Interactive Volume Rendering of Large Scalar Fields On Mobile Devices Using Subsampling Techniques

A Project Report

Submitted in partial fulfilment of the

requirements for the Degree of

Master of Engineering

in

Computer Science and Engineering

by

Debasish Tapna

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

JUNE 2011

To my

Mom, Dad and *Sister*

Acknowledgements

I express my profound gratitude and sincere thanks to Prof. Vijay Natarajan for his valuable guidance, caring supervision and priceless feedback for the progress of this project. I am immensely thankful to Prof. K. Gopinath for providing the mobile hardware to work on. He also provided his valuable suggestion and feedback. I am thankful for the superior lab facilities at the Visualization and Graphics Lab, CSA, IISc. I am also thankful to the students of IISc for their unparalleled cooperativeness and enthusiasm.

Abstract

We study the problem of subsampling of 3D scalar fields. Software tools for visualizing 3D scalar fields usually have processing and memory requirements depending upon the size of the scalar field for getting desired interactiveness. Small computing systems such as mobile devices have resource constraints. When the size of the scalar field is large or exceeds the available memory, the visualization tool is unable to achieve a good interactive rate. Appropriate subsampling of the scalar field can alleviate such a problem. The subsampled scalar field should have all the important features of the original scalar field. Topological properties in scalar fields carry important information about the scalar fields itself. So preserving these properties in the subsampled scalar field can help in retaining important features of the original large scalar field. We built a volume render for a mobile device. We propose three subsampling algorithms based on retaining the critical points. A method to evaluate the subsampled scalar fields based on mapping of branches using proximity. We also showed how our algorithms and evaluation method works on some of the dataset.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Scalar Fields

Experiments and simulations such as computational fluid dynamics generate 3D scalar fields. Study of objects or spaces in which scanning techniques such as laser scans, satellite imaging, 3D radar imaging and medical imaging etc are used also produces 3D scalar field. The scalar field is available as an array of samples(values) over the domain polygonal mesh. These meshes can be structured or unstructured. In this paper we will consider structured grids only. The methods of visualizing the scalar field are isosurface extraction[3] and volume rendering[3]. Isosurface extraction involves finding of positions in the scalar field where the value is at least equal to a given value and rendering them onto the screen. Volume rendering is the technique where beneficial portions of the scalar field are identified and rendered onto the screen.

1.2 Mobile Devices for Visualization

Mobile devices can be thought as small computing systems where the processing power, memory size and memory bandwidth is relatively less compared to the desktop computers. Most of these devices don't even support floating-point operations. Use of mobile devices has grown over the years due to its small form factor, ubiquity, low power

consumption and improvement in the processing and graphics performances. There have been several advances in application development on such devices. Khronos Group maintains the 3D application programming interface (API) OpenGL ES for 3D graphics programming in mobile and embedded devices. With these capabilities, scientific visualization is possible. The doctors and medical practitioners can use it for displaying CT and MRI scans to their patients with little effort. Scientists can use them for visualizing their simulated data when they are outside their laboratory. Engineers can use it for reference when they are on the fields away from bulky dedicated machines. Thus there is high potential of usage of such devices for scientific visualization purpose.

1.3 Subsampling

Subsampling is most common in 2D image processing, where the images are downscaled for space and low resolution requirements. It enables to accelerate the visualizing applications when the user doesn't need much of the detail. The subsampled scalar field should have the required amount of features from the original scalar field. Subsampled scalar field enables high degree of interactivity since it minimizes the amount of processing, data transfer between the processor and the memory and also the total memory requirement. Analyzing large scalar field may require huge amount of computation and completion time. If approximate results will serve the purpose then subsampled scalar field holds good i.e. it can be analyzed to get the results in much lesser time. Subsampling can be done in the preprocessing stage and its benefits are seen in the processing stage.

1.4 Topological Properties

Subsampling a scalar field can introduce geometrical error and or topological error. The topological properties of the scalar field capture features, which are useful in analysis and visualization of scalar fields. G. H. Weber et al.[4] and Takahashi et al.[5] used it

in volume rendering for creation of the transfer function. V. Natarajan et al.[6] used topological properties to analyze volumetric data. Hilaga et al.[7] showed that 3D shape datasets can be compared and matched based on these properties. Bajaj et al.[8] used it for simplification of data. We will focus on preventing topological errors in this paper.

1.5 Organization of this Thesis

The rest of the report is arranged as follows: section 2 gives the details of the volume renderer we built for a mobile device and the challenges we faced; chapter 3 provides the related work done in subsampling of scalar fields; chapter 4 provides an overview of topological properties of scalar fields; chapter 5 gives details of our work; chapter 6 showcases the results and finally chapter 7 provides the conclusion from the result and the future work.

Chapter 2

Volume Rendering on Mobile Devices

We implemented a texture-based axis aligned volume renderer for the mobile device *Nokia N900*.

2.1 Configuration of Mobile Devices

Mobile devices usually have low-powered single chip with multiple IP cores for different purposes such as general purpose processor, dedicated graphics processor, digital signal processor(s), and many other cores for encoding and decoding etc. The speed of the

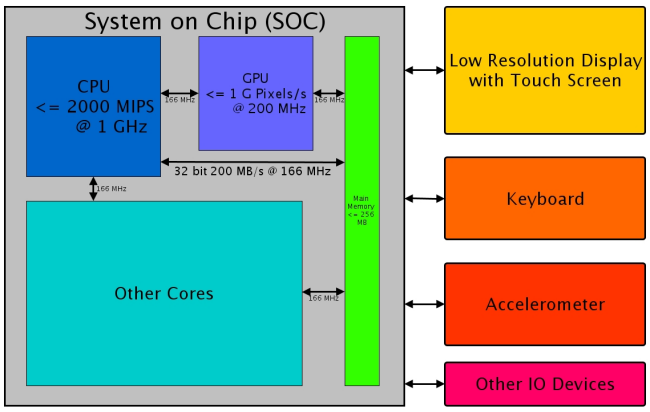

Figure 2.1: Diagram of a sample System on Chip (SOC) used in mobile devices

computing processor is at most 1 GHz (up to 2000 MIPS) with system bus frequency up to 166 MHz [12]. Not all mobile devices have a graphics accelerator. The mobile graphics processors are without any dedicated graphics memory and the most advanced mobile graphics processor can provide fill rate of 1 giga pixels per second with a clock speed of at most 200 MHz using the shared system memory. The system memory can be at most 256 megabytes. Memory interface is at most 32 bits and memory bandwidth is up to 200 MB per second [12]. The display screen dimension is small but some of them offer high resolution within that. The user inputs are provided using touchscreen and QWERTY keyboard. These capabilities are less compared to desktop systems due to small form-factor and low power consumption requirements.

2.2 Previous Work on Mobile Devices

Initially mobiles devices were used for displaying the rendered images which were processed and rendered in remote machines[9]. This is due to the fact that the minimum processing, memory and graphics support were not satisfied at that time. With the advent of better mobiles, it became favorable for processing and rendering in the device itself. The 3D models were loaded, processed on the mobile device and rendered them with several illustrating techniques at an interactive frame rate[10]. The first volume rendering of 3D scalar fields on mobile devices have been credited to Manuel Moser and Daniel Weiskopf[11]. They identified the limitations in hardware of mobile devices and used memory efficient techniques for rendering small sized scalar field of dimension at most 64^3. We used the techniques mentioned by them to build a volume renderer on a much better hardware platform. The scalar fields we used are larger up to 512^3. Our goal was to achieve at least 10 frames per second which can be considered as decently interactive.

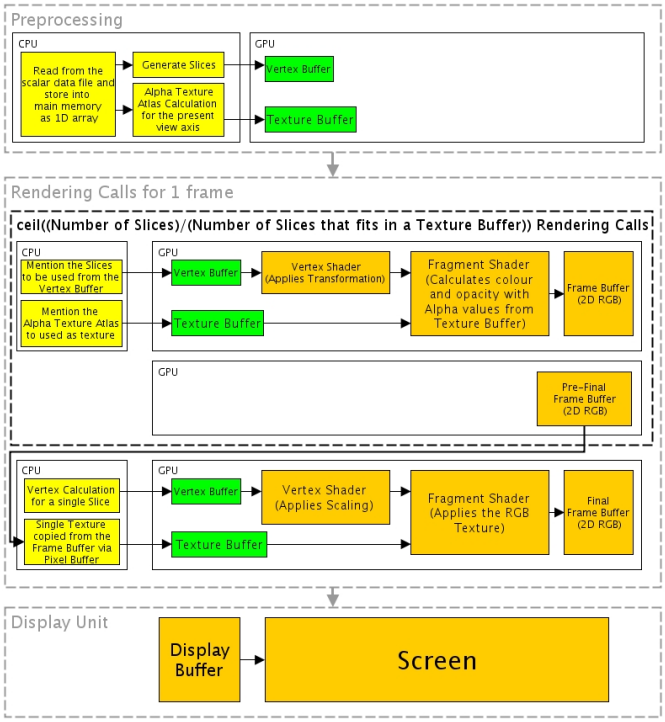

Figure 2.2: Volume rendering pipeline used in our implementation

2.3 Implementation

The platform used for implementation is Nokia N900. It has Texas Instruments' ARM based OMAP 3430 Cortex-A8 600 MHz processor, PowerVR SGX 530 GPU (clock speed of 110 MHz, fill rate of 500 mega pixels per second if clock speed is 200 MHz) with OpenGL ES 2.0 support, 256 MB main memory running at frequency 166 MHz and 768 MB virtual memory.

