2012

Visualization of Time-Varying Data from Atomistic Simulations and Computational Fluid Dynamics

Bidur Bohara
Louisiana State University and Agricultural and Mechanical College

VISUALIZATION OF TIME-VARYING DATA FROM ATOMISTIC SIMULATIONS AND COMPUTATIONAL FLUID DYNAMICS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

Division of Computer Science and Engineering
School of Electrical Engineering & Computer Science

by
Bidur Bohara
B.E., Tribhuvan University, 2006
August 2015

To my parents,
Bhim B. Bohara and Sushila Bohara,
for making all of this possible

Acknowledgements

Foremost, I am most indebted and thankful to my major advisor Prof. Bijaya B. Karki, who gave me the opportunity to be part of exciting research projects and who directed this dissertation. Your constant support and guidance throughout my doctoral studies has been crucial to my intellectual development. You have been my mentor in the truest sense and I will always be thankful to you.

I am very grateful to Dr. Gerald Baumgartner, Dr. Robert Kooima and Dr. Michael Malisoff, Dean's Representative, for being part of my thesis committee. I sincerely appreciate your contribution of time and thought, as well as personal encouragement. Dr. Werner Benger, with whom I had the opportunity to work during early years of my doctoral studies, is also due more than one word of thanks. Your passion and drive toward the field of visualization has always been a constant motivation all these years.

Judy Kahn and John Whittaker, I am very grateful to have you as my family in Baton Rouge and for all the love and care you have shown me. I would like to thank my coffee group for inspiring discussions everyday. I am also thankful to friends and colleagues for shaping me into the person I am today, knowingly or unknowingly.

Finally, my appreciation goes out to all my past and present roommates for keeping me sane during doctoral years.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BO	Bridging Oxygen
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
FO	Free Oxygen
FPMD	First-Principles Molecular Dynamics
GB	Giga Bytes
GLSL	Graphics Library Shading Language
GLUI	Graphics User Interface Library
GLUT	Graphics Utility Toolkit Library
GUI	Graphical User Interface
HDF5	Hierarchical Data Format v5
I/O	Input/Output
MD	Molecular Dynamics
NBO	Non Bridging Oxygen
RDF	Radial Distribution Function
UI	User Interface
VBO	Vertex Buffer Object

Abstract

Time-varying data from simulations of dynamical systems are rich in spatio-temporal information. A key challenge is how to analyze such data for extracting useful information from the data and displaying spatially evolving features in the space-time domain of interest. We develop/implement multiple approaches toward visualization-based analysis of time-varying data obtained from two common types of dynamical simulations: molecular dynamics (MD) and computational fluid dynamics (CFD). We also make application case studies.

Parallel first-principles molecular dynamics simulations produce massive amounts of time-varying three-dimensional scattered data representing atomic (molecular) configurations for material system being simulated. Rendering the atomic position-time series along with the extracted additional information helps us understand the microscopic processes in complex material system at atomic length and time scales. Radial distribution functions, coordination environments, and clusters are computed and rendered for visualizing structural behavior of the simulated material systems. Atom (particle) trajectories and displacement data are extracted and rendered for visualizing dynamical behavior of the system. While improving our atomistic visualization system to make it versatile, stable and scalable, we focus mainly on atomic trajectories. Trajectory rendering can represent complete simulation information in a single display; however, trajectories get crowded and the associated clutter/occlusion problem becomes serious for even moderate data size. We present and assess various approaches for clutter reduction including constrained rendering, basic and adaptive position merging, and information encoding. Data model with HDF5 and partial I/O, and GLSL shading are adopted to enhance the rendering speed and quality of the

trajectories. For applications, a detailed visualization-based analysis is carried out for simulated silicate melts such as model basalt systems.

On the other hand, CFD produces temporally and spatially resolved numerical data for fluid systems consisting of a million to tens of millions of cells (mesh points). We implement time surfaces (in particular, evolving surfaces of spheres) for visualizing the vector (flow) field to study the simulated mixing of fluids in the stirred tank.

Chapter 1
Introduction

1.1 Background

Visualization is a study of the transformation of data into visual representations. It is a process of conversion of data into a graphical form so that intrinsic information of the data is evident to the viewer. By displaying things which are otherwise unseen [3], visualization enhances our cognitive ability to gain insight from the data. Essentially, visualization has always been a part of human activity and the one we participate in by observing, seeing and imagining.

With the publication of "Visualization in Scientific Computing" in 1987 [3], visualization has been recognized as a scientific field on its own. Since then the field of visualization has expanded to include three major sub-fields: scientific visualization, information visualization, and visual analytics. Scientific visualization (evolved during 1980's) focuses on the visualization of spatial data associated with the scientific process. Information visualization focuses on the visual representation of non-spatial data commonly found in humanities, social sciences and many other disciplines. In case of scientific visualization, it is relevant and important that the rendered output reflects the physicality of the system being visualized. However, the same may not be required for information visualization, as it depends upon visual metaphors to represent the data. And the third, visual analytics, form a close bond between visualization and other disciplines. It facilitates analytical reasoning with the help of interactive visual interfaces.

Scientific discovery has moved into a new era of simulation-based research in all areas of science. These simulation-based researches can produce a tremendous amount of data. The steps that

follow are the validation of data, the analysis of data, and the discovery of new knowledge. Visualization is a fundamental tool that enables scientists to validate, visually analyze, and make sense of data. Therefore, visualization is widely used in essentially all areas of science and engineering including materials modeling and simulation, which is the target application of our work. For scientists, visualization can serve to formulate new hypotheses, assist in decision making, facilitate effectual communication of ideas, and help share knowledge and discoveries [4].

In this thesis, we present our work on visualization of time-varying data from two different dynamical simulation systems. First, we highlight our contributions toward visualizing position-time series data sets from molecular dynamics (MD) simulation. We have been developing and adding new features, constantly improving the space-time multiresolution atomistic visualization system [5], which is used for rendering and analyzing atomistic simulation data. Throughout this report, we use the term "atomistic" to mean molecular. The visualization system integrates a number of analysis and rendering tools together to extract and visualize a variety of intrinsic information in the time-varying positional data. Our target data is produced by the first-principles molecular dynamics (FPMD) simulations based on inter-atomic interactions derived within quantum mechanical framework. Given the high accuracy of FPMD simulations, it is important to extract the detailed information hidden in the data. For this purpose, instead of only rendering the position-time series data as it is, additional information is extracted by processing the data and rendering on real time. Structural analysis of the atomistic data is conducted by computing the radial distribution functions (RDF) and visualizing the coordination environments and clusters formed by constituent atoms. Alternatively, the atom trajectories are visualized to understand the dynamical behavior. We report improvement toward the structural and dynamical visualization of the atom trajectories from the FPMD simulations.

Second, we present an approach towards developing a visualization environment to explore the time-dependent features of the fluid flow by generating time surfaces of the flow. Time surfaces are higher dimensional extensions of timelines, which are the evolution of a seed line of particles in the flow of a vector field. In contrast to time lines, time surfaces may require the refinement of

triangle primitives of surface based on criteria such as triangle degeneracy, triangle area, surface curvature, etc. Our target data are time-dependent vector fields in the stirred tank, generated from Computational Fluid Dynamics (CFD) simulation.

The rest of this thesis is organized as follows. Chapter 1 introduces the field of visualization, problem domain of our research, and the motivation behind the visualization problem studied in this thesis. Chapter 2 presents a description on the atomistic data to be visualized, a review of previous studies related to atomistic visualization, and a description of various approaches for rendering and analyzing structural and dynamical behavior of data. In Chapter 3, we highlight results for application of the visualization system to a simulated silicate melt. Chapter 4 presents a description of data from CFD simulation and the design, implementation and analysis of the visualization approach. Finally in Chapter 5, we present conclusions and future work.

1.2 Motivation

Atomistic simulation based on first-principles molecular dynamics can produce large amounts of positional data for constituent atoms or molecules [6, 7]. Two approaches for a rapid navigation through the data are animations and trajectories rendering. Particle trajectories allow a complete representation of the position-time series by rendering positions of all atoms for whole time-steps as points or line segments so that full simulation information is contained in a single display. However, trajectory rendering becomes too crowded because there are so many trajectories that are long, distributed in 3D space, and overlapped with each other. The degree of crowdedness is expected to increase with the size of the data, which depends on the number of atoms and the number of time steps. The challenge is how we can render the atomic trajectories with a minimum clutter/occlusion and encode useful information along the trajectories.

Our objective is:

To overcome the clutter/occlusion problem associated with the trajectories' rendering of positional dataset while preserving the dynamical and structural information contained in the data.

We approach the atomic-trajectories occlusion problem in multiple ways. First we explore approaches such as selective trajectory rendering, region based rendering, and multiple positions merging to reduce the number of line segments. Then we extend the basic merging technique to an adaptive hierarchical scheme for merging multiple positions along trajectories, which significantly reduces the number of points/lines segments. The trajectory geometry and underlying atomic structure become increasingly visible after merging, so the nature and extent of atomic arrangements and movements can be better assessed.

Chapter 2
Atomistic Visualization

Molecular dynamics simulation is a technique to study the time evolution of atomic degrees of freedom by solving the Newtonian equations of motion numerically [6]. This approach allows us to study the atomic process in large and complex material systems at atomic time- and length scale through a microscopic view. The simulation treats atoms as classical particles, and the interatomic forces provide interactions. The atoms interact by exerting forces on each other and they follow Newton's equations of motion. Based on the interaction model and the equations of motion, the simulation computes the trajectory of atoms.

With rapid advances in the computational resources and capabilities, it is feasible to treat the interatomic interactions on first-principles basis derived within a quantum mechanical framework. This approach does not require any a priori knowledge of interatomic interactions, and the atomic forces are computed directly from the sophisticated electronic structure calculations of the system, in contrast to the classical simulations, where the empirical/semi-empirical parametrized (such as pair-wise) interaction models are used. Atomistic visualization provides a means to visually explore, discover, and understand the structural and dynamical properties of the simulated atomic system.

2.1 Molecular Dynamics Data

2.1.1 MD Simulation Method

In any MD simulation, including first-principles molecular dynamics, an atomic system is defined by a set of properties representing atomic positions, atomic types, and constraints that have to be satisfied. Let, $A = \{P, \Lambda, C\}$ describes the atomic system, such that P is the set of atomic positions in the simulation space. Λ is the set of atomic species in the system. The larger the number of species types the more complex the interactions among atoms in the system. C is the constraints that have to be satisfied in the system. For example, in the simulations based on canonical NVT ensemble, the number of atoms (N), volume (V), and temperature (T) of the system are fixed.

As simulation progresses, a new set of atomic system (A) is generated. The collection of atomic configurations (A) generated at fixed time-steps over a finite simulation period describes the system's dynamics (atomic trajectories).

$$D = \{A(t) \mid 0 \leq t \leq \tau\} \tag{2.1}$$

where τ is the total simulation time.

MD computes positions at time t, $p_{\mathrm{i}}(t)$, of each atom i by numerically integrating Newton's equations of motion:

$$m_i \frac{d^2 \vec{p_i}}{dt^2} = \vec{F_i} \tag{2.2}$$

where m_{i} is the mass of an atom i, and F_{i} is the force on an atom i due to all other atoms j N, which is computed as:

$$\vec{F_i} = \sum_{i \neq j} \vec{f_{ij}} \tag{2.3}$$

The interatomic forces f_{ij} at each time snapshot are computed directly from the sophisticated electronic structure calculations of the atomic system [5].

2.1.2 Description of Data

If A is the atomic system with n_α number of atoms of species type $\alpha \in \Lambda$, and $p_{\mathrm{i}}(t)$ is the position of atom i at time t, then positional data of n_α atoms at time-step t represent the snapshot of atomic system at that particular time:

$$P(t) = \{p_i(t) \mid 1 \leq i \leq n_\alpha\} \tag{2.4}$$

If a constant time interval of Δt is used for taking snapshots of an atomic system over the simulation period, then the dataset consists of n_{step} snapshots of the atomic system's configuration. Thus the complete position-time series dataset can be represented as,

$$D = \{P(j\Delta t) \mid 0 \leq j \leq n_{\text{step}}\}, \quad t = j\Delta t \tag{2.5}$$

Raw data of a simulated atomic system consists of normalized *XYZ* coordinates of the atoms' positions inside the simulation cell, as shown in Figure 2.1. The first six lines are metadata on how to interpret the raw data. Metadata consists of information about total steps in simulation runs, the number of atoms, dimension and volume of simulation cell, temperature condition and the simulated system's name. The *Konfig* line marks the beginning of a new snapshot, and the following rows of lines consist of position coordinates of all atoms for that snapshot. Furthermore, each line represents *XYZ* coordinates of an atom composed as three columns, as seen in Figure 2.1. For an atomic system of silica liquid with 72 atoms, there are as many lines of coordinate data for every snapshot denoted by *Konfig*. Notice that the coordinate values of raw data are normalized and are in the range $[0, 1]$, hence, should be multiplied by the simulation cell dimensions (lattice vector) prior to using in visualization/analysis applications. For positions data to be correctly associated with atom species, we require additional information that provides mapping of atom indices to the species kind. Usually, this information is provided as a separate input file.

```
60000
 84  84   1
 0.1230407E+02  0.1011058E-08  0.1011058E-08  0.1011058E-08  0.5000000E-15
  2500.000000000000
 CAR
Hydrous Silicate
Konfig=           1
  0.19629916  0.99787529  0.68324408
  0.78426848  0.60795386  0.74735296
  0.99897023  0.91388691  0.97186834
  0.27158085  0.61934914  0.65887542
  ------
  ------
Konfig=           2
  0.19612395  0.99831080  0.68359434
  0.78420407  0.60841625  0.74630483
  0.99912297  0.91405507  0.97169959
  0.27107856  0.61882268  0.65939377
  ------
  ------
```

Figure 2.1: Layout of raw positions data at multiple time snapshots.

Lattice Vector

The simulation space can be expressed as a 3D vector, $a \in R^3$ such that $\{a^{\mathrm{i}} \mid 0 \leq i < 3\}$ gives the length of side *i* of the simulation cell. Usually, all sides of the cell are of equal length but are also likely to be of different lengths. Here, vector $\boldsymbol{a}$ is said to be a lattice vector.

Periodic Boundary Condition

If atomic system simulation considers that a simulation cell is being replicated throughout space to form an infinite lattice, then as atoms move in the central cell, their periodic image in all other cells replicates their dynamics. Thus when an atom leaves a simulation cell, one of its periodic images enters from the opposite face. Such a system is said to follow the *Periodic Boundary Condition*. There are no walls at the boundary and the number of atoms in the cell stay constant.

