Scalable Methods for Visualizing Flow in a Pellet Filled Reactor

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Engineering

IN

Computer Science and Engineering

BY

Jaipreet Singh

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

June, 2017

Declaration of Originality

I, **Jaipreet Singh**, with SR No. **04-04-00-10-41-15-1-12031** hereby declare that the material presented in the thesis titled

Scalable Methods for Visualizing Flow in a Pellet Filled Reactor

represents original work carried out by me in the **Deparment of Computer Science and Automation** at **Indian Institute of Science** during the years **2015-17**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discusions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Prof. Vijay Natarajan Advisor Signature

DEDICATED TO

Visualization and Graphics Lab, IISc

&

Shell Technology Centre, Bangalore

Acknowledgements

I would like to express my sincere gratitude to Prof. Vijay Natarajan for his unmatched guidance and supervision. I have been extremely lucky to have work with him. I am thankful to people at Shell Technology Centre, Bangalore for their assistance and guidance. It had been a great experience to work with them. I would like to thank the Department of CSA for providing excellent study environment and infrastructure.

Abstract

Smoothed-particle Hydrodynamics (SPH) simulations generate large amounts of flow field data. Extracting knowledge from these volumes of data and visualizing the huge data are challenging problems as the simulation produces a collection of particles together with multiple physical quantities like density, velocity, pressure and temperature at each particle location. For simulating flows of fluid in a reactor filled with pellets, our aim is to visualize 100 million particles. As a first step, we develop a scalable parallel algorithm to convert the simulation output into a grid representation that is amenable to fast visualization. The parallel algorithm is implemented on a manycore (GPU) architecture. In the next phase, we develop fast visualizing techniques that helps in interactive visualization of the generated 3D grid.

Contents

Acknowledgements i

Abstract ii

Contents iii

List of Figures v

1 Introduction 1

2 Background 3

3 Related Work 6

4 Dataset Description 8

5 Methodology 9

6 Structured Grid Representation 11

6.1 Algorithm Flow . . . 12

6.2 Error Computation . . . 13

6.2.1 Root Mean Square Deviation . . . 13

6.2.2 Mean Absolute Difference . . . 13

6.2.3 Infinity Norm . . . 14

7 Visualization 15

7.1 Important Slices Animation . . . 15

7.2 Sub-volume Extraction . . . 16

7.3 3-Slice Intersection . . . 16

8 Experimental Results **17**
8.1 Structured Grid Representation 17
8.2 Grid Visualization 21
8.3 Execution Timings and Errors 25

9 Conclusion and Future Work **28**

A Algorithm Work-flow **29**

Bibliography **32**

List of Figures

3.1 Contribution from neighbouring Voronoi cells based on the area in original Voronoi diagram. (a) Original Voronoi diagram for sites p_1, p_2, p_3 and p_4. (b) Updated Voronoi diagram because of a new site p. (c) u_1, u_2, u_3 and u_4 is the contribution from p_1, p_2, p_3 and p_4 respectively. 6

6.1 (a) Contribution from neighbouring particles to a grid point. (b) Each thread launched computes an interpolated value for one grid point. 11

8.1 For computing the Sum of gaussians field, the parameters of the gaussian density functions were fixed in such a way that at $isovalue = 1$ we get the surface represented by the particles. For input file containing 5 million particles, we obtain the above surface representing boundary of reactor and the pellets within. 18

8.2 The sum of gaussians field for the input point cloud. (a) Front view. (b) Side view. (c) Back view. 19

8.3 Density scalar field generated using Shepard interpolation (a) Front view, (b) Side view. Velocity scalar field generated using Shepard interpolation (c) Front View, (d) Side View. 20

8.4 The sum of gaussians field for input file containing 9.5 million particles. 21

8.5 The sum of gaussians field for the input point cloud containing 21 million particles. (a) Entire scalar field. (b) A slice from the field. (c) Important slice from the field. 22

8.6 The sum of gaussians field for the input point cloud containing 21 million particles. (a) A sub-volume region within a field. (b) 3-slice intersection at global maximum value. 23

8.7 The sum of gaussians field for input point cloud containing 9.5 million particles. (a) Entire scalar field. (b) An important slice from the field. (c) Hotspot region for a slice. (d) 3-slice intersection at global maximum value. 24

8.8 For 21 million particle dataset, (a) Algorithm execution timings and (b) computed errors. 26
8.9 For 9.5 million particle dataset, (a) Algorithm execution timings and (b) computed errors. 27

Chapter 1

Introduction

Smoothed-particle Hydrodynamics (SPH) is a computational method used in fluid flow simulations. The SPH method works by dividing the fluid into particles. For a single SPH simulation, millions of particles are generated. Each particle will have some physical quantities associated with it like density, velocity, temperature, pressure, mass, location etc. Several files each containing millions of particles will be generated for different simulations. Handling such a huge data and then visualizing it interactively is a challenging task. Early analysis of the data generated by large scale simulations is crucial to effectively monitor the correct progress of simulations and the understanding of simulation results. Such early analysis capability requires a monitoring framework that allows analysis tasks to take place on-the-fly while data is generated.

Simulation data is provided for reactor filled with pellets. Pellets are cylindrical shaped objects that take part in reaction with the fluid inside the reactor. These pellets help in refining the fluid. This data can be very large. We would like to read the necessary data within seconds. Once we have processed the data and extracted all the useful and necessary information, we would like to develop efficient and fast methods for interactive visualization to explore various physical quantities available from the simulation. Visualizing these physical quantities will help in understanding the flow of fluid through pellets inside the reactor. Pellets can be arranged in different patterns and each pattern can be visualized and analyzed in order to get optimum results from the reaction. Current methods require large amount of time for processing and visualizing the data. Implementing interactive visualization techniques further help in understanding the data. The key idea is to represent the data into an efficient representation. In our implementation, we sample input particles onto a grid for fast and interactive visualization. Performing computations over all grid points is an expensive task. We use GPU to achieve parallelism so that each grid point computation can be handled by one thread. Parallelism helps in improving the execution time for grid generation.

We use Paraview, an open source application for scientific visualization, for visualizing the grid instead of a stand-alone visualization tool. We implement visualization modules on top of Paraview because it provides various inbuilt functions like isosurface extraction, volume rendering, subset extraction and slicing for a grid data and also it is a commonly used tool for visualization purposes. Forming a grid helps us to exploit these functionalities of Paraview.

In this project, we develop an efficient representation for processing and storing information for millions of particles. We also develop fast visualization methods that helps us in understanding and analyzing the data.

Chapter 2

Background

Physical quantities such as temperature, velocity, density and pressure are often represented as scalar fields within a domain. A *manifold* $\mathbb{M}$ is a topological space that locally resembles the Euclidean space. For an n-dimensional manifold, $\mathbb{M}^n$, each of its point will have a neighbourhood that is homeomorphic to the n-dimensional Euclidean space, $\mathbb{R}^n$. A *scalar field*, $\mathbb{S}$, on region $\mathbb{U} \subseteq \mathbb{M}$, is a function defined as

$$\mathbb{S} : \mathbb{U} \longrightarrow \mathbb{R}$$

In other words, every point in $\mathbb{U}$ will have some scalar value associated with it. In this report, we talk about scalar field defined on $\mathbb{R}^3$. In practice, scalar field is available as discrete samples at some vertices in the domain. Scalar values for the rest of the points in space is obtained by linear interpolation.