The volume renderer we built is Graphics Processing Unit(GPU)-based i.e. it utilizes the GPU for processing intermediate results and rendering the final result. The volume data was submitted to the GPU in the form 2D textures since 3D textures are not supported by the GPU hardware. 3D textures utilize trilinear interpolation to calculate value at a particular position and it is computationally costly. Slices of 2D textures were derived from the scalar field with respect to the axis alignment. There can be six types of axis alignments i.e. a positive x-axis, a negative x-axis, a positive y-axis, a negative y-axis, a positive z-axis and a negative z-axis. So this accounted for two footprints of

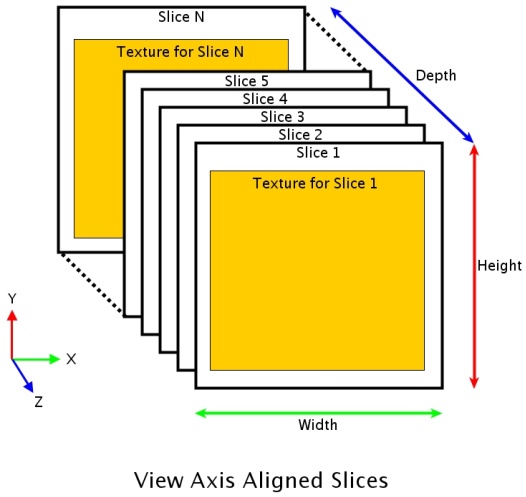

Figure 2.3: Slices axis alignment

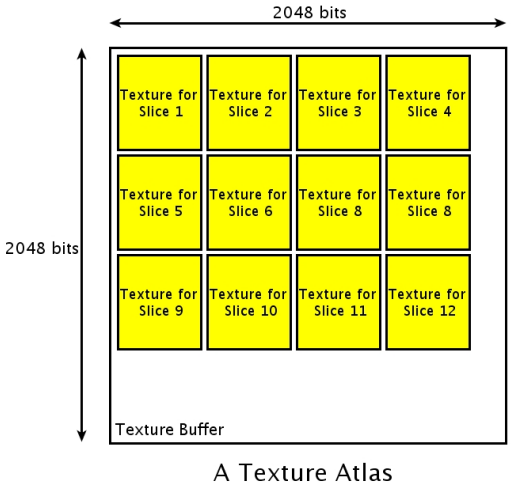

Figure 2.4: Texture Atlas

the same data in the memory at any instance i.e. one for the scalar field data and other for the set of particular axis-aligned 2D textures. If the value at each point in the scalar field(w×h×d) is of size b bits, then the memory requirement will be (2×w×h×d×b)/8 bytes. Thus a scalar field of dimension 256×256×128 is represented by 256 slices or 2D textures, if it is aligned to x-axis or y-axis otherwise it requires 128 slices for the z-axis alignment. These textures are preprocessed by GPU before further use in order to get better memory access through locality of reference.

These textures are applied to a graphic primitive i.e. rectangular polygons (may be created using two triangular polygons for each). The guidelines of PowerVR[15] suggests to decrease the rendering calls made to OpenGL ES. This will reduce the number of data transfer between CPU and GPU. In order to decrease the rendering calls we used vertex buffers to define the primitives and their texture coordinates. This is complemented with the use of texture atlas. The graphics hardware supports at most 8 texture buffers; this hinders the target of reducing the number of rendering calls. Texture atlas is used for fitting several slices in one texture buffer of a given size. An algorithm have been written for fitting maximum number of slices in a given texture buffer. In our case the texture buffer is of 4 mega bytes. This doesn't allow placing all the slices in one single texture buffer for scalar fields exceeding 4 megabytes. And hence use of texture atlas reduces

number of rendering calls to single call for scalar fields of dimension $\leq 128\times128\times256$. For large scalar fields texture atlas reduces the number of rendering calls to few.

The benefit obtained from using newer hardware is the support for OpenGL ES 2.0 which gives access to the programmable shader. The scalar field was not preprocessed to create textures using the RGBA values for each point depending upon the value at that point. If RGBA values were used in storing the textures, the memory requirement would have increased by five folds at any instance. This helped to reduce memory usage. The raw data is later processed in the programmable shader using transfer function table (in the form of 2D texture) to generate the RGBA values for each point (known as postclassification). This also provides quality rendered images.

When the zoom in feature is implemented using the scaling function of OpenGL ES, the size of the intermediate data (number of fragments of each OpenGl ES primitive) is increased through interpolation, which reduces the frame rates. Instead of using the scaling function, the image obtained from the initial rendering is scaled using the bilinear interpolation as proposed by Moser and Weiskopf. This can be done using the pixel buffer. The cost of this interpolation is less compared to the interpolation done for scaling function. The image quality degrades due to increased pixel size. This technique is useful when the user doesn't need much detail.

2.4 Results of Volume Renderer Implementation

The data of varied sizes were obtained from the Volume Dataset Repository (`http://www.volvis.org/`) at the WSI/GRIS, University of Tübingen, Germany. The table 2.1 displays the details for each of the data. Also the frame rates (average value for a number of readings) for each of the data have been provided. All the images were rendered using the same transfer function. These results infer that there is a decrease in frame rates as the size of the data is increased.

Table 2.1: Volume rendering results

Data name	Dimensions	Memory	FPS
Silicium	$98 \times 34 \times 34$	110 KB	34.75
Fuel	$64 \times 64 \times 64$	256 KB	35.56
CTA Ear	$128 \times 128 \times 30$	480 KB	20.19
Hydrogen atom	$128 \times 128 \times 128$	2 MB	13.58
Engine	$256 \times 256 \times 128$	8 MB	4.78
Skull	$256 \times 256 \times 256$	16 MB	2.34
Vertebra	$512 \times 512 \times 512$	128 MB	1.24

Figure 2.5: Nokia N900 with our volume renderer displaying the shockwave data

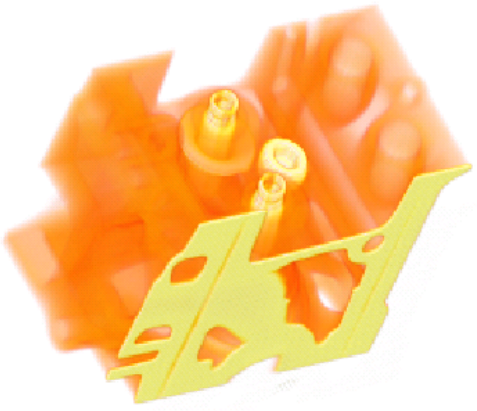

Figure 2.6: A volume rendered image of engine data on Nokia N900.

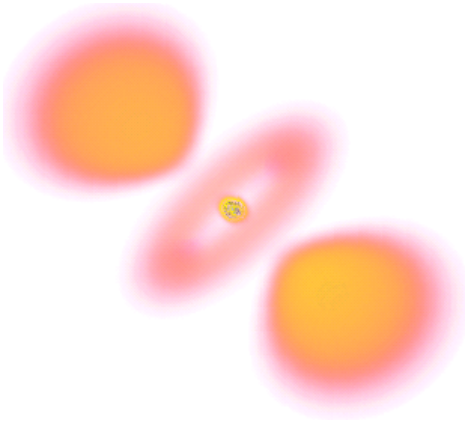

Figure 2.7: A volume rendered image of hydrogen atom data on Nokia N900.

Figure 2.8: A volume rendered image of bonsai data on Nokia N900.

2.5 Techniques for Large Scalar Field

The goal of at least 10 fps couldn't be achieved with the previously mentioned techniques. Several techniques have been proposed to handle large scalar fields in desktop computers[14]. Their applicability to mobile devices has been summarized below.

Bricking technique assumes the scalar field at least fits into the main memory. The scalar field is subdivided into small parts called bricks. The GPU memory is assumed to be small here. So these bricks are sent to the GPU memory in batches of some order and rendered. This is similar to what we did if the slices don't fit into a single texture buffer.

The multiresolution based volume rendering stores the scalar field in an octree. Depending upon the distance from the camera, it decides whether a particular child node of the octree is traversed (followed by processing and rendering) or not. This technique may increase the number of rendering calls depending upon the size of each child node. Also it increases the number of primitives in the graphics pipeline, which can further decrease the performance.

Compressed texture support of GPU can be utilized for improving performance with decreased quality of the rendered image. The hardware which we used supports compressed textures in RGB and RGBA formats only. This will neutralize the effects of memory saving done and the postclassification cannot be performed anymore.

In wavelet compression based technique, the scalar field is converted into hierarchical wavelet representation in compressed form using octree. When a child node is selected for display, it is decompressed first and then used by the rendering call. This reduces the memory footprint at the cost of increased processing. This technique also has the same problems as with the multiresolution technique.

The packing technique and the vector quantization technique uses compression but here decompression is done using shader(either vertex or fragment). The low clock speed, memory bandwidth and small number of stream processors in GPU of mobile devices may not deliver the desired performance due to increased floating-point computations.

The above approaches are not applicable in mobile devices due to the hardware limitations. This necessitates moving towards other solutions. In mobile devices, downscaling of images and video is quite common utilizing the fact that the display screens are smaller in size and minor decrease in detail will not affect user's perception. This motivates us to study and explore the possibilities in downsampling also known as subsampling.

Chapter 3

Topology of a Scalar Field

The topological properties of a scalar field refer to the topology of its level sets. Let $f : \mathbb{M}.\mathbb{R}$ be a piecewise scalar function defined over a mesh $\mathbb{M}$. The level set for a particular isovalue $v \in \mathbb{R}$ is defined as a set of points $p \in \mathbb{M}$ where $f(p) = v$. The connectivity among points of a level set can be defined based on certain scheme (distance or edges in the mesh $\mathbb{M}$ etc.). This will result in a graph with one or more connected components called contours (in 1D iso point, in 2D isoline, in 3D isosurface). The contours begin from points called minima and ends at points called maxima. Minima and maxima points together constitute the extrema points. The points where the contours join or split, are called saddles. There are various types of saddles depending upon the number of splits and joins. The saddles and the extrema points together are called the critical points. Other than critical points, there are points where contours are neither created or destroyed nor split or join. Such points are called regular points. Figure 3.1 demonstrates a 2D scalar field.