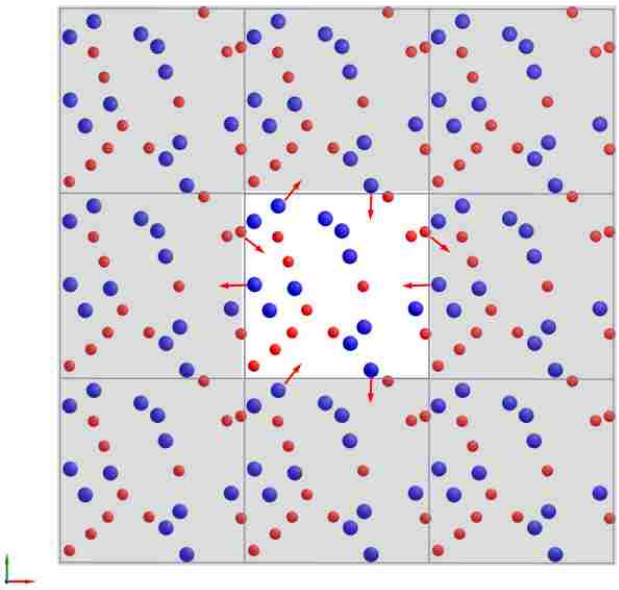


Figure 2.2: Schematic illustration of a periodic boundary condition in the simulation cell.

2.2 Related Work

Molecular visualization is one of the most widely studied scientific applications of computer graphics and visualization [5, 8–12]. To gain insight into MD and FPMD simulations, visualization has previously been exploited in various ways depending on the nature of the atomic systems studied. A few common examples include VMD [9], Molscript [13], XcrysDen [14], Atomeye [15], etc. In addition to rendering the atomic configurations in various forms, they also support animation and trajectories. To the best of our knowledge, not much work was previously done in manipulating the atomic trajectories.

Trajectory rendering is also relevant for other movement-related data such as traffic network consisting of roads and trails [16]. Previous works on visualizing large amounts of movement data (trajectories) suggest a method for temporal and spatial aggregation of movement data to transform the trajectories into aggregate flows between regions. These aggregation method are based on extracting significant points from the trajectories to convey the essential characteristics of the movement [17] with some visually clear representation. Removing selected trajectories or

portions of those trajectories, and line simplification of highly curved trajectories can help reduce the clutter to some extent [18]. Clustering similar trajectories with respect to material properties or visualization parameters can be abstracted to represent the principal paths in the flow and reduce the cluttering [19].

It is relevant to point out that the pathline or trajectory technique is widely used in flow visualization [20] with a lot of efforts made for correct illumination and 3D rendering of lines [21, 22] and encoding the relevant information [23]. The majority of flow visualization uses numerical integration methods to generate pathlines for representing the velocity vector of the flow [24–26]. For the particle-based flow fields, an alternative approach for tracing pathlines is to interpolate particle positions from the particle neighborhood in consecutive time steps [27]. In the case of MD simulations, the input position-time series directly represent the trajectory data so that trajectory rendering displays the spatio-temporal behavior of an atomic system.

2.3 Data Model

Molecular dynamics simulations can generate positional data for very large systems, even exceeding one billion atoms [6, 7, 28] and can extend for runs of more than one million time steps. Typically, most of the simulation packages export data in ASCII format, which is human readable. Just because ASCII files are human readable doesn't necessarily mean they are easy to read for computer programs. This section addresses the hurdles of reading atomistic simulations data from ASCII format for visualization.

Reading an ASCII file requires parsing of the text to retrieve data values. It can be tricky when an ASCII file contains metadata along with the data values, as parsing is sensitive to text layout. And slight changes in ASCII format can introduce an erroneous reading. One scenario is when a simulation application changes the format of an exported ASCII file. Then, reading an ASCII file may either fail or introduce an error in the data. Since ASCII format stores each character as a byte, the files are larger compared to other formats - in particular binary formats. Moreover, parsing a large text file is extremely slow and requires lots of memory space.

These hurdles with ASCII data format compelled us to consider alternative data formats, primarily those developed for storing scientific data. Besides, we also require data format to provide support for random partial access of dataset and independent access to metadata. Exploring Hierarchical Data Format (HDF5) was an important step toward high performance storage of data and metadata.

2.3.1 HDF5

Hierarchical data format v5 (HDF5) is "a versatile data model that can represent complex data object and wide variety of metadata" (www.hdfgroup.org). HDF5 is also a software library that provides well documented API, a file format that has no limits on the number of data objects and file size, and a container format that includes objects such as groups, datasets, and attributes.

HDF5 *datasets* are multidimensional arrays of atomic or compound datatypes. *Attributes* allow annotation of HDF5 objects with the metadata, and *groups* provide organization of datasets and attributes in association that defines a complete data. An HDF5 data model organizes these objects in a rooted-directed graph. The organization of data in HDF5 format and how data is tailored to these objects is up to the application scientist/user.

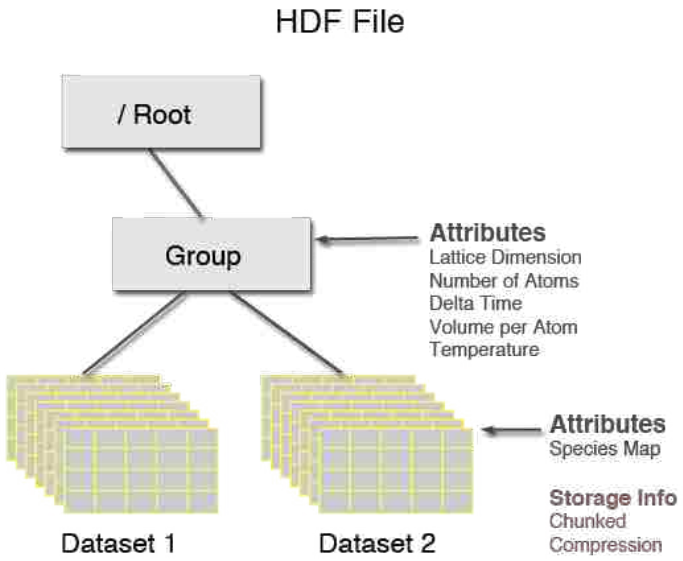


Figure 2.3: Hierarchical organization of HDF5 objects - groups, datasets, and attributes.

Since HDF5 is a container format that stores multidimensional arrays with multiple data layout possibilities, providing data in HDF5 format does not necessarily guarantee correct interpretation by the reading application. In other words, being able to open and interpret one HDF5 file doesn't mean that the program can interpret any other HDF5 files. Hence, both writing applications and reading applications should agree on the layout of the dataset and metadata to make the data readable. Otherwise, even though a file is in HDF5 format, the dataset will be unreadable to the application. With multiple possible ways to lay out data in the HDF5 format, the choice of the right layout is critical for the final performance. So combined with community specific convention and application specific semantics, HDF5 can provide robust and reliable foundation for high performance storage and access of data and metadata.

2.3.2 Data Storage Layout

The HDF5 data model provides various objects to map data into HDF5 format. Raw data is organized in a multidimensional array and stored as *datasets*. Multiple correlated datasets can be bundled together in a *group*, and a group can contain one or more datasets and/or groups. Finally, metadata that contains information on how to interpret data is mapped to *attributes* associated with groups/datasets. The raw data from atomistic simulations contains time-varying positional data of atoms. And we require the data model to provide random access to partial dataset and metadata, either a time-wise access of the ensemble, or access to all time-steps data of an individual atom.

The earlier approach was to group all data related to the same simulation time-step in a single *group* so that, a group object was created for each time-step which contained data of all atoms for that particular time. For an ensemble of a few hundred atoms, the datasets created for single time step were relatively small arrays. This way the application can have random access to datasets for any particular time step by traversing to the respective group, but, there are few tradeoffs with this approach. First, to generate a trajectory of a single atom, datasets at all time-steps will have to be read to extract positions(trajectory) data of a particular atom. This can be very inefficient because the reading program will open and read a whole dataset while it only needs to read a

subset of dataset. Repeating this behavior for thousands of time-steps will introduce a significant memory overhead as a result of copying numerous datasets into application memory. Second, for longer simulation runs with a large number of time-steps, this approach will create as many numbers of *group* objects. The storage overhead involved in representing many group structures in HDF5 format is significant, even more so when there are many groups, each containing small arrays (datasets).

To overcome these tradeoffs, our second approach to a data model is to map time-varying positional data onto a three-dimensional dataset. The raw data is organized as a three-dimensional array with the layout of $[NUM\_ATOMS] \times [NUM\_STEPS] \times [DIMS\_POSITION]$, where $DIMS\_POSITION = 3$ for atoms' coordinates in three-dimensional space. Finally, positions dataset and other datasets (if any) are all bundled together under the *root* group of an HDF5 file.

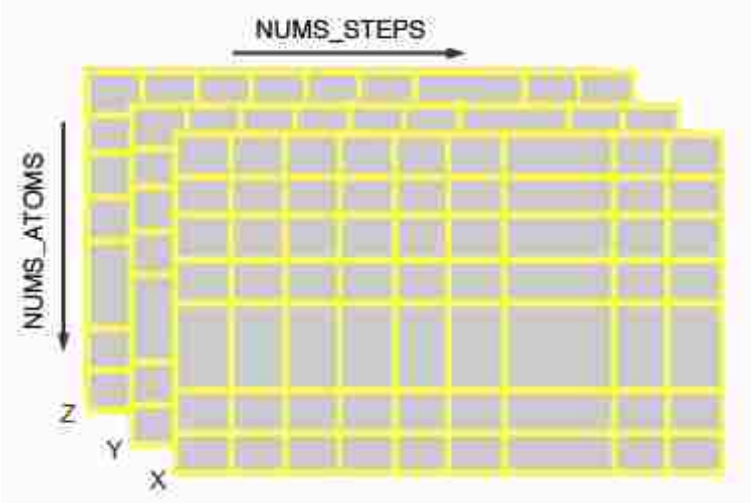


Figure 2.4: Data storage layout of a three-dimensional dataset that stores time-varying positional data of all atoms. Each row stores positions data of an atom for all time-steps, whereas each column stores positions data of all atoms at a particular snapshot. Depth of the dataset stores XYZ coordinate values.

2.3.3 Partial I/O

The HDF5 data model provides a data selection feature to access partial dataset using *hyperslab*. A hyperslab slab is used to access a subset of a large dataset, and is generally used for copying

selected data from memory (dataset) to dataset (memory). However, performance of partial I/O on a dataset is dependent on the storage strategy of HDF5 storage model, on whether the dataset array is stored in the file as contiguous blocks, or as chunked blocks. By default, data is stored contiguously as a single continuous array of bytes. It is the simplest model and has limitations while accessing subsets of a dataset. As an alternative strategy, a dataset can be stored as fixed-sized chunked blocks. When defined as a chunked dataset, each chunk is written or read in a single I/O operation and data belonging to a single chunk is stored contiguously. Since data is transferred in chunks, chunking strategy on a dataset can have a dramatic effect on the performance of partial I/O of a dataset. Further more, using hyperslab selection on an individual chunk will have a prominent effect on I/O performance by optimizing the partial access of the dataset.

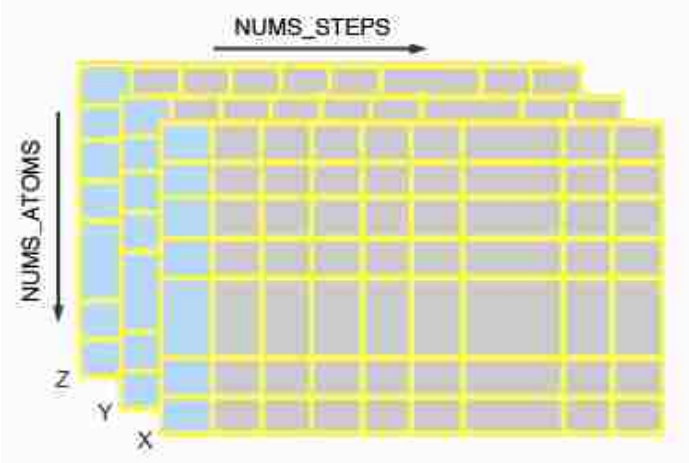


Figure 2.5: Hyperslab selection (blue cells) to read positions data belonging to the first snapshot.

Given three-dimensional dataset storing time-varying positional data of atoms, our visualization implementation requires two different data access strategies: time-wise sequential and per-atom wise trajectory access. Time-wise sequential access is favorable for processing individual time snapshots, for example, while rendering animation, computing radial distribution function, or rendering coordination environment. Efficient time-wise access of data for all atoms requires that time-step data be stored contiguously in a single chunk. This way, data is stored in chunked blocks with at least NUM_STEPS chunks per dataset. Hyperslab selection of a chunk will allow the

partial I/O access to the data belonging to a single snapshot. However, time-wise sequential access is not favorable while processing trajectory for an atom or for a set of atoms (either atoms of same species or atoms in coordination) because a whole dataset has to be copied to memory to extract subset of data, increasing memory and processing overhead.

Thus, per-atom wise access is more relevant while rendering the trajectory of an atom because processing trajectory data is more efficient if consecutive positions of an atom are stored contiguously in a dataset and/or in memory. Hence, in this case, the chunking strategy is to store all positions data (beginning with the first time step until the end) belonging to an atom as a single chunk. This way, a dataset is stored in chunks with at least NUM_ATOMS number of chunks. And the hyperslab selection of a chunk will allow the partial I/O access to the trajectory data of the corresponding atom.

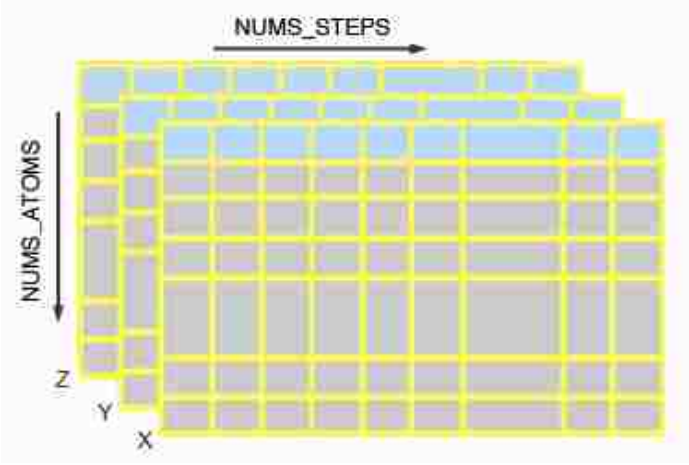


Figure 2.6: Hyperslab selection (blue cells) to read the trajectory data of the first atom.

