according to their span lengths. Within each span list, the cells are further grouped into different buckets based on their lower bounds.

During extraction for a given isovalue, the algorithm examines each span list. Within each span list, buckets that have bounds at and lower than the isovalue, depending on the span length of the list, are retrieved, and the elements inside are visited. Figure 2.7 show span filter structure with four span lists.

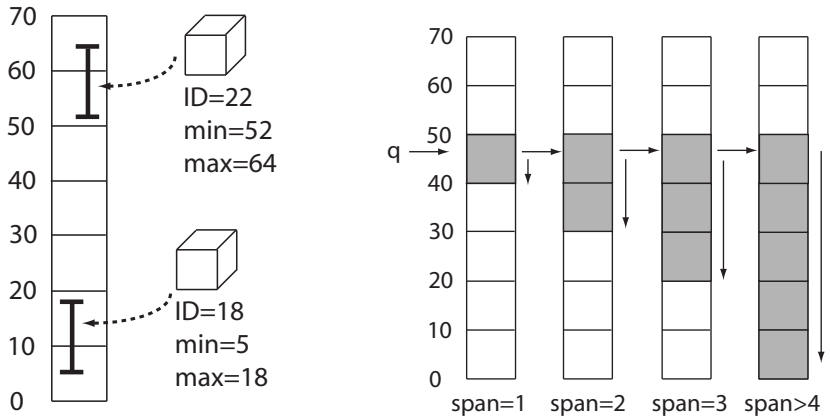


Figure 2.7: (a) Span range and buckets. Two cells with span length = 2. (b) Span filter data structure with four span lists. Search sequence for isovalue q (marked with arrows) starts at the bucket with the lowest span length and continues to the once with higher span length until all span lists are visited.

Shen and Johnson introduced *Sweeping Simplices* technique [47]. The technique uses a cell list division scheme, which assigns the cells into the groups at various level according to the min/max values. For each subgroup the minimum list (cells sorted by the min values) and the corresponding sweeping list (cells sorted by the max values) is constructed. During the active cells extraction, the minimum lists and sweeping lists are restricted by the selected isovalue. Then the efficient comparison scheme is used to quickly extract only the cells in both restricted lists, which are in fact the active cells.

The *Interval Tree* technique [11] guarantees worst-case optimal efficiency. Cells, represented by the *min*/*max* intervals are grouped at the nodes of balanced binary tree. Each node holds discriminant *d*. A list of the cells $I = c_i, i = 1..N$ at each node is divided into three groups: 1) $c_{i_{min}} \leq q \leq c_{i_{max}}$, 2) $c_{i_{min}} \geq q$ and 3) $c_{i_{max}} \leq q$. The cells from the second and third group belongs to the left and right subtree of the node while the cells from the first from the first group are kept in the node sorted in two lists according to their *min* values (AR list) and *max* values (DR list). Figure 2.8 shows example of the interval tree and stored intervals. For any isovalue query, at most one path from a root node down to a leaf node is traversed. During traversal, the AR and DR lists of the nodes along the path are traversed for active cells identification.

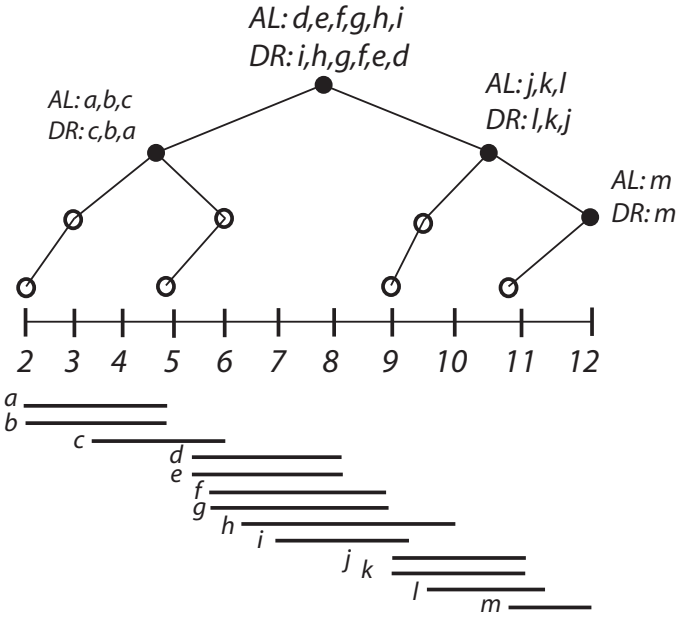


Figure 2.8: Interval tree built over 13 cells. White circles represent nodes with empty AR, DR lists.

Span-triangle data structure was introduced by von Rymon-Lipinski et al. [57]. Span-triangle is focused on isosurface extraction from medical volume datasets, sampled as either 8 bits or 16 bits integers, moreover cropped to an exploration range $[e_{min},\ e_{max}]$.

Skeleton of Span-triangle structure is base array. Each element in the base array corresponds to the cells x with *min* value $b = x_{min} - e_{min}$ and contains pointer to its span array and cell info array. An element of span array at position s holds pointer to the first item with value span $s = x_{max} - x_{min}$ in a cell info array. Figure 2.9 shows example of Span-triangle structure for exploration range $[e_{min},\ e_{max}] = [0, 3]$ and isovalue $v = 2$. Using fast Radix sort algorithm, the structure can be constructed in $O(\mathrm{N})$ time and takes $O(\mathrm{N})$ space.

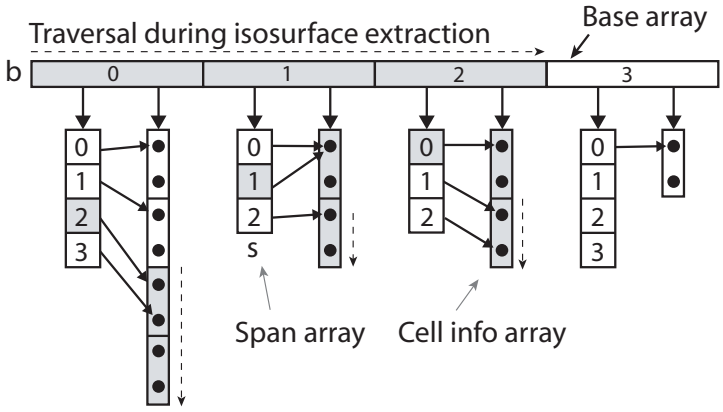


Figure 2.9: Sample of the span-triangle data structure for exploration range $[e_{min},\ e_{max}] = [0,\ 3]$ and isovalue $v\ =\ 2$. Active element of the data structure for isovalue $v\ =\ 2$ are emphasized by a gray background.

Waters et al. [60] use *fixed-sized buckets* to divide the list of cells in a scalar field. A record for one cell includes a cell's *min*/*max* and *ID* values. List of the cells is first sorted according to the minimum values. Such sorted list is then split into intervals of the same size B - buckets. Figure 2.10 illustrates organization of the cells in the *min/max* space and their assignment to the buckets. Similarly to Span Space the active cells are identified by restriction of the 2D space along the *min* (x) and *max* (y) axis. Buckets are traversed along the *min* (x) axe and tested for $y_{cell} = max_{cell} < max$ condition. The time complexity for creating the list is $O(N\ logN\ +\ \frac{N}{B}(B\ logB))$ and for identifying the active cells $O(K\ +\ B)$. Space complexity is $O(N)$.

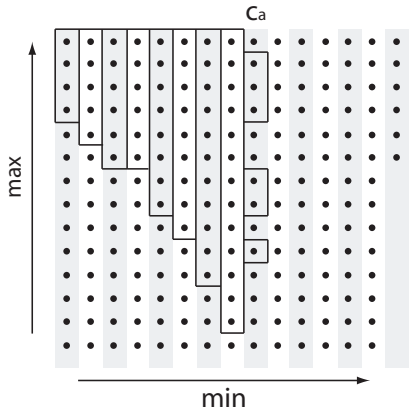


Figure 2.10: Isosurface extraction using fixed-sized buckets. The cells of scalar field are organized in a space similar to Span space and divided into the buckets of equal size. Parts of the buckets containing the active cells are emphasized by the black boxes.

Bordoloi and Shen [7] introduced the space efficient technique for isosurface extraction from large scientific datasets. The original mesh cells are first represented as the points in 2D *UV-space* (figure 2.11). U coordinate of the cell C is determined as $u = C_{maximum} + C_{minimum}$ and V coordinate as $v = C_{maximum}$ - $C_{minimum}$. Then the U and V axis are quantized, which results into a set of rectangular regions within UV-space. Given an isovalue, the decision boundaries for UV-space are computed. The cells stored in the rectangular regions within such decision boundaries are selected for extraction of isosurface geometry. Due to the quantization of the U and V axis the min/max values of the cells within each rectangular region of UV-space do not need to be saved. Only the space for cells *ID*s and the space or representing rectangular regions in UV-space are required.

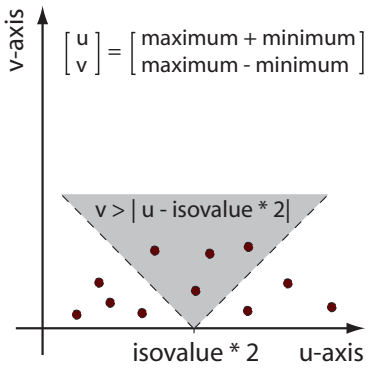


Figure 2.11: UV-space. Each cell is represented as a point with coordinates (u, v). Isosurface passes passes through the cells in the region $v \geq$ —u - *isovalue* * 2— (shaded).

2.5 Analysis of the existing methods

2.5.1 Space complexity

This subsection discusses space requirements of the methods described in the previous sections 2.2, 2.3 and 2.4. In the following, we assume that there are N cells in the dataset, ID of each cell is represented by c bytes number and minimum and maximum value of a cell need d bytes each. For example: a single triple (ID, min, max) needs $c + 2d$ bytes of memory.

Marching Cubes method traverses the input cells by simply reading the input dataset. Because no auxiliary search structure is built, the space requirements of the Marching Cubes are equal to the storage space needed for input dataset only.

Wilhelm and van Gelder in the original paper on BONO method calculate the ratio of BONO nodes against data points to be $\leq \frac{1}{6}$ (0.1615 exactly) for the regular volumetric datasets whose side contains at least 32 cells.

The NOISE and Fixed-sized buckets techniques store the (ID, min, max) record once for each cell. Thus, the run-time space requirements of the NOISE and Fixed-sized buckets are $(c + 2d)N$.

The ISSUE and Interval tree use two records for each cell: (min, ID) and (max, ID). Therefore, the ISSUE and Interval tree require storage space equal to $(2c + 2d)N$.

The space-efficient method of Bordoloi and Shen [7] first transforms the cells' extremal values into the 2D UV-space and then applies the quantization to divide the uv-space into a finite set of M x L buckets. The method requires the storage space for N cell IDs and $ML + L + 1$ quantization levels, where $ML + L + 1$ is typically equal to $N/100$. Table 2.1 summarizes space requirements of the selected methods.

Method	Space requirement	Space complexity
ISSUE	$4N$ + kd-trees at min=max	$O(N)$
Interval tree	$4N$ + tree overhead	$O(H + I)$
Fixed-sized buckets	$3N$ + min dictionary	$O(N)$
Quantized search	$N + ML + M + 1$	$O(N)$

N = # of cells H = # of interval tree nodes
I = # of distinct intervals in data M, L = # of quantization levels

Table 2.1: The space requirements of the selected methods.

2.5.2 Time complexity

There are two phases of the active cells search: construction of the auxiliary search structure and the search pass itself. Both phases are discussed in this subsection.

The most time-consuming parts of the construction phase are the sorting steps. The ISSUE method [46] requires two sorting passes for each non-empty lattice element (Row and Column data structures). The Interval tree method [11] needs to sort AL and DR lists of cells within each tree node. The Fixed-sized buckets method [60] sorts all cells by minimum value in $O(N\ log\ N)$ time, and then resorts the cells bucketwise by maximum value in $O(B\ log\ B)$ time for each of the $\frac{N}{B} + 1$ buckets (where B is the number of cells per bucket). In the Quantized search method [7] all cells are first sorted by their u value, then the u axis is quantized into M intervals and cells are sorted for the second time within each u-interval.

During the search phase K active cells are identified for a supplied isovalue. To identify K active cells a certain number of search structure elements E must be visited.

For Marching Cubes method $K = E$. Therefore, optimized methods were proposed. In generally all the other method provide nearly optimal search times almost proportional to the number of active cells K. Table 2.2 summarized time complexity of the extraction phases for a selected methods.

As can be seen from the table 2.2 the best search times can be theoretically achieved by the Bucket Search, because very few elements need to be visited and all the cells within visited elements are put into the list of active cells. However, this approach provides results with certain search error.

Method	Time complexity
Marching Cubes	$O(N)$
BONO	$O(K + \log(N/K))$
NOISE	$O(\sqrt{K} + K)$
ISSUE	$O(log(\frac{N}{L}) + \frac{\sqrt{N}}{L} + K))$
Buckets Search	$O(K + B))$
UV Search	$K + Q + err$

L = number of lattice elements along each axis
B = number of buckets
Q = number of visited quantization elements
err = search error (number of false positives)

Table 2.2: Time complexity of the active cells search of the selected methods.

2.5.3 Suitability and accuracy

The last criteria are suitability and accuracy. It might be misleading to judge described methods only by their time and space complexity. Marching Cubes and BONO methods are suitable (without any auxiliary data structure) only for regular (structured) volumetric datasets. On the other hand, the family of value-space methods is suitable for both structured and unstructured datasets because geometry doesn't play any role in the search structure.

All described methods are able to provide exact list of the active cells, except the UV-Search methods. UV-Search methods guarantees to provide all the active cells together with certain search error (small percentage of false positives). However, this search error is balanced with small space requirements and short search times.

2.5.4 Conclusions from analysis

All of the value-space methods summarized above report space complexity $O(N)$. However, in practice situation is different. If the cell IDs are represented by 32-bit integers and min-max values by 32-bit floats, then the ISSUE and Interval tree applied to a 512^3 regular floating-point dataset require approximately 2GB of memory only for search structure, which is four times the the size of the initial dataset. This high space requirement, makes the two mentioned methods unusable for moderate size 512^3 datasets on most of todays desktop computers (with 2GB of memory and 32 bit operating system).

Our first conclusion from the analysis above is, that there is still room for space optimization of the isosurface extraction methods.

As mentioned in the section 2.5.2, the most time-consuming part of construction of the search structure are sorting steps. All of the presented methods need to sort the $\{min, max\}$ pairs according to both extremal values - this means, that they require at least $2 * O(N\ log\ N)$ steps. Out of this we make our second conclusion:

There should exists a 1D space S, and such transformation $\{c_{min}, c_{max}\} \rightarrow S$ for all cells c in the dataset, that the active cells identification will be possible directly in S.

Coming out of the $R^2 \rightarrow R$ transformation of the cells' extremal values proposed in the previous paragraph, it is obvious that part of the original information has to be lost. Therefore, the resulting set of the active cells identified over S will include certain number of false positives (search error). The accuracy with which a search structure would be built over S is crucial for minimizing of the the search error. In order for such search method to be efficient, the search error should be in practice smaller than the error of the UV-search method of Bordoloi and Shen [7].

2.6 Proposed method

2.6.1 Overview

This section describes our newly proposed method for isosurface extraction. Motivation for this research came out of the conclusions made in the previous section 2.5.4.

The method is based on a simple transformation of the {minimum,maximum} information of each cell into a 1D space. The proposed transformation lowers the worst-case space requirement to $2N$. Construction of the search structure over 1D data requires only one initial $O(N\ log\ N)$ sorting step, which shortens and

simplifies the preprocessing compared to the existing methods.

Our algorithm achieves the near-optimal search times. The number of false positives during the active cell search (search error) decreases significantly compared to [7].

Once identified, the active cell IDs are transferred into our point-based visualization system. A range of point-based techniques has been developed for isosurface rendering [30], [21], [12], [42], [13], [8], [33]. We have adopted a technique similar to Iso-splatting [13] for interactive focus+context [58] visualization accelerated by modern GPU hardware. For high-quality isosurface rendering we use the method of Brentzen and Christensen [8].

2.6.2 Transformation

The main idea of the transformation is to convert the {minimum,maximum} pair of each cell into the parameter t (denoted t_c for cell c):

$$\{c_{min}, c_{max}\} \rightarrow t_c, t_c \in [0,1] \tag{2.1}$$

The transformation (Eq. 2.1) is done by quantization of the max-axis of the Span Space [32] to the M quantization intervals. M is the parameter to our method and its choice is explained later in the Sec. 2.6.5.

After the max-axis quantization the Span Space becomes a finite set of parallel horizontal lines. Parameter t from the transformation, is equal to 0 at the beginning of the bottom-most line, and is equal to 1 at the right end of the top-most line. The t-interval per one max-axis quantization interval is $t_{int} = 1 \ / \ M$. Figure 2.12 shows the span space turn into finite set of the quantized intervals along the *maximum* axis.

Considering the quantization described in the previous paragraph, the parameter t_c of a cell c is computed as follows:

$$t_c = t_i + t_0 \tag{2.2}$$

where:

$$t_i = \lfloor \frac{c_{max} - max_{min}}{max_{max} - max_{min}} \ / \ t_{int} \rfloor * t_{int} \tag{2.3}$$

$$t_0 = \frac{c_{min} - min_{min}}{min_{max} - min_{min}} \ * \ t_{int} \tag{2.4}$$

Such a 1D index is used for a fast construction of the search structure and for the identification of active cells. As can be seen, the transformation does not handle both extremal values in a symmetric way (only the max values are quantized), which results in the small search error rates of the method presented.

Once the parameter t is computed for each cell of a dataset, the cells are sorted by t in increasing order. Finally, a list of records is constructed. Each record contains parameter t and a list of IDs of the cells, which have this value

of parameter t. Since the list of cells is sorted by the t parameter, the records can be created by simply traversing the list of cells and grouping the cells with the same parameter t into the same record. An index of the first record on each max-axis quantization interval is placed into a simple search dictionary which helps during the active cell search.

2.6.3 Extraction

The goal of the extraction phase is to identify all active cells. For a supplied isovalue q, a cell c is defined to be *active* if: $c_{min} \leq q \leq c_{max}$.

Active cells for the supplied isovalue are collected by traversing the max-axis quantization intervals in the top-bottom order using two nested loops. The outer loop traverses the items of the search dictionary to determine the index of the first record on the current quantization interval. The inner loop collects the active cells from the records in the current quantization interval, until the condition $t_c \leq t_{limit}$ holds. The value of t_{limit} for the n-th quantization interval is computed as follows:

$$t_{limit} = (n * t_{int}) + \frac{q - min_{min}}{min_{max} - min_{min}} \; * \; t_{int} \tag{2.5}$$

The index of the last traversed quantization interval, "*final*", is determined by the selected isovalue q:

$$final = \lfloor \frac{q - max_{min}}{max_{max} - max_{min}} \; / \; t_{int} \rfloor \tag{2.6}$$

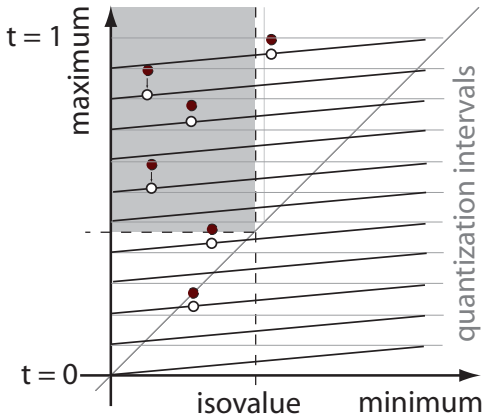


Figure 2.12: Span Space quantized along the *maximum* axis. Filled circles represent the original cells (min,max pairs). Empty circles show transformation of the original cells (represented by t value). Shaded area contains the active cells for given isovalue.

2.6.4 Incremental search

For small changes of isovalue q, it is often efficient to extract the active cells *incrementally*. For this purpose, we keep list L of the last visited record on each quantization interval. The intervals between the topmost one and the $final_{new}$ are marched from the record with index stored in L, activating the cells until the first inactive record is met. Cells from all the quantization intervals between $final_{old}$ to $final_{new}$ are deactivated.