Scalar fields are visualized by extracting and displaying isosurfaces or level sets. For a given *isovalue* v, *level set* L is the preimage of scalar field S

$$L(v) = S^{-1}(v)$$

Level sets of 2D scalar fields are 1D curves called *isocontours* and level sets of 3D scalar fields are 2D surfaces called *isosurfaces*.

Gaussian density function [1] can be used to represent the surface of a molecule. A molecule consists of atoms and each atom has some radius and position. SPH simulation data file contains information about particles' location. Each particle has a fixed radius. We use the same approach mentioned above to get the envelope surface represented by particles. Gaussian

density function [1] used in this report is defined as

$$D(x, y, z) = \sum_i b_i \; e^{-a_i r_i^2} \tag{2.1}$$

where gaussian is centered at r_i, b_i is the height of the gaussian bump and $1/a_i$ is the variance.

A surface can be defined as those points where this gaussian density function equals a given threshold value T:

$$F(x, y, z) = D(x, y, z) - T$$

Setting

$$T = b_i \; e^{-a_i R_i^2}$$

where R_i is the radius of particle, we get

$$a_i = -\frac{ln(T/b_i)}{R_i^2}$$

A blobbiness parameter, B_i, may be defined as

$$B_i = ln(\frac{T}{b_i})$$

so that solving for b_i we get

$$b_i = T \; e^{-B_i}$$

We set $T = 1$. A threshold of 1 is convenient since its logarithm is 0. Surface defined by one particle is then a quadric surface. Blobbiness B_i can be changed to decide the smoothness of the surface. In our implementation, B_i is set to -0.1.

SPH simulation generates data in the form of particles. This data contains particle information in $\mathbb{R}^3$. For visualization, it is desirable to have the input defined over a regular grid. An interpolation technique is required to generate uniform grid data from scattered data. Shepard's method is one of the earliest techniques used for interpolation [8, 4]. *Shepard Interpolation* [5] is the weighted average of the values available for n points. To obtain interpolated value at point p, we use

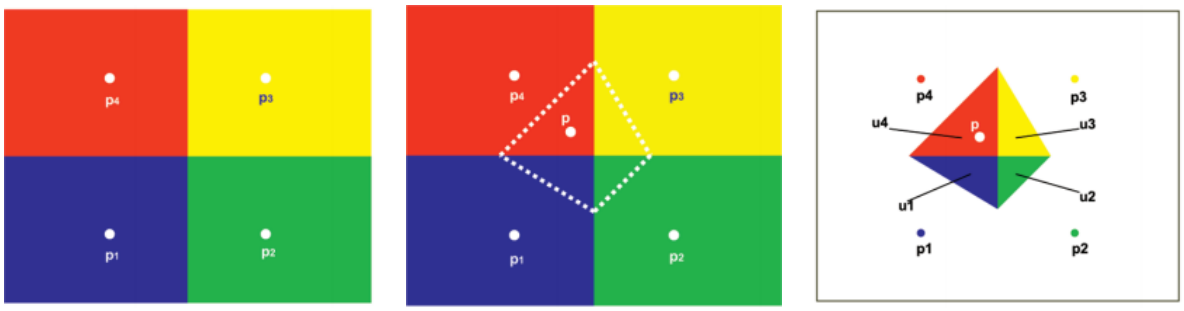
$$x_p = \frac{\sum_i u_i f(x_i)}{\sum_i u_i} \tag{2.2}$$

where $u_i = \frac{1}{d(p,x_i)}$, $d(p, x_i)$ is the distance between point p and x_i, $f(x_i)$ is the function value at x_i. This method is global because the interpolation function is evaluated for all grid points.

Chapter 3

Related Work

Several methods have been developed for converting scattered point data into a grid and then visualizing it. Sibson Interpolation [8, 4] starts out with Voronoi diagram for given points. When a new point is inserted, it updates the Voronoi diagram and calculates the contribution from neighbouring sites based on area/volume in the original Voronoi diagram. Figure 3.1(a) shows original Voronoi diagram for sites p_1, p_2, p_3 and p_4. In figure 3.1(b), a new site p is inserted and Voronoi diagram gets updated. Figure 3.1(c) shows the contribution from neighbouring sites of p based on their area in original Voronoi diagram.



(a) (b) (c)

Figure 3.1: Contribution from neighbouring Voronoi cells based on the area in original Voronoi diagram. (a) Original Voronoi diagram for sites p_1, p_2, p_3 and p_4. (b) Updated Voronoi diagram because of a new site p. (c) u_1, u_2, u_3 and u_4 is the contribution from p_1, p_2, p_3 and p_4 respectively.

Some methods have been developed to visualize the molecular surface given positions of atoms [2, 1, 3]. Molecular Surface definition is introduced in [1] called Metaballs. This is an implicit surface defined as all points $p \in \mathbb{R}^3$ which satisfy a certain equation $F(p) = 0$. Each particle i is represented by a density function $D_i(p)$ that degrades with distance to the atom

center a_i. The density value of all particles is added up for each point to a global density field $D(p) = \sum_i D_i(p)$. The isosurface is defined by a threshold value T as $F(p) = D(p) - T$. We have extended this approach to find the surface represented by particles using gaussian density function. Surface can also be extracted using Marching Cubes algorithm [3].

Shepard interpolation [5] is another technique to get interpolated values at grid points from a given point cloud. Values at grid points can be calculated using weighted average of the given points. The weight assigned is inverse of the distance between grid point and the particle. We use this interpolation to compute various fields. This approach can be implemented in parallel because the computation of interpolated value for a grid point is independent of other grid points.

Chapter 4

Dataset Description

Simulation data is available for a small cross section of a cylindrical reactor. The reactor is filled with cylindrical shaped pellets. The pellets and the boundary of reactor are represented as particles. The data file consists of 21 million particles and there are some physical quantities like location, density, velocity and temperature corresponding to each particle. Input data file uses XML syntax and support features like compression, binary encoding and little endian and big endian byte order. Each physical quantity is stored as an XML node and text part of node has information for each particle corresponding to physical quantity. Data file containing 21 million particles contains data in little endian byte order. Data corresponding to each physical quantity is compressed using *zlib* library and then encoded in *base64*. Size of data file containing 21 million particles and 6 physical quantities is 2GB.

Chapter 5

Methodology

Current technique visualize the data based on particles' location. This method works well for small number of particles but for millions of particles, this is not an efficient solution. Developing interactive visualization methods also becomes difficult for a large dataset. Main challenge in current approach is the scalability of algorithm i.e algorithm should be able to handle and process millions of particles within seconds. Next challenge is to make the visualization interactive for the user. We handle these challenges in two phases. First phase make use of GPU and CPU cores to achieve parallelism so that large datasets can be processed within few seconds. In second phase, we focus on developing methods for interactive visualization.