Evidently, only one chunking layout can be defined per HDF5 dataset, so what chunking strategy to apply may depend upon the total number of atoms and number of time-steps. For dataset with millions of time steps but very few atoms will require a different strategy than a dataset with millions of atoms for only few time-steps. It should be noted that defining more chunks per dataset will increase the file storage overhead. But external compression filters can be applied with chunks to compress a dataset and hence the HDF5 file. We have defined two datasets corresponding to

two different chunking strategies in a single HDF5 file. Consequently, there is a storage overhead due to redundancy in stored data which is not a better approach storage-wise. However, the improved efficiency in a partial I/O of dataset and the application memory usage well compensates the storage overhead in an HDF5 file.

2.4 Visualizing Structural Properties

2.4.1 Radial Distribution Function

Radial distribution function (RDF) is the simplest and most important quantity for the structural analysis of molecular dynamics data. It is also known as the pair-correlation function. The function gives the probability of finding a pair of atoms at distance r apart. Hence, RDF represents the average radial packing of the atoms and is frequently used for concisely characterizing the structure of complex systems such as liquids and colloidal suspensions [29]. Computing radial distribution function is essentially the first step towards the structural analysis of an MD system. The radial distribution function for an atomic species pair of α and β is defined as [30],

$$g_{\alpha\beta}(r) = \frac{1}{4\pi\rho_\beta r^2}\left[\frac{dN_\beta(r)}{dr}\right] \tag{2.6}$$

where ρ_β is the number density of species β and N_β is the number of species β atoms within a sphere of radius r around a selected atom of type α. In other words, for a species pair of α and β, $g_{\alpha\beta}(r)$ represents the probability of finding atoms of species β within the radial width dr at distance r from a reference atom of species α, as shown in Figure 2.7.

A distribution function that considers all atoms irrespective of their type is called total-RDF. Total RDF gives the total number of atoms per volume that can be found in the shell of width dr at a radial distance r. Since this function doesn't differentiate between atoms of different species type, it is usually not considered in studying multi species atomic systems. Likewise, the distribution function that is defined for atoms of species pair $\alpha - \beta$ is called partial-RDF. Since partial-RDF

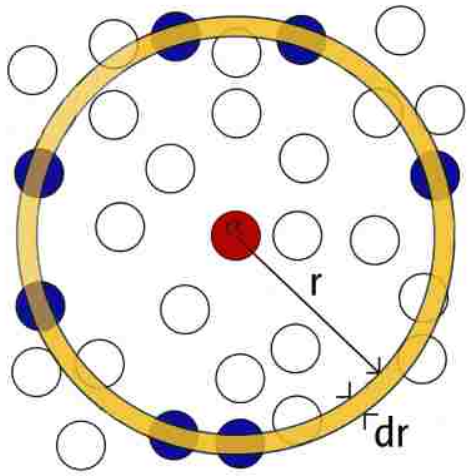


Figure 2.7: Computing $g_{\alpha\beta}(r)$ using spherical space discretization from a reference α atom [1].

takes the atomic species type into consideration, this function is mostly used in analyzing the distribution of atoms of one species in the neighborhood of another atomic species. It is subsequently used for the computation of coordination environment between an atomic species pair.

Cutoff distance (r_{start}) is the distance where the RDF first becomes a non-zero (Figure 2.8). It gives the minimum allowable distance between two atoms, since no two atoms can come closer to each other than a certain distance. And the minimum RDF distance (at lowest point) after its first peak is known as distance to first-minimum (r_{min}). We can say that atoms within distance of r_{min} are originally under the first peak in the atomistic system.

Furthermore, RDF is computed over an extended simulation period; otherwise, RDF may show highly fluctuating characteristics and may not be insightful. Computing radial distribution functions is one of the most computationally intensive tasks. So it is sensible to compute RDF only once for a given dataset and save distribution functions in a file for future references. Moreover, saved RDFs of an atomic system computed for different temperature and pressure conditions can be displayed together by overlapping multiple distribution functions in a single RDF plot. This will also allow us to easily compare the distribution functions across varying conditions, as shown in Figure 2.8. The vertical sliders in a graph window can be adjusted to select the cut-off values for corresponding coordination pairs to be used for structural analysis.

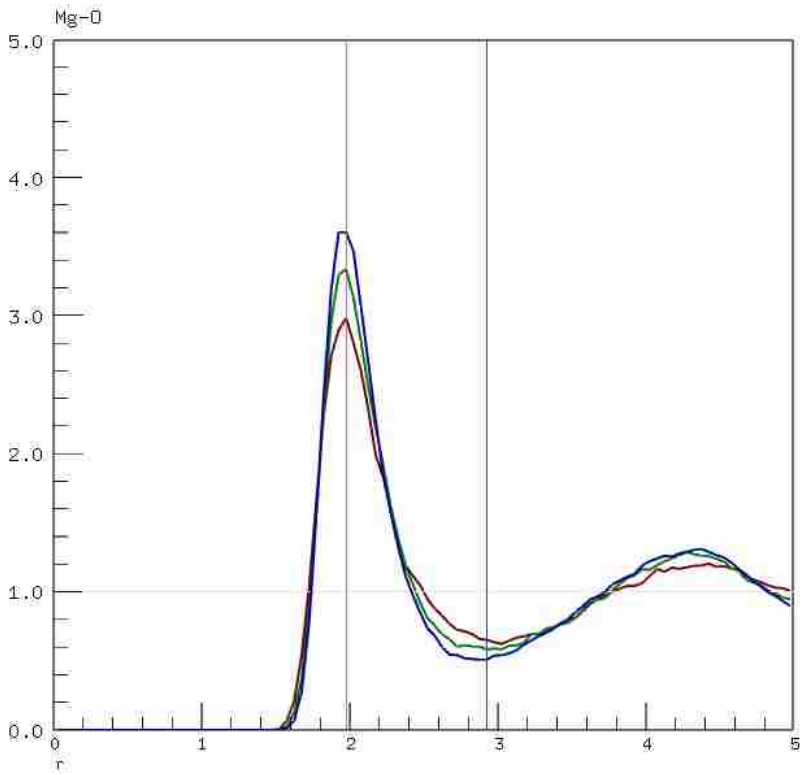


Figure 2.8: Superimposing multiple partial RDFs of Mg-O environment of same the atomic system simulated for different temperature conditions (2500K: blue, 3000K: green, and 4000K: brown functions).

2.4.2 Coordination Environment

Let r be the distance between atoms i and j of two different species types α and β, respectively. If atoms i and j are thought to be coordinated whenever $r \leq r_c$, then r_c is said to be the coordination distance. And the coordination environment for species pair α and β is defined as

$$C_{\alpha\beta} = 4\pi\rho_\beta \int_0^{r_c} r^2 g_{\alpha\beta}(r)\ dr \tag{2.7}$$

where $g_{\alpha\beta}(r)$ is the partial RDF between corresponding species pair, and $C_{\alpha\beta}$ gives the average coordination number for atom i with respect to atoms of type β.

The coordination distance r_c defines how far radially to consider for coordination. The farther we go from an atom, the more number of atoms will contribute for the coordination; hence, the

coordination number increases, and vice-versa. But, practically, the coordination distance (cutoff distance) is computed from the corresponding RDFs. Further more, the coordination state per atom as a function of time is given as

$$C^{i}_{\alpha\beta}(t) =\mid \{1 \leq j \leq N_{\beta} : d(i,j) \leq r^{\alpha\beta}_{min}\} \mid, for \;\; i = 1...N_{\alpha} \tag{2.8}$$

by counting the atoms of species β, which lie within a sphere centered at an atom of species α. The radius of sphere defined by corresponding cutoff (coordiantion) distance $r_c = r^{\alpha\beta}_{min}$. Moreover, the coordination environment is simply a distribution of atoms around each other and doesn't represent the actual bonding environment between atoms.

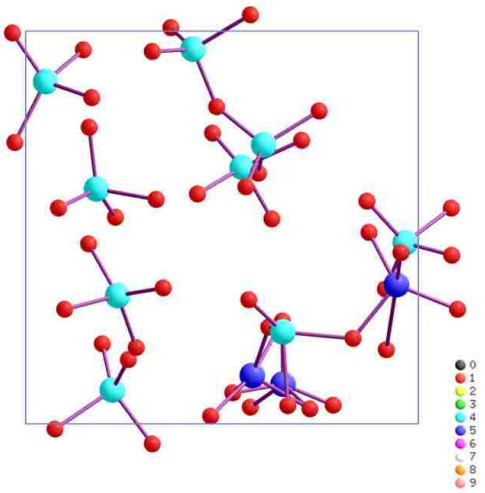


Figure 2.9: Coordination environment of Si-O species pair in hydrous silicate melt. Si atoms (large spheres) are color coded as per their coordination states (4-fold: cyan, 5-fold: blue) with respect to O atoms (small red spheres).

The coordination environment is more diverse in case of liquids in which coordination is a function of both space and time. For instance, Si-O coordination numbers of all Si atoms in a hydrous silicate melt are not the same, three Si atoms have a coordination number of 5 (encoded blue), and the rest of the Si atoms have a coordination number of 4 (Figure 2.9). Since both Si and O atoms in a liquid phase are constantly moving, the coordination state of a particular Si

atom changes with time. Hence, it is generally preferable to calculate an average value of the coordination number over the simulation period. This may give important information about the stable state of atoms of particular species in the coordination environment.

2.5 Visualizing Dynamical Properties

Atom trajectories allow a complete representation of position-time series data by rendering positions of all atoms for all time-steps as points or line segments. Since the entire data set is rendered, full positional information is contained in a single display. As one can realize from Figure 2.10, the main problem is that trajectory rendering becomes too crowded because there are simply many trajectories that are long, distributed in 3D space, and overlapped with each other. Note that the trajectories are unfolded so that they become continuous curves extending in the space both inside and outside the simulation cell. The challenge is how to render these trajectories with minimum clutter while preserving the dynamical and structural information contained in the data. Though the clutter cannot be be completely avoided, it may be reduced to an acceptable level via repetitive processing and analysis of data. This section proposes multiple approaches toward improving the rendering of trajectories from MD simulations.

2.5.1 Selective Trajectory Rendering

A straightforward way of reducing cluttering is to display fewer trajectories by selecting a group of atoms and/or by considering shorter time intervals than the entire simulation duration. One option is to render the trajectories for one atomic species to a single display, or the trajectories of different species to different viewports. But, the number of atoms for each species is not always small, so for each species, a subset of trajectories may be considered to further reduce cluttering. This subset must contain at least one trajectory. Likewise, a group of atoms whose trajectories are being rendered can be chosen randomly or according to some criteria. If the intention is to minimize the overlap among the trajectories, the selected atoms must be widely separated from each other (Figure 2.11, left). However, if one is exploring temporal behavior of a structural unit involving

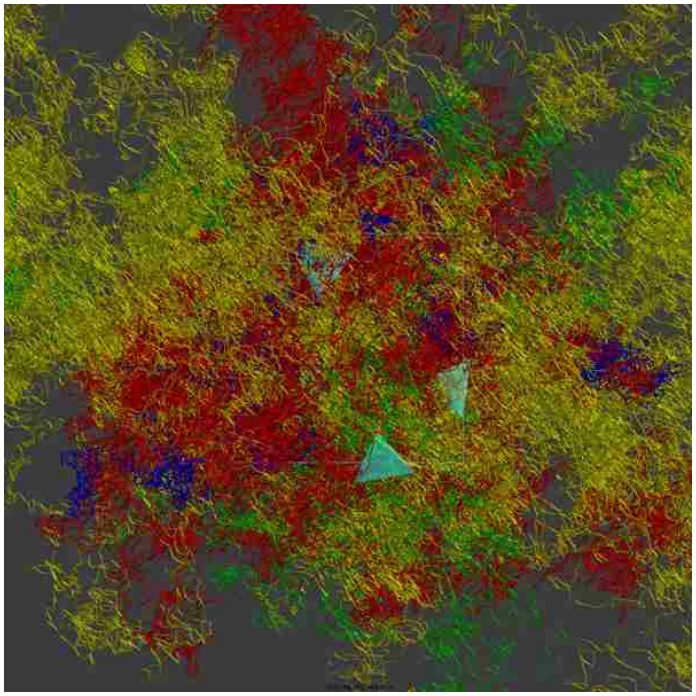


Figure 2.10: Atom trajectories and coordination structures are rendered together during visualization of hydrous silicate liquid simulation data.

atomic bonding, the participating atoms of different species should be selected. For example, an SiO_4 coordination unit (tetrahedron with Si atom at the center and O atoms at the vertices) involves five trajectories as shown in Figure 2.11 (right).

2.5.2 Region-Based Rendering

In general, different atomic species show different dynamical behavior. Some are highly mobile and others' motions are confined to small regions. Relative movenments of different species depend on factors such as pressure, temperature and composition. Because of the mobility differences, the trajectories of slow moving species tend to fall largely in the regions through which all other faster species traverse at some time intervals, thereby resulting in a high degree of cluttering. Thus, the trajectories of less mobile species in dense regions (containing crowded lines) are invisible to great extent. Only the portions of trajectories in far regions are clearly visible because only fewer atoms travel long distances. It is useful to render the trajectories of different

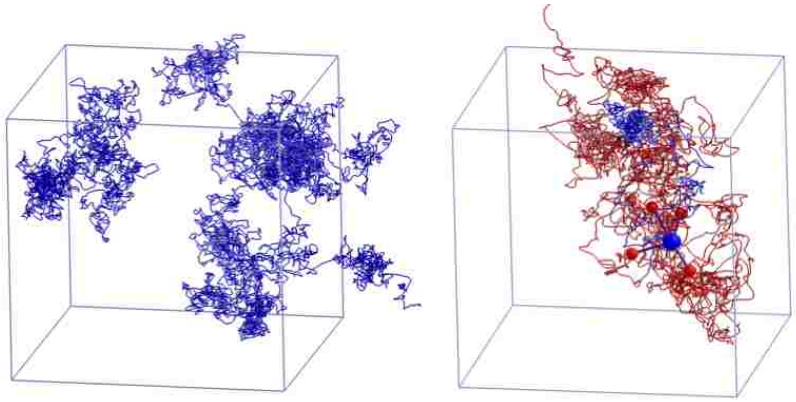


Figure 2.11: Trajectories of four well separated Si atoms (left). Trajectories of one Si atom (blue) and four O atoms (red) forming a SiO_4 coordination unit for 20,000 time-steps (right).

species in different non- overlapping regions so that the single species trajectories cannot obscure other species trajectories. The portions of the trajectories belonging to a region are discarded if they extend to other regions, i.e., if they fall outside the region defined for the corresponding species.