Similarly, when the isovalue is decreased to q_{new}, the quantization intervals are traversed backward from the position stored in L, deactivating the cells passed along the way to the new value of $limit_n$. Additionally, full extraction has to be done for the quantization intervals $final_{old}$ to $final_{new}$.

2.6.5 Number of quantization intervals

As stated in Sec. 2.6.2, the number of max-axis quantization intervals is an optional parameter to the method presented and depends on the data type of the processed dataset.

For datasets with byte data, it is sufficient to quantize the max-axis of the Span Space into 256 intervals to recognize clearly each possible maximum value. In other words, the $\mathcal{R}^2 \rightarrow \mathcal{R}$ transformation (Eq. 2.1) does not quantize the maximum values of cells (i.e. it preserves the original {minimum,maximum} information), and the active cells can be extracted with zero search error. The same is the situation for 2-byte integer datasets and the search structure with 65536 quantization intervals.

The number of quantization intervals for the floating-point data has been determined experimentally. The relationship between the chosen number of quantization intervals and the search error has been observed. In all performed measurements, the search error drops under 0.3% for more than 2^{16} quantization intervals, regardless of the input dataset (Fig. 2.19). Thus, for floating-point data we use 2^{16} quantization intervals. Comparison of the search error with the method [7] is provided in Sec. 2.7.2.

2.6.6 Search error

The search error is represented by the number of false positives that appears during active cells identification. In our method the false positives may appear when the last (the lowest) quantization interval is traversed for gathering the IDs of active cells.

Within this last interval the method can't check whether the original *max* value of an investigated cell is higher or lower than the supplied isovalue because the original *max* value has been quantized (this is where the part of information is lost). Figure 2.13 shows the area where the search error my appear.

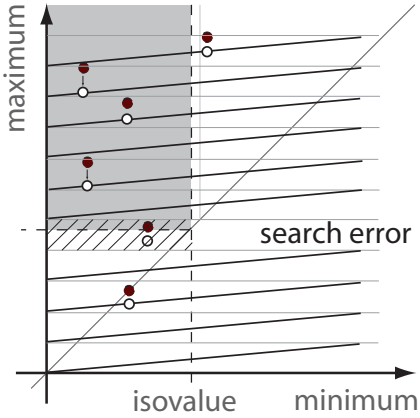


Figure 2.13: Span-space turned into finite set of quantization intervals along the *maximum* axis. The area of the lowest *max* quantization interval where the search error may appear is crosshatched.

2.6.7 Optimization

For floating-point datasets the number of different t parameters of the transformed cells can be very large, which may results in a large number of records. In the worst case, there is one record created for each cell. Therefore, we employ user-selected level of the t quantization $\triangle t$. Mathematically, the cells of any record $R = (c_0, ..., c_n)$ satisfy conditions: $t_{c_0} \leq t_{c_1} \leq ... \leq t_{c_n}$ and $|t_{c_n} - t_{c_0}| < \triangle t$. All cells of R are further represented by the parameter $t_R = t_{c_0}$.

Let's assume two records R_1 and R_2 on the same quantization interval, so that $t_{R_1} < t_{limit} < t_{R_2}$. Because each cell of R_2 has its $t > t_{R_2}$, R_2 can not contain any active cells with $t < t_{limit}$. Therefore, collecting the active cells from records with $t < t_{limit}$ guarantees that all active cells will be found when quantization is applied, avoiding cracks in the isosurface.

2.7 Comparison

We have implemented the ISSUE [46], Interval tree [11], Fixed-sized buckets [60], and Quantized search [7] algorithms to compare the method proposed against existing methods. The Span-triangle method [57] was omitted from the tests because it is aimed only to quantized data (byte or 16-bit integer).

First, the formal space and time complexities of the presented method are discussed, followed by the results of the tests. The comparison is always made against the method with best results according to give criteria (e.i. extraction time, search structure size, etc.).

2.7.1 Formal complexity analysis

Search structure

The construction of our data structure involves three steps. The transformation of $\{min, max\}$ values into parameter t requires $O(N)$ time, while the sorting pass requires $O(N\ log\ N)$ time. The creation of records is $O(N)$ step. *Thus, construction of our search structure can be done in $O(N\ log\ N)$ time.*

The most time-consuming parts of the construction phase are the sorting steps. The ISSUE, Interval tree and Fixed-sized buckets methods sort minimum an maximum values in two sorting passes. Similarly Quantized search method sorts first u values and then v.

Since there is only one initial sorting step in the proposed method, the construction time is significantly shorter when compared to the current state-of-the-art algorithms for active cell identification. This conclusion is supported by the measurements presented in Sec. 2.7.2.

All of the methods have space requirement equal to at least 3N. The exception is UV-search method [7] which balances the space requirement and search error.

Because our method stores the ID number and t parameter once for each cell, the temporal space required by our method is $2N$ words. However, due to the bucketization of cells into records, the final space requirements of the proposed method are between $1N$ and $2N$, which is competitive with the UV-search (see results in Sec. 2.7.2).

There is also a small space required for the search dictionary, which contains one 4-byte integer for each quantization interval. Thus, the space allocated for the search dictionary is 256 * 4-bytes = 1kB for byte data, and 256kB for 16-bit integer and floating-point datasets.

Active cells search

At run-time, active cells are identified by traversing the list of records. In the worst-case scenario, there is one record created for each cell. In such a case, we need to visit $K + E$ records, to extract K active cells. For each record visited, one comparison of its t_R parameter is performed. For each traversed quantization interval one check of the search dictionary and one failed t_R comparison are performed. E is the search error introduced by the small amount of records visited in the last quantization interval, which do not contain active cells. Because E is very small in practice (under 0.3%), the run-time complexity of our search algorithm is $O(K)$.

In a typical case, many records contain more than one cell. Therefore, to extract K active cells we usually need far less than K comparisons, which shortens the search process.

2.7.2 Search structure tests

The analyses from Sec. 2.7.1 are supported by the results of measurements. Tab. 2.3 summarizes seven datasets used during tests. The tests for Vertebra and $X^2 - Y^2$ datasets were done on the 64-bit Intel processor and 4GB of RAM. The tests for all other datasets were done on a desktop PC with Intel 3.2GHz processor, 2GB of RAM and ATI FireGL V5200 graphics adapter.

Dataset	Resolution	Description
Skull	256x256x256	hexahedral grid, byte
CT-head	256x256x113	hexahedral grid, 16-bit integer
FiveJets	128x128x128	hexahedral grid, 32-bit float
TeraShake	750x375x100	hexahedral grid, 32-bit float
Isabel	500x500x100	hexahedral grid, 32-bit float
Vertebra	512x512x512	hexahedral grid, 16-bit integer
$X^2 - Y^2$	512x512x512	hexahedral grid, 32-bit float

Table 2.3: Summary of datasets used during tests

Figure 2.14 provides a comparison of the construction time of the presented data structure versus four tested methods. For testing purposes, the number of quantization intervals for our method has been determined according to the data type of a dataset (Sec. 2.6.5). For the Fixed-sized buckets [60] the fixed bucket size of 8192 cells has been used as recommended by the authors of the method. For the Quantized search the (M,L)=(2000,200) quantization levels were constructed, because at this setup it achieves the run-time performance similar to our method (see Figures 2.16, 2.17, 2.18). As predicted by the analysis of the construction complexity, our search structure achieves the shortest construction times of all tested methods. Construction times provided by Fig. 2.14 include creation of the temporary and final data structure without data loading.

Figure 2.15 compares the size of the final data structures. Because we do not compress the cell IDs, the size of our search structure is always greater than $1N$ words. We need additional space to store the t_R parameter of each record, and the space for the search dictionary. Thus, the total space required by our method is between $1N$ and $2N$ words for the tested datasets. Only the Quantized search [7] achieves a comparable size of data structure. However, our method achieves much better construction times than the Quantized search and lower search error.

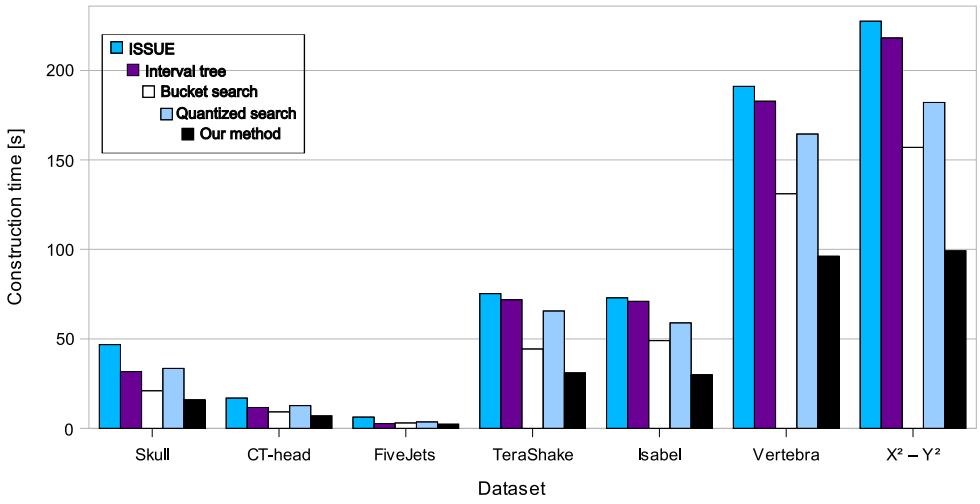


Figure 2.14: Comparison of the construction times. As predicted by the analysis of construction complexity, the search structure proposed achieves the shortest construction times of all tested methods.

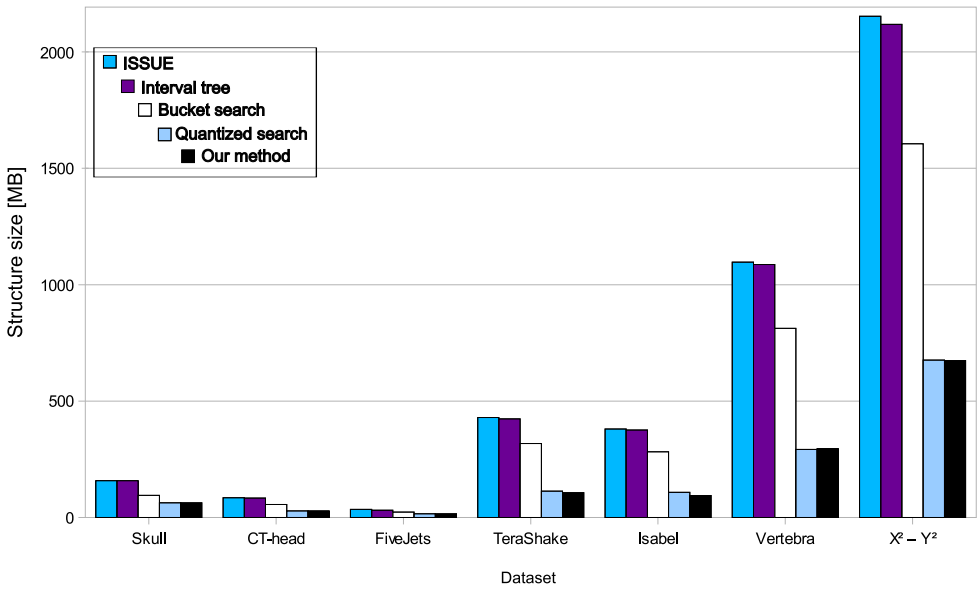


Figure 2.15: Comparison of the size of the final data structures. Only the Quantized search achieves size of the data structure comparable to our method, but at the cost of almost twice as long construction time (Fig. 2.14) and higher search error.

2.7.3 Extraction times test

The measurements of the search times are provided in this subsection. The search times of the presented method are compared against the Interval tree and the Quantized search with three different setups of quantization levels. The Interval tree method has been chosen because of its reported near-optimal search time $O(K + log\ N)$, where K is the total number of active cells. The Quantized search has been included into the measurements because, as well as the method presented, it uses quantization of the data to shorten the search time. For each dataset, 1000 isovalues from the value range of the data were randomly chosen. For each chosen isovalue, the average search time for 100 full extraction queries was recorded.

Figures 2.16, 2.17, 2.18 show that our method outperforms the Interval tree. In fact, the search times of our method are comparable to those of a Quantized search with M=2000 and L=200 quantization levels. However, as will be shown later in this section, the search error of the Quantized search for (M,L)=(2000,200) is significantly higher when compared with our method.

Note that the authors of the Quantized search method [7] do not provide any specific recommendation for the optimal setup of M and L parameters, while the value of the optional parameter in our method (the number of max-axis quantization intervals) is exactly given by the data type of the processed dataset (see Sec. 2.6.5).

Figure 2.19 shows that the search error decreases with an increasing number of quantization intervals. We used three floating-point datasets with a different value range. For all three datasets, the search structures with 2^{10} to 2^{19} quantization intervals were constructed. For each constructed data structure, the average search error for 200 full extraction queries (with random isovalues from the value range of the data) was recorded. The results indicate that the average search error remains below 0.3% for a search structure with more than 2^{16} quantization intervals, regardless of the input dataset.

Finally, Tab. 2.4 provides exact measurements of search error for floating-point datasets and selected isovalues. Note that the search error of our method is zero for byte and 16-bit datasets, because each possible maximum value is covered by one quantization interval; thus, there is no quantization of the input data at all. The search error of the Quantized search method has been measured for various quantization levels ranging from (M,L)=(500,50) to (4000,400). As can be seen, our method achieves much lower search error, even compared to the Quantized search with M=4000 and L=400.

Additionally, a C++ pseudocode is provided in the appendix B of this thesis to facilitate the implementation. Appendix A provides details about the test datasets and the isosurfaces extracted using the method proposed.

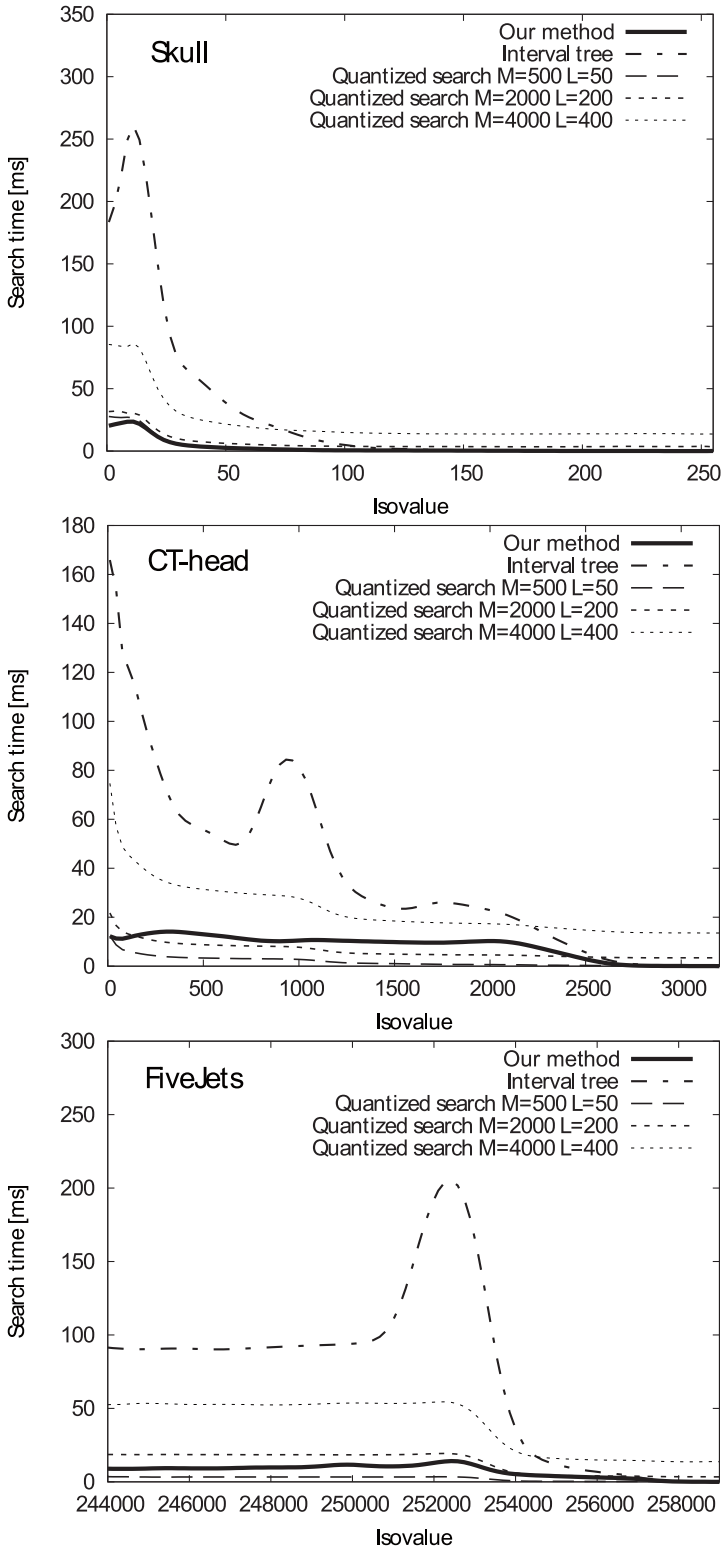


Figure 2.16: Comparison of the search times. The plots show that the search times of our method are comparable with Quantized search with M=2000 and L=200 quantization levels. As mentioned in Sec. 2.7.1 the run-time complexity of our proposed method is $O(K)$.

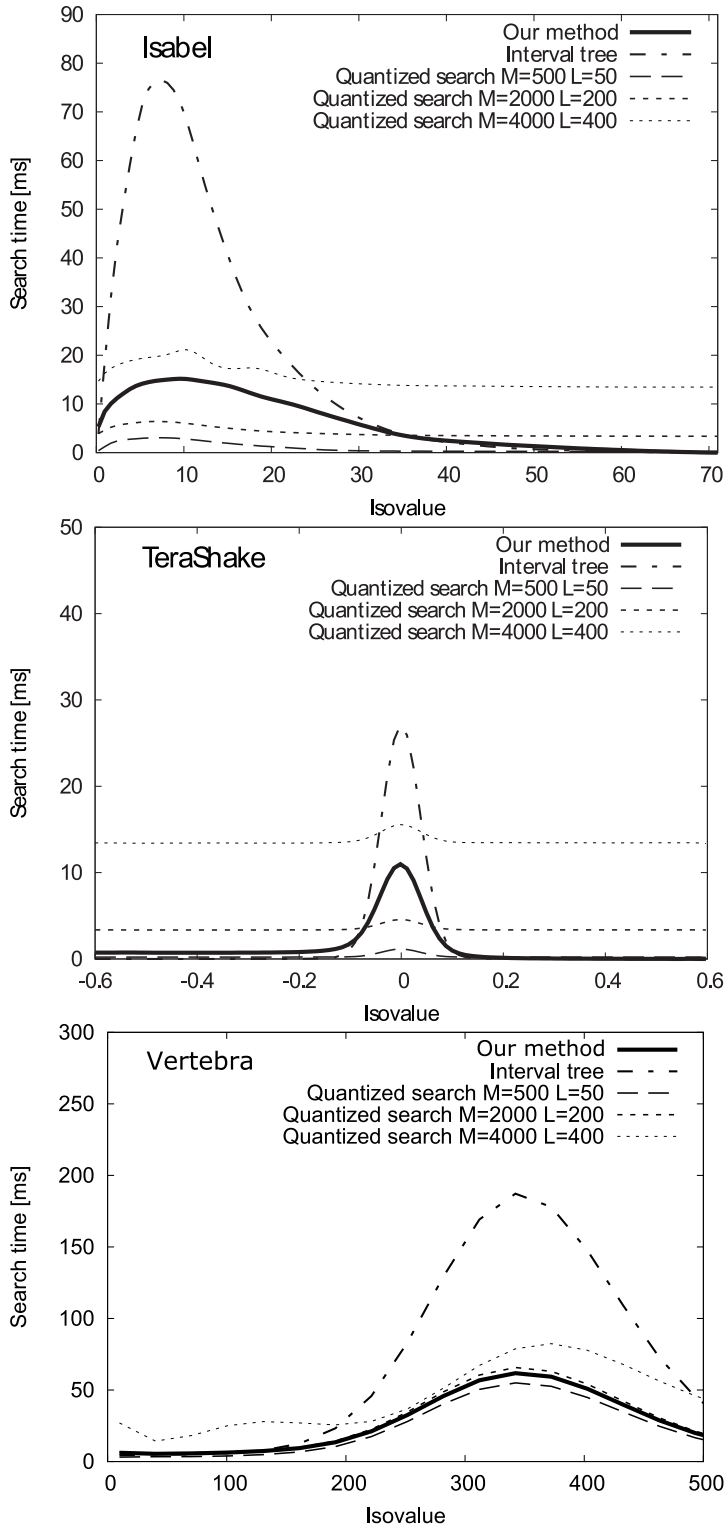


Figure 2.17: Comparison of the search times (part 2).