The first phase of the project focuses on design of the representation scheme. The simulation produces a collection of particles together with multiple physical quantities computed at each particle location. We build a 3-D grid for storing all variables associated with the flow in the reactor. Fast parallel methods have been developed in the past for computing a gaussian density function for molecular surface computation and visualization [2, 1]. We extend the same approach to compute the gaussian value for each grid point. In order to make our parallel algorithm more efficient, we develop an accelerated data structure which takes the contribution of only those particles that are in the neighborhood of a grid point. We don't consider far particles because their contribution towards a grid point will be negligible. The parameters for gaussian function are set in such a way that the isosurface at $isovalue = 1$ represents the envelope surface of the particles. We also compute various fields like density and velocity field over the grid using Shepard interpolation technique [5].

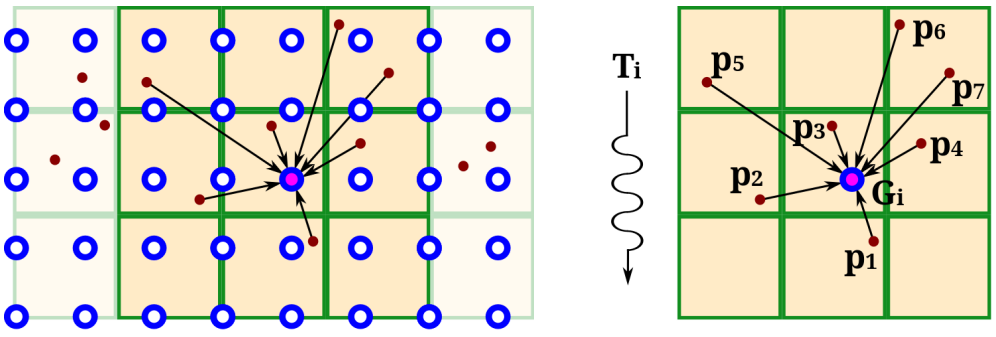
In second phase of project, we develop techniques within Paraview framework for fast and interactive visualization of the earlier generated grid. These methods helps us in understanding the distribution of function values within the field. These techniques involve interactive animation of important slices, identifying a hotspot region within a slice and getting intersection

of slices at global maximum function value. Filters available in Paraview are used to develop these methods.

Chapter 6

Structured Grid Representation

In first phase of project, we implement a GPU based algorithm which outputs a 3D grid for a given point cloud data. User has the choice of selecting a field for computation. We define *supercells* in our implementation as cubes having some fixed width. Supercells helps in making our parallel algorithm more efficient. Width of the supercells is set to *smoothing length*. Smoothing length is the cutoff radius which is used to identify neighboring particles, i.e only those particles that lie inside this radius for a particular point. We use contribution from only those particles that lie in neighboring supercells of a grid point. These supercells are numbered and each supercell may contain some particles and grid points depending on it's width. A supercell may even be empty. In Figure 6.1(a), green boundary cubes represents supercells. Based on a known particle location coordinate inside the grid, the supercell number in which that particle lies can be computed in $\mathcal{O}(1)$ time. The algorithm is implemented in C++ using CUDA.



(a) (b)

Figure 6.1: (a) Contribution from neighbouring particles to a grid point. (b) Each thread launched computes an interpolated value for one grid point.

Algorithm 1 Generating 3D grid from point cloud

Input: Simulation data file containing particles.

Output: VTK file containing scalar fields defined over a grid.

1: Compute range in X,Y,Z axes.
2: Input dimensions for axis having minimum range.
3: Compute dimensions for remaining two axes.
4: Arrange particles according to their supercell number.
5: Launch GPU kernel for chosen fields.
6: Write obtained outputs to VTK file.
7: Compute errors.

6.1 Algorithm Flow

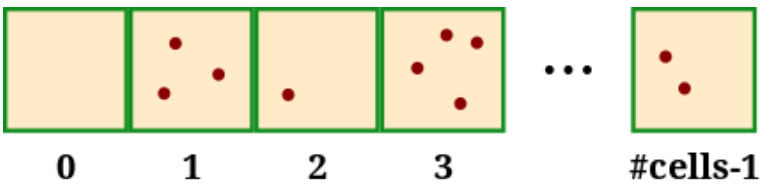
- Initially the input file is read and based on particles' locations, range of particles is calculated in all three axes. User is then asked to enter the dimension for axis having minimum range.

- Based on the dimension entered, step size s is computed and using this step size dimensions for other two axes are also computed.

- A border $(5 \times s)$ is added in each axis so that the effect of boundary particles is also visible.

- Based on particles' locations, all the particles are arranged into supercells so that for a given grid point, particles that are present in a supercell as well as neighbouring supercells, can be quickly accessed.

- An array is maintained to keep track of number of particles in each supercell. Particles are arranged in supercells as shown below. Index indicates supercell number and red dots within a supercell indicate particles.



- Kernels for selected fields are then launched. For each kernel, one thread is launched per grid point. To compute interpolated value at a particular grid point, contribution from neighbouring particles which lie in adjacent supercells is taken into consideration. This is done to reduce the computation overhead as particles that are far away will have negligible contribution.

- When a kernel is launched, many threads are spawned that perform the computation in parallel. Figure 6.1(b) shows a thread T_i performing computation on grid point G_i. Particles $p_1, \ldots, p_7$ denotes the neighbouring particles from adjacent supercells.

- Once we obtain interpolated value at all grid points for a field, those values are written to the output file.

- We also compute Root Mean Square Deviation(RMSD), absolute mean error and max difference error to look at errors which is introduced due to conversion from point cloud to 3D grid.

6.2 Error Computation

We use Shepard interpolation technique to convert a given point cloud into a 3D grid. This interpolation leads to some error. Once a 3D grid is obtained, we use trilinear interpolation to get interpolated value at the particle's location. Let this value be denoted by f' and actual value at that particle's location be denoted by f. We compute three different error measures to study the differences between the grid representation and the input particle data.

6.2.1 Root Mean Square Deviation

Root Mean Square Deviation(RMSD) is computed as follows:

$$RMSD = \sqrt{\frac{\sum_{i=1}^{N}(f_i - f_i')^2}{N}}$$

where f_i denotes the actual function value of i^{th} particle and f_i' denotes the interpolated value computed at i^{th} particle location. N denotes total number of particles.

6.2.2 Mean Absolute Difference

Mean Absolute Difference is given by:

$$AMD = \frac{\sum_{i=1}^{N}|(f_i - f_i')|}{N}$$

6.2.3 Infinity Norm

We compute Infinity Norm as follows:

$$MD = max(|(f_i - f_i')|)$$
$$for\ i = 1, 2, \ldots, N$$

Chapter 7

Visualization

In second phase of project, we develop methods using Paraview framework which helps us in getting a better insight at the data. We implement several algorithms in Python which allows us to look at interesting regions within the field such as hotspot regions, sub-volume extraction and important slices. All these algorithms are interactive and use some of Paraview's inbuilt filters.