Contour tree is an abstract way of representing a scalar field introduced by Van Krevald et al[19]. The contour tree captures the topological properties of a scalar field. A contour tree can be subdivided hierarchically into branches known as branch decomposition commenced by Pascucci et al[20]. A single branch consists of a pair of either saddle and extremum or extremum and extremum. A branch decomposed contour tree consists of a single branch having extrema as pair and the rest of the branches are pair

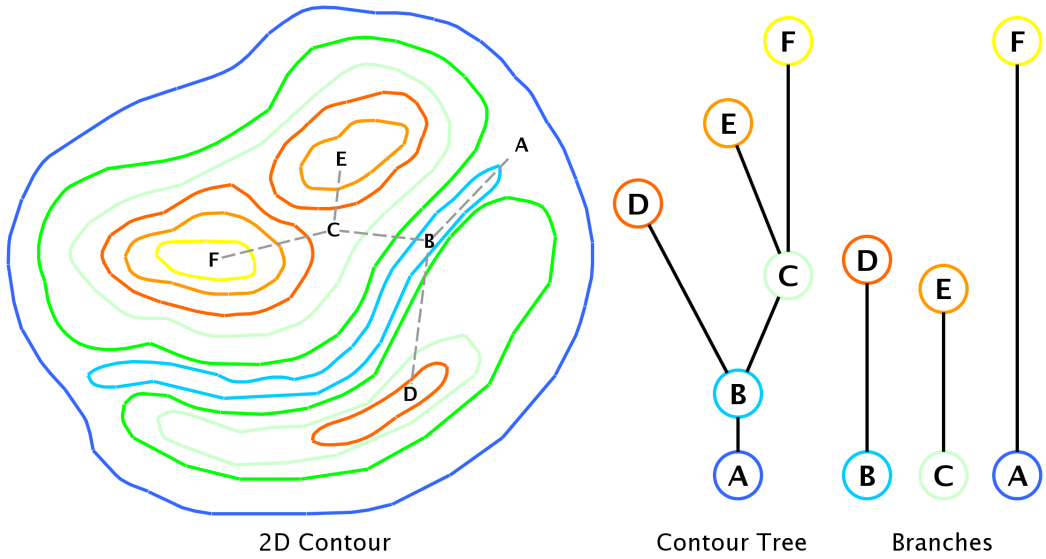

Figure 3.1: A terrain with contours (right picture) representing a 2D scalar field; The critical points are: A is minima, B and C are saddles, D, E and F are maxima; The graph on the right shows how the contours begins from A, splits at B and C, and finally ends at D, E and F. The contour tree and branches for the scalar field in the middle and right respectively.

by an extremum and a saddle. A distinct saddle can be paired up in multiple branches depending upon the connectivity in the contour tree. Each branch has a value associated with it called the persistence which is the absolute difference between its pairing points. In subsampling, we try to approximate a scalar field, such that the functions which generated the new scalar field and the original scalar field are close to each other.

Chapter 4

Related Work on Subsampling

Martin Kraus and Thomas Ertl suggested a topology-controlled downsampling technique[1] for structured volume grid. They selected a cube of a grid consisting of 8 corner points and classified each of the point as regular or critical (saddle or extremum) taking 18 points around its neighborhood. The critical points are the one which provides the topological properties, thus preserving them will preserve the topology of the scalar field itself. They provided an algorithm to choose the data value of the new point which will replace those eight points. Thus the original scalar field is downsampled to 1/8th of its size. Their classification of points is based on Weber[2] definition of regular and critical points. Weber classified each point considering 6 points (figure 4.1) from its neighborhood from the face centers. Martin Kraus and Thomas Ertl used 18 points (figure 4.2) which comprised of 6 face centered points and 12 edge centered points since it gave them better results compared to 6 points.

In the downsampling process, their algorithm partitions the scalar field into unit cubes with eight corner points and selects a critical point from that cube. The selection of a critical point among two or more critical points of a cube incurs loss in topology and hence do not guarantee complete topology preservation. The value at the selected critical point is taken as the value of the new point which will be replacing the corner points of the cube in the downsampled scalar field. If there is no critical point among the points in the cube, the average of their values is taken as the value of the new point.

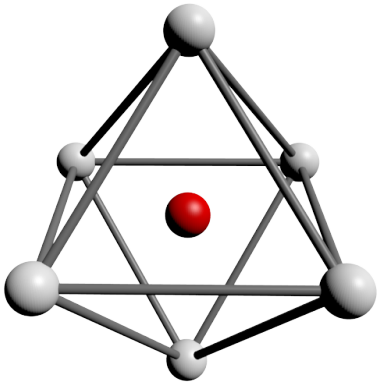

Figure 4.1: 6-points based neighborhood marked as white sphere; their connectivity is shown using grey edges; surrounded red sphere is in contention for classification.

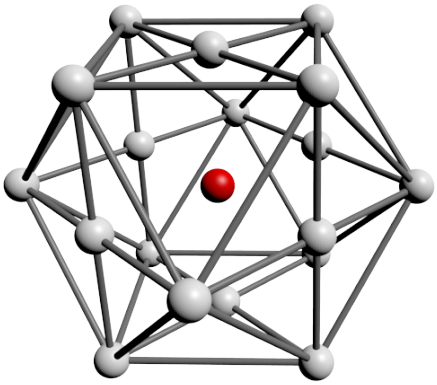

Figure 4.2: 18-points based neighborhood marked as white spheres; their connectivity is shown using grey edges; surrounded red sphere is in contention for classification.

Their approach has some drawbacks. No perturbation scheme had been used. Together with the classification scheme they described, it generates a huge number of critical points mostly extrema along the flat regions (region in scalar field having identical value). This result in extra constant factor computations required for processing the critical points. The avoidance of perturbation also makes the output absolutely implementation dependent, since there can be ties. Ties can occur when the absolute distance from the average value among two or more competing critical points are same. Use of average value for the cubic cell having all regular points causes some topological loss, and this loss can be avoided. They used subdivision of scalar field which doesn't create new points. This helps to reduce the computations at the cost of lesser topological preservation.

Chapter 5

Subsampling Algorithms

We devised three new subsampling algorithms, which try to preserve the topology of the scalar fields. These algorithms are extension of the work done by Kraus el al [1]. We also identified some measurement techniques that can be used for evaluating the subsampled scalar field when compared to the original scalar field. We applied the algorithms on various scalar fields obtained from data sets with size ranging from 41^3 to $256 \times 256 \times 128$. The algorithms have been compared based on the evaluation of the subsampled scalar field. Our goal was to build a subsampling algorithm which reduces amount of computation as well as memory footprint and produces minimum topological distortion. A topological distortion can be removal of original topological structures as well as introduction of new structures. This will enable it to be used on mobile devices.

5.1 Some Terms and Definitions

5.1.1 Perturbation

The topological properties are based on Morse Theory. In Morse Theory, each point in the scalar field should have a unique value. In practical cases, such uniqueness is not possible. To make the points unique, they need to be perturbed. One of the methods for perturbation is indexing the points. Therefore even if two or more points may have same value, their indices are different. In a rectilinear (structured) grid, all the original

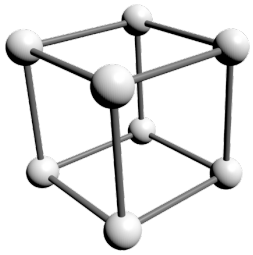

Figure 5.1: A locality

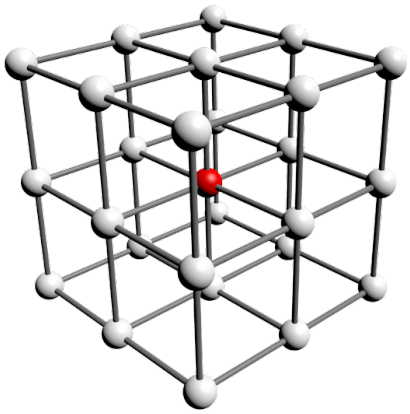

Figure 5.2: The 26 neighboring points around a subsampled point for the neighborhood set

points can be ordered by rows, then by columns and finally by slices or depths. Thus all these points can be indexed from 0 to $n-1$, where n is the total number of points in the rectilinear grid. Any new points introduced by subdivision are assigned a unique index $i \geq n$. This paper follows such method of perturbation.

5.1.2 Locality

Since we only consider rectilinear scalar field, the scalar field can be subdivided into cubic blocks each consisting of exactly 8 points. Such a cubic block is considered as a *locality*. In the subsampled scalar field, each such locality from the original scalar field will be replaced by a single point with a value decided by the subsampling algorithm. Each locality in the original scalar field is identified by a position in the subsampled scalar field. The points in a locality can be ordered. Let (x_o, y_o, z_o) be the lowest indexed point in the locality. Then the order will be: (x_o,y_o,z_o), (x_o+1,y_o,z_o), (x_o+1,y_o,z_o+1), (x_o,y_o,z_o+1), (x_o,y_o+1,z_o), (x_o+1,y_o+1,z_o), (x_o+1,y_o+1,z_o+1), (x_o,y_o+1,z_o+1). Let $L_o(x_s, y_s, z_s)$ be the locality with respect to the subsampled position (x_s, y_s, z_s). In a locality itself, there exists exactly 3 neighbors for each point. For example the point 0's neighbors in the locality are 1, 3 and 4. A locality in which all the 8 points are regular is known as a *regular locality* and subsampled position for this locality be L-regular. A

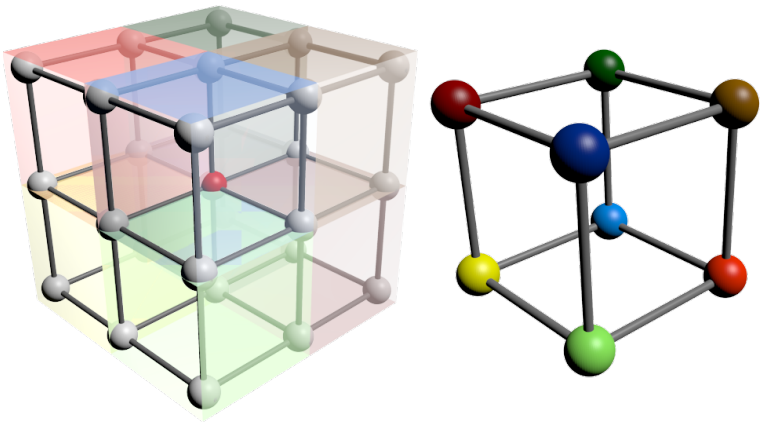

Figure 5.3: The 26 neighboring points around a subsampled point with the neighborhood set formation for each of 8 points in the locality

locality in which at least one of the points is critical is known as a *critical locality* and subsampled position for this locality be L-critical.

5.1.3 Permissible Value Assignment

For a given subsampling position with respect to a locality L in the original scalar field, the subsampling algorithm will calculate a value v. This value v is permissible if it is in the closed interval $[min(L), max(L)]$ otherwise non-permissible, where $min(L)$ and $max(L)$ is the minimum, maximum value respectively in the locality. Our entire algorithms use permissible values.