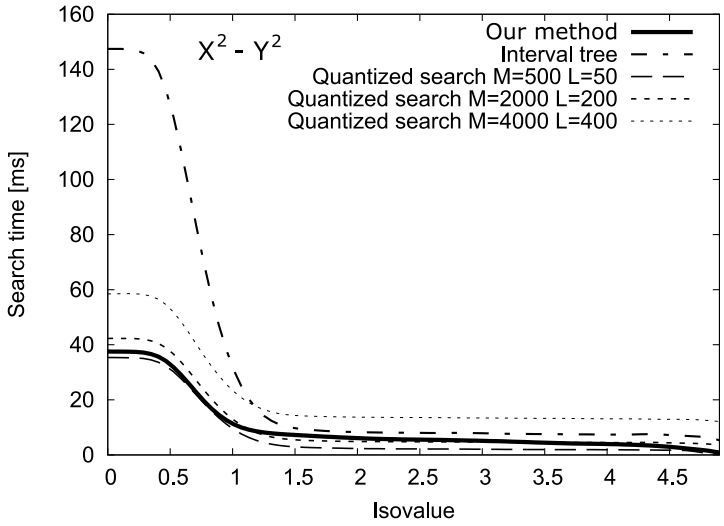


Figure 2.18: Comparison of the search times (part 2).

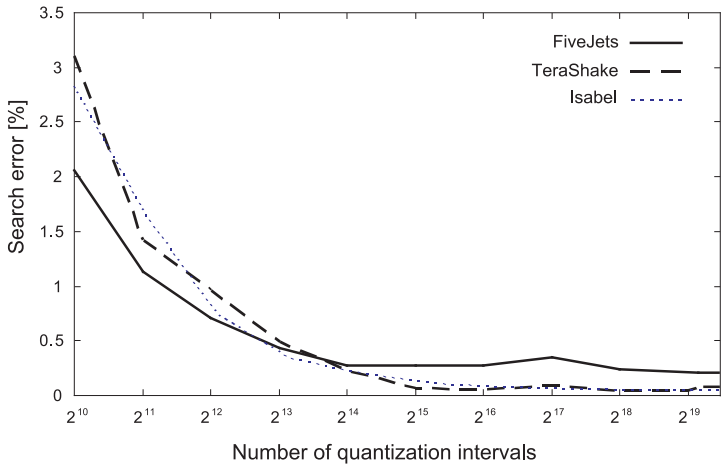


Figure 2.19: Plot of the search error versus number of quantization intervals for floating-point datasets. The results show that the search error falls below 0.3% for search structure with more than 2^{16} quantization intervals, regardless of the input dataset.

Dataset / Isovalue	Quantized search [%]			Our method [%]
	M=500 L=50	M = 2000 L=200	M = 4000 L=400	
FiveJets / 253947.5	1.81	0.86	0.97	0.025
TeraShake / 0.015	6.25	3.05	1.25	0.04
TeraShake / 0.08	5.29	4.95	4.26	0.05
Isabel / 16.3	1.63	0.53	0.48	0.18
Isabel / 25.6	5.44	1.57	1.25	0.21
$X^2 - Y^2$ / 3.5	3.61	2.77	1.01	0.17
$X^2 - Y^2$ / 4.1	2.81	2.01	1.32	0.31

Table 2.4: Comparison of the search error for the Quantized search and our method.

Chapter 3

Dynamic simulation mesh

3.1 Overview

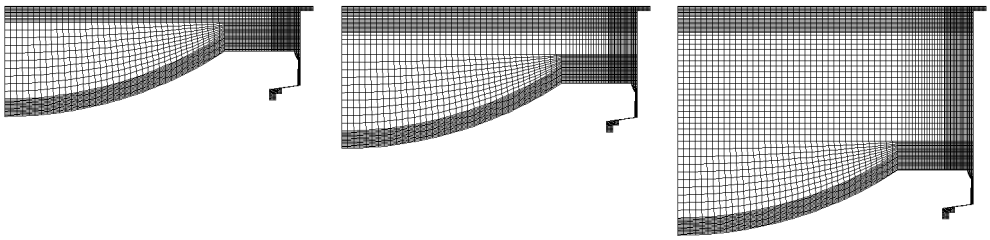
The research presented in this chapter focuses on isosurface extraction from time-varying datasets with dynamic geometry. This topic represents the second large part of the research presented in this thesis.

During late 90 and beginning of this decade, the computational power allowed for simulation of physical phenomena during certain period of time. Data that came out of this *time-varying* simulations were in generally magnitude larger than the static datasets from late 80 and early 90. Due to this fact, a demand for highly optimized isosurface extraction methods emerged.

Most of the time-varying simulations use static simulation mesh to discretize simulation domain. Most existing isosurface extraction methods targets this type of data.

Recently a new type of simulations appeared, that requires simulation domain to change its boundaries due to the moving parts or changing physical properties inside the domain. Changing domain boundaries cause the mesh cells to change their shape. This gradual mesh changing may end up in point where the shape of cells impacts the simulation results up to an unacceptable level. At such a point a change know as *re-meshing* has to take place. During re-meshing, the cells that no longer satisfy given quality criteria are removed from the mesh, replaced by a set of more suitable new cells or moved/re-shaped according to some simulation factors.

Figure 3.1 shows an example of dynamic mesh from a simulation of combustion process in an engine. The mesh changes its layout (number of cells and their geometry) according to the vertical position of a moving piston. Dynamic mesh provides better discretization of changing simulation domain and positively influences overall accuracy of the solution.



(a) time step 400 (b) time step 420 (c) time step 440

Figure 3.1: Example of dynamic mesh from a simulation of combustion process in an engine. The simulation mesh is reconstructed according to the vertical position of a moving piston. Layout of the simulation mesh is depicted for the time steps 400, 420 and 440.

The phenomenon of changing mesh complicates the problem of isosurface extraction. While the techniques for generating dynamic meshes are being rapidly developed, there is a lack of suitable visualization methods for this type of datasets.

The simplest solution to this problem is to treat each time step as a separate static subset of data. This solution is common in most of today's commercially available visualization systems. Though, this solution is very stable and easy to implement its drawbacks are high space a time requirements. Therefore, the research presented in this chapter focuses on development of the visualization techniques capable of interactive isosurface extraction from the datasets with dynamic simulation mesh.

Two novel approaches to the problem described in the previous paragraph will be introduced. Our first approach, focuses on geometry update between two adjacent time steps. By calculating back the inter-time step cells connectivity, the algorithm is able to smoothly visualize evolving isocontours even between defined time steps. The second approach transforms initial data into Span space and tries to minimize space requirement of the search structure allowing for fast isosurface extraction from any defined time step at the time of request.

The rest of this chapter splits into three logical parts. The first part (section 3.2) provides introduction into the theory and application of dynamic meshes. The second part (sections 3.4 and 3.5) describes existing visualization techniques for the time-varying datasets with static and dynamic simulation mesh In the third part (sections 3.6) and 3.7) two newly proposed methods for isosurface extraction from dynamic mesh data are introduced, followed by the practical test on the real-world dynamic mesh datasets.

3.2 Dynamic meshing

3.2.1 Introduction

Principle of dynamic meshing lies in the conditional update of generated initial simulation mesh at each time step. Criteria and conditions of mesh quality and mesh update depend strongly on particular application.

There are many kinds of conditional mesh update covered by the term *dynamic meshing*. In the simulations where the zones of frequent changes in data values are moving, the fixed uniform grids are computationally inefficient. Therefore, solution adaptive grids are often employed, refining the mesh right at the place of the high data changes. This is called *mesh adaptation* [1].

Another technique uses a fixed number of grid points, and let the grid points move in space or move entire grid. This kind of dynamic meshes is known as the *moving meshes*. Thompson et al. [53] provides a good survey of the area.

The most general case is to rebuild the whole mesh at each time step. This kind of mesh update, usually termed *re-meshing*, is used in simulations with rapidly moving domain boundaries (figure 3.1). Re-meshing changes both geometry and the number of cells during mesh update.

Many simulations, due to the complexity of simulated phenomenon, use combination of the techniques mentioned above. Typically the computationally expensive re-meshing is required in the mesh zones around places of fluid-structure interaction where mesh cells are deformed due to some dynamics of the structure. In the places further away from the simulation domain boundaries the shape of mesh cells is less influenced by the boundaries movement. In such place usually the mesh movement/adaptation techniques are applied to increase quality of the resulting mesh. Figure 3.2 shows an example of where combination of the mesh update techniques is used.

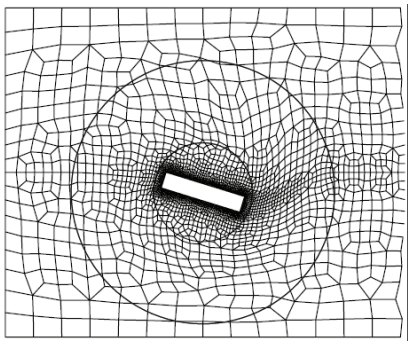


Figure 3.2: Example of the simulation where two mesh update techniques are used. Frequent mesh changes happen around moving rectangular body (inside the ring), while small or none mesh nodes displacement is applied outside the ring (picture from [15]).

3.2.2 Arbitrary Lagrangian-Eulerian (ALE) methods

In the following paragraphs the term *continuum* stands for a moving fluid or other material volume.

In the CFD applications the mesh generation process begins by choosing the right kinematical description of moving and deforming continuum. This description essentially determines the relationship between a continuum and a mesh [15]. There are two classical descriptions of motion:

- *Lagrangian description*, in which each node of simulation mesh is attached to a particle in a continuum. During simulation a mesh follows the continuum in its motion because mesh nodes remains attached to the same particles. The main drawback of this method is its inability to follow large deformations of simulation domain without frequent re-meshing.

- *Eulerian description* lefts a mesh fixed during the whole simulation. The physical quantities associated with particles moving through the fixed region of space are examined. A particular value of observed quantity at a mesh node at time t corresponds to the value of observed quantity at node position in time t. Eulerian method handle the simulation domain distortions relatively easy, but at the cost of lower resolution of flow details.

There are situations that would be difficult to analyze in either the Lagrangian reference frame or the Eulerian reference frame individually. *Arbitrary Lagrangian-Eulerian* (ALE) description of continuum movement combines the best features from both Eulerian and Lagrangian description. Figure 3.3 shows example of mesh nodes and particles movement in Eulerian, Lagrangian and ALE description. In the ALE description the nodes of simulation mesh may fully or partially follow a movement of continuum or may be fixed like in the Eulerian description. The movement of nodes in ALE description offer higher ability to handle the mesh distortions caused by large deformations or movement of simulation domain boundaries.

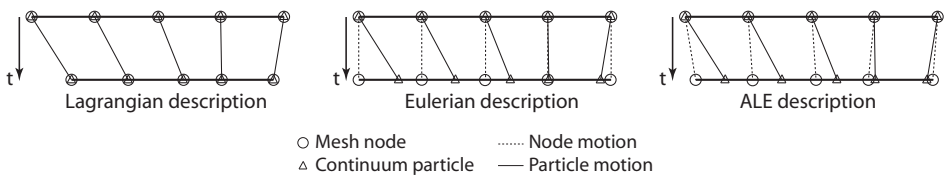


Figure 3.3: 1D example of the Lagrangian, Eulerian and ALE mesh update with respect to the actual particle motion (picture from [15]).

3.2.3 Mesh adaptation

The main objective of the *mesh adaptation* is to optimize the simulation mesh to achieve higher accuracy of the final solution, without excessive computational load. The mesh adaptation strategies typically fall into one of the following three categories:

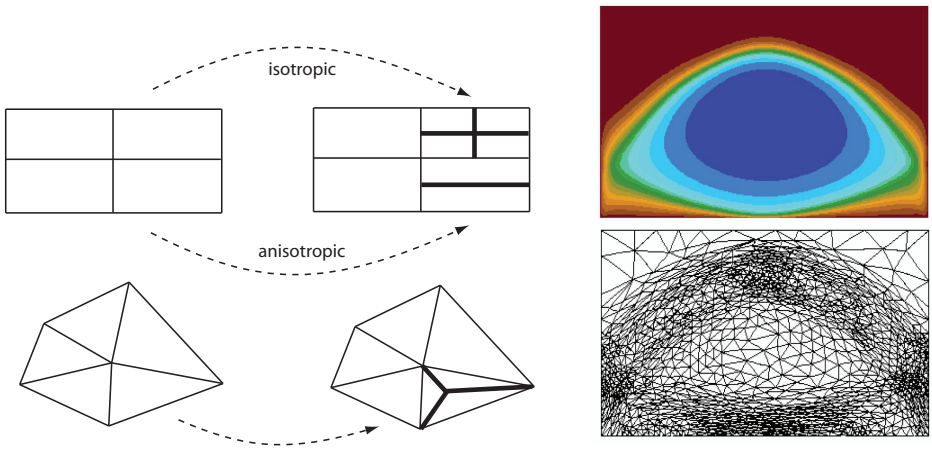
- *r-Refinement.* During this type of mesh adaptation the number of mesh cells remains the same. Typically, the mesh nodes are moved toward the zones with higher solution gradient.

- *h-Refinement.* h-Refinement modifies the cells connectivity by refining the cells in the zones with increasing solution gradient, while the cells in the zones with decreasing computational gradient are merged. The simplest strategy is based on the cells subdivision. The "parent cell" is subdivided into in generally N "child cells".

  There are two basic types of h-Refinement: isotropic and anisotropic. In isotropic refinement the new cells are added equally is all directions, while anisotropic adds the cells only in some directions (figure 3.4).

- *p-Refinement.* in p-refinement the increased resolution is achieved by adaptive increasing of the order of accuracy of the polynomial in each cell.

Combinations of the listed refinements types are also possible. In practice the combinations like hp-refinement or hr-refinement are often employed.



(a) (b)

Figure 3.4: (a) Examples of isotropic and anisotropic cell refinement process for 2D rectangular and triangular shaped cells. (b) Example of the solution adaptive grid. Picture from the ITAPS web page www.tstt-scidac.org.

3.2.4 Mesh regularization

Objective of the *mesh regularization* is to keep the simulation mesh as regular as possible by preventing excessive cells deformation caused by the boundaries movement. Watching and enforcing given cells shape quality criteria decreases numerical errors of the overall solution.

In generally the mesh regularization methods are classified depending on whether the boundaries movement is known before the simulation or is unknown.

If the boundaries movement is prescribed before the simulation then the mesh movement/update can be simply computed using interpolation techniques [23], [25], [41]. Interpolation of the mesh nodes velocity usually results into Eulerian description close to the moving boundaries, while Lagrangian description applies to the nodes far away from the moving boundaries. An example of the method that fits into this category is *Layering*.

Layering, updates hexahedral meshes by dynamically adding the layers of mesh cells near the moving boundaries (figure 3.5). Layer of the cells adjacent to the simulation domain boundaries (layer j in figure 3.5) can be split or merged with its adjacent layer (layer i in figure 3.5). When the layer j grows, its height h is checked by the ideal cell height h_{ideal} condition [16]:

$$h < (1 + \alpha)h_{ideal} \tag{3.1}$$

If cell height h does not meet condition 3.1, the layer j is split according to the user specified criteria and the new layer is build upon j. The mesh from figure 3.1 has been created by the layering method.

If the boundaries movement is not known *a priori* a parts of moving surfaces must be tracked during the simulation. In this case, usually the mesh nodes movement fits Lagrangian description near the moving boundaries (i.e. cells are bound and updated according along the boundary), while the internal nodes of the mesh are Eulerian. This is case of numerical simulations described by for example [39], [31], [24].

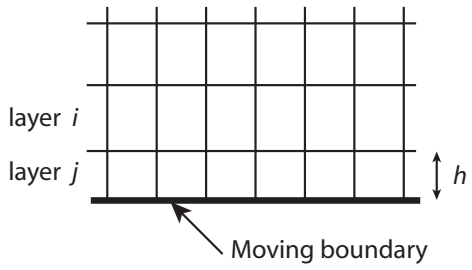


Figure 3.5: Layering. Layer j grows on the layer i according to the movement of domain boundaries. Height h of layer j is periodically checked until it fails to meet condition 3.1 - then the layer j is split [53].

The following two methodologies the Transfinite mapping and Spring smoothing assume that no information abound boundaries movement is known a priori. A short overview is given for both.

Transfinite mapping. This technique can be applied in the cases where boundaries are given by their exact geometric description. The general transfinite method describes an approximate surface or volume at a nondenumerable number of points (thus the term transfinite mapping) [15]. The nodal coordinates can be obtained explicitly once the boundaries have been discretized, thus the procedure is computationally cheap (see [22], [17], [18]). Figure 3.6 shows an example of the transfinite mesh.

Spring smoothing. Spring smoothing idealizes the connections between mesh vertices as a network of interconnected springs. The initial connection of the mesh nodes is considered as a stable state of the mesh. Deformation of the mesh boundaries causes displacement of the vertices by the boundaries and generates tension on the springs. The tension on the springs connected to i-th internal mesh vertex is expressed as a force $\vec{F_i}$ in the i-th vertex:

$$\vec{F_i} = \sum_{j}^{n_i} k_{ij}(\delta\vec{x_j} - \delta\vec{x_i}) \tag{3.2}$$

where n_i is the number of vertices connected to vertex i, $\delta\vec{x_j}$ is a displacement of neighbor n_j against the vertex x_i, k_{ij} is the spring constant (Equation 3.3). So, the resulting force in the i-th vertex is proportional to the displacement along the springs connected to the node i [16]. The position of the i-th vertex is then adjusted according to the vector and magnitude of the $\vec{F_i}$. This process of *force-relaxation* on the mesh springs is applied to all internal mesh vertices.

$$k_{ij} = \frac{1}{\sqrt{\vec{x_i} - \vec{x_j}}} \tag{3.3}$$

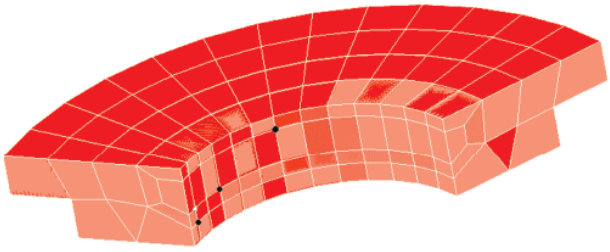


Figure 3.6: Example of a transfinite mesh created around circular shaped body [35].

3.2.5 Real-world applications of dynamic meshing

Four examples of the industrial applications of the dynamic meshing are discussed and illustrated in this section. This is to give reader clearer idea on how the techniques from the sections 3.2.3 and 3.2.4 are used in practice; and to support the conclusion made in the rest of this thesis.

Amsden [2] describes dynamic meshing techniques used in KIVA-3V program for simulation of vertical and canted valves of combustion engines. Especially useful in these simulation are *Layering* and local refinement and coarsening of a simulation mesh (figure 3.7). As stated by Amsden: *"This is not simply a matter of generating the initial grid: The grid must dynamically change during the run in response to the changing valve positions..."*.