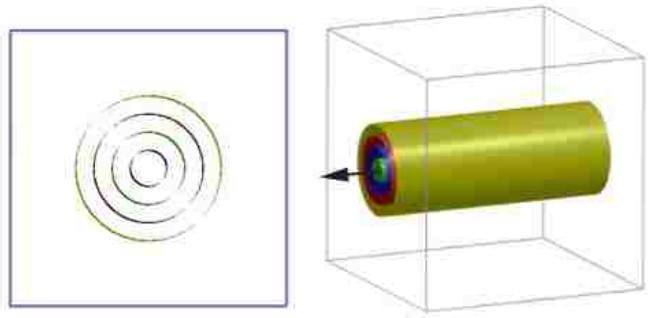


Figure 2.12: Schematic representation of cylindrical regions used in constraining trajectory rendering, a top 2D view (left), and a 3D view (right).

To implement a region-based trajectory rendering, we divide space into multiple regions or partitions. The number of regions is equal to the number of atomic species. In a 2D representation, these regions can be concentric circular shells (rings) about the center of the simulation box (Figure 2.12). These rings are assigned to different atomic species from inside to outside in an increasing mobility order. For the hydrous silicate liquid, the trajectories of slowest Si atoms are

displayed in the innermost ring whereas those of the fastest H atoms are displayed in the outermost ring (Figure 2.13). By default, the trajectories start from the initial positions of their respective atoms, which are distributed throughout the box. Inner circular regions may not encompass all atoms of their respective species and the corresponding trajectories. To accommodate most trajectories in their respective regions, we shifted all trajectories to make them appear as if they started at the center of the simulation box (Figure 2.13). This does not affect the geometry (structure) of individual trajectories, although they are rigidly displaced from their original positions to the center of simulation box.

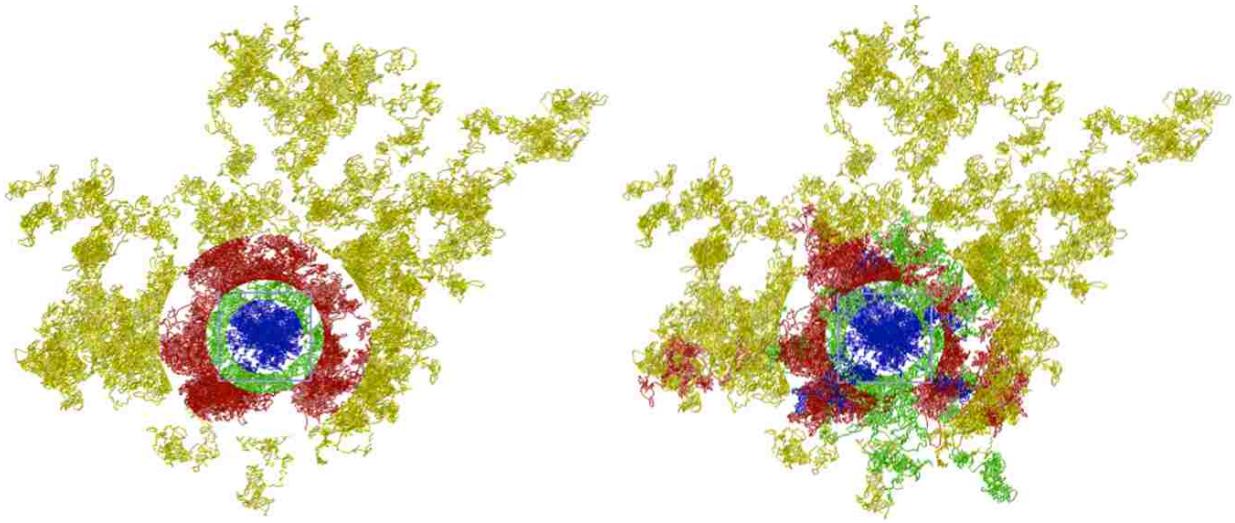


Figure 2.13: Region-based rendering of trajectories using four regions for Si, Mg, O and H atoms (in an inside-to-outside order) with strict boundaries (left) and open outside boundaries (right).

In 3D rendering, the situation is much more complicated. The use of concentric spherical shells about the center of the simulation box does not work because the outer shells occlude the inner shells. To overcome this problem, we use the concentric cylinders with their common axis being parallel to the view direction and passing through the center of the simulation box (Figure 2.12). All the trajectories start from the same point, the center of the simulation box, but they are visible only in their respective regions. For the best view mode (where the view direction or cylindrical axis is perpendicular to the screen) the layout of the regions is essentially equivalent to the 2D case described above, but the trajectories displayed in each region are 3D.

The cylindrical regions for different species have the same length but different circular cross-sections. The circular width for species α is given by Δr_α, where r_α and r_β define the inner and outer circular boundaries for the region and $\alpha < \beta$ in the mobility order. Thus, we define a separate region for each species to display their trajectories by restricting rendering to the specified region from both inside and outside (Figure 2.13, left). We can relax this restriction to allow some overlap between trajectories belonging to different regions. One useful variation is to have an open outside boundary so that the trajectories displayed in a region can also appear in one or more outer regions (Figure 2.13, right). Only a few long trajectories are expected to extend to outer regions, so they will not affect rendering of outer region trajectories much. The user can change these boundaries interactively to decrease and increase the region, and to even delete the region. The user can also exchange the species between successive regions. Such manipulation of species-based regions requires domain specific knowledge and can benefit from our understanding of the system as we visualize the trajectories.

2.5.3 Position Based Merging

Constituent atoms of a simulated material system are generally expected to show non-uniform movements. This means that the atoms spend longer a time in certain regions in 3D space than they do in other regions. In other words, the trajectories consist of extended regions as well as confined regions. Positions in slow regions where atoms are perhaps making oscillating (going back and forth) motions form short and crowded portions of the trajectories. A "proximity window" can be introduced based on the assumption of such confined atomic movements or localities so that positions lying in the window can be merged. This window can be modeled as temporal proximity or spatial proximity.

By setting up multiple, non-overlapping windows, one can dramatically reduce the number of positions needed to render the trajectory. In effect, this process replaces multiple line segments in each locality by a single line segment. Since multiple positions in each proximity window are merged, our merging method reduces the amount of data used to render the trajectories with desired

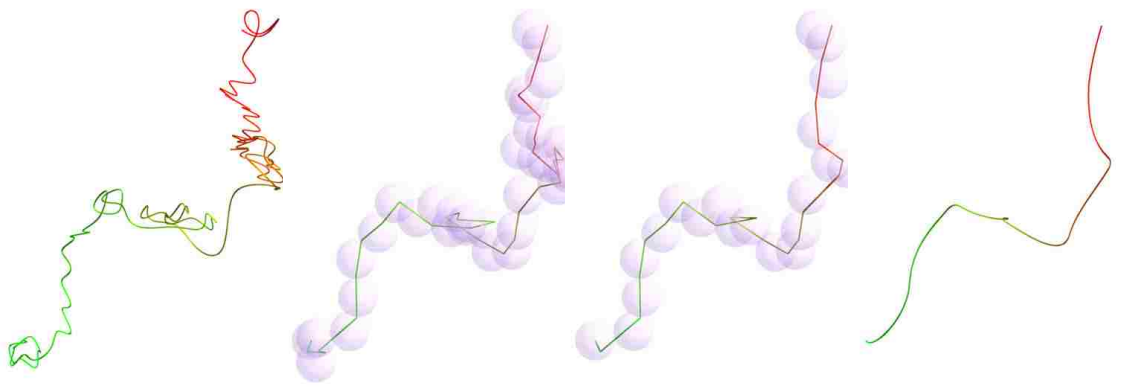
clutter reduction. This also means that a smaller amount of positional information is copied to the video memory for rendering. However, it is important that the 3D geometry of the trajectory be preserved during clutter reduction through merging multiple positions.

Temporal Proximity

We first consider the temporal proximity, which involves positions occupied by the concerned atom over a finite time interval. A time window is defined in terms of the number of successive MD steps, e.g., every 100 steps form a window. For each window, we find a representative point by calculating the mean (centroid) of all positions occupied at time-steps belonging to that window. A single position is thus generated per window so the number of windows ($N_{\text{STEP}}/N_{\text{TW}}$) gives the number of positions to be rendered, where N_{STEP} is the total number of steps and N_{TW} is the size of the time window. Line segments are drawn between two successive mean positions to render the trajectories. The length of the line segments can vary significantly since the atoms move at different speeds at different times. Generally, N_{TW} needs to be very large in order to reduce the clutter in highly confined regions. This means that line segments in extended regions (where the atoms are moving fast without reversing directions) can be unusually long so the time-window-based merging may not work well for the entire trajectory length.

Spatial Proximity

Perhaps a more appealing approach is to apply spatial proximity, which considers positions located in close vicinity of each other irrespective of time when these positions are occupied by the concerned particle. We define spatial proximity in terms of a finite space window (cut-off distance) so that all positions lying within such windows are considered for merging. The idea is to scan successive time-steps to pick up positions whose distances measured from the reference position are shorter than the cut-off distance and merge them together. The locality being approximated is thus a spherical region of radius equal to the cut-off distance (Figure 2.14).



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

Figure 2.14: Space proximity based position merging along trajectory. Transparent spheres represent space windows (or cutoffs) used for a two-level merging. An original trajectory before merging (a), and the same trajectory after first merging with cutoff spheres shown (b), second merging with cutoff spheres shown (c) and smoothing (d).

Our position-merging algorithm for each trajectory works as follows:

Step 1: Set the space window at the starting (reference) position of an atom trajectory.

Step 2: Scan subsequent time steps to pick up positions lying inside the window until an outside position is encountered for the first time.

Step 3: Merge/map all positions found in the step 2 to single position.

Step 4: Start at the first position not yet picked up (new reference position), and do the steps 2 and 3.

Step 5: Move the window in the time order with merging performed at each successive location (repeating step 4) until the end.

Thus, all positions are processed to generate a reduced set of new (merged) positions. If successive positions are well separated because either the cut- off distance is too small or atoms move too fast, merging is not effective at all, and every window contains a single but original position. On the other hand, if atoms have covered small distances because either atom are slowly moving/oscillating or the cut- off distance is too large, an arbitrarily large number of positions can be found in the window under consideration to map to a single position. Note that processing atomic

positions in the time order ensures the correct geometry of the trajectories while merging nearby positions as much as possible.

2.5.4 Adaptive Hierarchical Merging

After the first run of position merging using a cu-toff distance, the clutter, though it might have already reduced considerably, is still not at an acceptable level, and merged positions are still too close. So it may be desirable to apply additional merging with the same or larger cutoff. The next-level merging then considers all new positions derived in the previous merging by placing a space window of the same or bigger size at the first merged position. Figure 2.14 illustrates two-level merging. As in the original merging, all positions lying within the current window at each successive location are picked up and merged. It is likely that the positions even after the second merging are still closely spaced, and/or again it may be desirable to apply an even larger cut-off. A third/higher level merging is then performed for these positions, which have undergone merging more than once. This adaptive-hierarchical processing will continue until the number of repeatedly merged positions does not decrease anymore. This situation arises relatively fast if the cut-off distance is constant from one level to the next. However, if the cut-off distance is increased progressively, the merging process may continue over many levels. The user has to specify when to stop either by not increasing the cut-off further or by visually assessing the crowdedness and correctness of the trajectories being increasingly approximated with multi-level positions merging.

2.5.5 Merging with Constraints

Merging is expected to be sensitive to information that we want to display (color code) along the trajectories. Examples of such information include coordination states, bond events, merge count, mean squared error, etc. Such information is computed during the merging process from an original set of positions and appropriately mapped to the merged positions. Information extraction for coordination states involves the positions of nearby atoms. Atomic coordination $(C_{\alpha\beta})$ is the

number of the nearest neighbor atoms of the same species (α) or other species (β) around a given atom of species α, and is calculated as Equation 2.8.

Coordination is of a multivariate type as it can take multiple values for a given window. Our approach to impose a coordination constraint while merging is that successive positions within a space window are merged only if they are in the same coordination state. A position with a different coordination state than the previous positions will start a new space window for merging the next sequence of positions of the same coordination state. Because of this additional constraint imposed, merging with a given cut-off is expected to result in more crowded trajectories than original merging (Figure 2.26).

2.6 Design and Implementation

We use the space-time, multi-resolution atomistic visualization system [5] to gain insight into the structural and dynamical information of the atomic positions-time data set. All of the rendering/-analysis algorithms described so far have been implemented in the visualization application.

Our visualization application is developed using C++ and OpenGL (for graphics context). OpenGL provides functionality for rendering graphics onto the display window. We also use utility libraries such as GLUT and GLUI that provide additional rendering algorithms and user interface support. We started porting our application from an earlier implementation that uses GLUI libraries to the new implementation based on Qt framework for better user interface (UI) options. Features for visualizing dynamical properties has been imported to the Qt-based implementation and we are in the process of porting structural visualization modules. Qt (*http://qt-project.org*) is a cross-platform application and UI framework for developing C++ based application, and has support APIs for OpenGL based rendering. Moreover, GLSL shading language is used for a hardware-accelerated rendering purpose whenever possible. The use of shaders means that we were able to implement illumination (shading) model for lines, line smoothing using Bézier curves, and rendering of atomic spheres as quadratic surfaces in graphics hardware. We define shaders to program various stages of OpenGL rendering pipeline in GPU.

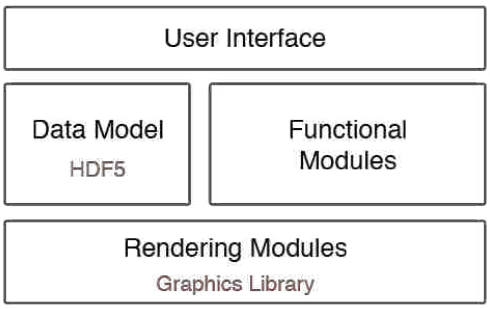


Figure 2.15: Multi-layer design of interactive atomistic visualization application.

2.6.1 Visualization System Design

The overall structure of the atomistic visualization system is outlined in Figure 2.15. The visualization system is designed in a multi-layer fashion with three main layers. The bottom most layer, rendering modules, is responsible for rendering images to the frame buffer using low level OpenGL API. This layer is composed of modules that implement the rendering algorithms and data structures, and calls functions of the graphics library. OpenGL API is a library of function calls that enables us to create an interactive graphics application for rendering images and 3D geometric objects. OpenGL is a operating system independent and supported across windowing systems. However, it does not provide functions for creating a window/frame to display images, hence depends upon other libraries and operating systems for creating the display window. We use Qt and its OpenGL related classes to create OpenGL context and display windows for drawing graphics objects.