5.1.4 Probable Identical Branch Neighbourhood Set

When there is more than one critical points in a locality, only one of which can survive (*owned*) in the subsampled position and the rest (*disowned*) may take the position in the surrounding subsampled L-regular point position with some loss in topology. In other words, these disowned critical points may get replaced by some other point of the same branch to which it belonged. The new critical points can be the L-regular points surrounding the L-critical point. This will result in some topological loss. Such

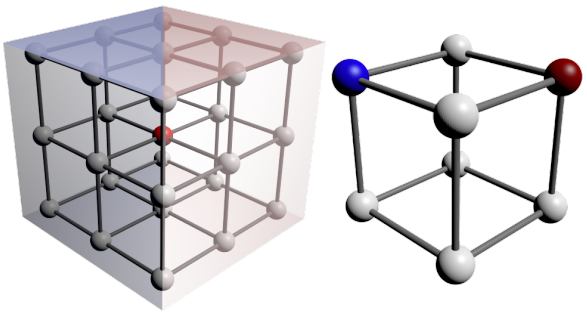

Figure 5.4: A neighborhood set formation for 2 critical points in a locality

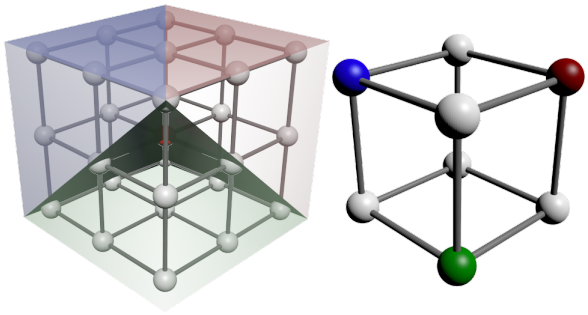

Figure 5.5: A neighborhood set formation for 3 critical points in a locality

topological loss can be minimized by mapping each disowned critical points to the L-regular points in the subsampled position due to regular locality and giving them a value close to the disowned points. The set of such subsampled point position for a disowned point is termed as *probable identical branch neighborhood set* or just *neighborhood set* in short.

There is a set of subsampled positions which can be a probable position for a disowned point. Since there are 8 points in one locality, there can be at most 8 critical points in a locality and at most 26 regular subsampled point positions around a given subsampled position (see figure 5.2). The neighborhood set comprises of points from these 26 points and each of these points should be L-regular and associated to some regular locality in the original scalar field.

The neighborhood set for each critical point in a locality can be found using polarity of the critical points. So any subsampled position lying exactly between two or more poles, will be common to those poles. The neighbor in between two poles is shared (see figure 5.3). It also provides the initial polar sets used for finding neighborhood set. Some examples of neighborhood set calculation are shown in figure 5.4 and 5.8. An algorithm for finding the neighborhood set for each critical point of a locality is shown in algorithm 1.

Algorithm 1 : Neighborhood Set Formation Algorithm

1: **procedure** FINDNEIGHBORHOODSET(Point P) ▷ *P is a subsampled position*
2: Identify the set $S_{critical}$ of critical points in P's corresponding locality in original scalar field.
3: Let R_P be the set of points in subsampled scalar field which are P's neighbors and corresponds to regular locality in original scalar field.
4: **for** each of the point P_i in $S_{critical}$ **do**
5: Put the initial set of points in N_{P_i} which $\in R_P$ and resides in the polar set for P_i.
6: **end for**
7: **for** each of the point P_i in $S_{critical}$ **do**
8: If the one step neighbor of P_i is not critical add all points of the polar set for that position to N_{P_i} which do not belong to any polar set of critical points.
9: **end for**
10: **for** each of the point P_i in $S_{critical}$ **do**
11: If the two step neighbor(through one step non critical neighbor) of P_i is not critical add all points of the polar set for that position to N_{P_i} which do not belong to any polar set of critical points and one step neighbors.
12: **end for**
13: **return** $N_{P_0}, N_{P_1}, ..., N_{P_{|S_{critical}|}}$ ▷ *returns neighborhood set for each critical point in the locality for the point P*
14: **end procedure**

5.2 Using the Neighborhood Set

We used the neighborhood set in our algorithms. The decision on the appropriate critical point whose value is to be placed at L-critical point is based on the loss for each L-regular point. The loss for each L-regular point with respect to a critical point for its neighborhood set is found. This loss denotes the topological loss that is responsible for selecting an L-regular point for a critical point if it is disowned. The L-regular point which gives the minimum loss is taken as ideal choice. The critical point which gives the maximum minor loss is chosen (owned) for the L-critical position. The rest of the critical points (disowned) are replaced or survived through the L-regular points having the minor loss. Since an L-regular point can be in multiple neighborhood sets for a critical locality (having more than 2 critical points), a L-regular point can be provide minor loss for 2 or more disowned points. Such ties can be resolved by forming a bipartite graph. One set with disowned critical points and other with the L-regular points. The edges are

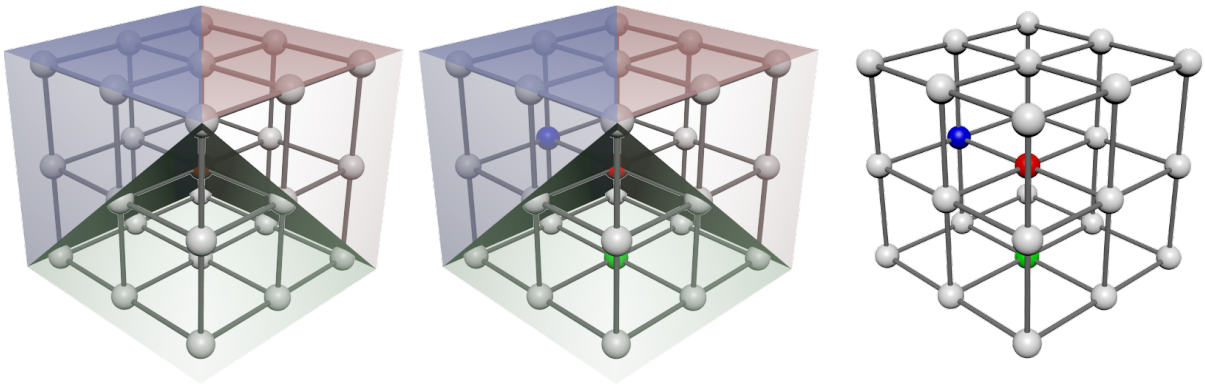

Figure 5.6: A neighborhood set formation for 3 critical points in a locality shown in Fig 5.8; followed by selection of owned critical point(red); the disowned critical points (blue and green) are survived using one of the L-regular point from their neighborhood set

between the disowned critical points and the L-regular points of their neighborhood set. The edges are given some weights, which are the losses for the L-regular and disowned critical point of that edge. The perfect matching of this bipartite graph gives the solution.

5.3 Classification of Points

Each point in a mesh or scalar field can be classified as Regular, Saddle, Minima or Maxima. We used a classification scheme based on Face-Centered 24-fold subdivision Carr et al [16]. This subdivision was selected because it is consistent with the mesh and thus it will result in less artifact and accurate classification of the points. According to this subdivision, an original (non-new) point in the mesh will be surrounded by at most 26 points. Among these 6 will be the original points, and rest of the points will be newer. The newer points include 8 body-centered points and 12 face-centered points. The figure 5.7 shows the neighborhood of an original point. Each point in a scalar field can be classified based upon the points in its neighborhood. The neighborhood forms a blanket around the point to be classified. This blanket can be decomposed into simplicial mesh to obtain the connectivity, where two points are connected if there is a path formed by edges between them. The connectivity thus obtained is useful in classification. A critical point can be an extremum or a saddle point.

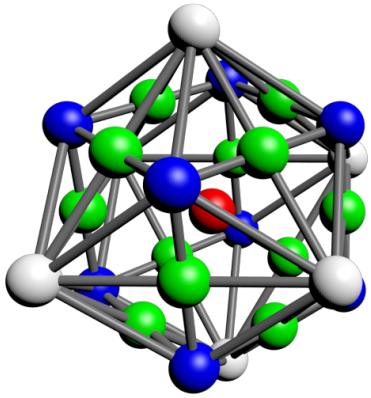

Figure 5.7: 26 points based neighborhood due to Face-Centered 24-fold subdivision. The navy blue colored spheres are body-centered points; green colored spheres are face-centered points; white sphere are the original points; the internal red sphere is the original point in contention for classification.

This neighborhood contains all the six points considered in Weber's classification. Such a neighborhood of points can be created from a structured grid using 8 trilinear interpolations (for the body-centered points) and 12 bilinear interpolations (for the face centered points). Trilinearly interpolated points have degree 6 each (with 3 nearest bilinearly interpolated points and 3 nearest original points). Bilinearly interpolated points have degree 4 each (with 2 nearest trilinearly interpolated points and 2 nearest original points). Original points (used in Weber's classification) have degree 8 each (with 4 nearest trilinearly interpolated points and 4 nearest bilinearly interpolated points).

The classifier algorithm for a given point P is shown in algorithm 2. The point classification algorithm identifies all the points which are either greater or less than to P (in terms of value; if values are same, index number is used to break the tie) and creates two sets of points. This is followed by removal of those edges which connects two points of different sets which lead to disconnected graph. There are three types of connected components in G used in classification of P. If G comprises of single component with either all greater (minima) or all less than or equal (maxima), we denote P as extremum. When there are two components with one group as all greater and other as all less or equal, we denote P as regular. If G has more than two components, it is saddle. This algorithm can be slightly modified to identify flat regions in the original scalar field by

Algorithm 2 : Point Classification Algorithm

```
1: procedure CLASSIFY(Point P)
2:     Identify the set S of points in P's neighborhood.
3:     Define the connectivity graph G among the points in S with edge set E.
4:     for each of the point Q in S do
5:         If Q is greater than P (in case of same value, ties broken using indices), then
       Mark Q as positive.
6:         Otherwise, Mark Q as negative.
7:     end for
8:     for each edge e(A,B) in E where A and B are two end points of e do
9:         If the marking on the points A and B are different then delete edge e from E.
10:    end for
11:    Let the number of connected components in G be C.
12:    if C==1 then
13:        P is an extremum point.
14:    else if C==2 then
15:        then P is a regular point.
16:    else
17:        P is a saddle point.
18:    end if
19: end procedure
```

checking that all the comparisons involved tie break or not. If yes, the point P can be considered as regular. This can help in increasing number of *regular locality* and reduce some computations involved in processing the *critical locality*.

The algorithm is of the order of O(n) where n is the number of edges considered in the connectivity since the number of edges will be always greater than or equal to number of points based neighborhood. Since this algorithm will be called once for each of the points in the scalar field, the 6 points based neighborhood will be providing the result quickly, followed by the 18 points based and then the 26 point based solutions.