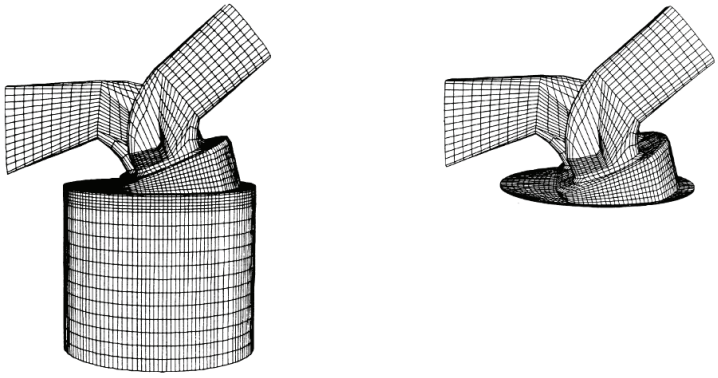
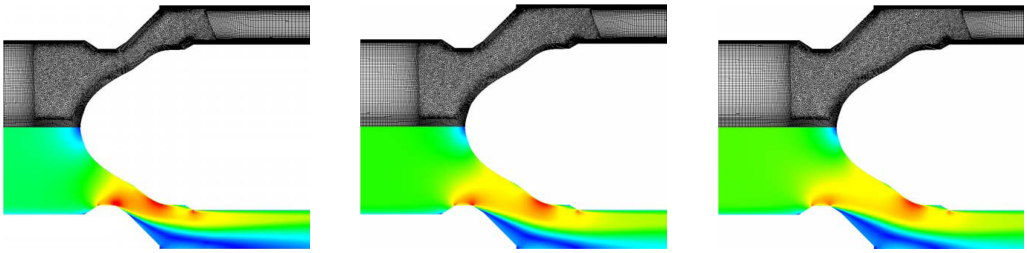


Figure 3.7: Mesh layout of the valve with piston down (left image) and up (right image) [2].

Cavallo et al. [10] used dynamic mesh in the simulation of a flow through the split-body valve. The mesh is adaptively refined and coarsened in the zones where a moving inner part of the valve creates moving mesh boundaries (figure 3.8).



(a) 50% open (b) 70% open (c) 80% open

Figure 3.8: Layout of the dynamically updated simulation mesh and direct rendering of the data from a split-body valve simulation (picture from [10]).

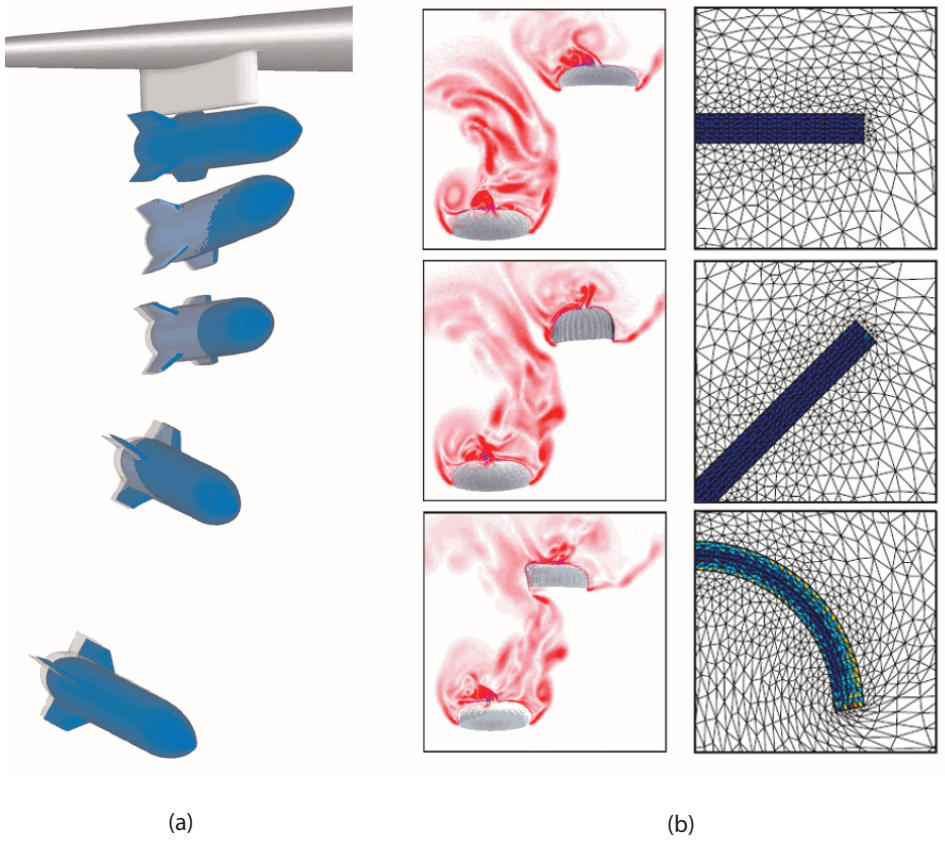


Figure 3.9: (a) Munition separation in various stages of the simulation. The mesh has to dynamically adapt around falling object [50]. (b) Left column: modeling the flow around dynamic canopies of the deployed parachute group. Right column: Details of the dynamic mesh adapting around changing parachute shape [51].

Snyder and Sverdrup [50] discuss advantages of dynamic meshing against other methods for simulation of munition separation from under an aircraft wing. Rapid movement of falling munition requires continual modification of the simulation mesh (figure 3.9a).

Stein et al. [51] use dynamic simulation mesh to model aerodynamic around changing parachute structure. Mesh moving methods along with local re-meshing of the fluid domain are used in this simulation (figure 3.9b).

3.3 Isosurface extraction from dynamic mesh data

3.3.1 Problems

Typically after the simulation mesh is generated, the cells correspondence is lost. This fact defines the starting point for the isosurface extraction from dynamic mesh data: *a dataset with different mesh at each time step without any defined relation of the cells along the time dimension.*

There are two major problem that need to be addressed by the methods for isosurface extraction from the datasets with dynamic simulation mesh:

1. *Changing mesh geometry.* The number of the cells changes between successive time steps. Geometry and position of the cells (and possibly topology of the mesh) may also vary during the course of simulation.

2. *Large data volumes.* The datasets with dynamic mesh are usually a magnitude larger when compared to those with static mesh.

Two problems listed above lead to the specific challenges that need to be overcome by the efficient solution to the isosurface extraction from dynamic simulation mesh:

- Representation of the dynamic mesh
- Measuring similarity in the dynamic mesh
- Suitable search structure for active cells identification

Each of these three challenges is discussed in the following sections.

3.3.2 Representation of the dynamic mesh

The total number and position of the cells may vary between successive time steps. Mathematically: for two successive time steps C_t and C_{t+1} represented by the disjunctive set of cells $C_t = \{c_i, i = 1..n\}$ and $C_{t+1} = \{c_j, j = 1..m\}$ holds that $n \neq m$.

To find the representation of the dynamic mesh for the total number of T time steps means to find such mapping Λ that:

$$\Lambda_t \colon C_t \longrightarrow C_{t+1},\ t \in (1,\ T-1) \tag{3.4}$$

$$\Lambda = \Lambda^{-1} \tag{3.5}$$

Equality 3.5 states that the reconstruction of the cells correspondence should give the same result whether being done along or against the time dimension.

There are four variations of the Λ mapping above on the level of single cells:

1. The cell $c_i \in C_t$ does not map on any cell of C_{t+1}:

$$\lambda_0 \colon c_i \longrightarrow \emptyset \tag{3.6}$$

2. The cell $c_i \in C_t$ maps on exactly one cell $c_j \in C_{t+1}$:

$$\lambda_1 \colon c_i \longrightarrow c_j,\ c_i \in C_t,\ c_j \in C_{t+1} \tag{3.7}$$

3. The cell $c_i \in C_t$ maps on more than one cell from C_{t+1}:

$$\lambda_{split} \colon c_i \longrightarrow \{c_j : c_j \in C_{t+1}\},\ c_i \in C_t \tag{3.8}$$

4. The two or more cells $c_i \in C_t$ maps on one cell from C_{t+1}:

$$\lambda_{merge} \colon \{c_i : c_i \in C_t\} \longrightarrow c_j,\ c_j \in C_{t+1} \tag{3.9}$$

3.3.3 Similarity in the dynamic mesh

Let's represents a cell c as a couple $c = (g, v)$, where g represents the cell's geometry and v defines cells values. A similarity δ of two cells c_1 and c_2 then can be defined by the means of geometric and value similarities as a function composition:

$$\delta(c1, c2) : G \circ V \longrightarrow R,\ \delta \in (0, 1) \tag{3.10}$$

$$G \colon (c_{1g}, c_{2g}) \longrightarrow R \tag{3.11}$$

$$V \colon (c_{1v}, c_{2v}) \longrightarrow R \tag{3.12}$$

δ=1 represents maximum similarity (c_2 is an exact image of $c1$) and suggests that only one λ mapping exists and it is λ_1. δ=0 represents no similarity in neither geometric nor value sense and suggests that only one *lambda* mapping exists and it is λ_0.

When generalizing δ similarity for the two mesh setups at successive time steps, the minimal and maximal similarity $\Delta \in (0, 1)$ of two meshes can be defined as:

$$\Delta = 0 \Leftrightarrow \forall c \in C_t\ \exists\ \lambda_0 \colon c \longrightarrow \emptyset \tag{3.13}$$

$$\Delta = 1 \Leftrightarrow \forall c \in C_t\ \exists\ \lambda_1 \colon c \longrightarrow s \in C_{t+1},\ \delta(c, s) = 1 \tag{3.14}$$

Between the similarity extremes defined above lies the average case when $0 < \Delta < 1$. In this case certain amounts of all four cases of λ mappings exist between the cells at adjacent time steps and the similarity δ of the mapped cells is between 0 an 1.

3.3.4 Geometric considerations

The purpose of this section is to analyze the impact of the mesh update methods (section 3.2) on the mesh similarity Δ at successive time steps. Such analysis is necessary for the space and time efficient design of the isosurface extraction method from the datasets with dynamic mesh. In the following, the cells geometry changes during mesh adaptation and mesh regularization are discussed.

There are two ways in which the mesh adaptation is usually done (section 3.2.3). The first way is to adaptively refine/coarse single cells in the areas with high solution gradient. This process is equal to the set of the λ_{split} mappings (refinement) or the set of λ_{merge} mappings (coarsening). In any case, this changes in the mesh are local and most of the cells in the mesh keep their position and shape (equivalent to the Eulerian description of the mesh nodes).

The second typical way of performing the mesh adaptation is Lagrangian or ALE nodes displacement toward the zones with high solution gradient. This operation results in a set of λ_1 mappings whereas the cells similarity δ in the area of adaptation decreases proportionally to the velocity magnitude of the nodes movement (figure 3.10a).

The situation is more complicated in the case of mesh regularization. Changes in cells position, shape and total number strongly depend on the used mesh update method and the speed of boundaries movement. Figure 3.10 shows each of the three cases listed below (figure 3.10b).

- Typically the mesh zones near the moving boundaries (if the boundary movement is not known *a priori*) features increased number of the λ_0 mappings of the cells.

- Further away from the moving boundaries the severity of the mesh changes decreases and λ_0 mappings become rare. Important aspect of the cells geometry change in the transition zone between Eulerian and Lagrangian nodes movement is to keep the quality of the cells in order produce accurate solution. Such quality enforcement may result in adding/removing the cells which are due to their shape no longer computationally feasible (e.g. too thin or too large). The result of the addition/removal of the cells is increased number of the λ_{split} and λ_{merge} mappings along with decreased similarity δ of the mapped cells.

- The influence of the boundaries movement on the position and shape of the cells decreases with the distance of these two elements. The update of cells far away from the moving boundaries is small resulting in the λ_1 mapping and cell similarity δ close to 1.

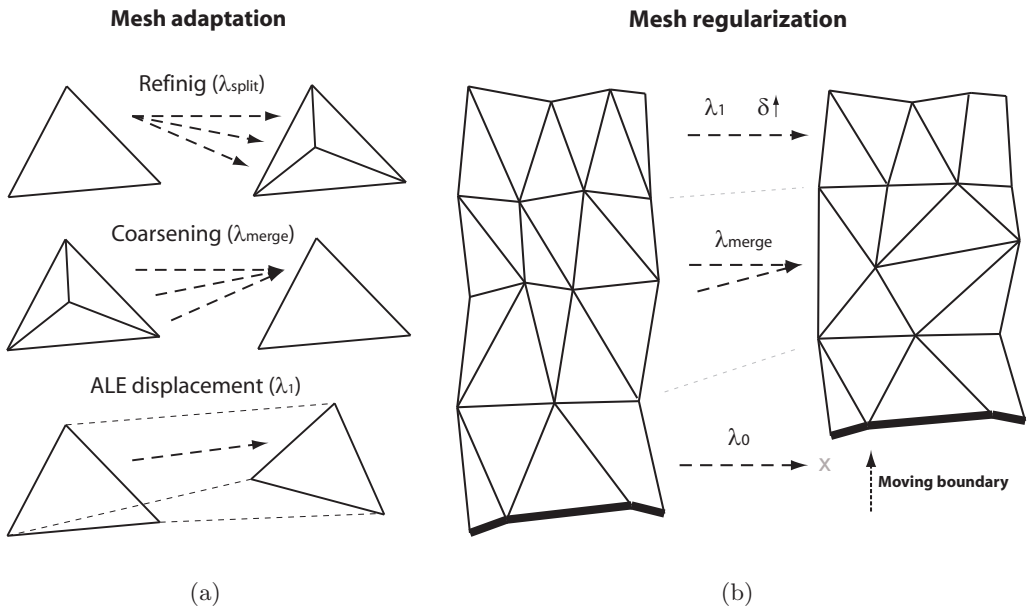


Figure 3.10: Illustration of the geometric changes in term of λ mapping in the mesh during (a) mesh adaptation and (b) mesh regularization.

3.3.5 Identification of the active cells

Most of the existing techniques for active cells extraction from time-varying data assume static simulation mesh. The methods exploit the fact that the correspondence between the cells at successive time steps is implicitly given. In the sense of the equations 3.6 to 3.9 it means that only λ_1 mappings exist between the cells at successive time step and these mapping are known *a priori*. Dynamic mesh update however requires λ_0, λ_{split} and λ_{merge} mappings between the cells at successive time step.

In the section 3.4 an overview of the existing methods for static simulation mesh is provided, along with the conclusion that all of this methods fail when applied on the dynamic mesh datasets. Thus, the two main challenges of the suitable isosurface extraction method are: *(1) how to find back lost correspondence of the cells at the successive time steps and (2) to exploit geometric and value coherence of the corresponding cells along the time dimension.*

Keeping geometry of all mesh cells in the main memory could be very space-consuming (if not impossible for large datasets). Thus, only the space efficient structures holding the minimal amount of information necessary to identify the active cells, could be in the main memory. Geometry of the active cells then can be read from the disk using I/O efficient out-of-core techniques.

3.4 Existing methods for time-varying datasets with static mesh

The explanation of the principles of dynamic meshing provides a minimum knowledge necessary for understanding of the requirements imposed on the methods for isosurface extraction from such kind of datasets. This section provides overview of the existing methods for isosurface extraction from time-varying data with static simulation mesh. These methods provide a good starting point for the further research presented later in this thesis.

3.4.1 Temporal Hierarchical Index Tree

For a time-varying field, a cell may has multiple corresponding points in the Span space for the different time steps. To characterize a cell's scalar variation over time, the area over which the corresponding points spread in the Span space is measured.

In *Temporal Hierarchical Index Tree* (THIT) [44] the cells are first assessed by their temporal variation. Criterion for low temporal variation is that the points corresponding to the same cell are located with 2x2 elements of lattice subdivision of the Span space.

The Temporal Hierarchical Index Tree places the cells with low temporal variation over time closer to the root of the tree. For other cells that do not satisfy the criterion of low temporal variation the root time interval is divided in half. Process continues recursively into each of two subtrees. The leaf nodes contain cells with the highest scalar variation over time, so that the cells' time-specific extreme values are used.

Search index for each node of the THIT is created by the ISSUE [46] algorithm. The min/max values in the nodes closer to the root node are used to refer to a cell for more than one time step, which contributes to the lower overall size of the tree structure.

Given an isosurface query at time step t, THIT is traversed and the nodes that contain the active cells are visited. In the visited nodes the ISSUE search for the active cells is performed.

This method accelerates isosurface extraction from time-varying data. However at each time step the entire data domain (time step) is loaded into the main memory. The isosurface extraction process potentially needs to access all of the time steps in the time-varying dataset, which may cause memory overhead.

3.4.2 Temporal Branch-on-Need Tree (T-BON)

The *Temporal Branch-on-Need Tree (T-BON)* [52] extends the three dimensional Branch-On-Need Octree (BONO) [63] for dataset. The method focuses on minimizing the number of I/O operations, by reading from disk only those portion of search structure and data necessary to construct the current isosurface.

First a BONO tree [63] is computed for each time step. Information about general infrastructure of a BONO tree is saved to the disk only once for the entire dataset. Then the extreme values for nodes are computed and stored separately for each time step (figure 3.11).

During isosurface extraction a query in the form (*isovalue, time step*) is processed by first testing if the iso-range of the root node of the BONO tree for desired time step covers the isovalue. If so, root node's children are tested for iso-range as well. Once the process hits a leaf node which covers desired isovalue the disk block which contains the node's data (cell values) is added to the list. Once this process is over, all of the blocks in the list are read from the disk at once. This clearly minimizes the I/O access. However, T-BON does not exploit any temporal coherence in the input data (except regular mesh geometry).

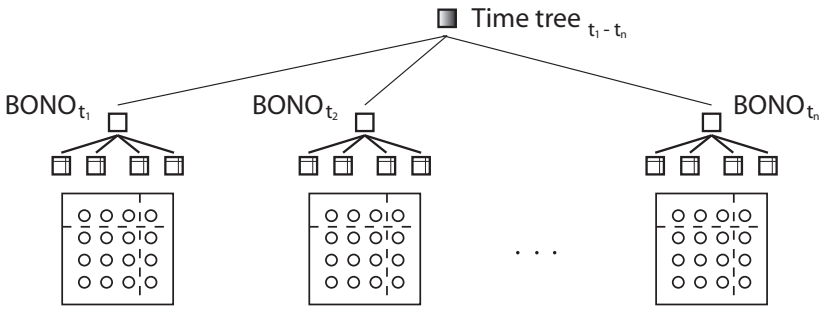


Figure 3.11: T-BON method. A separate BONO tree is constructed for each time step. Geometry of the tree structure is saved only once, while actual values are stored separately for each time step.

3.4.3 Time-space partitioning tree

Time-space partitioning tree (TSP) [45] is the octree-based data structure, efficiently encoding coefficients of the spatial and temporal variation of the data. TSP tree is a standard full octree that recursively subdivides a data volume until a predefined minimum size of sub-volume is reached.

Each node of TSP tree stores a binary tree, recursively bisecting time-span $[0, t]$ of the entire dataset until the unit time step is reached. For each node of binary time-tree associated with each TSP tree node the coefficients of *spatial* and *temporal variations* are computed from the original time-varying data.

During the volume-rendering-based visualization the partial images from sub-volumes are cached for the current time step. When another time step is selected a TSP tree is traversed. The test whether particular sub-volume's spatial and temporal variations exceeds the user-selected thresholds is performed, in case of which this sub-volume is rendered again. Otherwise the cached rendered partial image is used, speeding up the visualization process.

TSP enables user to manage the trade-off between the visualization speed and accuracy through the thresholds of spatial and temporal coherence. Similar error-based isosurfacing method [59] is described later in this section. Figure 3.12 shows example of the TSP tree for datasets with three time steps.

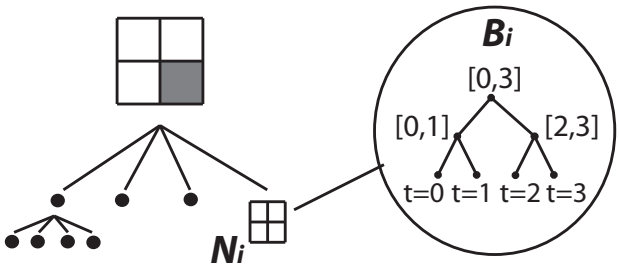


Figure 3.12: TSP tree is a standard full octree (left side of the picture). Each node of TSP tree has assigned a binary time tree (right side). In this case, the dataset consists of four time steps (picture from [45]).

3.4.4 4D approach

In 1996 Weigle and Banks introduced *recursive contour meshing* [61] for decomposing n-dimensional simplex into a set of (n-1)-dimensional simplices. Their method also includes the 4D case. Based on this work, two years later, Weigle and Banks published the method [62] for isosurface extraction from time-varying scalar fields.

Basic idea of their 4D method [62] is to look at the time-varying data as to be the one static 4D dataset. Each sample of the original dataset is identified by three spatial coordinates x, y, z and one time coordinate t. This interpretation of 3D time-varying data together with the usual way of extracting isosurfaces from the mesh composed of n-dimensional simplices (over n-dimensional data) leads them to construction of a mesh of 4-simplices. To find an isosurface $f(x,y,z,t) = 0$, they apply two constraints to the 4-simplicial mesh.