Paraview is an open source application for scientific visualization. It is used to analyze and visualize data sets. Data exploration can be done interactively in 3D or by writing programmable filters using Paraview's batch processing capabilities. Paraview provides it's own python shell for writing scripts. We use this functionality to write our python programs which help us in visualizing the above generated grid interactively and efficiently.

We can visualize the generated fields in "volume rendering" mode, an inbuilt function of Paraview. It's a good way to visualize an entire scalar field but it doesn't help when we want to look at some small portion/region of the field which is important from data analysis point of view. To be able to better understand the field, we look at some of the methods which we have implemented.

7.1 Important Slices Animation

Our field is represented as a 3D grid. A "slice" can be defined as a plane obtained from within the grid. For eg. let's say our grid range goes from $z = 0$ to $z = 5$ in Z axis. Then a plane, say $z = 2$, will give us a slice (Figure 8.5(b)).

Looking at a single slice may not provide us sufficient information about the data. So we consider multiple slices and go through them as an animation. A grid may contain many slices so going through all the slices is not desirable. Therefore we would like to choose only those

slices which are important (Figure 8.5(c)).

We define important slices as those slices which contain grid points that have their function value above a certain threshold value. This threshold can be set between 0 and 1. For eg. threshold of 0.8 would mean that we want to select those slices which contain points whose function value is in top 20 percent.

7.2 Sub-volume Extraction

The "volume rendering" functionality in Paraview helps us in viewing the entire field. "Sub-volume extraction" is basically a subset of the original field. This feature helps us in viewing the hotspots within a field.

For a given slice, we look at the maximum function value. We use Paraview's filter called "Extract subset" to get a sub-volume around the grid point that has maximum function value. When we go through all important slices, we can use this feature to get all important hotspots for the field. The dimensions of this sub-volume can be modified by the user (Figure 8.6(a)).

7.3 3-Slice Intersection

The idea here is again to get a better understanding of the data. In this feature, we look at global maximum value(s) of the field and construct three axes parallel slices orthogonal to each other and passing through that point. This helps us in viewing the surrounding area of grid point through slices (Figure 8.6(b)).

To understand the entire work flow, refer to A.

Chapter 8

Experimental Results

We perform computations on input file containing 21 million particles and 6 scalar fields and on another input file containing 9.5 million particles and 4 scalar fields. We use Paraview to visualize our results for both phases of the project. We load the generated output file in Paraview and visualize each scalar field separately.

8.1 Structured Grid Representation

We look at various scalar fields like sum of gaussians, density and velocity field generated using our parallel algorithm.

Figure 8.1 shows the surface represented by particles for data file containing 5 million points. The boundary of the reactor is visible and hollow cylindrical pellets inside the reactor are also visible. We have used gaussian density function [2, 1] to extract the envelope surface of particles.

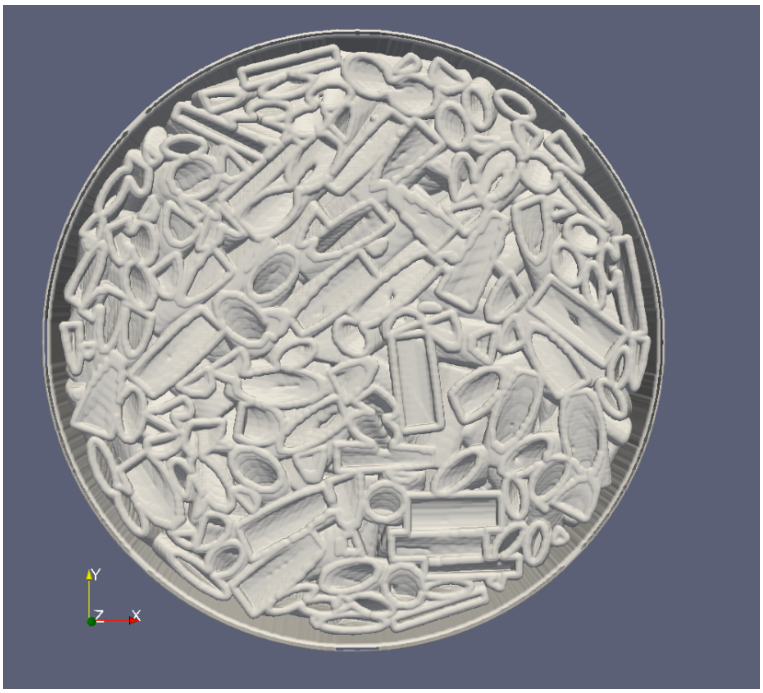
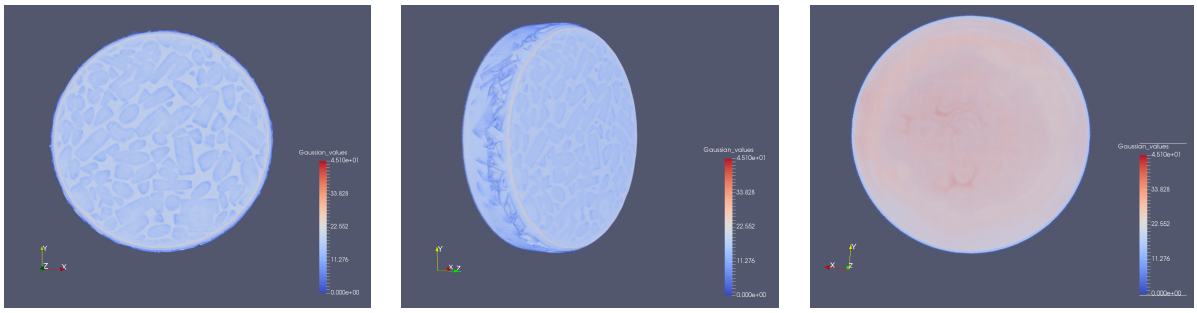


Figure 8.1: For computing the Sum of gaussians field, the parameters of the gaussian density functions were fixed in such a way that at $isovalue = 1$ we get the surface represented by the particles. For input file containing 5 million particles, we obtain the above surface representing boundary of reactor and the pellets within.

For input file containing 21 million particles, the algorithm generates a sum of gaussians field (Figure 8.2). Three different views are shown. Red colour in the back view represents high density of particles in the neighbourhood of a grid point. Blue colour represents less particle density. The given data file shows the start of flow in the initial time step. So we see more number of particles in figure 8.2(c).