We used this algorithm in our subsampling algorithms. We also found that the algorithm 2 without indices gives good result for few of the datasets. But due to increase in computation, we didn't use it in this thesis.

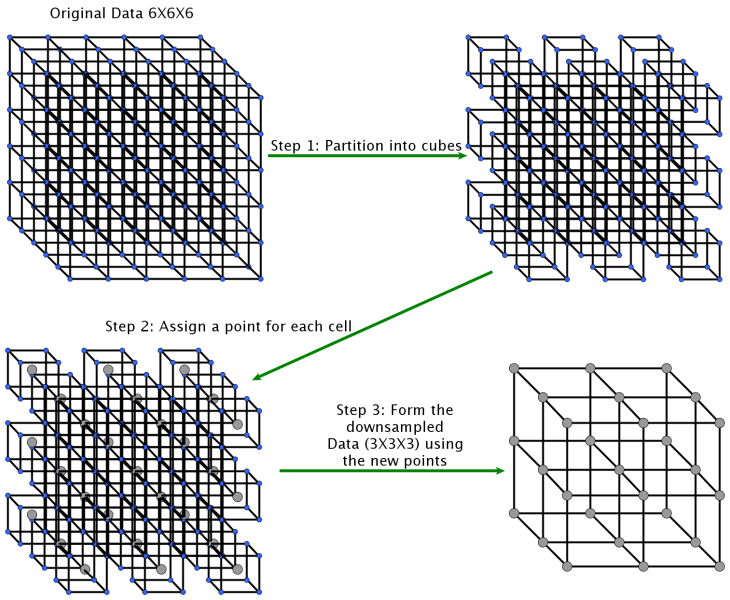

Figure 5.8: A general subsampling method

5.4 Subsampling Algorithms

5.4.1 The General Algorithm

The algorithm identifies the subsampling point position for each locality of the original rectilinear grid. Then for each locality, all the points are classified. Based on the type of points in a locality, the subsampled point's value is calculated. It was Kraus et al [1] who devised this technique. The general algorithm is shown in algorithm 3 which identifies the subsampled position and calls the new value assigning method for that position.

5.4.2 Kraus' Improved (KImp)

In this paper we show some variety of ways in which the values of the points in the subsampled scalar field can be calculated. The first algorithm is a similar to Kraus'[1] but with the above mentioned classification and perturbation of point through indexing. Kraus gave highest priority to the extrema, followed by saddle. If there is one or more extrema in a locality, the extremum with maximum distance from the average of the extrema is chosen whose value will be set to the new subsampled point. If there isn't

Algorithm 3 : General Subsampling Algorithm

```
1: procedure SUBSAMPLE(A 3D structured scalar field D)  ▷ The original data with dimension width, height and depth
2:     for k = 0 to depth/2 -1 do
3:         for j = 0 to height/2 - 1 do
4:             for i = 0 to width/2 - 1 do
5:                 D'+=getValueOfTheNewPoint(i, j, k, D)
6:             end for
7:         end for
8:     end for
9:     return D'.  ▷ A subsampled 3D structured scalar field (data) with dimension width/2, height/2 and depth/2
10: end procedure
```

any extremum but saddles and or regular, the saddle with maximum distance from the the average of the saddle is chosen whose value will be set to the new subsampled point. In case of a regular locality, the average of all the eight points is taken as the value of the new subsampled point. The algorithm is shown in algorithm **KImp**.

5.4.3 Loss Based Point Selection without Regular Neighbor Modification (LSWM)

The second algorithm is based on different selection process for the each critical locality. The neighborhood set for each of the critical point is found out. This is followed by calculation of the probable loss incurred if the critical point is not chosen for the subsampled position with respect to the present locality. The difference d_{r_e} between the L-regular neighborhoods r_e from a neighborhood set of an maximum extremum e is $r_e.max$ if $r_e.max \leq e.value$ else 0. The difference d_{r_e} between the L-regular neighborhoods r_e from a neighborhood set of an minimum extremum e is $r_e.min$ if $r_e.min \geq e.value$ else 0. The difference d_{r_s} between the L-regular neighborhoods r_s from a neighborhood set of a saddle s is $r_s.max$ if $r_s.max \leq e.value$ or $r_s.min$ if $r_s.min \geq e.value$ else 0. The loss is calculated for a critical point e for each point from its neighborhood set by the expression $abs(e.value - d_{r_e})$ and the minimum (*minor loss*) among them is chosen for comparison among the critical points. If there are extrema among the critical points

Algorithm 4 : Kraus' Improved Point Selection Algorithm [**KImp**]

procedure GetValueOfTheNewPoint(A point with location i, j, k and the input scalar field (data))
    Let $P_1, P_2, P_3, P_4, P_5, P_6, P_7$ and P_8 be the points of the locality which will be replaced by the new point.
    Classify each of the above points using the algorithm **Classify** 2.
    if All of $P_1, P_2, ..., P_8$ are regular **then**
        return the average of the values of all the points.
    else if Some of $P_1, P_2, ..., P_8$ are saddles but not extrema **then**
        if Only one saddle in the locality **then**
            return the value of the saddle.
        else if More than one saddles **then**
            return the saddle's value which is farthest from the average of all the saddles in this locality with ties broken using indices.
        end if
    else if Some of $P_1, P_2, ..., P_8$ are extrema **then**
        if Only one extrema in the locality **then**
            return the value of the extremum.
        else if More than one extremum **then**
            return the extremum's value which is farthest from the average of all the extrema in this locality with ties broken using indices.
        end if
    end if
end procedure

in a locality, only the extrema are considered for comparison. So if the extrema exists, the extremum having the maximum *minor loss* is chosen for subsampled position. The ties are resolved through indices value. When all the critical points in the locality are saddles, the saddle having the maximum minor loss is chosen for subsampled position. The regular locality in original scalar field is replaced by the average in the subsampled position.

This strategy assumes that all the critical points which are not chosen (disowned) will survive in the subsampled scalar field by some new points if there exists non-empty neighborhood set and these will incur a loss of at least *minor loss* for each them. Thus the loss is never overestimated. The detailed algorithm is shown in algorithm **LSWM**.

Algorithm 5 : Loss Based Point Selection Algorithm [**LSWM**]

1: **procedure** GetValueOfTheNewPoint(A point with location i, j, k and the input scalar field (data))
2: Let $P_1, P_2, P_3, P_4, P_5, P_6, P_7$ and P_8 be the points of the locality which will be replaced by the new point.
3: Classify each of the above points using the algorithm **Classify** 2.
4: **if** All of $P_1, P_2, ..., P_8$ are regular **then**
5: **return** the average of the values of all the points.
6: **else if** Only one of $P_1, P_2, ..., P_8$ is critical **then**
7: **return** the value of the only critical point.
8: **else if** Some of $P_1, P_2, ..., P_8$ are saddles but not extrema **then**
9: Let $P_1, P_2, ..., P_i, i \leq 8$ be the saddles.
10: **return Select**$(P_1, P_2, ..., P_i, i \leq 8)$ 6
11: **else if** Some of $P_1, P_2, ..., P_8$ are extrema **then**
12: Let $P_1, P_2, ..., P_i, i \leq 8$ be the extrema.
13: **return Select**$(P_1, P_2, ..., P_i, i \leq 8)$ 6
14: **end if**
15: **end procedure**

5.4.4 Loss Based Point Selection with L-regular Modification for Disowned Extrema only using Hungarian Algorithm (LSEM)

The third algorithm does exactly like the second algorithm but it additionally doesn't use average value for certain regular locality. These are the localities from the neighborhood locality of the disowned extrema. Only those disowned extrema are considered whose neighborhood set is non-empty. Here the problem converts to the assignment problem such that the total loss is least for a locality.

A well known algorithm to solve this is the Hungarian algorithm[18] in $O(n^3)$ where n is the number of vertices. The Hungarian algorithm performs perfect matching in bipartite graphs with minimum edge weight. Here the bipartite graph can be formed using the set of disowned extrema for a locality and their points from their respective neighborhood set. If there is a single disowned extremum in a locality, only the regular locality from the neighboring set which gives *minor loss* is taken as the replacement position. If there is a tie among the regular localities for same minor loss, the one with

Algorithm 6 : Select Owned Point Among the Critical Points Algorithm

1: **procedure** SELECT($P_1, P_2, ..., P_i, i \leq 8$) ▷ A set of critical points from a locality of one type only extremum or saddle
2: Let $N_{P_1}, N_{P_2}, ..., N_{P_i}, i \leq 8$ be the neighbor set for the respective extrema/saddles using algorithm 1.
3: Let $N_{P_1}, N_{P_2}, ..., N_{P_i}, i \leq 8$ be the neighbor set for the respective extrema/saddles using algorithm.
4: Let $min(d_{r_1}), min(d_{r_2}), ..., min(d_{r_i}), i \leq 8$ be the respective *minor loss*es for the extrema/saddles.
5: **return** the value of the extremum/saddle with maximum *minor loss* in this locality with ties broken using indices.
6: **end procedure**

the minimum index is chosen. When there are more than two disowned extrema in a locality, Hungarian algorithm is called with the points as the vertices and edges between an extremum and it's neighboring regular locality with the loss value as its weight. There may be a tie such that two or more disowned extremum have the same regular locality for individual minor loss. This is taken care by the Hungarian algorithm.

This algorithm assumes survival of all the disowned extrema which has non-empty neighborhood set and got matched by the Hungarian algorithm. There exists a problem with this approach. If not all of the disowned extrema are matched i.e. the number of regular neighboring localities are less or most of them having common regular neighboring locality, the loss will be greater. But such a case is very rare. There may the case where the two disowned extrema from different locality are matched to same regular subsampled position. We have not handled this situation since it will require additional storage; as a result the value for the last matching will remain in the disputed regular subsampled position. The detailed algorithm is shown in algorithm **LSEM**.