In the first pass a set of *iso-volumes* $f_1(x,y,z,t) = 0$ are extracted by imposing *isovalue constraint* over the 4-simplices. This results in a set of 3-simplices intersected by the isosurfaces of the given isovalue.

To produce isosurface for the particular time step c_t the second pass is done, imposing the *time constraint* $f_2(x,y,z,t) = t - c_t = 0$. This produces the resulting isosurface composed of 2-simplices for desired isovalue and time. Although this method elegantly captures temporal coherence in a dataset, its high computational demands makes it impractical for the large datasets.

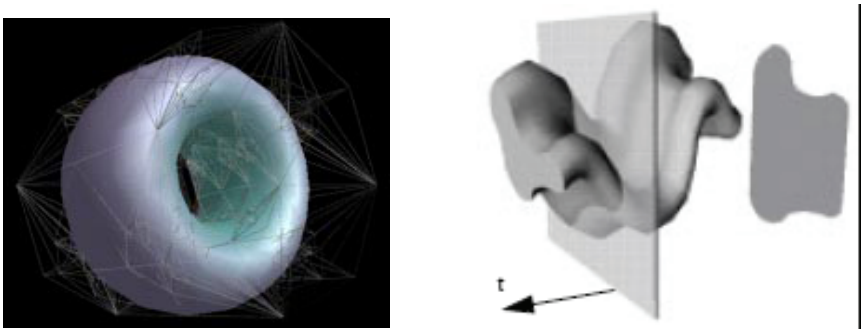


Figure 3.13: (Left) Isosurface satisfying 2 constraints in 4-dimensional space. 3-simplexes (wire frames) are the result of imposing first (*isovalue*) constraint. Extracted isosurface satisfies the second (*time*) constraint. (Right) A time-varying 2D contour sweeps out surface in (x, y, t)-space. The silhouette of the surface (projected down the t-axis) forms the envelope of the swept curve. Analogy can be used for (x, y, z) data plus time dimension (pictures from [62]).

3.4.5 Out-of-core visualization

Recent advances in computational performance enable scientists to perform large-scale simulations, producing datasets which are usually a magnitude larger than is a size of physical memory of workstation-class computers. Thus, various *out-of-core* visualization approaches have been proposed. Principle of the out-of-core approaches is to split original dataset into a range of files and using only those of them needed for visual analysis.

The main issues solved when using out-of-core approach is to minimize the number of disk I/O operations necessary to access required data. The out-of-core methodology is usually used as the extension of the existing methods, enabling them to work with large-scale dataset.

As an example of the out-of-core method the work of Reinhard et al. [40] is briefly described. Their method is focused on fast out-of-core isosurface visualization. The method first partitions the whole dataset into a range of small files. Each file contains the data for one time step and a certain range of isovalues. Efficient 28 bytes-per-voxel data format is used to store time-varying data in the files, reducing the necessary disk-memory traffic. For visualization of the isosurfaces they use ray-tracing based method of Parker et al. [36]. The technique has been implemented and tested on 32 processors SGI Origin 2000 computer, with 12 GB memory. Computational power along with the out-of-core nature of the technique allows visualization of the isosurfaces from such a large datasets at interactive frame rates.

3.4.6 Adaptive extraction of time-varying isosurfaces

Solution to the problem of fast isosurface extraction from large time-varying datasets proposed by Gregorski et al. [20], utilizes adaptive mesh refinement scheme and an out-of-core approach. The key of the adaptive extraction method is the recursive mesh refinement scheme which splits initial volume into a set of tetrahedra. These are further grouped into the diamonds organized in a hierarchical structure.

During the preprocessing phase an isosurface approximation error, minimum and maximum data values enclosed by each diamond and normalized gradient vector at each data point are computed. At runtime, the refinement process creates a set of tetrahedra, describing domain around the isosurface, based on user supplied isosurface approximation error, isovalue and time step.

If another time step is selected, the *min*, *max* and *error* tolerance values of the visible diamonds are checked. If these fit the user-defined threshold values, all of the tetrahedra within such diamonds are used for isosurface extraction. Otherwise, the refinement process (sequence of *splits* and *merges*) is initiated.

When the isovalue is changed then the refinement process can start either from the root diamond of a diamond hierarchy or from a current refinement of a mesh, checking the min, max and error tolerance values of the diamonds. The way in which the refinement process is initiated depends on the data difference between time steps t_i and t_{i+1}.

3.4.7 Difference intervals

Waters et al. introduced the *Difference intervals* technique [59] for better utilization of spatial and temporal coherence in a dataset. Static mesh with time-varying scalar values is assumed. Input of the method are desired isovalue and time step. Isosurface evolution is visualized by first computing the active and inactive cells from an entire dataset (or only from a selected time span) and then visualized using this information.

In a preprocessing time a set of active cells is first computed for the first time step, using an arbitrary existing technique. Active / inactive cells are then computed from this initial information for the next time step. Movement of cell in the *Span space* is observed and classified into 11 cases (figure 3.14), which results into designation of each cell with either *add* (cell became active) or *remove* (cell became inactive) operation. Eventually, a cell can holds its status in a next time step. The changes between t_i and t_{i+1} are encoded into an operation set D_i. In a preprocessing step the operation sets are computed for all the adjacent pairs of time steps. The technique for temporal compression of a sequence of operation sets is also discussed.

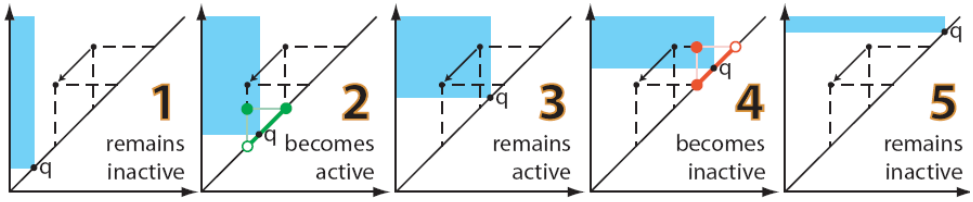


Figure 3.14: Examples of the *add* and *remove* operations. These operations are then encoded in the operation sets and used for playback of the evolving isosurfaces (picture from [59]).

Visualization utilizes the encoded operation sets to update only the necessary portions of the isosurfaces according to the *add* or *remove* operation assigned to each single cell. In this way an interactive playback of isosurface evolution of constant isovalue q is possible in both forward and backward direction.

A point-based rendering technique is utilized for visualization of the isosurface. Once the desired isovalue is changed the preprocessing steps has to be done with this new parameter again.

3.4.8 Persistent hyperoctree (PHOT)

Persistent hyperoctree (PHOT) [48] introduced by Shi and allows to extract a set of the active cells and simultaneously only those of them that are visible from the current point of view (relevant cells). By saving the time to extract occluded cells and more efficient tree construction, significant speed up in isosurface extraction is achieved.

By deleting the nodes which contain only the inactive cells in their sub-volume and collapse the nodes that contain only one children in an octree a *compact octree* is made (figure 3.15).

Each PHOT's node has $8+k$ *jumpers* (k is a small constant) with associated version number. Root node of hyperoctree represents an entire hyper volume and has 16 children, each representing one hyperoctant.

To make hyperoctree persistent all the cells are collected and two copies are held for each of them. One copy holds a cell's *min* value as its key and the other one holds the *max* value. The cells are sorted according to their key values in ascending order and the list is traversed. If a cell has its *min* value as its key it is stored in the current version of the tree, otherwise the new version is created and the cell is stored into it. When traversing a PHOT, the root node with the largest version number smaller or equal to isovalue is identified and its subtree is traversed by following the latest jumpers no later than isovalues.

For 3D isosurface extraction by 4D slicing along the time axis a 4D cells are constructed and indexed by their value ranges using PHOT. Given an isovalue and time parameters the active and relevant cells are extracted, and then sliced along the time axis to determine the isosurface at given time.

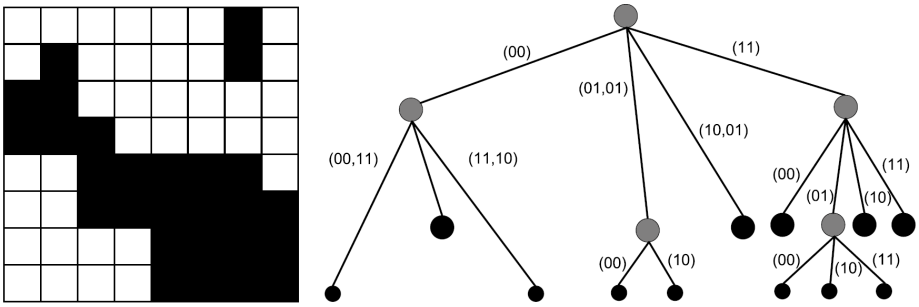


Figure 3.15: Compact hyperoctree for the set of active cells (the black ones on the left sub-figure). As can be seen from the right sub-figure the nodes which contain only one children are collapsed (picture from [48]).

3.4.9 Conclusion

Previous sections review 8 major techniques for isosurface extraction from the time-varying datasets with static mesh. The purpose of this section is to asses usability of the described methods for the datasets with static simulation mesh.

In the following paragraphs a brief summary of the way in which the techniques exploit data coherence along the time dimension is provided. Finally, the conclusion from this summary is made.

In the Temporal Hierarchical Index Tree a cell's temporal variation is measured as a displacement of its corresponding points (for the various time steps) in the Span space. Since the correspondence of the mesh cells between adjacent time steps is in generally unknown for a dynamic simulation mesh, such temporal variation measurements in Span space can not be done, which is why the THI tree is unusable for the datasets with dynamic simulation mesh.

T-BON method uses BONO tree for spatial indexing of the mesh cells. BONO tree is created only once for the whole datasets, so the simulation grid has to remain static during the course of simulation. Only the min/max values of the cells are stored separately for each time step. Because the T-BON method does not exploit any temporal coherence of the data between adjacent time steps, a set of different BONO trees can be used to represent dynamic simulation mesh. However, this idea implies for a large storage space (no spatial or value coherence is taken into account) and moreover, BONO tree is suitable for regular grid datasets rather than unstructured dynamic meshes.

TSP tree is a time supplement octree. The mesh cells are spatially indexed by the standard octree. So, each leaf node of the TSP tree corresponds to a particular mesh cell. A binary time tree stored at each TSP node keeps an information about variation of the cells' min/max values along the time dimension. Because of the variable number of the mesh cells in the dynamic simulation mesh, a TSP tree should has to be rebuilt at each time step to reflect the changes in mesh layout.

4D approach of Weigle and Banks represents 3D time-varying dataset as the 4D static data. The original 3D gridded time-varying data are represented as a mesh composed of 4D cells - the *hypercubes*. To construct a hypercube, a correspondence of the two original 3D cells for each hypercube has to be known *a priori* and must be 1:1. Thus the simulation mesh has to stay unchanged for any two adjacent time steps. Moreover, high computational and space requirement of the method makes it impractical for rather large datasets with dynamic simulation mesh.

In the Adaptive extraction approach of Gregorski et al. the original mesh cells are grouped into a hierarchy of diamonds. Since the hierarchy of diamonds does not change during the simulation, the simulation mesh is required to remain static.

The Difference intervals method encodes the changes of a cell status in the successive time steps as either *add* (cell become active) or *remove* (cell become inactive) operation. In order to encode the status of all cells in the dataset the assumption of static 1:1 correspondence between the cells at adjacent time steps has to be satisfied. This requires the simulation mesh to keep the fixed number of mesh cells during the whole simulation.

Persistent Hyperoctree (PHOT) is derived from the standard hyperoctree, in which each node has 16 children (hyperoctants). Since the original 3D octants are merged into the 4D hyperoctants a data grid is assumed to be regular and fixed during the whole simulation.

Given the description of the way a spatial and values coherence is exploited in the techniques listed above, it is apparent that all of them require static simulation mesh with implicit cells correspondence. Thus a simple conclusion can be made from the description above: *The assumption of the static geometry prevents the usage of these methods over the datasets with dynamic mesh.*

The next section provides overview of the naive approach and one existing method capable of active cells identification from the datasets with dynamic mesh.

3.5 Existing methods for time-varying datasets with dynamic mesh

3.5.1 Naive approach

As can be seen from the list of the techniques above, the active cells identification in the dynamic meshes can not be simply solved by applying the existing techniques designed for static simulation meshes. Thus a common approach to the this problem today in most of the commercially available visualization systems is to treat dynamic mesh as a sequence of the static meshes, which will be termed in the following as the *naive approach*.

In the naive approach the search structure for active cells identification is built independently for each time step (figure 3.16). Any of the existing techniques for static simulation mesh described in the chapter 2 can be used (including the newly proposed approach in the section 2.6).

The advantage of the naive approach is that it is simple to implement and relatively stable. Obvious disadvantage is its high computation cost and long time required to produce animation of the evolving isosurfaces. This is due to the fact that spatial and temporal coherency in the datasets is not exploited in any way.

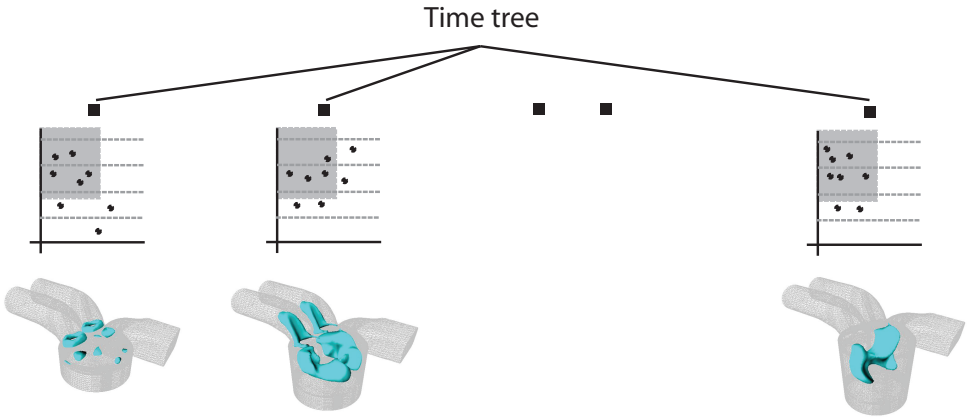


Figure 3.16: The naive approach. A search structure is built separately for each time step. Resulting set of search structures is indexed by a simple two level 'time tree'.

3.5.2 A method for layered meshes

Doleisch et al. [14] introduced a method for visualization of datasets with dynamic simulation mesh. Their method is designed specifically for the *layered meshes* (chapter 3.2.4).

Continuous intervals are assumed in the dataset within which a number of mesh cells and their correspondence between adjacent time steps remains static.

Such intervals are called *topology zones*. Cells are not matched or tracked over the topology zones borders (*rezone points*).

Within a topology zone a mesh layout is expected to change linearly, so the shape of cells as well as the values associated with mesh vertices can be interpolated from the *key geometries* at the borders (or inside) of a topology zone. Within a topology zone any of the existing methods for isosurface extraction from static datasets can be used (chapter 2). Figure 3.17 illustrates the principle of the method.

The assumption of the *topology zones* can be used only for a specific subset of the datasets with dynamic simulation mesh. However, in generally, each time step might represent a different topology zone.

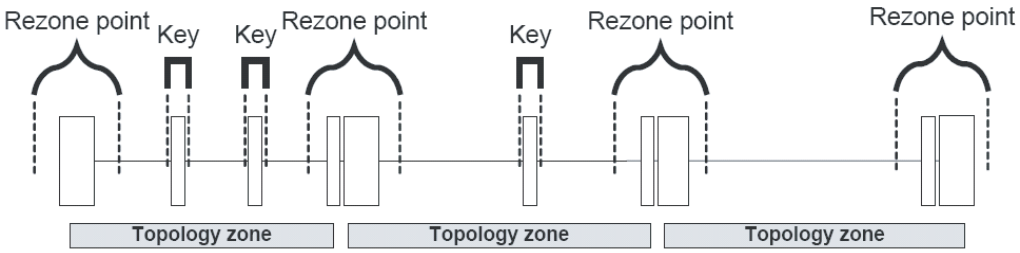


Figure 3.17: Principle of the *topology zones* technique [14]. Each topology zone consists of at least two *key geometries*. Borders of the topology zones are called *rezone points* (picture from [14]).

3.6 Proposed method 1: Isolines extraction

3.6.1 Overview

The goal of this method is the extraction of isocontours from the two-dimensional time-varying datasets with dynamic simulation mesh. Such datasets are often used in engineering practice to investigate physical phenomena 'per slice'.

The key idea of the method is to find the mapping Λ (section 3.3.2) between adjacent time step by establishing the edge-edge correspondence between the mapped cells. Reconstruction of the mapping Λ is based on the data values similarity and spatial distance of the potentially correspondent cell at two successive time steps. The method is able to handle all four cases of the λ cell mappings (equations 3.6 to 3.9) under the assumption of the low temporal distortion of the data (reasonably high similarity δ of the mapped cells).

The proposed method tries to reconstruct the cell-cell correspondence between the time steps. Out of the computed correspondence a cell strips can be made which follow the gradient of data values. In the following such cell strips will be referred to as *iso-components*. In the case of 2D mesh, an iso-component is basically a strip of neighboring triangles, covering certain range of the isovalues. The iso-component has the inner (IE) and the outer (OE) envelope (figure 3.18).

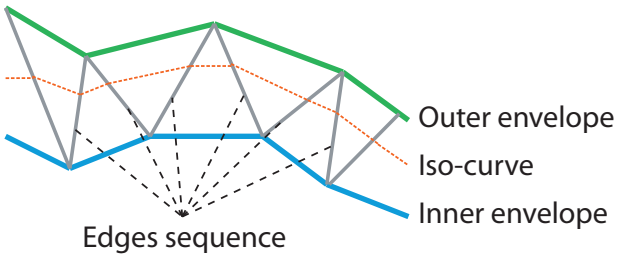
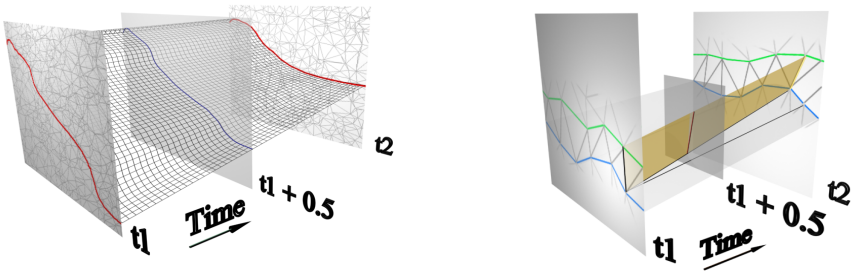


Figure 3.18: Iso-component. In 2D mesh an iso-component is a strip of neighboring triangles with inner and outer envelope.

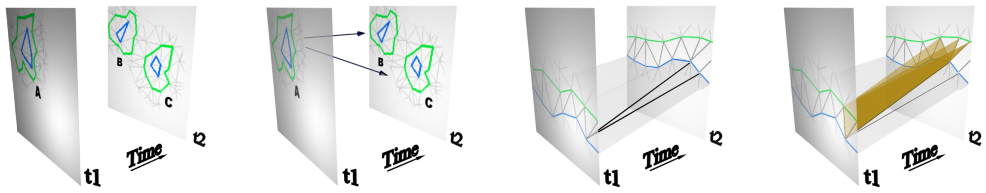
Computed iso-components allow for reconstruction of the edge-edge correspondence of the mapped cells. The edges mapping then can be directly used to interpolate points of the isocontours for the desired isovalue.

A *brute-force* approach to the problem would be to first extract isocontours for desired isovalue at two successive time steps and then map them to create their smooth shape morphing. Disadvantage of the this brute-force solution is that the isocurves and their morphing have to be recomputed every time the user changes isovalue. Figure 3.19 compares the *brute-force* solution and principle of our method.



(a) (b)

Figure 3.19: (a) The *brute-force* approach to the problem. (b) Our method.



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

Figure 3.20: Four stages of our method: (a) extraction of the iso-components, (b) tracking of the iso-components between successive time steps, (c) mapping of the inner envelopes vertices between corresponding components, (d) final edge-to-edge mapping of the components.

3.6.2 Preprocessing

The preprocessing described in this section applies for a pair of two consecutive time steps. Thus, it has to be done separately for each pair of time steps in the datasets. Such separate computations suggest application of the parallel computing with all its benefits.