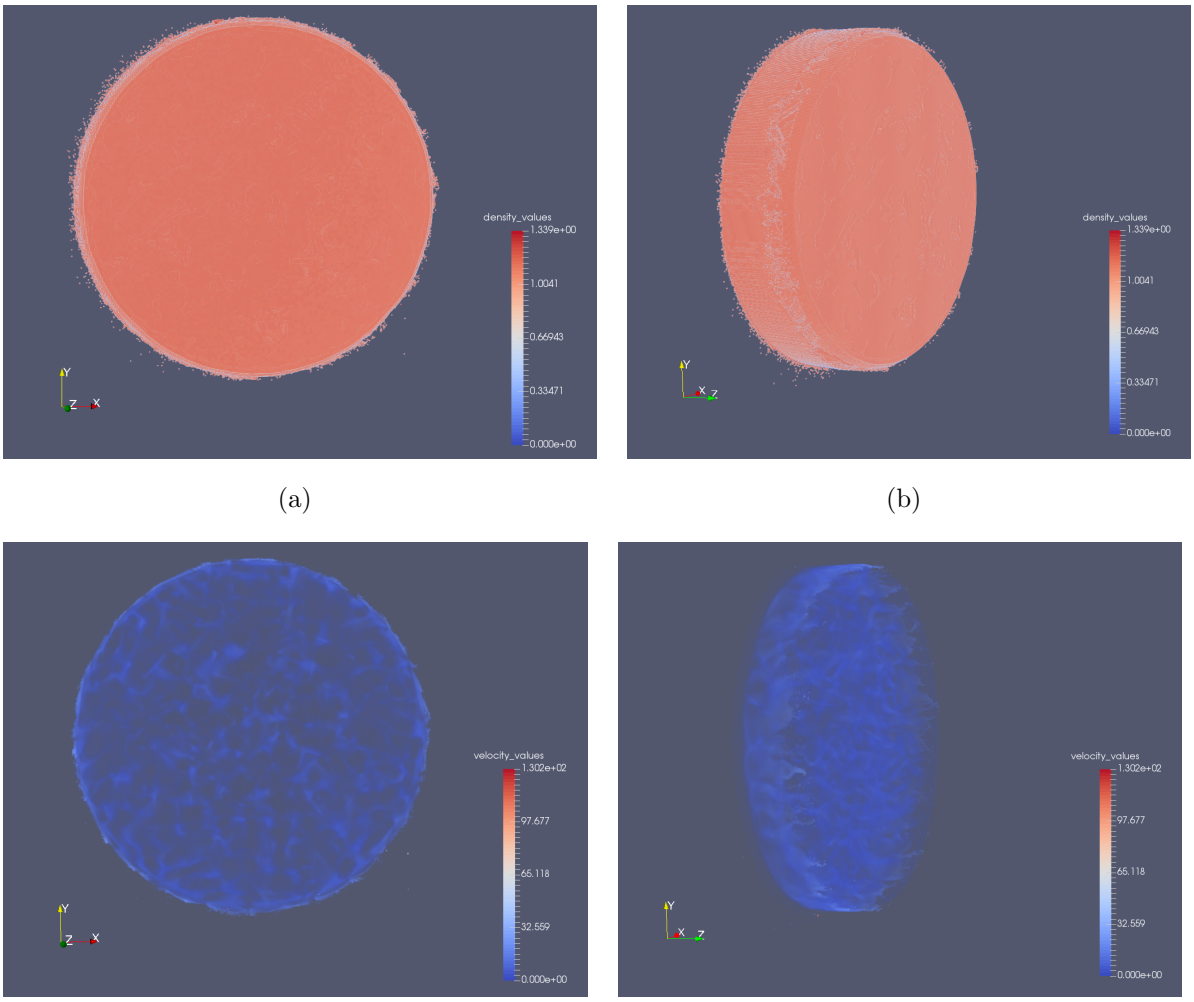


(a) (b) (c)

Figure 8.2: The sum of gaussians field for the input point cloud. (a) Front view. (b) Side view. (c) Back view.

Figure 8.3 contains two scalar fields. Figure 8.3(a) and 8.3(b) shows the density scalar field. In the given data file, particles' density values ranged from 1.08 to 1.33886. For those grid points which don't have any particles in their neighbourhood, density is set as 0. For other grid points, density is computed using Shepard interpolation. So most of the particles appear red.

Figure 8.3(c) and 8.3(d) represents velocity scalar field. For now only magnitude of the velocity is considered to compute the velocity value at a grid point. Shepard interpolation is done for computing this field as well. The data file shows the initial flow, so some particles have velocity and others don't. Accordingly grid points also get interpolated velocity value.



(a) (b)

(c) (d)

Figure 8.3: Density scalar field generated using Shepard interpolation (a) Front view, (b) Side view. Velocity scalar field generated using Shepard interpolation (c) Front View, (d) Side View.

Figure 8.4 shows the sum of gaussians field computed for dataset containing 9.5 million particles. We also measure the time taken by our algorithm on different datasets for various resolutions.

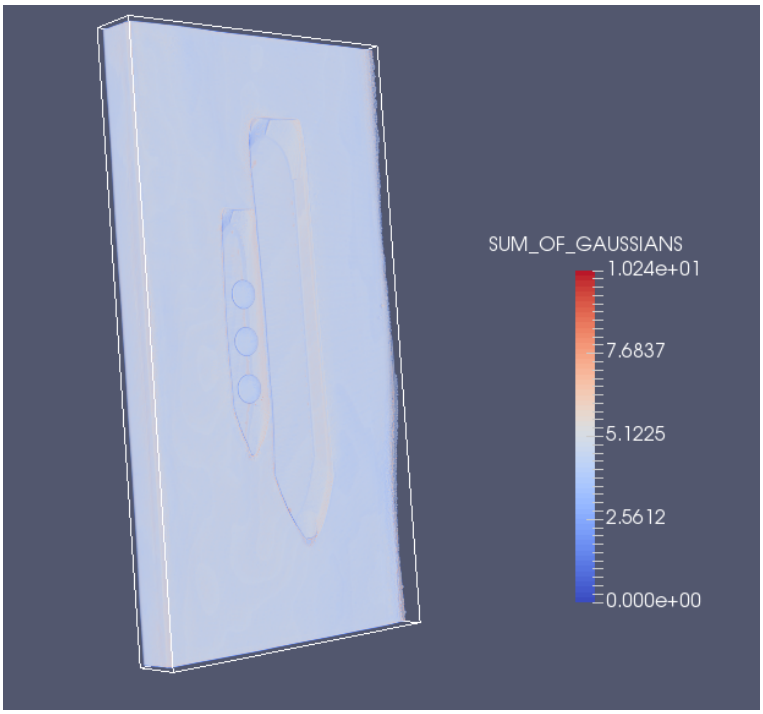
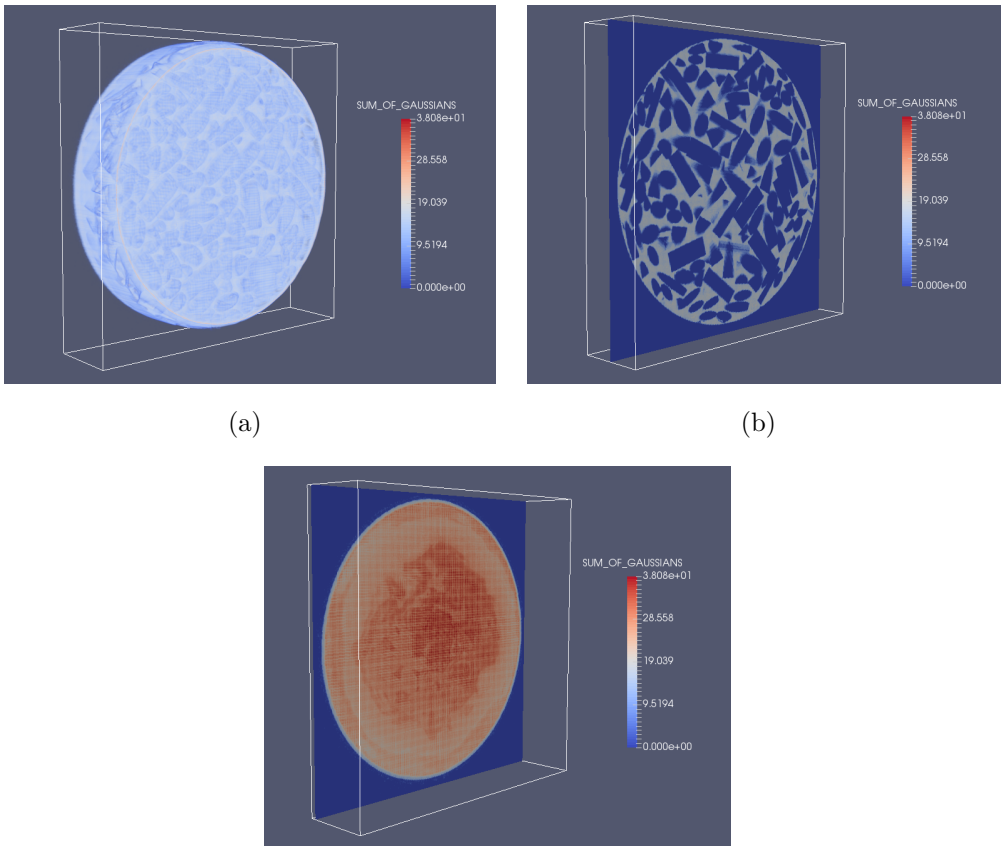