Functional modules constitute classes that implement analysis and visualization algorithms. Modules in this layer have direct access to the data model and rendering algorithms. Some of the functional modules evaluate radial distribution functions, coordination environment, mean-squared displacement, adaptive-positions merging, etc., of constituent atoms, whereas other modules just rearrange data layout for rendering algorithms. In general, processing data generates additional information that is useful for understanding the system from a new perspective. Visualization algorithms map this information in a meaningful way so that the information is physically repre-

sented by rendering algorithms. Algorithms that process data either for analysis or visualization purposes are part of this layer.

The visualization system supports the HDF5 file as the input file format. In general, HDF5 format is a container format, and the raw data is mapped to the format with various HDF5 objects. In our data model, raw positional data is stored in a three-dimensional dataset with the chunked storage layout. In addition, the metadata with information on how to interpret raw data is stored as attributes of the group containing dataset. When data for a time-step in requested, a subset of the data set belonging to that time is selected using hyperslab selection and copied to the application memory. Similarly, when a trajectory data for an atom is requested, the consecutive positions information of that atom is selected from the dataset and copied to the requesting module.

The top-most layer defines user interface and is the front end of the visualization system. We use Qt's set of GUI components/widgets for building user interface of the application. Besides, Qt also provides OpenGL related classes to define the graphics context for rendering 3D scenes and images. A set of *event* related classes is available to capture keyboard and mouse events on widget and process the triggered event accordingly. For example, mouse movement while holding the left mouse button is used to rotate the 3D scene. We use a signals and slots mechanism provided by Qt to communicate between two different object/modules. Usually, when one object is modified, it is required that other objects be notified. Consequently, signals and slots mechanisms grant this to happen across different objects.

2.6.2 Graphics Rendering Pipeline

The OpenGL rendering pipeline is a high-level model which describes how a set of input values (vertices, colors, normals, etc.) are transformed at multiple stages to draw a final image on the screen. As the word pipeline suggests, the input values are processed sequentially in a pipeline, while each stage contributes towards how the final image is drawn. An earlier version of OpenGL had a fixed-function pipeline, which means that all of the pipeline stages that OpenGL supported

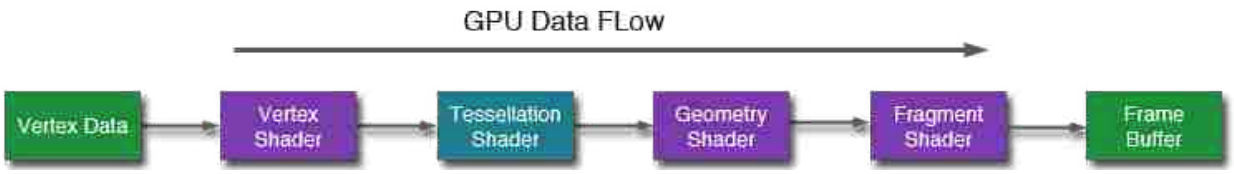


Figure 2.16: Programmable rendering pipeline available with OpenGL 4.1. In our implementation we only use vertex, geometry and fragment shader stages of the pipeline.

were fully defined and an application could only modify the output result by changing the input values. OpenGL version 2.0 provided support for the first programmable rendering pipeline, which allowed different stages of pipeline to be programmed using small programs called shaders. Shaders were responsible for implementing features to process the OpenGL scene as required by the application. In this version, two stages are made programmable: vertex shader and fragment shader. However, with advancement in graphics hardware over recent years, more pipeline stages are made programmable and new programmable stages were also added. Figure 2.16 shows the programmable pipeline stages supported in OpenGL version 4.1. In our implementation we define shader programs for three different stages: vertex shader, geometry shader and fragment shader.

In general, vertices data generated by the application in CPU is transferred to the memory of graphics hardware. The data flows through various stages of GPU to generate an image on screen or in a reserved memory space called a frame buffer. The vertices information is then streamed to the GPU memory as an ordered list through the vertex buffer objects. This way blocks of application memory values are copied to the GPU memory instead of streaming single vertex information at a time. Once copied to the GPU memory, the shaders in a pipeline process the scene as defined by the application. The vertex shader receives a stream of input vertices, additional attributes per vertex, and transformation matrices. Then it processes data on a per-vertex basis and applies transformation matrices to the input vertices. Output of vertex shader is a stream of transformed vertices that is passed to the next stage in the pipeline. Usually, if there is no shader program defined for tessellation- and geometry shaders, then the output of the vertex shader is directly passed to a fragment shader for per-fragment processing. A geometry shader processes input vertices in a per-primitive basis and receives as input all vertices belonging to a primitive. This stage is op-

tional, and if not defined in the rendering pipeline, the vertices are forwarded to a fragment shader. This stage processes input vertices to generate additional vertices per primitive, generate more primitives, or cull existing primitives as defined in the shader program by application. In other words, geometry shader can create new geometry objects by itself and cut off existing geometry. In addition, this stage distinctly defines input and output primitive types which can be a different type. For example, the shader can take input vertices as a triangle primitive and generate output vertices as line primitives.

Before the fragment shader, the stream of vertices from either a vertex shader or geometry shader are composed to predefined primitives objects. For polygon rendering, the vertices are composed to a set of triangles. Similarly for trajectory, the vertices are composed to a set of line segments. The primitives pass through a rasterization stage which generates fragments to fill up individual primitives. Finally, these generated fragments values are streamed to the fragment shader. In the fragment shader, each fragment is assigned a color value for rendering. Usually, fragment shaders also evaluates lighting and texture mapping per fragment.

2.6.3 Quadratic Spheres

In earlier approaches to display atoms as spheres, we used *glutSolidSphere()* or *gluSphere()* to render polygonal spheres. Usually, polygonal-based objects are composed of numerous triangle primitives, which can be refined to increase the number of triangles and hence the resolution of an object. This approach works fine while rendering together a few hundred atoms as spheres, but fails to scale well with the increase in the number of atoms to hundreds/thousands and many more.

As an improved alternative to polygonal spheres, we implemented the GPU-based rendering of quadratic surface as a sphere (Figure 2.17), described in detail in [31]. The ability to program different stages of GPU means that we can implement hardware-accelerated rendering of quadric surfaces, defined by the following quadratic equation:

$$Ax^2 + 2Bxy + 2Cxz + 2Dx + Ey^2 + 2Fyz + Gy + Hz^2 + 2Iz + J = 0 \qquad (2.9)$$

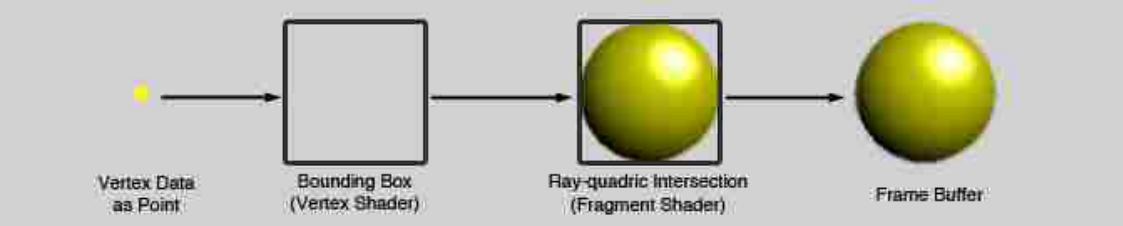


Figure 2.17: Schematic representation of quadric sphere rendering in programmable pipeline from a point sprite.

The roots of the quadratic equation define the surfaces, and the shape of surface (quadric primitive) is determined by the value of coefficients A to J.

For rendering quadric spheres, the vertex shader takes the radius and center position of a sphere as input parameters, in addition to model-view and projection matrices. The vertex shader then computes the bounding box for each quadric to be rendered. Next, the fragment shader evaluates the ray-quadric intersection at each fragment inside the bounding box by solving the quadratic equation. If the ray does not intersect the quadric then the corresponding fragment is culled; otherwise, the fragment is part of the quadric and is rendered. The two real roots of the quadratic equation correspond to the intersection points, so the smaller root is the closer intersection point and hence is the depth value of the fragment. Finally, the fragment shader computes a normal vector at individual visible fragments for the lighting evaluations. This implementation for rendering spheres as quadratic surfaces produces resolution-independent, high-visual quality output. The vertex and fragment shader sources for quadratic spheres are shown below:

Vertex Shader: Quadric Spheres

```
#version 410 compatibility
uniform mat4 modelviewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 viewPort;  // Contains Width and Height of the Viewport
attribute vec3 in_Position;
attribute vec4 in_Color;
attribute float in_Radius;
out vec4 iPMT_c3;
out mat4 MT_it;
out float radius;
```

```
out vec3 color;

void main(void)
{
  mat4 T = mat4(in_Radius);
  mat4 MV = mat4(modelviewMatrix);
  mat4 P = mat4(projectionMatrix);
  T[3][0] = in_Position.x;
  T[3][1] = in_Position.y;
  T[3][2] = in_Position.z;
  T[3][3] = 1.0f;

  mat4 PMT = P * MV * T;
  mat4 iPMT = inverse(PMT);
  vec4 r1 = vec4(PMT[0][0], PMT[1][0], PMT[2][0], PMT[3][0]);
  vec4 r2 = vec4(PMT[0][1], PMT[1][1], PMT[2][1], PMT[3][1]);
  vec4 r3 = vec4(PMT[0][2], PMT[1][2], PMT[2][2], PMT[3][2]);
  vec4 r4 = vec4(PMT[0][3], PMT[1][3], PMT[2][3], PMT[3][3]);
  vec4 D = vec4(1.0f, 1.0f, 1.0f, -1.0f);
  vec4 dr1 = D * r1;
  vec4 dr2 = D * r2;
  vec4 dr3 = D * r3;
  vec4 dr4 = D * r4;
  // Solve equation to get left and right border of bounding box
  float coef_a, coef_b, coef_c;
  vec4 roots = vec4(1.0f);
  coef_a = dot(r4, dr4);
  coef_b = dot(r1, dr4);
  coef_c = dot(r1, dr1);
  roots.x = (coef_b+sqrt(coef_b*coef_b-coef_a*coef_c))/coef_a;
  roots.w = (coef_b-sqrt(coef_b*coef_b-coef_a*coef_c))/coef_a;
  coef_b = dot(r2, dr4);
  coef_c = dot(r2, dr2);
  roots.y = (coef_b+sqrt(coef_b*coef_b-coef_a*coef_c))/coef_a;
  roots.z = (coef_b-sqrt(coef_b*coef_b-coef_a*coef_c))/coef_a;

  vec4 outPos=vec4(dot(r1,dr4),dot(r2,dr4),dot(r3,dr4),dot(r4,dr4));
  outPos = outPos/outPos.w; // Divide xyz with w component
  radius = max(abs(roots.w-roots.x)*viewPort.x,
               abs(roots.z-roots.y)*viewPort.y) * .5f;
  gl_PointSize = radius;
  gl_Position = outPos;

  iPMT_c3 = iPMT[2];
  MT_it = transpose( inverse(T) * inverse(MV) );
  color = vec3(in_Color.xyz);
}
```

Fragment Shader: Quadric Spheres

```
#version 410 compatibility
uniform mat4 modelviewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 viewPort;  //Contains Width and Height of the Viewport
in mat4 MT_it;
in vec4 iPMT_c3;
in float radius;
in vec3 color;
out vec4 vFragColor;

float specular = 150;
vec3 lightPos = vec3(1.3f, 1.5f, 1.0f);  //Single light source

void main(void)
{
  vec4 pixelCoord = gl_FragCoord;
  mat4 P_inv = inverse(projectionMatrix);
  //Convert Window coordinates to Clip coordinates
  pixelCoord.x = pixelCoord.x * viewPort.z - 1.0;
  pixelCoord.y = pixelCoord.y * viewPort.w - 1.0;
  pixelCoord.z = (2 * pixelCoord.z - gl_DepthRange.far
                - gl_DepthRange.near)/gl_DepthRange.diff;
  pixelCoord.w = 1.0f;

  vec4 p_e = P_inv * pixelCoord;
  vec4 xp_ = transpose(MT_it) * p_e;
  vec4 c3 = iPMT_c3;
  //Solve equation to get ray-quadric intersection
  vec4 D = vec4(1.0f, 1.0f, 1.0f, -1.0f);
  vec4 dc3 = D * c3;
  vec4 dxp = D * xp_;
  float coef_a = dot(c3, dc3);
  float coef_b = dot(c3, dxp);
  float coef_c = dot(xp_, dxp);
  float det = coef_b * coef_b - coef_a * coef_c;
  if(det < 0.0) discard;

  vec2 roots = vec2(0.0f);
  roots.x = (-coef_b+sqrt(coef_b*coef_b-coef_a*coef_c))/coef_a;
  roots.y = (-coef_b-sqrt(coef_b*coef_b-coef_a*coef_c))/coef_a;
  float depth = min(roots.x, roots.y);
  gl_FragDepth = gl_FragCoord.z + depth;
  // Fragment Coord in parameter space
  pixelCoord = xp_ + depth * c3;  //depth is fragment "z" component
```

```
  vec4 n_e = MT_it * pixelCoord;
  vec3 n = normalize(n_e.xyz);
  vec3 l = normalize(lightPos);
  vec4 tpe = inverse(transpose(modelviewMatrix)) * pixelCoord;
  vec3 v =  normalize(tpe.xyz);
  vec3 h = v + l;
  h = normalize(h);
  //Smooth out the edge of the Sphere quadrics
  float mag = dot(n.xy, n.xy);
  float blur = smoothstep(0.8, 0.9, mag);
  if(mag > 0.9) discard;

  //Blinn-Phong Lighting Model
  float diffuse = max(0.0, dot(l, n));
  float spec = max( pow(dot(n, h), specular), 0.0);
  vec4 varS = vec4(1.0f)*spec;
  vFragColor.xyz = vec3(color)*diffuse + varS;
  vFragColor.a = 1.0f - blur;
}
```

2.6.4 Compiling Atom Trajectories

The positional data from HDF5 format is separately read for each atom in a per-atom-wise access of dataset. This way, consecutive positions of an atom from start time-step to the end time-step are returned in a contiguous array with a single read operation on the dataset. Thus, for compiling trajectories for multiple atoms, as many reads on the dataset are needed.

Additional information such as velocity, curvature, tangent, coordination state, etc. along atomic trajectories can be computed for rendering purposes and encoded along the trajectories using color a map. In particular, tangent at positions along trajectory are used to illuminate trajectory line segments during rendering. A tangent at any point along the trajectory is computed from the positions' information at previous and current time-steps, so that it is the slope of line passing through the positions at these time-steps.