5.4.5 Loss Based Point Selection with L-regular Modification for Disowned Critical Points (LSCM)

This algorithm has similarity with previous. The difference is that it modifies regular points not only for extremum but for saddles too. The extrema are given more priority

Algorithm 7 : Loss Based Point Selection Algorithm with L-regular Modification for Disowned Extrema [**LSEM**]

1: **procedure** GetValueOfTheNewPoint(A point with location i, j, k and the input scalar field (data))
2: Let $P_1, P_2, P_3, P_4, P_5, P_6, P_7$ and P_8 be the points of the locality which will be replaced by the new point.
3: Classify each of the above points using the algorithm **Classify** 2.
4: **if** All of $P_1, P_2, ..., P_8$ are regular and there is no assignment for this locality by algorithm 9 **then**
5: **return** the average of the values of all the points.
6: **else if** Only one of $P_1, P_2, ..., P_8$ is critical **then**
7: **return** the value of the only critical point.
8: **else if** Some of $P_1, P_2, ..., P_8$ are saddles but not extrema **then**
9: Let $P_1, P_2, ..., P_i, i \leq 8$ be the saddles.
10: **return Select**$(P_1, P_2, ..., P_i, i \leq 8)$ 6
11: **else if** Some of $P_1, P_2, ..., P_8$ are extrema **then**
12: Let $P_1, P_2, ..., P_i, i \leq 8$ be the extrema.
13: **return SelectnModifyAll**$(P_1, P_2, ..., P_i, i \leq 8)$ 9
14: **end if**
15: **end procedure**

than the saddles. The Hungarian algorithm is run twice when there is a mix of extrema and saddles in the locality. In the first run the matching is done for the disowned extrema. This is followed by second run for the disowned saddles. This increases the computation required for processing the critical points. Like algorithm **LSEM**, this algorithm too does not handle disputed regular subsample positions and allows it to be overwritten. The detailed algorithm is shown in algorithm **LSCM**.

5.5 Evaluation of the Subsampled Scalar Field

We use contour tree to evaluate the subsampled scalar field. Both the subsampled and the original scalar fields are converted into meshes. The meshes are constructed conforming to the face-centered 24-fold subdivision [16] by introducing all the face center points and a single body center point to reduce artifacts in the resulting scalar field. This results in neighborhood structure for a non-interpolated point as shown in figure 5.7. The neighborhood structure for a face centered point and body-centered point is shown

Algorithm 8 : Loss Based Point Selection Algorithm with L-regular Modification for Disowned Critical Points [**LSCM**]

procedure GETVALUEOFTHENEWPOINT(A point with location i, j, k and the input scalar field (data))
    Let $P_1, P_2, P_3, P_4, P_5, P_6, P_7$ and P_8 be the points of the locality which will be replaced by the new point.
    Classify each of the above points using the algorithm **Classify** 2.
    if All of $P_1, P_2, ..., P_8$ are regular and there is no assignment for this locality by algorithm 9 **then**
        return the average of the values of all the points.
    else if Only one of $P_1, P_2, ..., P_8$ is critical **then**
        return the value of the only critical point.
    else if Some of $P_1, P_2, ..., P_8$ are saddles but not extrema **then**
        Let $P_1, P_2, ..., P_i, i \leq 8$ be the saddles.
        return SelectnModifyAll$(P_1, P_2, ..., P_i, i \leq 8)$ 9
    else if Some of $P_1, P_2, ..., P_8$ are extrema **then**
        Let $P_1, P_2, ..., P_i, i \leq 8$ be the extrema.
        return SelectnModifyAll$(P_1, P_2, ..., P_i, i \leq 8)$ 9
    end if
end procedure

in figure 5.9 and 5.10 respectively.

A contour tree can be decomposed into branches using the branch decomposition algorithm given in [17]. The branch decomposed contour tree is produced for the meshes of original and subsampled scalar field, which were used for evaluation and comparison. The branches due to subsampled scalar field are mapped to the branches due to original scalar field. In the branch having the pair of extremum and extremum, the least valued extremum is not considered as extremum in the evaluation. Thus from each of the branches, an extremum is extracted. This extremum can either maximum or minimum.

The branches to be mapped must have the same type of extremum. Given a position in the subsampled mesh, it is always possible to find the locality in the original mesh which has been substituted with the point in subsampled position. This locality contains 15 points i.e. 8 old points at the corners, 6 new points on the face and 1 new point in the center of the locality also known as body center. Thus for an extremum point e_{Sub} of a branch belonging to a branch decomposed contour tree of subsampled scalar mesh, there exist a locality in the original scalar mesh. This locality can contain a set of extremum

Algorithm 9 : Select Owned Among the Critical Points and Modify L-regular Point for Disowned Algorithm

procedure SELECTNMODIFYALL($P_1, P_2, ..., P_i, i \leq 8$) ▷ A set of critical points from a locality
  Let $N_{P_1}, N_{P_2}, ..., N_{P_i}, i \leq 8$ be the neighbor set for the respective extrema/saddles using algorithm 1.
  Let $min(d_{r_1}), min(d_{r_2}), ..., min(d_{r_i}), i \leq 8$ be the respective *minor loss*es for each critical point respectively.
  Let P_{mml} (*owned*) be the critical point (extremum is preferred if exists) with maximum *minor loss* in this locality with ties broken using indices.
  Match the rest of the extremum (if exists) using Hungarian algorithm for minimum total loss.
  for each matched critical points P_c with regular locality r_c **do**
    if P_c is a Maxima and $r_c.max \leq P_c.value$ **then**
      Set the value at $r_c.value = P_c.value$.
    else if P_c is a Minima and $r_e.min \geq P_c.value$ **then**
      Set the value at $r_c.value = P_c.value$.
    end if
  end for
  Remove from $N_{P_1}, N_{P_2}, ..., N_{P_i}, i \leq 8$, the regular locality which are matched to the extrema.
  Match the rest of the saddles (if exists) using Hungarian algorithm for minimum total loss.
  for each matched critical points P_c with regular locality r_c **do**
    if P_c is a Saddle and $r_c.max \leq P_c.value$ and $r_c.min \geq P_c.value$ **then**
      Set the value at $r_c.value = P_c.value$.
    end if
  end for
  return $P_{mml}.value$. ▷ the value of the *owned* point
end procedure

E_{Org} of same type as e_{Sub}. Now since all the extremum in E_{Org} belong to certain branch in the branch decomposed contour tree of the original scalar mesh. Now to map e_{Sub} to one of $e_{Org} \in E_{Org}$, certain conditions must be checked i.e. the value at e_{Org}must be close to the value at e_{sub}. If there is a tie i.e. more than one extremum in E_{Org} are equally closer to e_{Sub}, the persistence of the branches of those extrema will be considered and the closest among them get mapped. If there still exists a tie, it can be broken by indices i.e. the closest numbered extremum gets mapped.

When E_{Org} is empty, the next set of points will be the blanket of points covering the

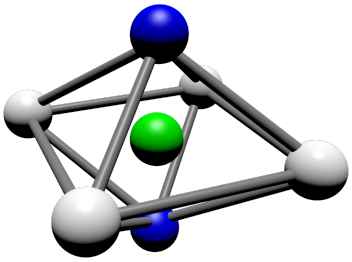

Figure 5.9: The neighborhood structure for a face-centered point of size 6

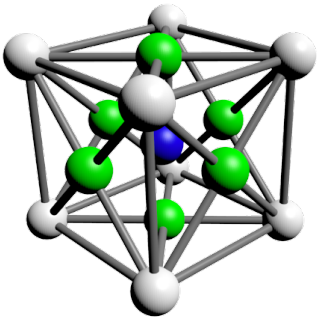

Figure 5.10: The neighborhood structure for a body-centered point point of size 14

current locality from outside. The set of points can be labeled with level. The initial set of points from locality will be labeled as level zero. Next sets of points will be at level 0.5, 1.0, 1.5. 2.0 and so on. This process continues until all the points in original mesh are exhausted.

It can so happen that an extremum from original mesh being considered for mapping with an extremum from the subsampled mesh which is already mapped to some other point. In this case, the tie can be broken by the level at which the point is considered with respect to the subsampled extremum points. If there is again a tie with respect to the level, it can be broken by the closeness of the persistence. And if still there is a tie, it is sure to be broken using the indexing of the subsampled extremum points i.e. lower indexed point will get mapped to the disputed extremum from original mesh. This leaves one of the subsampled extremum point unmapped which included in queue for the mapping process again.

Note that in this mapping process, some of the extrema in subsampled and original mesh may not get mapped. Once the mapping is done, the difference in the persistence of the branches of the mapped points are collected and added up with the of persistences for the unmapped extremum points to give the total persistence loss. The total persistence loss value can be normalized per point by dividing it with the sum of total mapping and unmapped points.

This evaluation technique gives same importance to the retention of original topological structures as well as to the introduction of new topological structure based upon persistence only.

Chapter 6

Results

The data have been obtained from the Volume Dataset Repository (*http://www.volvis.org/*) at the WSI/GRIS, University of Tübingen, Germany. All data have been subsampled once i.e. the subsampled data is having half the dimension of the original on each axis.

6.1 Small sized Scalar Fields

In the small scalar fields segment, the algorithm which works best are **KImp**, **LSWM**, **LSEM** and **LSCM**. Table 6.1 shows the comparison of branch details and persistence loss per branch among the original and subsampled scalar fields.

For nucleon (figure 6.1), we find algorithms **KImp**, **LSWM** and **LSEM** are introducing new topological conveyed by the increased total branch persistence exceeding the original. The algorithm **LSCM** resulted in total persistence below that of original but the branch count is higher compared to others. All the algorithms are tied for this data. Visually the outputs of algorithms **KImp**, **LSWM** and **LSEM** look nearly identical for nucleon and they produced higher total branch persistence compared to **LSCM**.

For fuel (figure 6.2), the algorithm **KImp**, **LSWM** and **LSEM** works best. The algorithm **LSCM** introduces less branches but the total branch persistence is high compared to others, this and together with persistence loss per branch suggests that it introduces topological error. Visually the outputs of algorithms **KImp**, **LSWM** and **LSEM** look

nearly identical for fuel too.

For clayleycube (figure 6.3), due to simplicity of the scalar field the all the algorithm fetches same total branch persistence. Even same branch count for the first 3 algorithms and they look identical visually. The algorithm **LSCM** provides higher branch count and is able to get best evaluation.

For neghip (figure 6.4), the algorithm **LSEM** fared well without exceeding others in terms of branch details. The algorithm **LSCM** provides higher branch count but losses some structures (the small lobes in the right and left) visible from the figure itself.

For silicium (figure 6.5), the results of last three algorithm look identical. The result due to algorithm **KImp** shows some visual difference in the form of small bubbles at different regions, it also exceeded the original total branch persistence. The algorithm **LSEM** too exceeded original total branch persistence but it is closer to that value. In this case algorithms **LSWM** and **LSEM** take the top positions.

In this segment, overall the algorithm **LSEM** did well. It topped in four cases and came closer to the best in the rest.