Figure 8.4: The sum of gaussians field for input file containing 9.5 million particles.

8.2 Grid Visualization

Visualization method which was used previously was based on visualizing particles. The results can be visualized in Paraview but Paraview's inbuilt functions like Slice filter, which provides a slice from the field, cannot be used directly for unstructured data. Our approach converts the particle data into structured grid data which is comparatively easy and fast to visualize in Paraview. We visualize the generated grid using Paraview's framework and it's inbuilt functions. We implement various methods for interactive visualization.

Figure 8.5(a) shows sum of gaussians field represented as a structured grid and figure 8.5(b) shows one of the slices from the field. Figure 8.5(c) displays one of the important slices from the field.

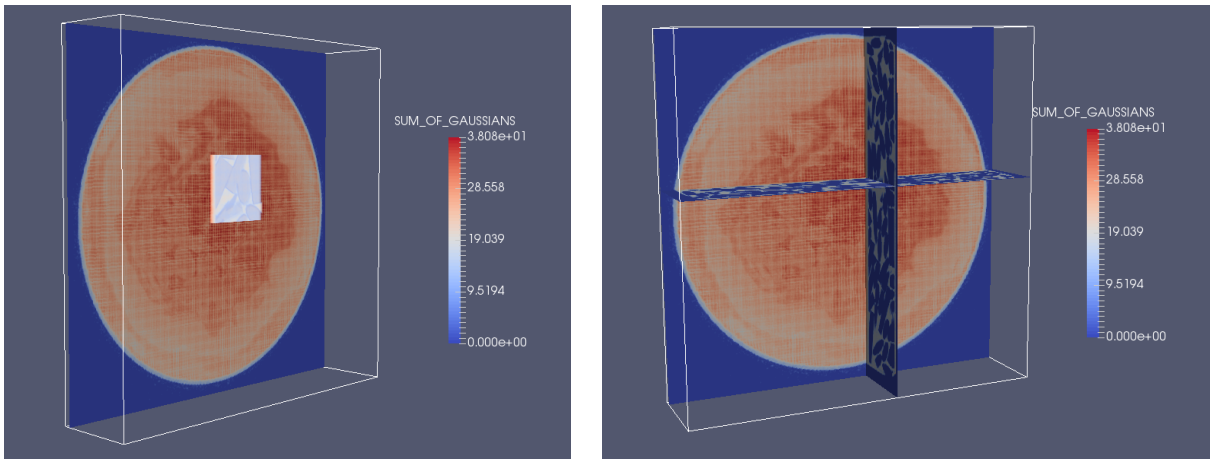


(a) (b)

(c)

Figure 8.5: The sum of gaussians field for the input point cloud containing 21 million particles. (a) Entire scalar field. (b) A slice from the field. (c) Important slice from the field.

Figure 8.6(a) shows the important slice along with extracted sub volume around the maximum function value point in that slice. Figure 8.6(b) shows 3-slice intersection view and the intersection of these slices is at grid point that has the global maximum function value.



(a) (b)

Figure 8.6: The sum of gaussians field for the input point cloud containing 21 million particles. (a) A sub-volume region within a field. (b) 3-slice intersection at global maximum value.

We test our visualization methods on another dataset containing 9.5 million particles. Figure 8.7(a) shows sum of gaussians field represented as a structured grid. Figure 8.7(b) shows one of the important slices from the field. Figure 8.7(c) shows a sub-volume region extracted around the grid point that has maximum function value among other grid points in selected slice. Figure 8.7(d) shows the 3-slice intersection at grid point that has global maximum function value.