In the case of position-based merging, current trajectory positions are processed as per the merging algorithm to generate a new set of positions, which can be processed further in hierarchical levels to generate reduced sets of trajectory positions. After each level of merging, the

associated information (such as color) for trajectory positions are updated. The number of positions in processed trajectory vary between trajectories depending upon the shape and extent of the individual trajectory and the selected cut-off distance. We keep track of the number of positions per processed trajectory, which will be used for indices information while drawing trajectories as line segments. We also save the original positions information so that the user will have an option to revert back to the original trajectories.

We use a vertex buffer object (VBO) to transfer the positions data to a graphics processing unit for rendering. However, the color information per position is passed to GLSL shaders as vertex attributes, and the tangent vectors along trajectory are computed using positions information in the geometry processor of the GLSL pipeline. For trajectory rendering, we define vertex, geometry, and fragment shaders to program the respective processors in the rendering pipeline. A vertex shader takes positions and colors as shader *attributes*, and the projection matrix and model-view matrix are copied as shader *uniforms*. In fact, the consecutive input positions are distributed among three sub-arrays in order, so that each array contains a set of control positions required for generating cubic Bézier curves. For a Bézier curve as shown in Figure 2.18(a), the control points P1 and P4 will be members of the first array, and control points P2 and P3 will respectively be members of the second and third array, where P1, P2, P3, and P4 are all positions along the trajectory. The first array is passed as a input to the vertex shader, and the other two are passed as attribute arrays to the shader. Consequently, the array of every vertex attribute will be divided into three sub-arrays and passed as different attributes to the vertex shader. The vertex shader simply outputs per-vertex position and attributes information to the geometry shader, the next stage in our rendering pipeline.

The output of vertex shader is copied to the geometry shader as arrays of input variable, and the size of each array depends on the input primitive of the geometry shader. Our geometry shader has *lines_adjacency* as input primitive and input arrays for positions, first control points, and second control points. Since input primitive type is *lines_adjacency*, each array will receive four consecutive elements per primitive. Given an array of input vertices, the geometry shader generates Bézier curves for trajectory smoothing and emits new vertices along the curve to the fragment shader, the

next stage in rendering pipeline. Finally, the fragment shader implements the lighting model to compute color value and light intensity for each rasterized fragment to be rendered.

Trajectory Smoothing

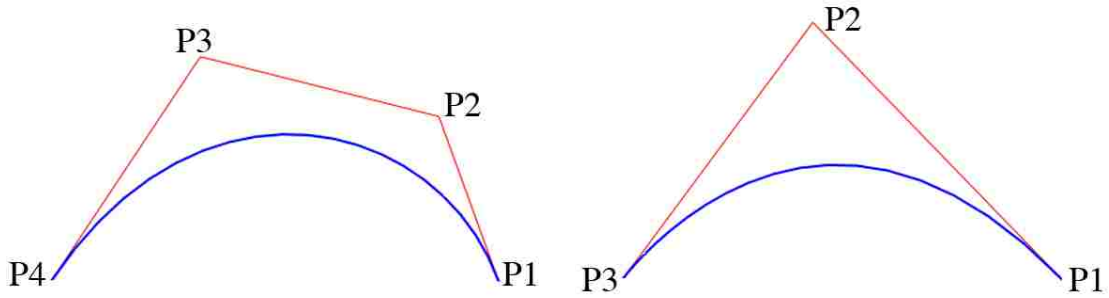
Successive positions generated after several levels of hierarchical merging are well separated from each other so the trajectories are no longer smooth and can be visually distracting. We perform a piece-wise smoothing of line segments by generating quadratic and cubic Bézier curves [32], as shown in Figure 2.18. Four successive positions are used at a time as control points to generate a cubic Bézier curve, given by

$$B(t) = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t)t^2 P_3 + t^3 P_4 \tag{2.10}$$

where $t \in [0, 1]$. $P_1, P_2, P_3,$ and P_4 are four control points such that while t increases from 0 to 1, the curve departs from P_1 towards the next control points (Figure 2.18a). If the last remaining positions are less than four, we use lower order equations. However, the points where two successive curves meet are not necessarily smooth, will likely form pointed artifacts along the trajectory. We use a quadratic Bézier curve to guarantee the smoothness of curves at these points of intersection (Figure 2.18b), as given by:

$$B(t) = (1-t)^2 P_1 + 2(1-t)t P_2 + t^2 P_3, \quad where \quad t \in [0, 1]. \tag{2.11}$$

We evaluate Bézier curve equations in the geometry processor of the graphics processing unit. The algorithm is implemented in a geometry shader which takes input arrays of per-vertex information and emits a new set of vertices along the curves to be rasterized. Deferring this process to GPU has performance advantages over CPU implementation, as a geometry shader uses accelerated hardware to separately generate smooth curves for each line segment. Another advantage is that we can render a high-resolution trajectory from a small number of input vertices, which amounts to a fraction of output vertices generated by the geometry shader. This means that only



(a) (b)

Figure 2.18: Cubic Bézier curve (a) and quadratic Bézier curve (b), drawn in blue, represent a smooth approximation of respective line segments, drawn in red.

a few bytes of data representing trajectory positions need to be copied from CPU to GPU, saving in bandwidth. Additionally, the geometry shader computes and emits a tangent vector per vertex along the trajectory, relieving CPU from more computation and less data transfer to GPU. The tangent vectors are used for lighting calculation in the fragment shader. Snippets of vertex and geometry shader sources for rendering smooth trajectory are shown below.

Vertex Shader: Trajectory Rendering with Bézier Curves

```
#version 410 compatibility
attribute vec3 qt_Vertex;
attribute vec3 qt_Cntrl01;
attribute vec3 qt_Cntrl02;
attribute vec4 qt_VertexColor;
uniform mat4 qt_ProjectionMatrix;
uniform mat4 qt_ModelViewMatrix;
out vec4 varCntrl01;
out vec4 varCntrl02;
out vec4 varColor;

void main(void)
{
  gl_Position= vec4(qt_Vertex, 1.0);
  varCntrl01 = vec4(qt_Cntrl01, 1.0);
  varCntrl02 = vec4(qt_Cntrl02, 1.0);
  varColor    = qt_VertexColor;
}
```

Geometry Shader: Trajectory Rendering with Bézier Curves

```
#version 410 compatibility
layout (lines_adjacency) in;
layout (line_strip, max_vertices = 64) out;

uniform mat4 qt_ProjectionMatrix;
uniform mat4 qt_ModelViewMatrix;
in vec4 varColor[];
in vec4 varCntrl01[];
in vec4 varCntrl02[];
out vec4 fColor;
out vec3 fTangent;

vec4 cubicBezierPosition( vec4 v[4], float t )
{
  vec4 p;
  float OneMinusT = 1.0 - t;
  float b0 = OneMinusT*OneMinusT*OneMinusT;
  float b1 = 3.0*t*OneMinusT*OneMinusT;
  float b2 = 3.0*t*t*OneMinusT;
  float b3 = t*t*t;
  p = b0*v[0] + b1*v[1] + b2*v[2] + b3*v[3];
  return p;
}
vec4[2] computeCubicCurve(vec4 pos[4], int deltaCount)
{
  int uNum = deltaCount;
  float dt = 1.0/float(uNum);
  float t = 1 * dt;
  vec4 lastPos[2];
  vec4 tangent;
  vec4 prevPos = pos[0];
  for(int i = 0; i < uNum; ++i)
  {
    t = i * dt;
    vec4 xyzw = cubicBezierPosition(pos, t);
    if(i == uNum-2) lastPos[0] = xyzw;
    if(i == uNum-1) lastPos[1] = xyzw;
    tangent = normalize(xyzw - prevPos);
    prevPos = xyzw;
    gl_Position = qt_ProjectionMatrix * qt_ModelViewMatrix *  xyzw;
    fColor = varColor[0];
    fTangent = tangent.xyz;
    //Render from i+2 to N-2, inclusive
    if(i > 1 && i < uNum-1) EmitVertex();
  }
```

```
  return lastPos;
}
void computeCubicIntersection(vec4 pos[4], int deltaCount)
{
  int uNum = deltaCount;
  float dt = 1.0/float(uNum);
  float t = 0;
  vec4 tangent;
  vec4 prevPos = pos[0];
  for(int i = 1; i <= uNum; ++i)
  {
    t = i * dt;
    vec4 xyzw = cubicBezierPosition(pos, t);
    tangent = normalize(xyzw - prevPos);
    prevPos = xyzw;
    gl_Position = qt_ProjectionMatrix * qt_ModelViewMatrix *  xyzw;
    fColor = varColor[1];
    fTangent = tangent.xyz;
    EmitVertex();
  }
}
void main( )
{
  vec4 pos[4];
  int uNum = 12;
  //Cubic curve with four control points
  pos[0] = gl_in[0].gl_Position;
  pos[1] = varCntrl01[0];
  pos[2] = varCntrl02[0];
  pos[3] = gl_in[1].gl_Position;
  vec4 p1[2] = computeCubicCurve(pos, uNum);
  //Compute the second and third points of next cubic curve
  vec4 p2[3];
  .......
  .......
  //Use point p1[2] and p2[2] to generate a cubic bezier
  //curve that smooths intersection between two curves
  pos[0] = p1[0];
  pos[1] = p1[1];
  pos[2] = p2[0];
  pos[3] = p2[1];
  computeCubicIntersection(pos, 6);
  //End of line strip primitive
  EndPrimitive();
}
```

Trajectory Illumination

While rendering line primitives, we implement the modified Phong illumination equation as derived in [21]. The Phong illumination model (or Phong reflection model) computes the light intensity at any point on the surface using the equation:

$$I = I_{ambient} + I_{diffuse} + I_{specular} = k_a + k_d\, L \cdot N + k_s\, (V \cdot R)^n \tag{2.12}$$

where N is the normal vector, L is light direction, V is the viewing direction, R is the unit reflection vector, and $k_a, k_d,$ and k_s are respectively ambient, diffusion and specular reflection constants. However, for line primitives there are more than one possible normal and reflection vectors, so we selected ones that are co-planer to $L - T$ vectors, where T is the tangent vector. After considering proper normal and reflection vectors, the diffuse and specular term in the illumination equation can be presented as,

$$L \cdot N = \sqrt{1 - (L \cdot T)^2} \tag{2.13}$$

$$V \cdot R = (L \cdot T)(V \cdot T) - \sqrt{1 - (L \cdot T)^2}\sqrt{1 - (V \cdot T)^2} \tag{2.14}$$

Details on the implemented illumination model for line primitives can be read in [21].

Trajectory Overlapping

While rendering a coordination environment or coordination cluster between atom species pair (α, β), additional dynamical information can be superimposed by rendering trajectories of selected atoms of interest. Usually, rendering the trajectories of atoms of species type α, in a coordination pair $\alpha - \beta$, is more relevant. Additional structural information such as the coordination state can be encoded along the atomic trajectories.

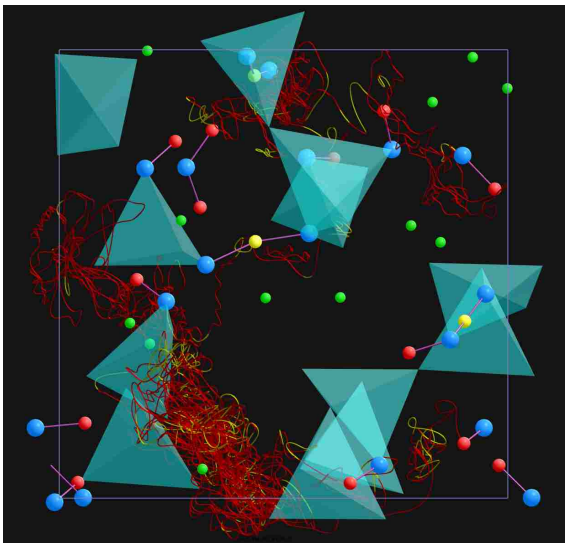


Figure 2.19: Visualizing Si-O polyhedra and H-O spheres encode the coordination informations with respect to O, and Mg (green), Oxygen (blue) as spheres. The trajectory of a single H atom (coordination number color coded) is superimposed over the coordination environments.

2.6.5 Frenet Ribbon

Visualization of a trajectory usually involves rendering line segments along the trajectory positions. However, adding extra width to the trajectory either as a cylindrical tube or as a ribbon is often expected for richer visual quality. For this purpose, we evaluate a Frenet-Serret frame along the trajectory to produce a ribbon structure for rendering. The tangent(T), normal(N) and binormal(B) unit vectors collectively form an orthogonal basis that defines the Frenet-Serret frame (Figure 2.20). Thus, first we need to compute these vectors along the trajectory to be able to produce and render ribbons, also called "Frenet Ribbons."

Given a set of positions along the trajectory, we should be able to compute tangent, binormal and normal vectors at these positions. If $P1, P2$ and $P3$ are consecutive positions, then the tangent vector at $P2$ is evaluated as $\hat{T} = \hat{v1} + \hat{v2}$ where $\hat{v1} = (P3 - P2)$, and $\hat{v2} = (P2 - P1)$. Likewise, the binormal vector can be evaluated as a cross product between $v1$ and $v2$, $\hat{B} = \hat{v1} \times \hat{v2}$ or $\hat{B} = -\hat{v1} \times \hat{v2}$. And since tangent, binormal and normal vectors form an orthogonal basis, we can evaluate normal vector as $\hat{N} = \hat{T} \times \hat{B}$. Once we compute these vectors, the Frenet

ribbon is generated along the trajectory length, and the width of the ribbons is symmetrically extended toward the binormal vector. A normal vector is passed to the fragment shader for correct evaluation of lighting on ribbon primitives. However, this approach with Frenet-Serret frame fails when positions are in a straight line as binormal vectors will be undefined. In reality, numerical calculations usually introduce small errors, and it is less likely to have three consecutive positions perfectly aligned in a straight line so that binormal vectors at these positions are undefined.

An evaluation of Frenet-Serret frames and the Frenet ribbons was implemented in the geometry shader of the programmable pipeline. The geometry shader takes input positions along the trajectory and produces a ribbon geometry to be rendered. It first computes tangent, binormal and normal unit vectors and uses these vectors to generate geometry primitives for the Frenet ribbons. The fragment shader then takes the normal unit vectors and evaluates light intensity for ribbon fragments. Since new geometries are produced in the graphics hardware, rendering Frenet ribbons is just about as fast as rendering line primitives for atom trajectory. Moreover, while rendering ribbon structure extra information can be encoded as varying widths of the ribbon, in addition to color encoding on the ribbon (Figure 2.30).