The process of establishing edge-edge correspondence consists of the four following steps (figure 3.20):

1. Extraction of the iso-components
2. Tracking of the iso-components between adjacent time steps
3. Mapping of iso-components' inner envelope vertices
4. Establishment of the edge-edge correspondence

Each step of these four is detailed below.

1. Extraction of the iso-components

Iso-component extraction begins by sorting a list L of all different isovalues from both adjacent time steps. The values in L defines boundaries of intervals that are used for further processing. The middle value of each interval within L is taken as a seed value for iso-components extraction.

To locate unique set of seed cells for each iso-component a Contour tree (section 2.3) is built. Once a seed cells are determined for each seed value the iso-components are extracted using continuation method (figure 3.21).

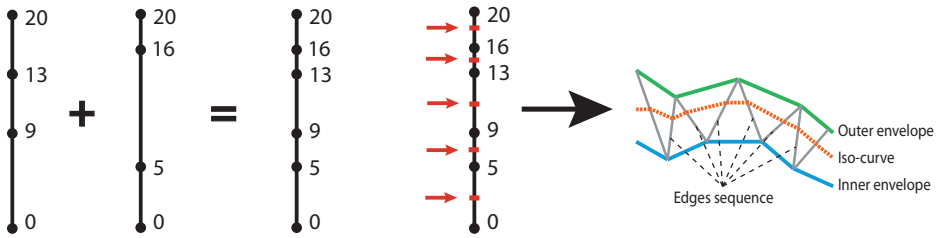


Figure 3.21: Iso-components extraction. All the different isovalues from two adjacent time steps are merged and sorted into one list L. Iso-components are then extracted using continuation method and middle values of the intervals in the list L.

2. Tracking of the iso-components between adjacent time steps

The feature tracking techniques [43] are employed to find the best matching successors for each extracted iso-components. In our implementation, the area overlapping test is used [49]. Depending on the application, more complex feature tracking techniques can be used without disturbing the overall concept of our method.

3. Mapping of iso-components' inner envelope vertices

Once the successors of the iso-components are determined, the process of mapping of their IE vertices takes place. This mapping provides a guide to the final edge-to-edge mapping.

Process of IE vertices mapping is based on the notion of *candidate area* (figure 3.22a), which is defined for each mapped IE vertex. Successor of the vertex has to lie within this area. Employing this principle, the closest from such restricted set of candidate vertices is chosen as the successful one. During the mapping process the projections (Fig. 3.22b) of vertices from successive time slices are considered rather than computing with the real space-time elements.

(a) (b)

Figure 3.22: (a) Candidate area of the vertex q (shadow region). (b) Projection of slicechord S.

The process of the IE vertices mapping is essentially a problem of linear morphing of two polygons. Our solution to this polygon morphing problem is inspired by the approach of Bajaj et al. [5]. They assume 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). Thus, our handling of the candidate areas follow slightly different rules than originally proposed by Bajaj. Details and all special cases treaded can be found in [37].

4. Establishment of the edge-edge correspondence

Connecting the IE vertices of successive iso-components by the slicechords enables us to map the edges of related iso-components. Edges from connected vertices are mapped. By this way a list of each extracted iso-component edges mapping onto the edges of its successive iso-components are build up. Extracted lists of iso-intervals, iso-components and edge-to-edge mapping are saved and can be used anytime for the final visualization.

3.6.3 Visualization

Having determined the set of iso-components for each isointerval and a set of edge-to-edge mapping for each iso-component, resulting isocontour evolution can be visualized.

During the visualization, queries of the form query(isovalue, 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 isovalue is selected from the earlier time-slice. As depicted by the figure 3.23, a list of selected iso-components edges is traversed and the points C and E are interpolated out of the point-pairs A-B and D-F according to the desired isovalue. Point R is then interpolated out of C-E according to the desired time. Extracted R points are then connected by line, approximating resulting isocontour.

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 isocontour 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 isovalue is changed and fits into the range of isovalues 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.

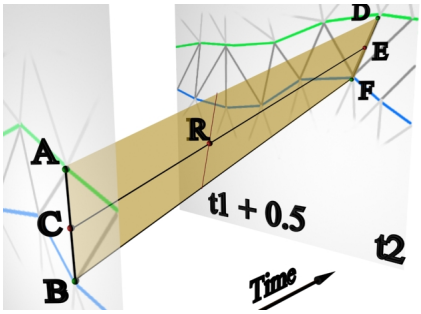


Figure 3.23: Visualization. First the vertices C and E are interpolated out at the edges of successive iso-components based on selected isovalue. User-selected time rules the interpolation of point R (on the connection of C and E).

3.6.4 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. Such iso-volume has only one envelope composed of 2D triangular mesh. This fact represents the complication: how to map the tetrahedral cells inside the iso-volume? Even if these geometric obstacles would be overcome, it is computationally not feasible to implement this method for 3D datasets.

3.6.5 Tests

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

The *Airfoil* dataset 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 dataset consists of 200 time steps each represented by the triangular adaptive mesh with 16 000 to 17 000 vertices per time step. 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 isocontour evolution in between them by our method. Figure 3.24 shows original data from the Airfoil dataset, dynamic mesh and evolving isocontours.

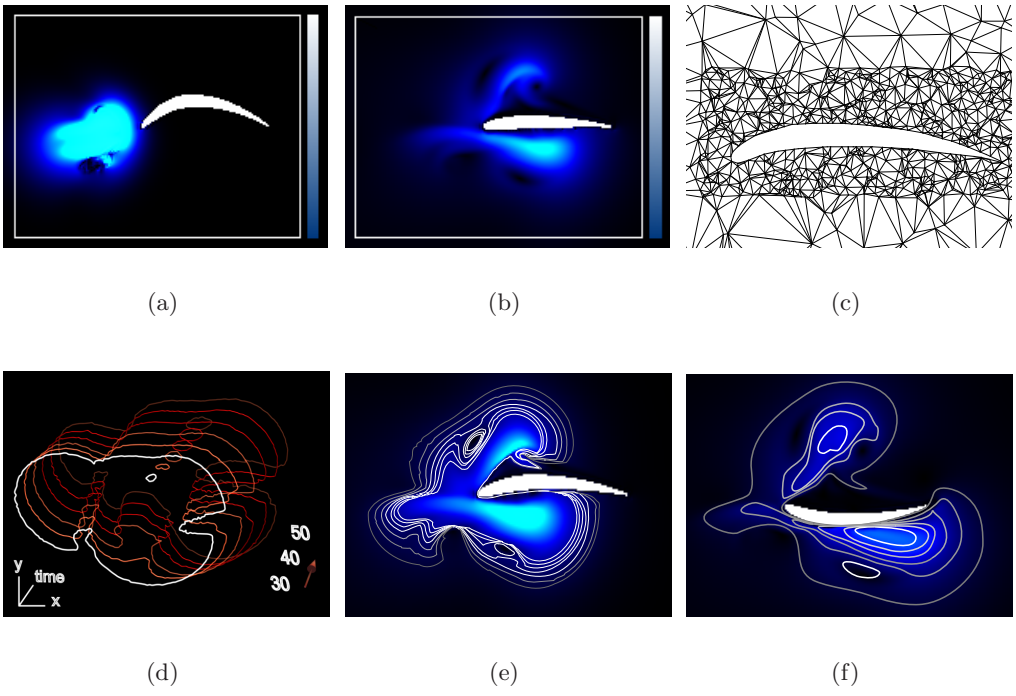
The *Payload* dataset originate from CFD simulation of payload release from under an aircraft wing. The dataset consists of 500 time steps from which every 10th time step has been saved and used as an input of our method. Number of the samples at each time step vary between 60 000 and 62 000. Simulation mesh adapts itself around falling payload as the simulation time proceeds.

Table 3.1 summarizes the total preprocessing times for both 2D datasets and provides extraction times for the selected isovalues and time steps.

		Dataset	
		Airfoil	Payload
Preprocessing time [minutes]		13.3	42.6
Extraction times			
isovalue	time step		
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 3.1: Performance results for both 2D dataset. Extraction times were measured for various isovalues and time steps.

Figure 3.25 shows the GUI of the application, implementing our isocontour extraction method. Isocontours rendered in the visualization window in figure 3.25 are extracted from the Payload dataset.



(a) (b) (c)

(d) (e) (f)

Figure 3.24: Airfoil dataset. (a,b) Input data (speed magnitude) at the time steps 10 and 100, (c) dynamic adaptive mesh (time step 80), (d) isocontour evolution computed by the proposed method between the time steps 30 and 50, (e)-(f) extracted isocontours at the time steps 74 and 186.

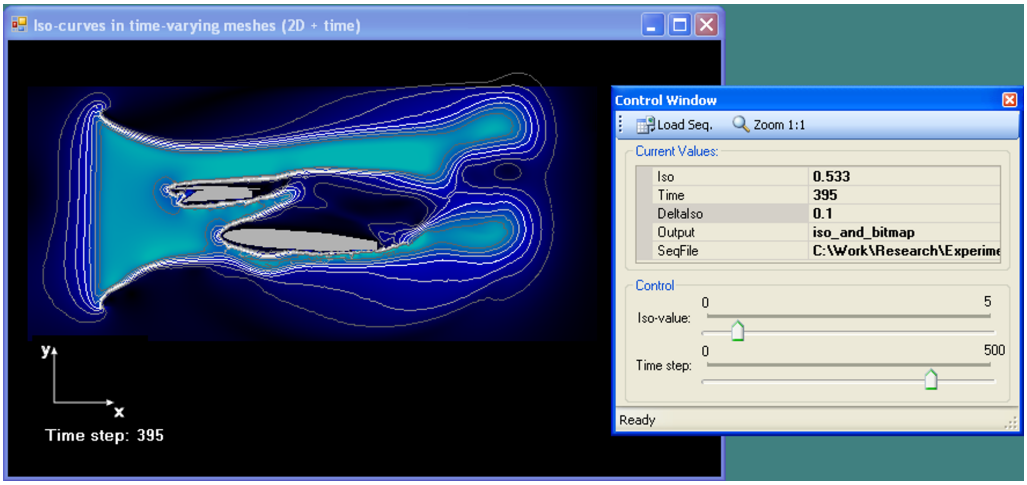


Figure 3.25: Isocontours extracted from payload-release simulation (iso-value=0.533, time step: 395). Blue rendered original data at the background (speed magnitude values) are just for illustration and comparison purposes.

3.7 Proposed method 2: Value-space approach

3.7.1 Overview

Z-Diamonds [38] is simple and efficient algorithm for extraction of the isosurfaces from the datasets with dynamic mesh. Z-Diamonds method works over triangular or tetrahedral meshes.

The method makes no assumption about the way a simulation mesh changes between adjacent time steps. We also do not attempt to reconstruct the evolving isosurfaces in between time steps defined in a dataset. Figure 3.27 outlines the principle of the Z-Diamonds method.

3.7.2 Preprocessing

Since a simulation mesh dynamically changes, we do not try to match the original mesh cells between adjacent time steps. Instead the mesh at each time step is preprocessed 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 3.26). Diamonds are not matched or tracked between adjacent time steps.

Pairing of the tetrahedral mesh cells into the diamonds has several advantages over the simple tetrahedral representation of a mesh. 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. During the visualization, geometry of the diamonds is loaded from disk.

The "five-vertex" representation of the diamonds reduces the I/O traffic during loading.

(a) (b)

Figure 3.26: 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.

In our algorithm, the diamonds for each time step are built and stored into 2D table (right side of the figure 3.27). To keep the diamonds with similar values close to each other in the table, we sort the diamonds by their minimum value and then store them one-by-one row-by-row. The result of the diamonds building process is a set 2D tables, one for each time step. Independent processing of the data for each time step suggests usage of the parallel computing.

Once the diamond building has finished, construction of a search structure for identification of active diamonds takes place. This search structure is the TSP tree of Shen et al. [45]. Instead of dividing volume, here we index a set of 2D tables spread along the time dimension. Thus, in our algorithm the TSP tree is quadtree rather than octree for each time step. So, each leaf node (x,y) of TSP tree corresponds to the elements at (x,y) in the diamond tables. TSP tree holds only the min/max values of the diamonds. Geometry of the diamonds is stored in the files, separately for each time step.

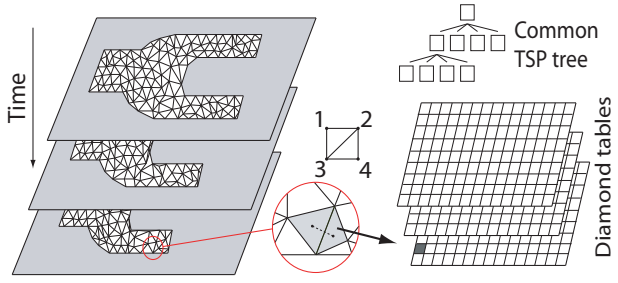


Figure 3.27: Outline of the principle of the Z-Diamonds method. Original mesh cells are paired into diamonds. Min/max and ID values of the diamonds are then organized in the diamond tables (one for each time step). A common TSP tree is used to index the diamonds stored in the diamond tables for all time steps.

3.7.3 Active cells identification

During the visualization phase the queries in the form (*isovalue, time step*) are processed. IDs of the active diamonds are extracted by traversing a common TSP tree using specified *isovalue* and *time step*. Geometry of the active diamonds is then dynamically loaded from a disk in an out-of-core fashion. Isosurface geometry can be extracted from the loaded active diamonds using techniques like Marching tetrahedra [54] or Marching Diamonds [3].

Since we extract isosurfaces only at the discrete time steps we do not deal with the topological changes of the isosurfaces at intermediate time steps. We assume that the datasets processed by our method are sampled sufficiently along the time dimension.

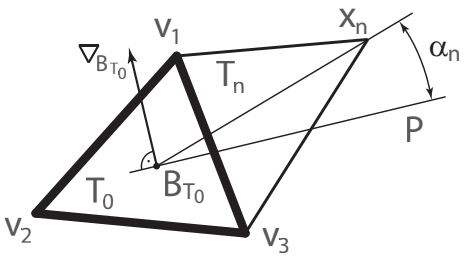
3.7.4 Optimizations

Optimized diamonds building

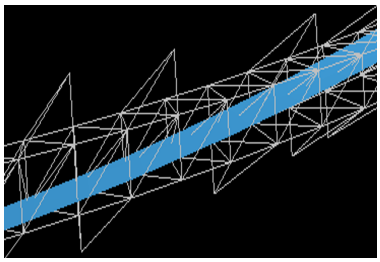
In the preprocessing phase of the Z-Diamonds method, the neighboring tetrahedra are paired into diamonds. When looking for a suitable neighbor of tetrahedron T_0 to create a new diamond, that neighbor of T_0 is chosen, whose additional vertex has scalar value closest to those of T_0. In this way, there is in average 13% of diamonds left, which contain only one tetrahedra intersected be the isosurface. The second tetrahedron is loaded from disk, but is left unused. This increases the amount of unused data loaded during visualization.

The way in which diamonds are built can be optimized by pairing those diamonds which have higher probability that the range of isosurfaces intersecting one of them will continue to the other one (i.e. following the gradient of data). Optimized algorithm has the following four steps:

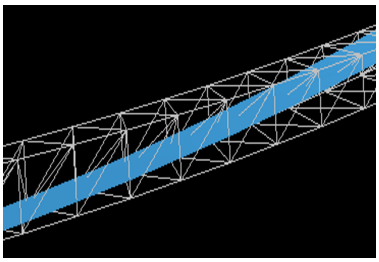
1. Interpolate gradient $\nabla B_{T_0} = \frac{1}{3}\sum_{i=1}^{3} \nabla v_i$ at barycenter B_{T_0} of T_0 (Fig. 3.28a).
2. Construct the plane P with normal vector ∇B_{T_0}, passing through the barycenter B_{T_0}.
3. For each neighbor T_i of T_0, do compute angle α_i between the plane P and line B_{T_0}, x_i.
4. Create the new 3D diamond from tetrahedron T_0 and its neighbor T_n, which has not been assigned to some other diamond yet, and which satisfies condition $\alpha_n = min(\alpha_i)$, i = 1,2,3,4.



(a)



(b)



(c)

Figure 3.28: (a) Optimized pairing of the triangular cells in 2D mesh. That neighboring triangle of T_0 is chosen which satisfies condition $\alpha_n = min(\alpha_i)$, i = 1,2,3,4. (b, c) A set of active diamonds for one isosurface. Active diamonds are rendered with (left) and without (right) optimized diamond building.

I/O optimized loading of diamonds

The process of loading geometry of diamonds consumes significant part of the processing time of each isosurface query (in average 70 to 80%). Thus, we propose optimized way of loading diamonds.

The optimization of geometry loading is achieved by the usual out-of-core practice of reading block of diamonds at once instead of separate access for each diamond. Once read, geometry of the diamonds is cached at the corresponding leaf nodes of the common TSP tree. A flag is kept at each TSP leaf node, indicating pre-loaded geometry, which can be used during visualization without additional disk access.

As mentioned before, the diamonds are sorted by their *min* value before their geometry is stored into files. This increases probability that cached diamonds are intersected by the same isosurface. Thus, usage of the pre-loaded and cached diamonds is maximized during isosurface visualization.

3.7.5 Tests

Original Z-Diamonds method and proposed optimizations have been implemented in C#.NET. The tests were run on Intel 2.4 GHz workstation with 2GB of RAM and ATI FireGL 5200 graphics adapter.

Table 3.2 provides overview of the datasets used for tests as well as the preprocessing times. Table 3.3 draws the isosurfaces extraction times during interactive visualization. Query execution times stated in the table 3.3 include time for active cells extraction by traversing a TSP tree, loading of active diamonds geometry from disk and extraction of isosurface geometry from loaded active diamonds.

Dataset	# of time steps	# of cells per time step	Dataset size	Preprocessing time	Size of preprocessed data
Motor	148	40k to 115k	3.2GB	86mins 32s	1.2GB
Wind tunnel	700	400k to 430k	7.5GB	120mins 8s	3.4GB

Table 3.2: Datasets used for during the tests of the Z-Diamonds method with the preprocessing times and sizes of the preprocessed datasets.

Dataset / isovalue / time step	Extraction time	# of active diamonds	# of triangles on the isosurface
Motor / 391.124 / 30	612ms	15,900	36,768
Motor / 392.345 / 130	514ms	6,741	15,398
Wind tunnel / 8.612 / 451	289ms	3,133	4,932

Table 3.3: Extraction times, numbers of active diamonds and numbers of triangles on the resulting isosurfaces for selected isovalues and time steps for *Motor* and *Wind tunnel* datasets.

Optimizations

Figure 3.29 shows the comparison of extraction times before and after proposed optimizations of the Z-Diamonds method for the first 100 time steps of the *motor* data set. Table 3.4 provides precise measurements of extraction times of the isosurfaces for selected isovalues and time steps for both data sets. The overall speedup of the Z-Diamonds method after the proposed optimizations ranges from 63 to 74%. Figure 3.30 shows isosurfaces extracted using the Z-Diamonds method.

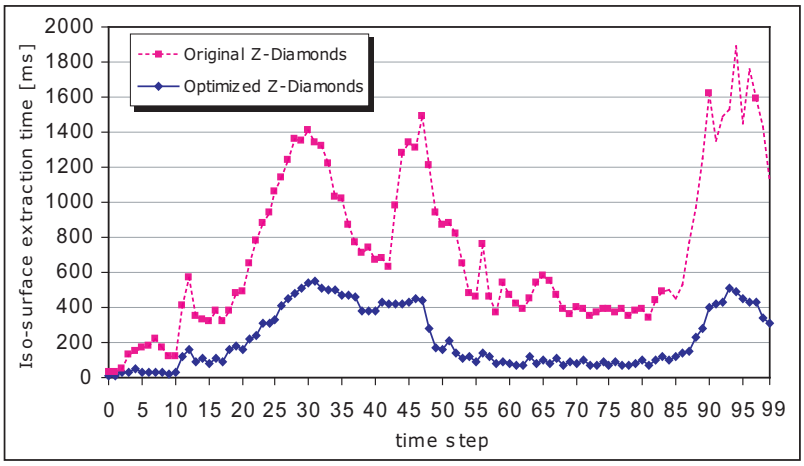
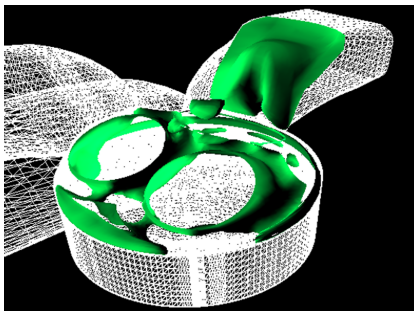


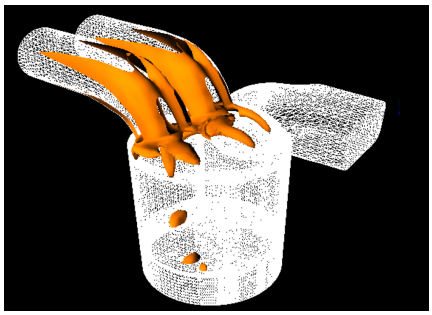
Figure 3.29: Extraction times for the first 100 time steps of the *Motor* data set.