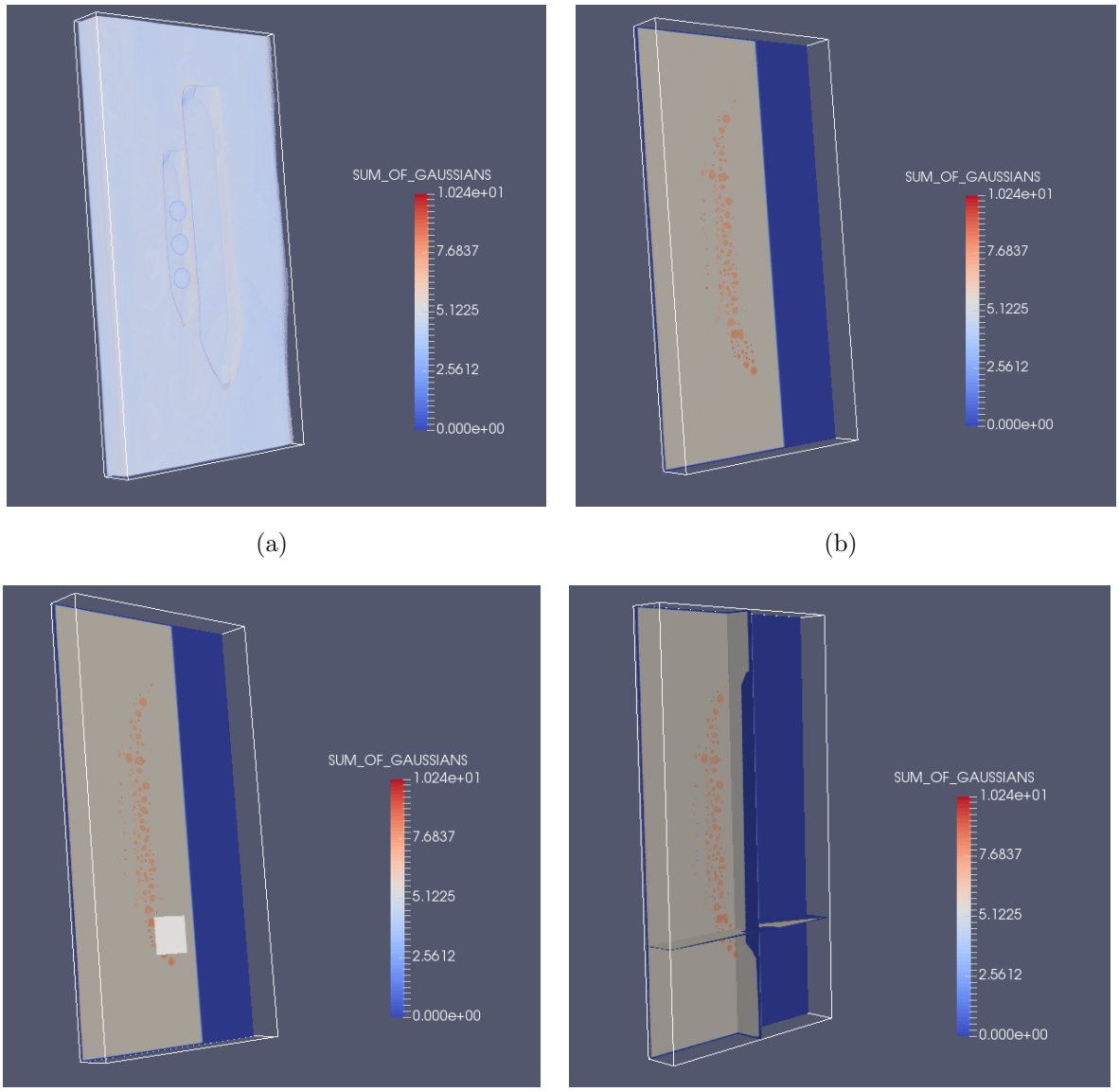


Figure 8.7: The sum of gaussians field for input point cloud containing 9.5 million particles. (a) Entire scalar field. (b) An important slice from the field. (c) Hotspot region for a slice. (d) 3-slice intersection at global maximum value.

We can use these visualization methods for any scalar field to find important slices and hotspot regions present in the field. These methods helps us in getting better insights and understanding about the field.

8.3 Execution Timings and Errors

Table in Figure 8.8(a) shows time taken and GPU memory consumed for different grid resolutions for computing Density field. GPU time includes time taken to transfer data from CPU to GPU, time taken by the kernel and time taken to transfer data from GPU to CPU. CPU computation time is time taken by all the computations which are performed on the CPU. Timings are for Density field. Other fields show similar results. Number of particles are 21 million.

Our main focus is on reducing the overall execution time of the algorithm. Size of the input file is 2GB and it contains information for 21 million particles. We are able to read and store the data in 6-7 seconds (File reading time). This process involves decoding the data from base64 and decompressing it using zlib library. This time is similar for all the grid resolutions as it is independent of that. $478 \times 478 \times 110$ is a low resolution grid. Time taken to compute interpolated values for all grid points is 0.5 seconds and memory consumed is 1.1 GB. As we increase the grid resolution, GPU computation time and memory consumption increases but even for a high resolution grid like $903 \times 903 \times 200$, the change is not much. We are able to compute function value at grid points for high resolution in less than 3 seconds and memory consumed is around 1.7 GB, which is easily manageable.

Table in Figure 8.8(b) shows the error values observed for different grid resolutions. We compute RMSD, absolute mean error and max difference error. Observation shows that as we increase the grid resolution, error decreases. Error is computed for Density field. Number of particles are 21 million. For low resolution grid, RMSD error is less than 3% and as we increase the resolution, RMSD error drops down to 0.03%. Acceptable error value is less than 5%.

Resolution	File reading time	GPU computation time	GPU memory consumed	CPU computation time
478x478x110	6.39 s	.553 s	1115.37 MB	5 s
573x573x130	6.4 s	0.766 s	1179.45 MB	5 s
667x667x150	6.3 s	1.12 s	1270.93 MB	5 s
761x761x170	6.4 s	1.586 s	1394.22 MB	4 s
855x855x190	6.38 s	2.116 s	1552.44 MB	4 s
903x903x200	6.4 s	2.43 s	1649.19 MB	5 s

(a)

Resolution	RMSD	Absolute Mean Error	Max Difference
478x478x110	0.0248642	0.00504691	0.781813
573x573x130	0.00755889	0.00116491	0.267804
667x667x150	0.00181417	0.00027898	0.0655413
761x761x170	0.00037723	0.000101894	0.0326146
855x855x190	0.000321428	8.96055e-05	0.0336068
903x903x200	0.000316099	8.70503e-05	0.0322915

(b)

Figure 8.8: For 21 million particle dataset, (a) Algorithm execution timings and (b) computed errors.

We measure execution timings for structured grid construction and errors on a different dataset containing 9.5 million particles. File size is 750 MB. This dataset contains 4 fields. We construct grid for velocity field. Figure 8.9(a) shows execution times and memory consumption for constructing grid over different resolutions. As we increase the resolution, increase in GPU computation time and GPU memory consumed shows same trend as observed for previous dataset.

Figure 8.9(b) shows different types of errors that we compute on the dataset. We compute RMSD, absolute mean error and max difference error for velocity field. We observe that as we increase the resolution, error decreases. For high resolution grid, RMSD error value is less than 5% which is acceptable.

Resolution	File reading time	GPU computation time	GPU memory consumed	CPU computation time
469x879x80	2 s	0.643 s	658.364 MB	2 s
535x1005x90	2 s	0.905 s	719.652 MB	2 s
601x1130x100	2 s	1.17 s	796.032 MB	2 s
734x1382x120	2 s	1.962 s	1009.81 MB	2 s
866x1633x140	2 s	3.009 s	1313.16 MB	2 s
1065x2011x170	2 s	5.38 s	1977.37 MB	2 s

(a)

Resolution	RMSD	Absolute Mean Error	Max Difference
469x879x80	0.0570539	0.0261558	3.61139
535x1005x90	0.0553036	0.0252846	4.27229
601x1130x100	0.0536974	0.0244511	4.00107
734x1382x120	0.0507763	0.0229742	4.16059
866x1633x140	0.0481528	0.0217172	2.96464
1065x2011x170	0.0447949	0.0200584	2.98

(b)

Figure 8.9: For 9.5 million particle dataset, (a) Algorithm execution timings and (b) computed errors.