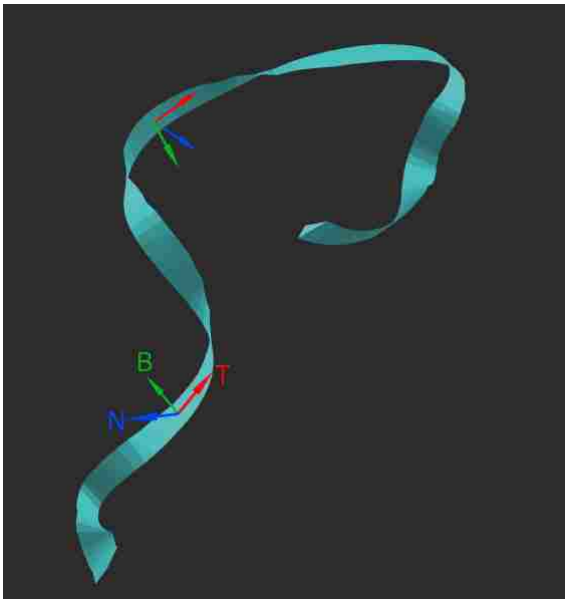


Figure 2.20: Evaluation of Frenet-Serret frame along the curve to generate Frenet ribbon representing trajectory. Three orthogonal vectors T-N-B collectively form the Frenet-Serret frame.

2.7 Results and Discussion

How much the trajectories are cluttered/crowded depends on the size of the data and the nature of the atomic movements. We present our case studies by visualizing complete position-time series of two material systems (silicate and silica liquids).

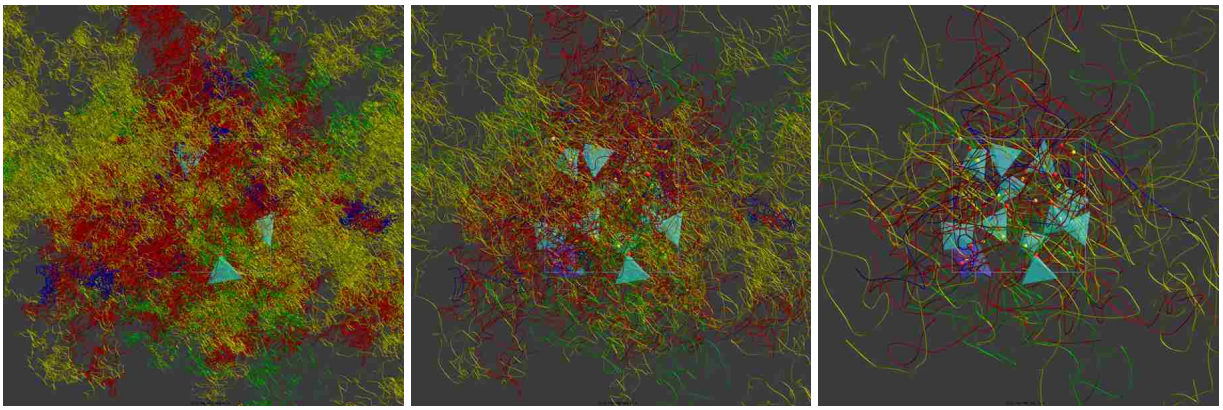


Figure 2.21: Visualization of molecular dynamics simulation data of hydrous silicate liquid: atom trajectories and structure are rendered together. As multiple positions along the trajectories are merged with our adaptive hierarchical merging (described in section 2.4.4), the structure (consisting of Si-O polyhedra, H-O bonds, and floating Mg atoms) becomes increasingly visible (left to right).

2.7.1 Temporal Proximity Versus Spatial Proximity

Figure 2.22 shows H atoms' trajectories approximated using two merging approaches. The sizes of the space window and time window used were chosen in such a way that the total numbers of final merged positions are almost equal between them. One can see that the processed trajectories with spatial proximity approximate the original trajectories better than those with temporal proximity. In particular, segments representing extended trajectory regions (labeled A, B, C, D) show notable differences, as they tend to be straight lines because more distant positions fall in the time window and are merged. The extent and shape of the trajectories appear to be better preserved when position merging is performed using space window.

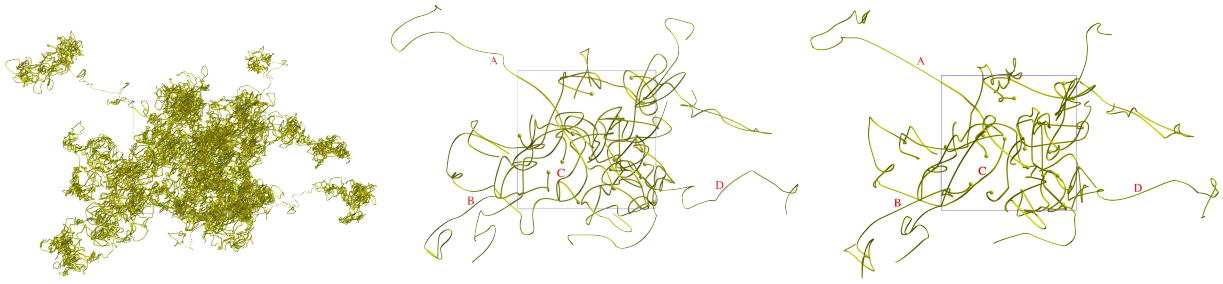


Figure 2.22: Comparison of trajectories (original on left) processed with spatial proximity (middle) and temporal proximity (right). Lines segments labeled as A, B, C, D are marked for comparisons.

2.7.2 Effects of Space Window

For a given data set, the size of the space window (cut-off) applied controls the degree of clutter reduction to a great extent. The larger the cut-off, the more positions are merged along the trajectories and less crowdedness (Figure 2.23). However, too wide space windows can cover multiple localities, and new positions produced by the merging process may be too few to approximate the trajectory geometry to any acceptable level (Figure 2.23, right). For this reason, there is a trade-off between cut-off (clutter measure) and the trajectory geometry.

The choice of the space cut-off requires domain specific knowledge and some pre-processing of a given position-time series data (such as radial distribution function and mean square displacement). The adaptive approach allows the user to adjust the space cut-off by considering a few factors such as the number of trajectories, the shape and extent of trajectories, the desired level of clutter reduction, and the degree of trajectory approximation. For instance, if a single trajectory (one selected atom) or a few trajectories (atoms forming a structural unit or cluster) are visualized, a small cut-off may reduce the associated clutter to an acceptable level. However, if many (corresponding to all atoms of one species) or entire trajectories are rendered, relatively large cut-offs are needed.

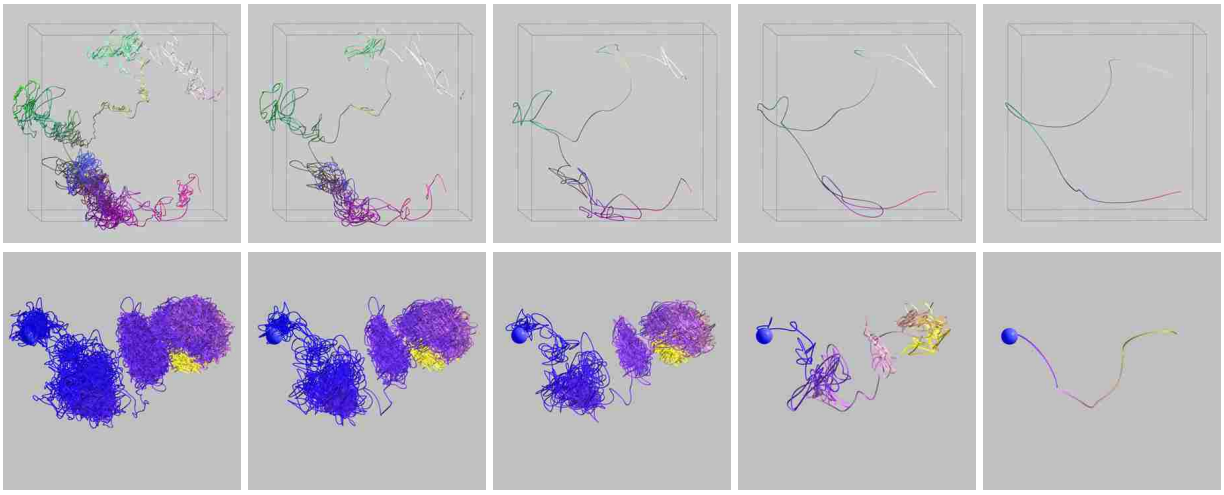


Figure 2.23: Effects of the space window (cutoff) size on the trajectory geometry. One H trajectory (top row) and one Si trajectory (bottom row) are processed using four different cutoffs. On increasing cutoff from left to right, the trajectories become simpler with local features disappearing more and more. The final trajectory (right) in each case fails to preserve the overall geometry. The color encodes time information along the original and processed trajectories.

2.7.3 Trajectories Rendering

A detailed visualization of trajectories of hydrous silicate ($MgSiO_3$) and pure silica (SiO_2) liquids shows us that the atomic movements involve two types of motion: extended portions of the trajectories represent rapid continuous motion whereas the crowded portions represent confined (oscillating) motion. The atoms jump from one confined region to another in a relatively short time interval.

For the hydrous system, each trajectory of each species (H, Mg, Si, and O) is obtained by rendering 60,000 successive positions. Figure 2.24 shows 16 H trajectories obtained by processing about one million positions in the total. They start from the respective initial atomic positions all lying within the supercell, and they eventually extend beyond the super-cell reaching 3 or 4 times the supercell length. Note that H atoms are the most mobile species. It is difficult to trace these trajectories except in the outermost regions. Also, any atomic structures such as initial configurations shown by atomic spheres can hardly be seen. The H trajectories are approximated using our adaptive position merging approach with a cut-off 1.5Å and five levels of merging (Figure 2.24,

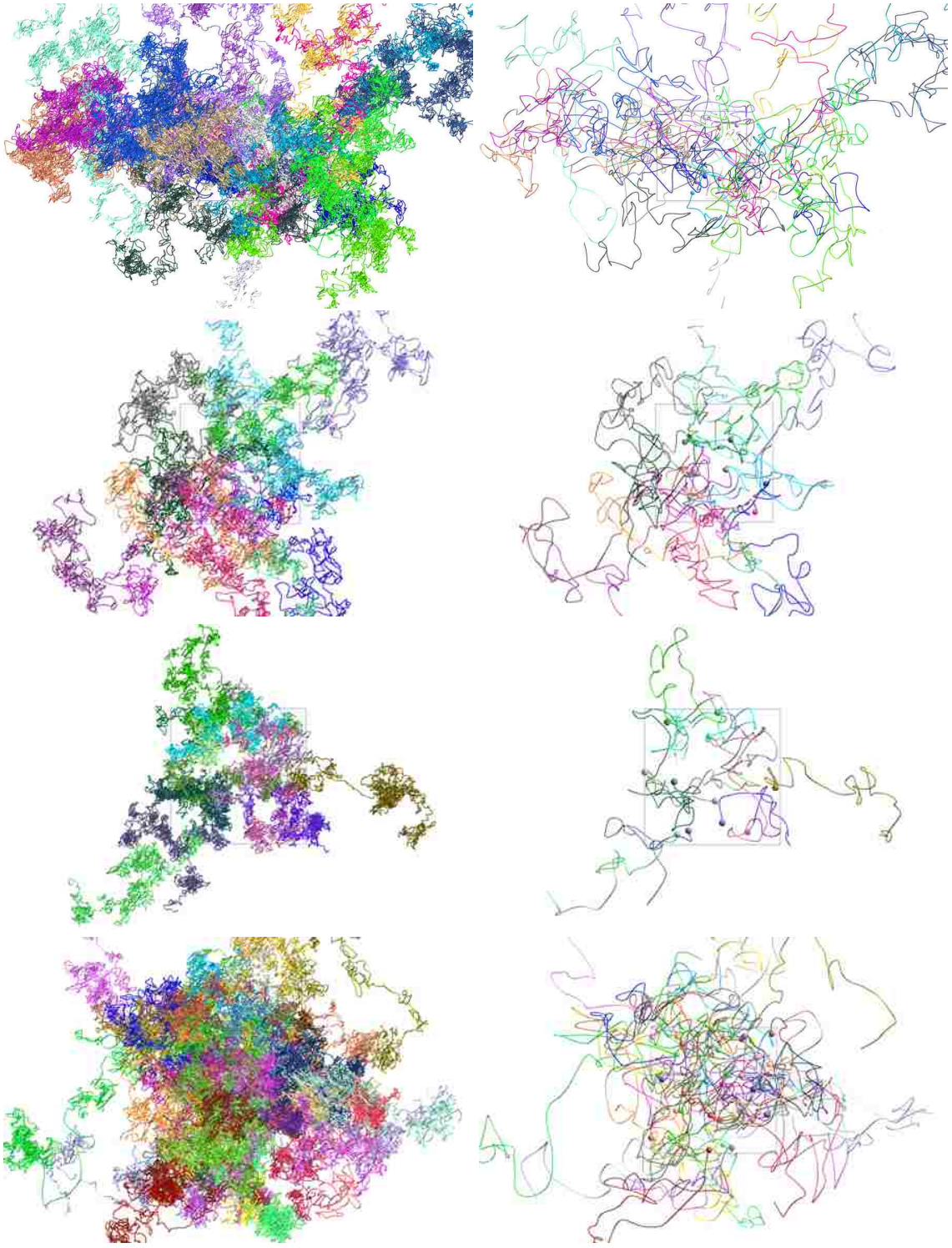


Figure 2.24: Trajectories (each colored differently) of 16 H (top row), 12 Mg (second row), 12 Si (third row), and 44 O (bottom row) atoms in hydrous silicate liquid; original (left) and after hierarchical merging (right).