Data set isoval. / time step	# of active diamonds	Extr. time original Z-Diamonds	Extr. time optim. Z-Diamonds	speed-up [%]
motor				
341.14 / 9	16,296	682 ms	212 ms	69%
392.2 / 12	10,102	412 ms	153 ms	63%
512.9 / 63	23,395	817 ms	238 ms	71%
wind tun.				
80.6 / 120	10,208	587 ms	147 ms	74%
96.5 / 213	8,296	496 ms	131 ms	73%

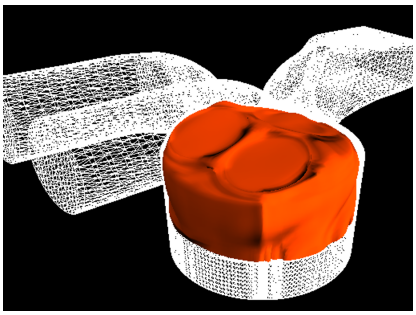
Table 3.4: Comparison of the extraction times of the original and optimized Z-Diamonds method for the *Motor* data set with calculated overall speedup in %.



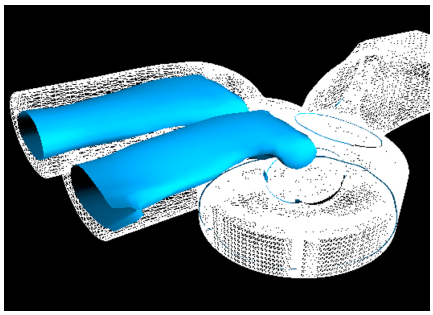
(a) Motor 3D (Temperature), Isovalue 341.14, time step 9



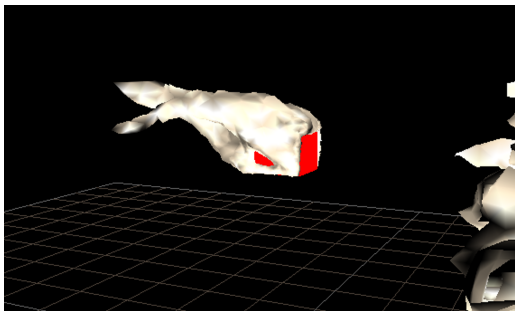
(b) Motor 3D (Temperature), Isovalue 391.142, time step 30



(c) Motor 3D (Temperature), Isovalue 512.9, time step 63



(d) Motor 3D (Temperature), Isovalue 392.345, time step 130



(e) Wind tunnel, Isovalue 8.612, time step 451

Figure 3.30: Isosurfaces for selected isovalues and time steps extracted by the Z-Diamonds method.

Chapter 4

Conclusions and future work

In this thesis we have presented our research made in the field of Isosurface Extraction. The focus has been put on two specific types of data-sets: static data-sets and time-varying data-sets with dynamic simulation meshes.

State of the art methods for Isosurface Extraction from static data-sets have been presented and evaluated in this work. Based on this evaluation, it has been concluded that advancements can be done in the space and time efficiency of the presented methods.

The method presented in this work (section 2.6) uses transformation of the initial static data into an alternative 1D space. Search structure for active cells identification is built over transformed data. Active cells are identified directly in the alternative space.

The proposed transformation of the input static data shortens preprocessing time considerably when compared to the best times achieved by the existing state of the art methods. The space requirements and extraction times of the proposed method are comparable to, or better than, the best existing space efficient method, while the search error of our method is considerably lower. The relative simplicity of the method proposed allows its easy implementation. The Isosurface Extraction from time-varying data-sets with dynamic meshes is the focus of the second part of this work. Dynamic meshing has numerous applications in engineering practice. While the methods for dynamic mesh generation were studied intensively during the '80s and '90s, there is still lack of suitable visualization techniques.

We have studied the principles and applications of dynamic mesh generation. This basic knowledge of the problematics is necessary to understand the requirements of suitable visualization methods.

Two principally different methods have been proposed for Isosurface Extraction from dynamic mesh data. Both presented methods address the same basic

problem: to find back lost correspondence of the mesh cells between the different mesh setups at consecutive time steps.

The first approach investigates geometrical and value similarity of the cells from adjacent time steps. Based on this, the correspondence of the cells is established. The method works for 2D data-sets with dynamic meshes. The biggest disadvantage of our first approach is complicated and long preprocessing before actual isocontours can be extracted. Therefore, our second proposed method targets the same goal of establishing the inter-time step cell correspondence but with shorter preprocessing, better algorithmic complexity and easier implementation.

The solution first transforms the cells in space efficient form into the min-max space. The correspondence is then calculated based on the cells data similarity, rather than on their geometric position. Once the cell correspondence is established, the search structure is built, exploiting temporal coherence of the input data.

Both proposed methods represent one of the first Isosurface Extraction methods published, which focus specifically on the data-sets with dynamic meshes. As we have observed from the reviews of the work presented, as well as after the cooperation with the engineers from practice, the need for such methods is high. However, very little research in the area has been done yet. Therefore, the work presented in this thesis, along with the very few existing methods, form the basis of a completely new chapter in the Isosurface Extraction.

The first topic that we would like to investigate in further research is metrics for measurement of the dynamic mesh similarity suitable for simulation data-sets. Out of this it should be possible to describe a model of the dynamic mesh, which will allow for space and time efficient construction/visualization of the evolving isosurfaces.

Appendix A

Test datasets and isosurfaces

All isosurfaces shown in this appendix were extracted using the newly proposed methods from this thesis.

Skull - static regular dataset, 256x256x256 cells, byte values
Rotational C-arm x-ray scan of phantom of a human skull. Siemens Medical Solutions, Forchheim, Germany. www.volvis.org

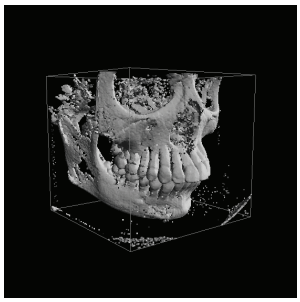


Figure A.1: Skull dataset, isovalue 60.

CT-head - static regular dataset, 256x256x113 cells, 16-bit integer values
CT study of a cadaver head. Data courtesy of North Carolina Memorial Hospital. Downloaded from Stanford volume data archive.

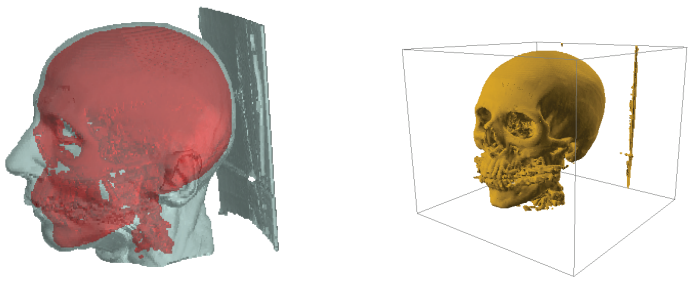


Figure A.2: CT-head dataset. Isovalues: gold 850, red 1994 and gray 850.

Vertebra - static regular dataset, 512x512x512 cells, 16-bit integer values
Rotational angiography scan of a head with an aneurysm. Michael Meiner, Viatronix Inc., USA.

Figure A.3: Vertebra dataset. Isovalue 1000.

FiveJets - time-varying dataset (static mesh), 128x128x128 cells, 32-bit float
2000 time steps. Energy values datasets of five jets entering rectangular region. Downloaded from the Volume data repository of UC Davis.

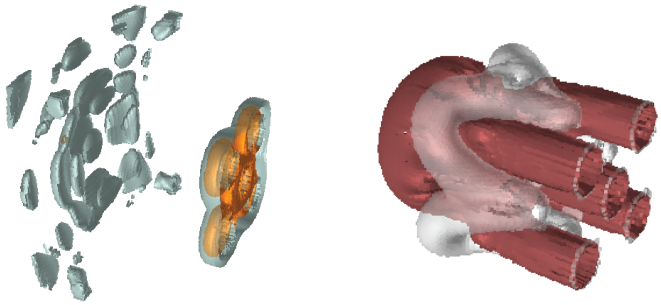


Figure A.4: FiveJets dataset. Isovalues: (left) time step=364 orange 250235, gray 253290, (right) time step=1763 red 254052 gray 251200.

Isabel - time-varying dataset (static mesh), 500x500x100 cells, 32-bit float
48 time steps. National Center for Atmospheric Research, USA.

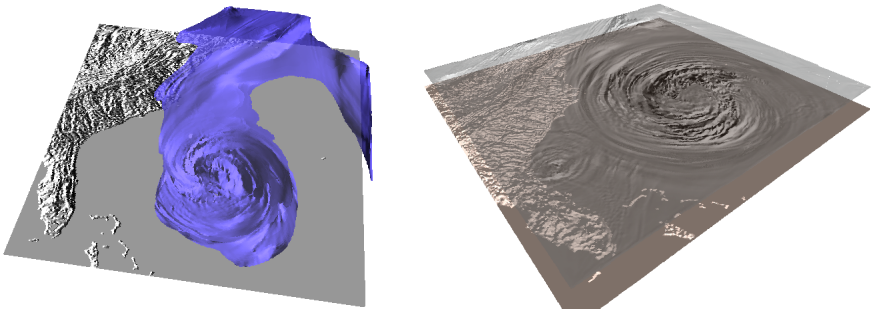


Figure A.5: Isabel dataset. Temperature isovalue 22 at (a) time step 4 (b) time step 8.

Terashake - time-varying dataset (static mesh), 750x375x100 cells, 32-bit float. 227 time steps. Magnitude 7.7 earthquake along San Andreas fault, California. Data courtesy of San Diego Supercomputer Center, USA.

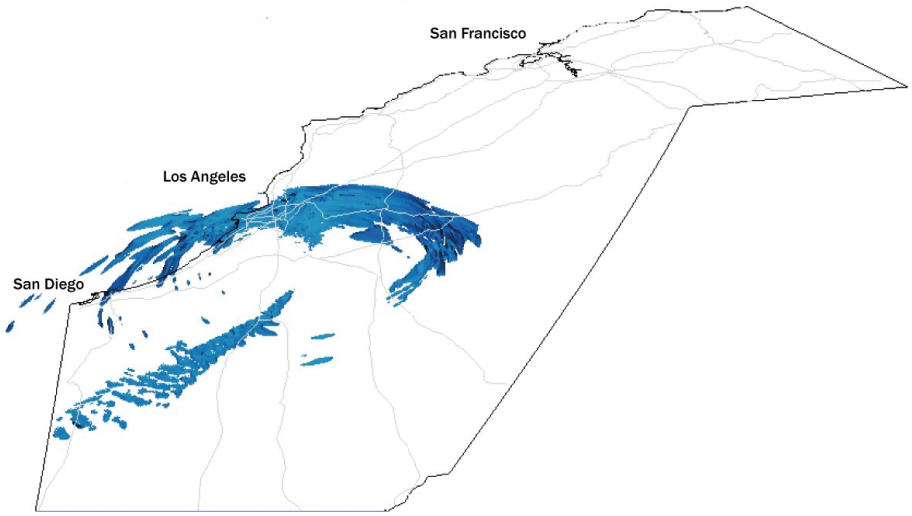


Figure A.6: Terashake dataset. Velocity magnitude isovalue 0.052.

Motor3D - time-varying dataset (dynamic mesh), 40000 - 115000 cells, 32-bit float. 148 time steps. Total size of the dataset is 3.2GB. Simulation of the combustion process. Data courtesy of Randy P. Hessel from Engine Research Center, University of Wisconsin-Madison, USA.

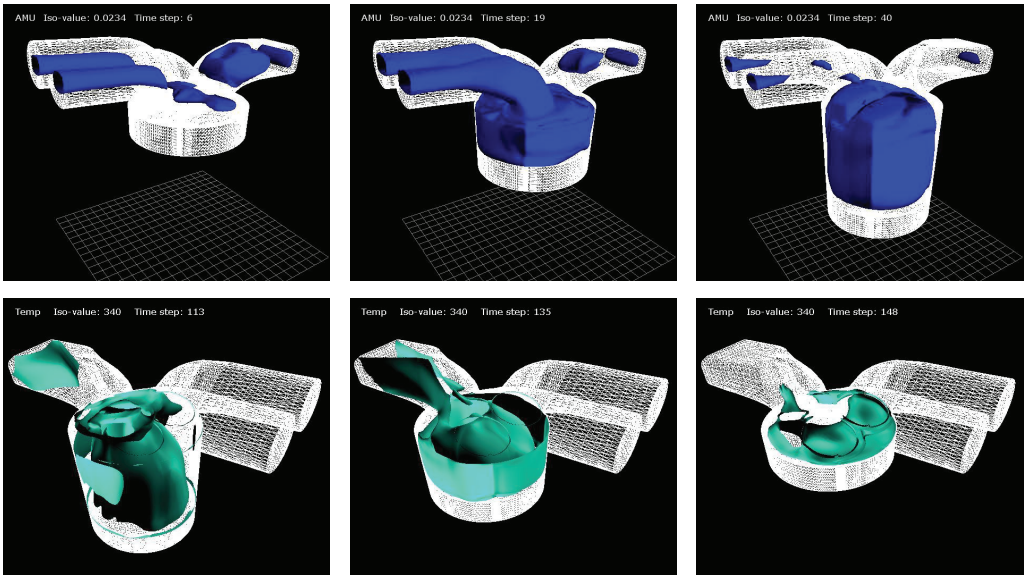


Figure A.7: Motor3D dataset. (Top row) AMU isosurfaces for isovalue 0.0234 at the time steps (from left to right) 6, 19 and 40. (Bottom row) Isosurfaces of the temperature 340 degrees Celsius at the time steps 113, 135 and 148.

WindTunel - time-varying dataset (dynamic mesh), 400 000 to 430 000 cells, 32-bit float. 700 time steps. Total size of the dataset is 7.5GB. Simulation of the debris in the flow. Data courtesy of New Technologies Research Center, University of Wes Bohemia, Czech Republic.

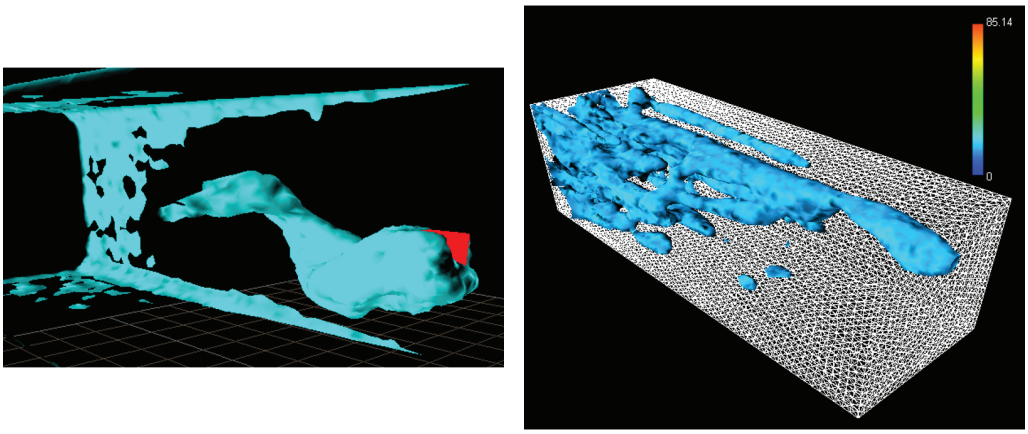


Figure A.8: WindTunel dataset. Isosurfaces of (left) Total pressure 12.711 at time step 688, (right) wind velocity magnitude around debris isovalue 17 at time step 28.

Appendix B

Implementation details

The pseudo-C++ code is provided in this appendix to facilitate implementation of our method for isosurface extraction from static datasets proposed in the section 2.6.

A C++ class `Helix` implements the construction and extraction algorithms. Helix structure is internally stored as a single list of helix records.

```
struct Record {
  float  t;
  int[]  ids;
};
```

Additionally, the class `Helix` contains the following member variables, which aid in the construction and the extraction process.

```
Record[] helix;    // list of records (helix)
int   n_rev = 256; // total number of helix revolutions
int   n_rec = 0;   // counter of helix records
int[] dict;        // search dictionary

// delta t per one helix revolution
float t_rev = 1.0 / n_rev;
```

Construction

Method `construct` takes a list and the number of cells as its input parameters. We assume that the functions `sort_by_t()` and `floor()` (returns the largest integer less than or equal to the specified number) are provided. Function `compute_t()` computes helix parameter t of a given cell according to the Eq. 2.2 to 2.4.

```
void Helix::construct( Cell* cells, int N ) {
  int rev = 0;   // index of current revolution
  int prev = 0;  // index of prev. revolution

  // Compute t for each cell
  foreach ( Cell c in cells )
    compute_t(c);
  sort_by_t( cells, N );

  // Create search dictionary
  dict = new int[n_rev + 1];
  for (int i = 0; i < (n_rev+1); i++)
    dict[i] = -1;

  // Construct helix
  for (int i = 0; i < N; ) {
    Record r = new Record();
    r.t = cells[i].t;

    // Store all consecutive cells with the same
    // parameter t into record r.
    while ( i < N && cells[i].t == c.t )
      r.ids->add(cells[i++].id);

    // Update search dictionary
    rev = floor( r.t / t_rev );
    if (rev != prev)
      dict[rev] = n_rec;

    helix->add(r);
    n_rec++;
    prev = rev;
  }
}
```

Extraction

Desired isovalue `q` is the only parameter of the extraction method. Helix revolutions are traversed in top-bottom order in the outer loop using search dictionary. Active cells from current revolution are collected by the function `add_cells` in the inner loop.

```
void Helix::search( float q ) {
  int rev = n_rev + 1; // current revolution
  int rec;             // index of current helix record
  float limit;         // t limit for current revolution

  float t_0 = ((q - min_min) / (min_max - min_min)) * t_rev;
  float t_q = (q - max_min) / (max_max - max_min);
  int   final = ( t_q / t_rev ) + 1;
```

```
  while ( rev > final && rev > 0 ) {
    rec = dict[--rev];

    if ( rec >= 0 ) {
      limit = ( rev * t_rev ) + t_0;

      while ( helix[rec].t < limit && rec < n_rec )
        isosurface->add_cells( helix[rec++].ids );
    }
  }
}
```

Appendix C

Practical applications and review

Within the scope of the research presented in this thesis the Z-Diamonds method (section 3.7) has been used and validated by the engineers in practice. During two months of cooperation the method has been applied on the 2D and 3D transient CFD datasets from the combustion simulation generated by Dr. Randy Hessel from the Engine Research Center, University of Wisconsin-Madison, USA.

Various pictures and movies of the evolving isosurface of the temperature, pressure and AMU (total viscosity) have been produced. Accuracy and quality of the produced isosurfaces have been validated and assessed by Dr. Hessel. His review is provided in this appendix.

University of Wisconsin-Madison
Engine Research Center

1500 Engineering Dr.
Madison, WI 53706
erc_office@engr.wisc.edu
608.263.2735 *(phone)*
608.263.9870 *(fax)*
http://www.erc.wisc.edu

Ing. Slavomir Petrik
Centre for Computer Graphics and Data Visualization
Computer Science Dept.
University of West Bohemia in Pilsen
Czech Republic

Review of Iso-Surface Visualizations Created by Slavomir Petrik

Dear Project Committee:

My name is Dr. Randy Hessel. I am a Research Scientist at the University of Wisconsin-Madison located in Madison, Wisconsin, USA. I work in the University's Engine Research Center. Here we have been performing flow and combustion research on internal combustion engines since the 1940s. Two of my specialties are computer modeling and visualization of flow and combustion in engines.