Dataset	**Original**	**KImp**	**LSWM**	**LSEM**	**LSCM**
nucleon	**3827**	434	433	435	466
$41 \times 41 \times 41$	**551.25**	553.5	554.0	554.0	528.75
		0.14	0.14	0.14	*0.14*
fuel	**12103**	2984	2983	2981	2961
$64 \times 64 \times 64$	**1005.0**	914.25	914.0	912.75	921.0
		0.062	0.062	0.062	0.064
clayleycube	**17172**	4133	4133	4133	4206
$64 \times 64 \times 64$	**442.0**	442.0	442.0	442.0	442.0
		0.029	0.029	0.029	*0.022*
neghip	**16507**	2789	2810	2802	2895
$64 \times 64 \times 64$	**4571.25**	3933.25	3922.75	3920.75	3914.75
		0.28	0.26	*0.25*	0.26
silicium	**3578**	1028	996	996	1009
$98 \times 34 \times 34$	**12503.2**	14043.8	13329.2	13329.0	13433.2
		0.66	0.54	*0.54*	0.77

Table 6.1: Branch count, total branch persistence and persistence loss per branch for small scalar fields

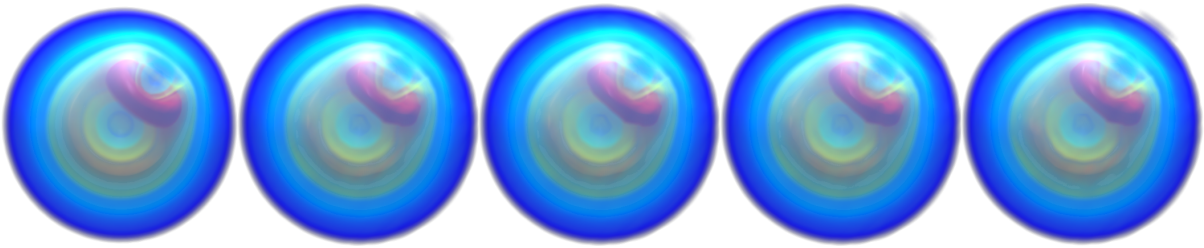

Figure 6.1: Volume rendered images of nucleon data: (from left) original, algorithms **KImp**, **LSWM**, **LSEM** and **LSCM**

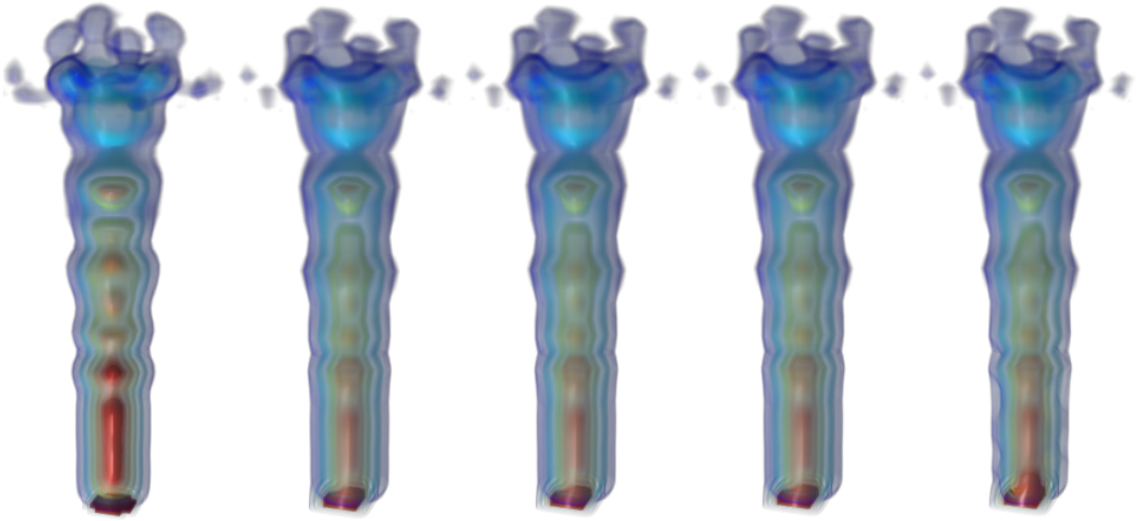

Figure 6.2: Volume rendered images of fuel data: (from left) original, algorithms **KImp**, **LSWM**, **LSEM** and **LSCM**

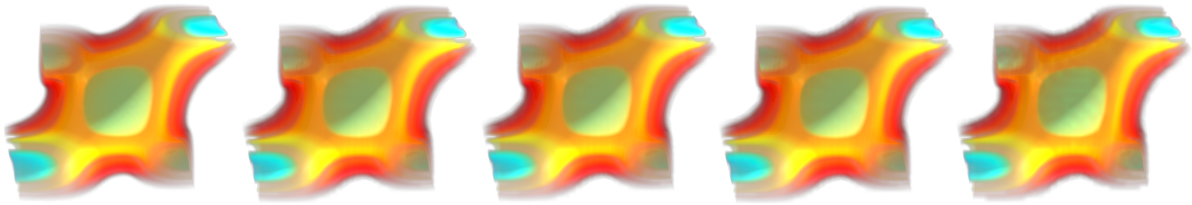

Figure 6.3: Volume rendered images of clayleycube data: (from left) original, algorithms **KImp**, **LSWM**, **LSEM** and **LSCM**

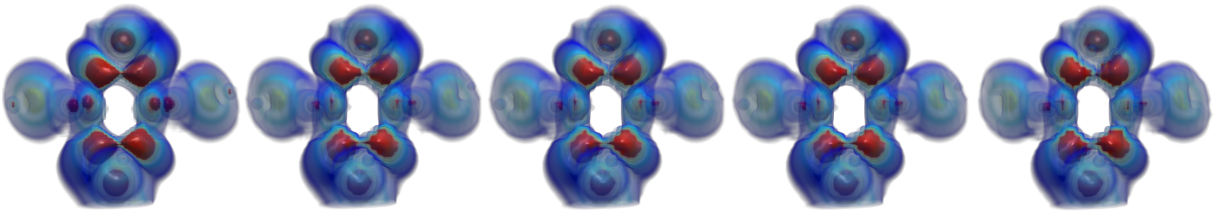

Figure 6.4: Volume rendered images of neghip data: (from left) original, algorithms **KImp**, **LSWM**, **LSEM** and **LSCM**

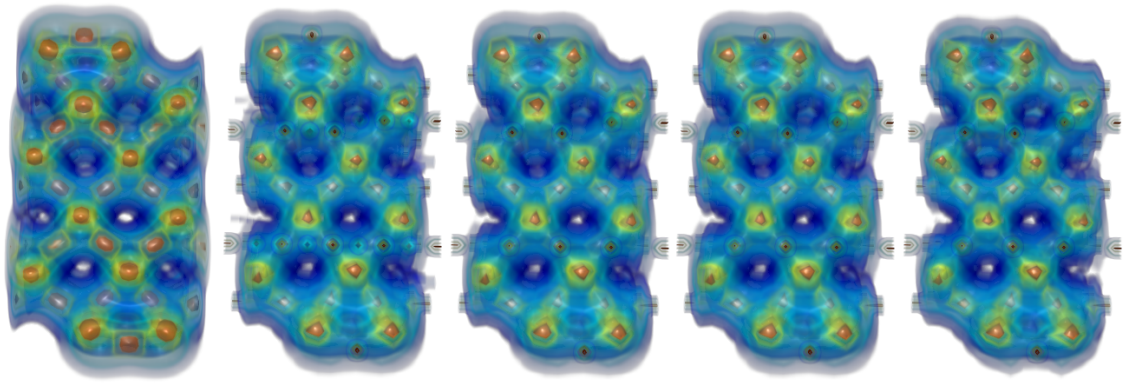

Figure 6.5: Volume rendered images of silicium data: (from left) original, algorithms **KImp**, **LSWM**, **LSEM** and **LSCM**

6.2 Medium sized Scalar Fields

The subsampling methods favoring the test dataset in medium sized scalar fields segment are algorithms **LSWM**, **LSEM** and **LSCM**. Table 6.2 shows the comparison of branch details among the original and subsampled scalar fields.

The teddy bear data (figure 6.6) shows high susceptibility to subsampling. The persistence loss per branch is more than one unit. The algorithm **LSCM** kept branch details low and still managed to be at the top. Visually the amount of distortion increases as we move from left to right.

For the bonsai data (figure 6.7), the algorithm **LSEM** bettered from others. The algorithm **LSCM** kept low on branch details. The algorithm **LSEM** produce more branches than the others. Visually all of them differ.

The shockwave data (figure 6.8) is the least affected by the subsampling algorithms and is visible from the images too. The algorithms **LSWM** and **LSEM** share the top place here by greater margin from others. Interestingly the subsampled outputs produced by them were identical when compared using hash function. Visually the 3^{rd} and 4^{th} are close to 1^{st}.

The engine data (figure 6.9) is subsampled well by algorithms **KImp**, **LSWM** and **LSEM**. The algorithm **LSCM** produced most non smooth result compared to others. Visually the 2^{nd} 3^{rd} and 4^{th} have little dissimilarity.

Like in small segment, here too the algorithm **LSEM** scored better than others.