Experiments for measuring execution timings and errors for structured grid construction are performed on an Nvidia Tesla K40 graphic card.

Chapter 9

Conclusion and Future Work

We are able to convert the given point cloud data into a 3-D grid and compute interpolated value for each grid point for several scalar fields within seconds using out parallel implementation. We are able to visualize those scalar fields using Paraview.

In the later phase of the project, we develop data visualization methods within the Paraview framework for interactive and more intuitive visualization of the data. The above developed methods can be used at *Shell* for visualization purposes.

In future, there can be methods to compute features of interest such as pores formed by the pellets which will explore the use of topological methods such as Morse-Smale complex [6, 7]. The above developed methods are for single time step. New methods can be implemented for time varying data as well.

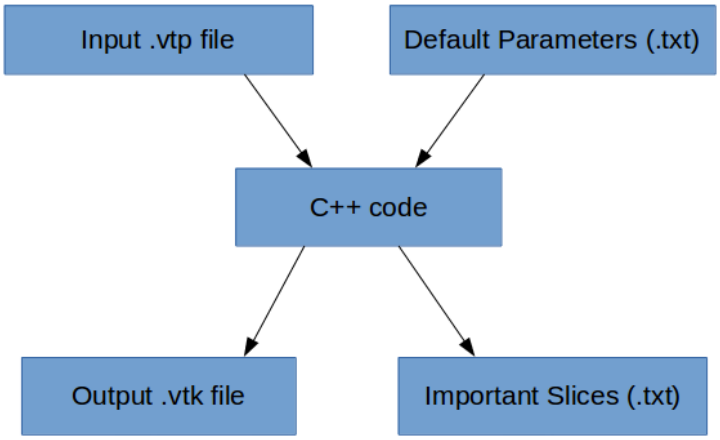
Appendix A

Algorithm Work-flow

We have simulation data file that contains information about the particles. Input file is a "vtp" file. *vtp* files use XML syntax to store the data. Data consists of various fields like velocity and density. Each of these fields is represented as a node and text of node contains the data corresponding to that field. This data is first compressed using zlib library and then encoded in base64. To fetch the data, we need to decode the data and then decompress it. This is repeated for all fields that we want to compute. This vtp file is given as an input to our algorithm for generating structured grid. We use C++ and CUDA for implementation.

We maintain a text file which contains several parameters that we set as default to our program. We provide the name of input file as one parameter. We pad our structured grid using some border. Width of the border is provided as a parameter that can be changed if needed. We get the particle location data from file and compute range of values in each axis. Based on this range, we select the axis that has minimum range and user is asked to enter the dimension for this axis. By default, we set this dimension to 120. This can be changed in the parameter file. We compute three types of errors. Computation of errors is set as another parameter in the file i.e whether to compute errors or not. We also compute important slices in our algorithm. We set a threshold between 0 and 1 to determine which slices to select as important. For eg. let global maximum function value for a field be f_{max}. Let's say that we will consider those slices as important slices that contain some grid point which will have it's function value in top 20%. Then our threshold will be set to 0.8 i.e all those slices will be marked as important slices if they contain atleast one point which has function value greater than $0.8 \times f_{max}$. This is done for one axis. Similarly we set thresholds for other two axes as well. Last parameter is the particle radius. For simulations on different datasets, particle radius may also change. We set this as a parameter so that it is convenient to change.

Our algorithm takes two files as input - Data file and default parameters file. It uses the



data file to convert point cloud in to structured grid using an interpolation technique. This conversion is done on GPU. The structured grid is written to a file and saved as "vtk" file. VTK file has various attributes like DATASET, DIMENSIONS and SPACING. We set DATASET as STRUCTURED_GRID and mention the dimensions i.e resolution of the grid. SPACING is the uniform step size of the grid. We write our output in binary format for each field.

We also compute important slices in our algorithm. This computation is also done on GPU. One thread is launched for each grid point. Slice numbers of all the important slices are written onto a file. We also compute the indices of grid points that have global maximum function value. We create a single text file which contains all this information for each field. Following figure sums up the above description.

We visualize the output vtk file in Paraview. We develop various visualization methods that help us in understanding the data. Paraview provide it's own python shell to write scripts. We develop these methods using python language within Paraview. We use Paraview 5.2.1 version.

We implement a method to interactively visualize important slices. This method takes important slices text file as input and works on corresponding vtk file loaded in Paraview. This method is saved as a *macro* in Paraview. A macro is a python script that is saved as a button in Paraview. Macros are loaded automatically when Paraview is launched.

We also implement sub-volume extraction method for a slice. This method finds the grid point that has maximum function value in that slice. We use subset extraction filter, an inbuilt function of Paraview, to extract a sub-volume around that grid point. The extents of this sub-region can be set accordingly.

We implement 3-slice intersection method. This method takes important slices text file as input and reads indices of grid points that have global maximum function value. Using the index of grid point, we construct three slices orthogonal to each other such that their normal

is parallel to corresponding X-axis, Y-axis and Z-axis , passing through that point.

All these methods within Paraview are saved as macros. Macros are python scripts that are loaded automatically when Paraview is launched. These macros are displayed as buttons in Paraview. We also save visualization methods as *state* files. State file saves all the information about a field as a *.pvsm* file. Visualization methods can be launched by either executing macros or by loading desired state file. Each output vtk file has its corresponding important slices text file.

Bibliography

[1] James F Blinn. A generalization of algebraic surface drawing. *ACM transactions on graphics (TOG)*, 1(3):235–256, 1982. 3, 4, 6, 9, 17

[2] Michael Krone, John E Stone, Thomas Ertl, and Klaus Schulten. Fast visualization of gaussian density surfaces for molecular dynamics and particle system trajectories. *EuroVis-Short Papers*, 2012:67–71, 2012. 6, 9, 17

[3] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987. 6, 7

[4] Sung W Park, Lars Linsen, Oliver Kreylos, John D Owens, and Bernd Hamann. Discrete sibson interpolation. *IEEE Transactions on Visualization and Computer Graphics*, 12(2): 243–253, 2006. 4, 6

[5] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524. ACM, 1968. 4, 7, 9

[6] Nithin Shivashankar, M Senthilnathan, and Vijay Natarajan. Parallel computation of 2d morse-smale complexes. *IEEE Transactions on Visualization and Computer Graphics*, 18 (10):1757–1770, 2012. 28

[7] Nithin Shivashankar, Pratyush Pranav, Vijay Natarajan, Rien van de Weygaert, EG Patrick Bos, and Steven Rieder. Felix: A topology based framework for visual exploration of cosmic filaments. *IEEE Transactions on Visualization and Computer Graphics*, 22(6):1745–1759, 2016. 28

[8] Robin Sibson. A vector identity for the dirichlet tessellation. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 87, pages 151–155. Cambridge University Press, 1980. 4, 6