Previously I had sent Slavomir Petrik results from a transient CFD simulation of flow in a three valve research engine. Nodal connectivity and the number of computational cells representing the flow domain changed during the simulation. The entrance to the intake ports and exit of the exhaust port were modeled as pressure boundaries. Flow responded to piston and valve motion. These moving surfaces governed the flow and were the source of changing nodal connectivity and changing number of cells.

I am very pleased with the quality of iso-surface visualization that Slavomir is able to produce. In my work I use commercial visualization software packages. To test the accuracy of an iso-surface created by Slavomir, I read the same dataset that I sent him into one of the commercial packages. I then recreated the web site image, http://herakles.zcu.cz/~spetrik/media/zdiamonds/temp__12_392_228.jpg. The commercially generated image is included in this mailing. The two images are essentially indistinguishable. There are minor differences, but I can not say that one is more accurate than the other.

I have also compared the quality of the two images on my computer screen. Slavomir's image quality is superior. With proper use of shading his image brings out more of the iso-surface three-dimensionality. Features easily overlooked in the commercial image stand out in Slavomir's image. (One example of this is the waviness in the front-bottom of the iso-surface.) While bringing out more three-dimensionality, Slavomir's image is also smoother, which is visually more appealing. Similar image quality is apparent in the animations found on the web site.

November 23, 2007
Page 2

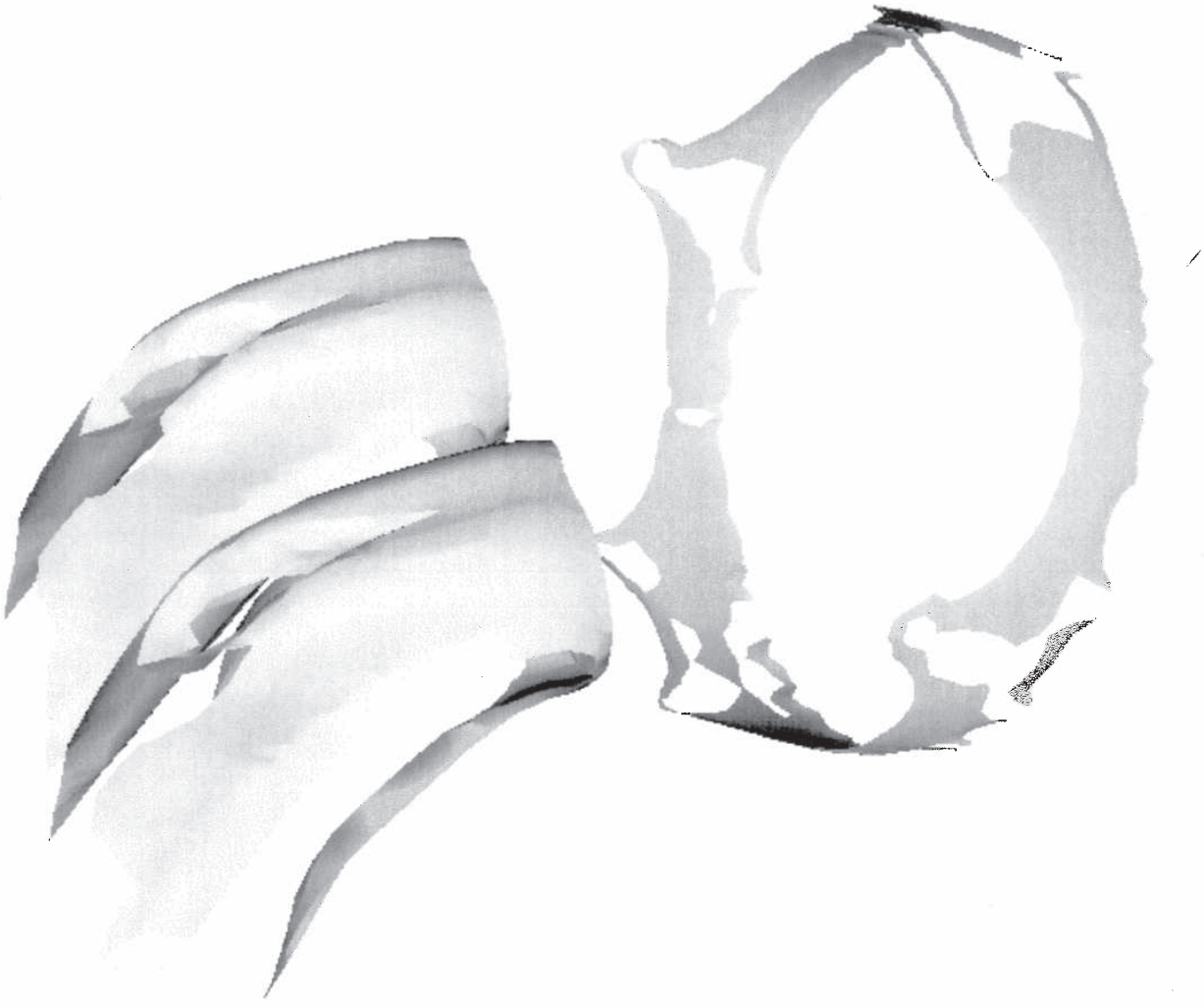
The user interface, which is demonstrated on the web site, also looks to be very user-friendly. And, even more importantly, the response time for transient iso-surface creation and translation, rotation and zoom response to mouse movement is very fast.

I use iso-surfaces in my research extensively to track how scalars change in magnitude and location in the transient flow field of an internal combustion engine. Slavomir has demonstrated that his tool is can perform very well for this type of application with transient data with varying connectivity.

Respectfully,

Randy Hessel

Randy P. Hessel, Associate Scientist
University of Wisconsin-Madison
Engine Research Center



Appendix D

List of author's publications

Related publications

1. Petrik S., Skala V.: Space and time efficient isosurface extraction, *Computers and Graphics*, 32(6):704-710, 2008 [Impact factor 0.787].

2. Petrik S., Skala V.: Z-Diamonds: A Fast Isosurface Extraction Algorithm for Dynamic Meshes. *Proceedings of the IADIS Computer Graphics and Visualization 2007*, Lisbon, Portugal, pp. 67-74, 2007, ISBN 978-972-8924-39-3.

3. Petrik S., Skala V.: Isocontouring in Time-varying Meshes. *Proceedings of SCCG 2007*, Budmerice, Slovakia, pp. 216-223, 2007, ISBN 978-80-223-2292-8. Also published by ACM Press.

4. Vokorokos L., Petrik S.: Proposed Parallel and Distributed Architectures for Behavioral Animation. *Proceedings of 9th IEEE International Conference on Intelligent Engineering Systems*, Cruising on Mediterranean Sea, 2005 [Thomson Reuters (ISI) proceedings].

5. Vokorokos L., Bliš̆tan Peter, Petrik S., Ádam N.: Utilization of Parallel Computer System for Modeling of Geological Phenomena in GIS. *Metalurgy Journal*, vol. 43, no. 4, pp. 287-291, 2004 [Impact factor 0.247].

6. Vokorokos L., Petrik S., Ádam N.: Aplikácia 3D informačných systémov v oblasti prevádzky výrobných technológií. *Proceedings of the 6th International Scientific Conference: New Trends in the Operation of Production Technology 2003*, Prešov, Slovensko, pp. 426-431, 2003.

Technical Reports

7. Petrik S., Skala V.: Isosurface extraction in time-varying data (State of the Art and Future Research), *Technical Report No. DCSE/TR-2007-06*, University of West Bohemia in Plzen, Czech Republic, June 2007.

8. Petrik S., Skala V.: Report on Videoconferencing Systems, *Technical Report No. DCSE/TR-2006-04*, University of West Bohemia in Plzen, Czech Republic, June 2006. Updated on May 18, 2007.

Project assignment

- 2006 - present: *VIRTUAL* - Virtual Research-Educational Center of Computer Graphics and Visualization, MSMT Czech Rep. No: 2C 06002, http://virtual.zcu.cz,
- 2005 / 2006: *3DTV* - Integrated Three-Dimensional Television - Capture, Transmission and Display, FP6-2003-IST-2, Network of Excellence, No:511568, http://3DTV.zcu.cz,
- 2005 / 2006: *INTUITION* - Network of Excellence on VIrtual Reality aNd VirTUal Environments ApplIcaTIONs for Future Workspaces, FP6-2003-IST-2, No:507248-2, http://intuition.zcu.cz.

Bibliography

[1] F. Alauzet, P. J. Frey, P. L. George, and B. Mohammadi. 3D transient fixed point mesh adaptation for time-dependent problems: Application to CFD simulations. *Journal of Computational Physics*, 222(2):592–623, 2007. [cited at p. 39]

[2] A. A. Amsden. 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, May 1997. [cited at p. 44]

[3] J. C. Anderson, J. Bennett, and K. I. Joy. Marching Diamonds for unstructured meshes. In *IEEE Visualization 2005*, 2005. [cited at p. 68]

[4] C. L. Bajaj, V. Pascucci, and D. R. Schikore. Seed sets and search structures for optimal isocontour extraction. Technical report, Texas Institute of Computational and Applied Mathematics, 1999. [cited at p. 14]

[5] Ch. Bajaj, E. J. Coyle, and K.-N. Lin. Arbitrary topology shape reconstruction from planar cross sections. *Graph. Models Image Process.*, 58(6):524–543, 1996. [cited at p. 63]

[6] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. [cited at p. 16]

[7] U. Bordoloi and H.-W. Shen. Space efficient fast isosurface extraction for large data sets. In *IEEE Visualization'03*, pages 201–208, 2003. [cited at p. 15, 20, 21, 23, 24, 26, 27, 28, 29, 31]

[8] J.A. Brentzen and N. Christensen. Hardware accelerated point rendering of isosurfaces. In *Proc. WSCG 2003*, 2003. [cited at p. 24]

[9] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Comput. Geom. Theory Appl.*, 24(2):75–94, 2003. [cited at p. 14]

[10] P. Cavallo, A. Hosangadi, and V. Ahuja. Transient simulations of valve motion in cryogenic systems. In *AIAA Fluid Dynamics Conference and Exhibit*, 2005. [cited at p. 44]

[11] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997. [cited at p. 15, 18, 21, 27]

[12] H. E. Cline, W. E. Lorensen, S. Ludke, C. R. Crawford, and B. C. Teeter. Two algorithms for the three-dimensional reconstruction of tomograms. *Medical Physics*, 15:320–327, 1988. [cited at p. 24]

[13] C. S. Co, B. Hamann, and K. I. Joy. Iso-splatting: A Point-based Alternative to Isosurface Visualization. In *Proceedings of Pacific Graphics 2003*, pages 325–334, 2003. [cited at p. 24]

[14] H. Doleisch, M. Mayer, M. Gasser, P. Priesching, and H. Hauser. Interactive feature specification for simulation data on time-varying grids. In *SimVis'05*, pages 291–304, 2005. [cited at p. 58, 59]

[15] J. Donea, A. Huerta, J.-Ph. Ponthot, and A. Rodriguez-Ferran. *Arbitrary Lagrangian-Eulerian methods, Encyclopedia of Computational Mechanics*. John Wiley and Sons, 2004. [cited at p. 39, 40, 43]

[16] J. Donea, A. Huerta, J.-Ph. Ponthot, and A. Rodriguez-Ferran. *Modeling Flows in Moving and Deforming Zones, Fluent Manual*. Fluent Inc., 2005. [cited at p. 42, 43]

[17] L.E. Eriksson. Practical three-dimensional mesh generation using transfinite interpolation. *SIAM J. Sci. Stitist. Comput.*, 6(3):712–741, 1985. [cited at p. 43]

[18] M.S. Gadala, M.R. Movahhedy, and J. Wang. On the mesh motion for ale modeling of metal forming processes. *Finite Elem. Anal. Des.*, 38(5):435–459, 2002. [cited at p. 43]

[19] Richard S. Gallagher. Span filtering: an optimization scheme for volume visualization of large finite element models. In *VIS '91: Proceedings of the 2nd conference on Visualization '91*, pages 68–75, Los Alamitos, CA, USA, 1991. IEEE Computer Society Press. [cited at p. 15, 17]

[20] B. Gregorski. Adaptive extraction of time-varying isosurfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):683–694, 2004. Student Member-J. Senecal and Member-M. A. Duchaineau and Member-K. I. Joy. [cited at p. 54]

[21] J. P. Grossman and William J. Dally. Point sample rendering. In *Proc. Rendering Techniques 98 (Eurographics)*, pages 181–192, 1998. [cited at p. 24]

[22] R. Haber and J.F. Abel. Discrete transfinite mappings for the description and meshing of three-dimensional surfaces using interactive computer graphics. *Int. J. Numer. Methods Eng.*, 18(1):41–66, 1982. [cited at p. 43]

[23] A. Huerta and W.K. Liu. Vicious flow structure interactions. *J. Press. Vessel Technol.-Trans.*, 110(1):15–21, 1988. [cited at p. 42]

[24] A. Huerta and W.K. Liu. Vicious flow with large free-surface motion. *Comput. Methods Appl. Mech. Eng.*, 69(3):277–324, 1988. [cited at p. 42]

[25] J. Huetink, P.T. Vreede, and J. van der Lugt. Progress in mixed eulerian-lagrangian finite element simulation of forming processes. *Int. J. Numer. Methods Eng.*, 30(8):1441–1457, 1990. [cited at p. 42]

[26] Takayuki Itoh and Koji KOYAMADA. Isosurface generation by using extrema graphs. In *VIS '94: Proceedings of the conference on Visualization '94*, pages 77–83, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. [cited at p. 13]

[27] Takayuki Itoh and Koji Koyamada. Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, 1995. [cited at p. 13]

[28] Takayuki Itoh, Yasushi Yamaguchi, and Koji Koyamada. *Fast Isosurface Generation Using the Cell-Edge Centered Propagation Algorithm*. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2000. [cited at p. 13]

[29] Takayuki Itoh, Yasushi Yamaguchi, and Koji Koyamada. Fast isosurface generation using the volume thinning algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(1):32–46, 2001. [cited at p. 14]

[30] M. Levoy and T. Whitted. The use of points as display primitives. Technical report, Technical report 85-022, Computer Science Department, University of North Carolina at Chapel Hill, 1985. [cited at p. 24]

[31] W.K. Liu and H.G. Chang. Efficient computational procedures for long-time duration fluid-structure interaction problems. *J. Press. Vessel Technol.-Trans.*, 106:317–322, 1984. [cited at p. 42]

[32] Y. Livnat, H.-W. Shen, and Ch. R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996. [cited at p. 15, 16, 24]

[33] Yarden Livnat and Xavier Tricoche. Interactive point-based isosurface extraction. In *Proc. IEEE Visualization '04*, pages 457–464, 2004. [cited at p. 24]

[34] W. E. Lorensen and H. E. Cline. Marching Cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87*, pages 163–169, New York, NY, USA, 1987. ACM Press. [cited at p. 8, 12]

[35] N. Mukherjee. High quality bi-linear transfinite meshing with interior point constraints. *Proc. 15th Int. Meshing Roundtable*, pages 309–324, 2006. [cited at p. 43]

[36] S. Parker, P. Shirley, Y. Livnat, Ch. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *IEEE Visualization '98*, pages 233–238, 1998. [cited at p. 53]

[37] S. Petrik and V. Skala. Iso-contouring in time-varying meshes. In *Proceedings of the SCCG 2007*, pages 216–223, 2007. [cited at p. 63]

[38] S. Petrik and V. Skala. Z-Diamonds: A fast iso-surface extraction algorithm for dynamic meshes. In *To appear in proceedings of the IADIS Computer Graphics and Visualization 2007*, 2007. [cited at p. 66]

[39] W.E. Pracht. Calculating three-dimensional fluid flows at all flow speeds with an eulerian-lagrangian computing mesh. *J. Comput. Phys.*, 17:132–159, 1975. [cited at p. 42]

[40] E. Reinhard, Ch. Hansen, and S. Parker. Interactive ray tracing of time varying data. In *EGPGV '02: Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pages 77–82, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association. [cited at p. 53]

[41] A. Rodriguez-Ferran, A. Perez-Foguet, and A. Huerta. Arbitrary lagrangian-eulerian (ale) formulation for hyperelastoplasticity. *Int. J. Numer. Methods Eng.*, 53(8):1831–1851, 2002. [cited at p. 42]

[42] S. Rusinkiewicz and M. Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *Proc. SIGGRAPH '00*, pages 343–352, 2000. [cited at p. 24]

[43] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing features and tracking their evolution. *Computer*, 27(7):20–27, 1994. [cited at p. 62]

[44] H.-W. Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 159–166, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press. [cited at p. 50]

[45] H.-W. Shen, L.-J. Chiang, and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 371–377, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press. [cited at p. 51, 52, 67]

[46] H.-W. Shen, Ch. D. Hansen, Y. Livnat, and Ch. R. Johnson. Isosurfacing in span space with utmost efficiency (ISSUE). In Roni Yagel and Gregory M. Nielson, editors, *IEEE Visualization '96*, pages 287–294, 1996. [cited at p. 15, 16, 21, 27, 50]

[47] H.-W. Shen and Ch. R. Johnson. Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In *IEEE Visualization'95*, pages 143–150, 1995. [cited at p. 15, 17]

[48] Q. Shi and J. JaJa. Isosurface extraction and spatial filtering using persistent octree. In *IEEE Visualization 2006*, Los Alamitos, CA, USA, 2006. IEEE Computer Society Press. [cited at p. 55, 56]

[49] D. Silver and X. Wang. Tracking and visualizing turbulent 3d features. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):129–141, 1997. [cited at p. 62]

[50] D. O. Snyder and J. Sverdrup. Dynamic meshing models store separation from transonic aircraft. In *Journal articles by fluent software users*. Fluent Inc., 2005. [cited at p. 45]

[51] K. Stein, T. Tezduyar, and R. Benney. Computational methods for modeling parachute systems. *Computing in Science and Engg.*, 5(1):39–46, 2003. [cited at p. 45]

[52] P. Sutton and Ch. D. Hansen. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (t-bon). In *VIS '99: Proceedings of the conference on Visualization '99*, pages 147–153, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press. [cited at p. 50]

[53] J. F. Thompson, B. K. Soni, and N. P. Weatherill. *Handbook of Grid Generation*. CRC Press, New York, 1999. [cited at p. 39, 42]

[54] G. M. Treece, R. W. Prager, and A. H. Gee. Regularised marching tetrahedra: improved iso-surface extraction. *Computers and Graphics*, 23(4):583–598, 1999. [cited at p. 68]

[55] M. van Kreveld, R. van Oostrum, Ch. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry*, pages 212–220, New York, NY, USA, 1997. ACM Press. [cited at p. 14]

[56] M. van Kreveld, R. van Oostrum, Ch. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry*, pages 212–220, New York, NY, USA, 1997. ACM Press. [cited at p. 14]

[57] B. von Rymon-Lipinski, N. Hanssen, T. Jansen, L. Ritter, and E. Keeve. Efficient point-based isosurface exploration using the span-triangle. In *IEEE Visualization'04*, pages 441–448, Washington, DC, USA, 2004. IEEE Computer Society. [cited at p. 15, 18, 27]

[58] Lujin Wang, Ye Zhao, Klaus Mueller, and Arie E. Kaufman. The magic volume lens: An interactive focus+context technique for volume rendering. In *Proc. IEEE Visualization '05*, pages 367–374, 2005. [cited at p. 24]

[59] K. W. Waters and Ch. S. Co. Using difference intervals for time-varying isosurface visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1275–1282, 2006. [cited at p. 52, 54, 55]

[60] K. W. Waters, Ch. S. Co, and K. I. Joy. Isosurface extraction using fixed-sized buckets. In *IEEE VGTC Symposium on Visualization*, pages 207–214, 2005. [cited at p. 15, 19, 21, 27, 29]

[61] Ch. Weigle and D. C. Banks. Complex-valued contour meshing. In Roni Yagel and Gregory M. Nielso, editors, *IEEE Visualization '96*, pages 173–180, 1996. [cited at p. 52]

[62] Ch. Weigle and D. C. Banks. Extracting iso-valued features in 4-dimensional scalar fields. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 103–110, New York, NY, USA, 1998. ACM Press. [cited at p. 52, 53]

[63] J. Wilhelms and A. van Gelder. Octrees for faster isosurface generation. *ACM Trans. Graph.*, 11(3):201–227, 1992. [cited at p. 12, 13, 50, 51]