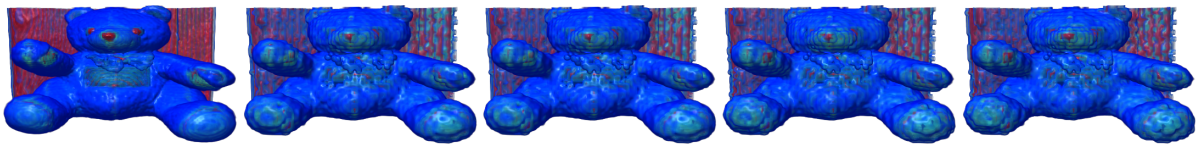

Figure 6.6: Volume rendered images of teddy bear data: (from left) original, algorithms **KImp**, **LSWM**, **LSEM** and **LSCM**

Dataset	Original	KImp	LSWM	LSEM	LSCM
shockwave	**80659**	19709	19748	19748	19916
64 × 64 × 512	**546.5**	550.25	550.75	550.75	551.25
		0.0066	*0.0003*	*0.0003*	0.0067
Teddybear	**198634**	28122	28380	28488	27361
128 × 128 × 62	**292169.0**	107024.0	100607.0	97164.8	87691.0
		1.35	1.33	1.32	*1.31*
bonsai	**850181**	243253	244752	244120	231835
256 × 256 × 128	**889336.0**	509469.0	516423.0	505964.0	477885.0
		0.74	0.74	*0.73*	0.80
engine	**746683**	145761	145186	145042	144332
256 × 256 × 128	**165761.0**	106724.0	108251.0	107186.0	95036.0
		0.16	0.16	*0.16*	0.18

Table 6.2: Branch count, total branch persistence and persistence loss per branch for medium sized scalar fields

Figure 6.7: Volume rendered images of bonsai data: (from left) original, algorithms **KImp**, **LSWM**, **LSEM** and **LSCM**

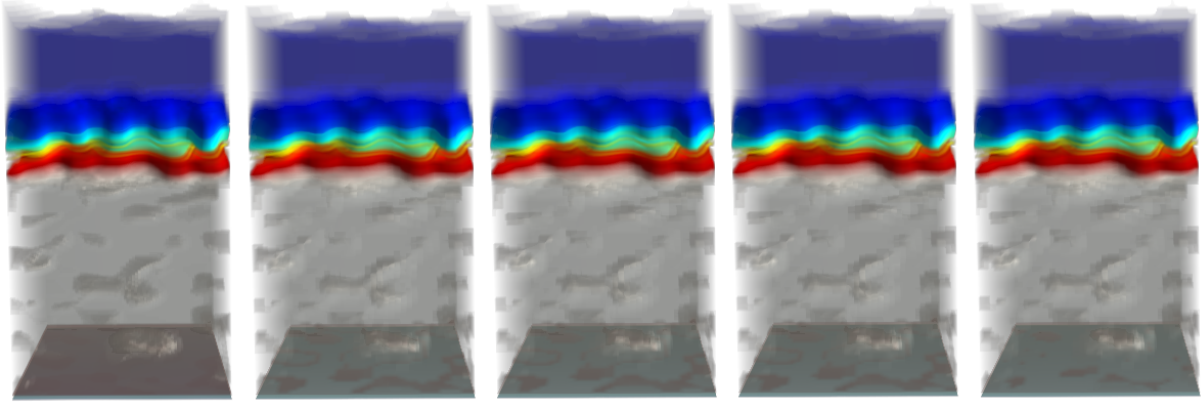

Figure 6.8: Volume rendered images of shockwave data: (from left) original, algorithms **KImp**, **LSWM**, **LSEM** and **LSCM**

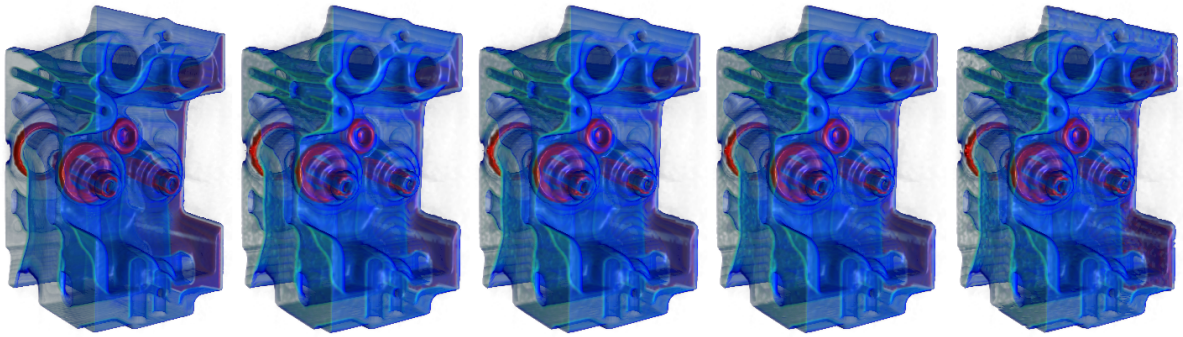

Figure 6.9: Volume rendered images of engine data: (from left) original, algorithms **KImp**, **LSWM**, **LSEM** and **LSCM**

6.3 Appropriate Subsampling Algorithm Selection

According to the results, we suggest some appropriate subsampling algorithms depending on the type of the scalar field resulting from the data. The scalar field with simple structures such as in the nucleon data should be should be subsampled with algorithm **LSCM** or average based subsampling method. For scalar fields with very low or very high critical points count with uniform distribution, the algorithm **LSCM** is better (for example clayley cube and teddy bear). The scalar fields having small number of critical points concentrated at one or more regions, the algorithm **LSEM** holds good (example shockwave) in this case. For any other type of scalar fields, the algorithm **LSEM** is appropriate.

If the least computation requirement is the only concern, then average value based subsampling algorithm is the best. In addition if the quality is a concern but with lesser priority compared to the computation, the algorithm **KImp** is appropriate. The increasing order based on the amount of computation required is as follows: **KImp**, **LSWM**, **LSEM** and **LECM**.

Chapter 7

Conclusions and Future Work

The evaluation method described by us correctly maps the original topological features with that of the subsampled data. It is sensitive to the noises introduced as well as original structures retained in the subsampled data. Our point selection method based on minor loss followed by appropriate modification of the L-regular point's value provides a better subsampling solution in most of the cases. All the three algorithms have both time complexity and space complexity of $O(\mathrm{n})$. Thus they can be used on mobile devices. All the three algorithms are based on locally optimal solution, thus they do not guarantee globally optimal solution.

The future work can be finding globally optimal subsampling algorithm which produces minimum topological loss for any scalar field. Such algorithm may have high computation and memory requirements. Making it suitable for mobile devices reducing floating point calculations will be another challenge. When OpenCL will be supported by mobile device's GPU[13], a subsampling algorithm can use it to reduce subsampling time. The evaluation method is extensive and takes long processing time for large data. The possibility to have more efficient evaluation method with the same degree of accuracy can be a problem to work on. Noises are inherent in data. Thus incorporating noise correction in subsampling as well as in the evaluation method will be challenging. The classification of scalar fields resulting from the data and based on that appropriate subsampling algorithm selection is another problem to venture upon.

References

[1] Martin Kraus and Thomas Ertl, "*Topology-Guided Downsampling*", In Proceedings of International Workshop on Volume Graphics 01, pages 129-147, 2000.

[2] Gunther H. Weber, Gerik Scheuermann, Hans Hagen and Bernd Hamann, "*Exploring Scalar Fields Using Critical Isovalues*", in Proceedings of IEEE Visualization Conference '02, pages 171-178, 2002.

[3] Charles D. Hansen and Chris R. Johnson, "*The Visualization Handbook*", Part II Scalar Field Visualization: Isosurfaces, pages 39-123; Volume Rendering, pages 127-258.

[4] Gunther H. Weber, Scott E. Dillard, Hamish Carr, Valerio Pascucci and Bernd Hamann," *Topology-Controlled Volume Rendering*", IEEE Transactions on Visualization and Computer Graphics, volume 13, number 2, pages 330-341, 2007.

[5] S. Takahashi, Y. Takeshima, I. Fujishiro, " *Topological Volume Skeletonization and its Application to Transfer Function Design*", Graphical Models, volume 6, number 1, pages 24-49, 2004.

[6] Vijay Natarajan and Valerio Pascucci, "*Volumetric Data Analysis using Morse-Smale Complexes*", International Conference on Shape Modeling and Applications 2005 (SMI' 05), pages 322-327, 2005.

[7] M. Hilaga, Y. Shinagawa, T. Kohmura and T. L. Kunii, "*Topological Matching for fully automatic similarity estimation of 3D shapes*", in Proceeding ACM SIGGRAPH '01, pages 203-212, 2001.

[8] Chandrajit L. Bajaj and Daniel R. Schikore, "*Topology Preserving Data Simplification with Error Bounds*", Computers & Graphics, volume 22, number 1, pages 3-12, 1998.

[9] Hong Zhou, Huamin Qu, Yingcai Wu and Ming-Yuen Chan, "*Volume Visualization on Mobile Devices*", Appeared in the Proceedings of 14th Pacific Conference on Computer Graphics and Applications (Pacific Graphics'06).

[10] Jingshu Huang, Brian Bue, Avin Pattath, David S. Ebert and Krystal Thomas, "*Interactive illustrative rendering on mobile devices*", IEEE Computer Graphics and Applications, volume 27, number 3, pages 48-56, 2007.

[11] Manuel Moser and Daniel Weiskopf, "*Interactive Volume Rendering on Mobile Devices*", In Proceedings of "Vision, Modeling, and Visualization VMV '08 Conference", pages 217-226, 2008.

[12] Jukka-Pekka Vihmalo and Veli Lipponen, "*Memory technology in mobile devices−status and trends*", Solid-State Electronics, Volume 49, Issue 11, pages 1714-1721, 2005.

[13] `http://www.imgtec.com/Newsletters/with_imagination/issue7_10.pdf`, "*With Imagination*", issue 7, page 3, 2010.

[14] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Christof RezK-Salama and Daniel Weiskopf, "*Real-Time Volume Graphics*", A. K. Peters Ltd., pages 441-459, 2006.

[15] `http://www.imgtec.com/factsheets/SDK/POWERVRSGX.OpenGLES2.0ApplicationDevelopmentRecommendations.1.8f.External.pdf`, "*POWERVR SGX OpenGL ES 2.0 Application Development Recommendations*", page 6, 2010.

[16] Hamish Carr, Torsten Moller, Jack Snoeyink, "*Artifacts Caused by Simplicial Subdivision*", IEEE Transactions on Visualization and Computer Graphics, volume 12 number 2, pages 231-242, 2006.

[17] Hamish Carr, Jack Snoeyink and Ulrike Axen, "*Computing contour trees in all dimensions*", Computational Geometry: Theory and Applications - Fourth CGC workshop on computional geometry archive, volume 24 issue 2, pages 75-94, 2003.

[18] H.W. Kuhn, "*The Hungarian Method for the assignment problem*, Naval Research Logistic Quarterly, volume 2, pages 83-97, 1955.

[19] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D.R. Schikore, *Contour Trees and Small Seed Sets for Isosurface Traversal*, Proc. 13th Ann. ACM Symp. Computational Geometry(SoCG), pages 212-220, 1997.

[20] V. Pascucci, K. Cole-McLaughlin, and G. Scorzelli, *Multi-Resolution Computation and Presentation of Contour Trees*, Lawrence Livermore Natl Laboratory, Technical Report UCRL-PROC-208680, pages 452-290, 2005.