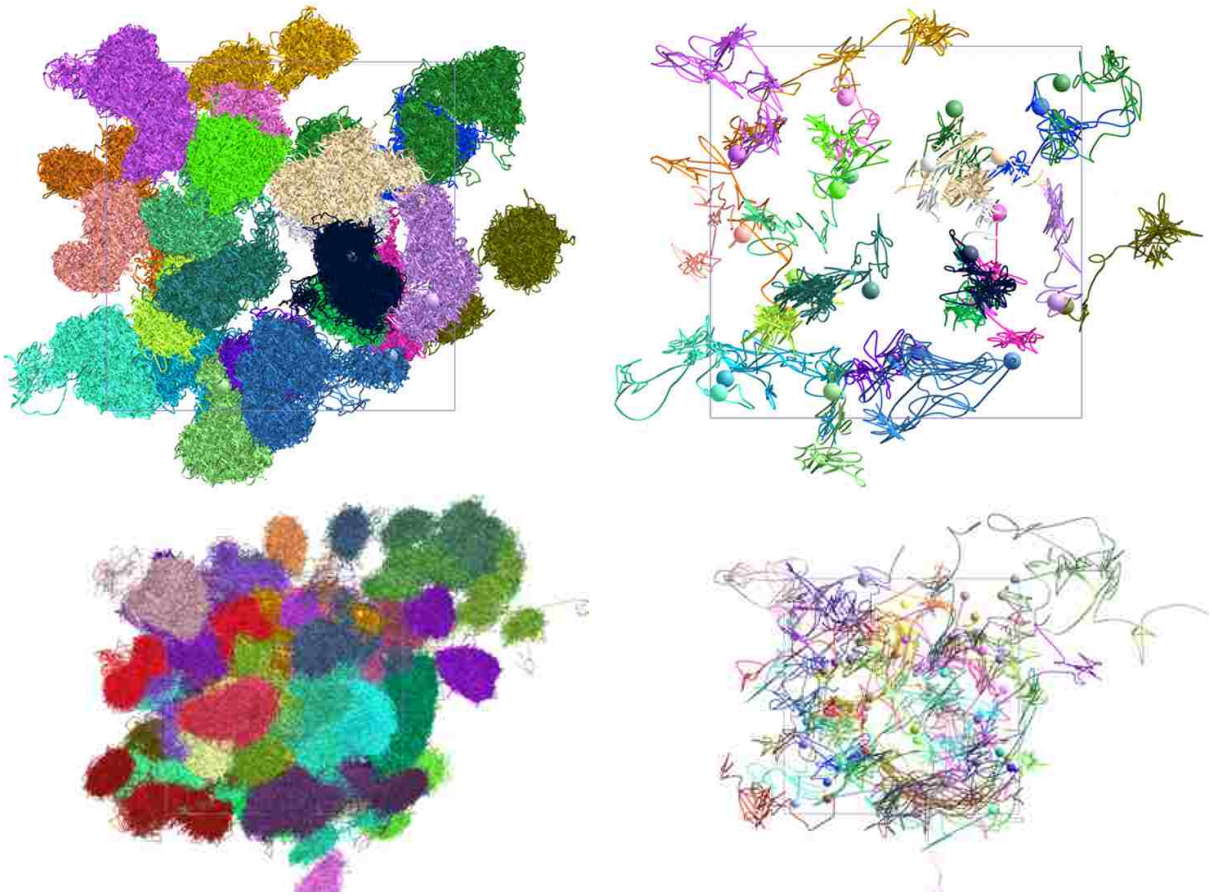


Figure 2.25: Trajectories of 24 Si (top row) and 48 O (bottom row) atoms in silica liquid; original (left) and after hierarchical merging (right).

right). This process suppresses local wiggling features and reduces the number of positions along the trajectories. Individual trajectories can be seen almost everywhere, and all atomic spheres are visible. During the visualization process, the user can explore the complete trajectory geometry with the aid of operations like rotation, translation, scaling, and highlighting.

The atoms in the silica liquid are very slow, covering relatively small distances even over much longer simulation duration. In other words, the atoms are trapped locally most of the time. The Si trajectories obtained by rendering one million positions per atom are highly confined to narrow regions, and they extend very little outside the supercell (Figure 2.25). Different trajectories tend to form distinct regions displayed with different colors, but these regions overlap among each other. Hence, It is difficult to assess the shape and extent of these trajectories because the geometric

structures are almost invisible. Our adaptive hierarchical position merging approach is expected to be useful.

2.7.4 Performance Analysis

The effectiveness of position merging along the trajectories can be assessed by counting the number of final (merged) positions used for trajectory rendering, examining the geometries (shape and extent) of individual trajectories, and having a measure (qualitative) of clutter reduction in visualization of trajectories and atomic structures. Key factors relevant for this assessment include space window size (cut-off distance), hierarchical merging level, and also imposed information constraint. Here, we present a performance analysis considering two cases: weak merging (with a narrow space window) and strong merging (with a wide space window) for each species trajectory for both material systems.

As shown in Table 2.1, the effects of window size (or cut-off) are substantial. For H trajectories, merging with a cut-off of 0.8Å involves 5 hierarchical levels and yields 16,450 final points compared to about 1 million original positions. The visibility of the trajectories improves considerably and some atomic spheres start to appear. When a larger cut-off of 1.5Å is used (again with 5 hierarchical levels) the number of final positions drops to 3,170 − a reduction by a factor of 300 with respect to the original positions. As a result, the individual trajectories and underlying atomic structure become visible. For O trajectories, the number of final positions drops from about 9,000 (weak merging) to 1,770 (strong merging), compared to 2.6 million original positions. Similarly, the number of final positions drops by a factor of 4 and 6 between the two cases of merging for Mg and Si trajectories, respectively. A relatively larger effect is seen for Si trajectories because most of them are more confined.

Position merging along atomic trajectories for silica liquid is much more sensitive to window size than that for hydrous silicate liquid. A weak merging along Si trajectories using a cut-off of 0.5Å saturates at the 8th level and generates around 57 thousand positions, compared to the original 24 million positions. The number of reduced (final) positions drops to around 4 thousand

Table 2.1: Number of atomic positions, original and after merging, of individual atom species in hydrous silicate liquid (top part), and silica liquid (bottom part) for weak merging (small cut-offs) and strong merging (large cut-offs). The number of positions after first-level merging at corresponding cut-off is presented inside the parentheses. The last column represents the number of positions for a constrained merging with large cut-off.

Original Positons		Weak Merging			Strong Merging			Constrained Merging
		Cut-off	Levels	Positions	Cut-off	Levels	Positions	
H	960,000	0.8	5	16,450 (24,710)	1.5	5	3,170 (7,160)	15, 630 (16,830)
Mg	720,000	0.5	4	6,818 (7,975)	1.0	4	1,585 (2,520)	14,000 (14,015)
Si	720,000	0.5	4	2,920 (4,540)	1.0	3	510 (740)	1,975 (2,115)
O	2,640,000	0.8	5	8,945 (14,125)	1.5	4	1,770 (2,615)	5,480 (6,170)
Si	24 millions	0.5	8	57,420 (507,490)	0.7	5	3,865 (12,055)	51,530 (75,110)
O	48 millions	0.8	8	93,880 (989,570)	1.2	6	3,150 (12,290)	–

with a somewhat larger cut-off (0.7Å). The overall reduction in the number of positions needed for trajectory rendering is by a factor of about 6,200. Thus, a much smaller set of processed positions, which represents only 0.016% of the original positional data, is rendered, thereby dramatically reducing clutter in rendering of Si trajectories. Since O atoms cover longer distances, we use larger cut-offs of 0.8Å (weak merging) and 1.2Å (strong merging). The weak merging yields little less than one hundred thousand positions and the strong merging case yields much fewer positions (3,000 positions) for a reduced trajectory, which represent about 0.0066% of the original positional data and result in a much improved visibility.

The number of hierarchical levels needed to saturate merging varies somewhat depending on the space cut-off and atomic species. This number spans the range of 3 to 5 for two cases (weak and strong merging) considered for all species of hydrous silicate. The number is higher (5 to 8 levels) for Si and O trajectories of silica. Hierarchical merging tends to continue deeper for the cases of highly confined trajectories, and the number of new (merged) positions along trajectories decreases as merging goes deeper the merger hierarchy. As shown in Table 2.1, the numbers of positions differ significantly between the first level and the final level of hierarchical merging in both weak and strong merging.

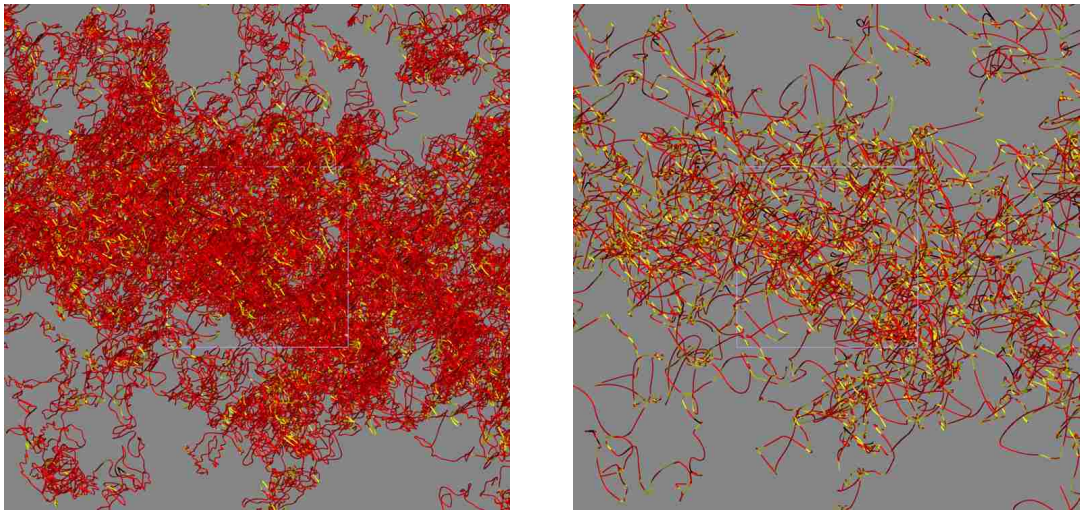


Figure 2.26: Trajectories of H atoms with color-coded coordination information with respect to O atom (red: singly coordinated, yellow double coordinated), before (left) and after (right) constrained merging with cut-off of 1.50Å.

Finally, we consider the sensitivity of position merging to a coordination information constraint (Figure 2.26) with coordination constraint). As shown in Table 2.1, fewer levels are needed to saturate merging in the hierarchy, and the number of final positions generated by merging along H trajectories, with H-O coordination as constraint, increases from around 3,000 (for original merging) to above 15,000 for a cut-off of 1.5Å. Similar increases in the number of final positions were found for other cases.

2.7.5 Metrics for Merging

Adaptive hierarchical merging of positions approximate original trajectories to lessen the clutter. Each level of merging will increasingly reduce the number of positions and line segments for rendering, with local features of trajectories disappearing more and more. Measuring deviation of approximated trajectory with respect to the original trajectory can be an effective metric for deciding whether to continue further merging or not. We consider a couple of metrics for measuring the deviation of a new trajectory from the original. In this case, deviation is simply the euclidian distance from new positions to the original positions.

$$P_{RMSD} = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(p_{i,t} - \overline{p_i})} \tag{2.15}$$

where $p_{i,t}$ is the position of an atom i at time snapshot t.

One approach is to compute the root-mean-squared deviation/error (Equation 2.15) of new positions with respect to original positions, which were merged to produce the new position. The value of deviation varies along the trajectory positions depending upon the movement of an atom over time as well as the cut-off window size. As expected, the mean-squared deviation increases with bigger cut-off window size, as shown in Figure 2.27 (weak merging vs. strong merging), and with multiple levels of hierarchical merging (Figure 2.29). For a cut-off window, the median deviation after first level of merging is usually less than half of the window size; however, deviation along simplified trajectories increases with multiple levels of merging, thus, median deviation,

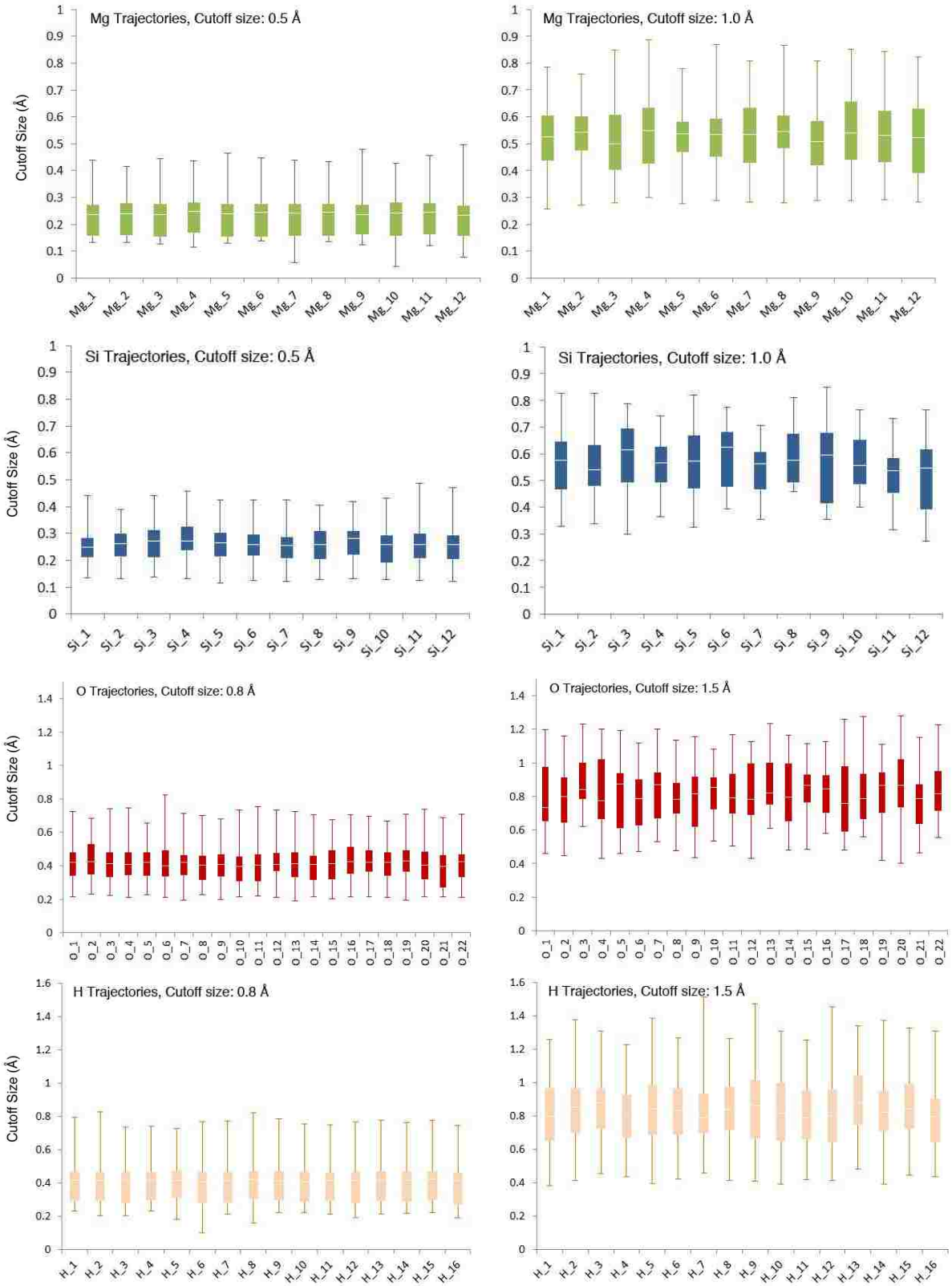


Figure 2.27: Box plots of root mean squared deviation (RMSD) along simplified Mg, Si, O, and H trajectories, in hydrous silicate melt, after weak-merging (left column) and strong-merging (right column).

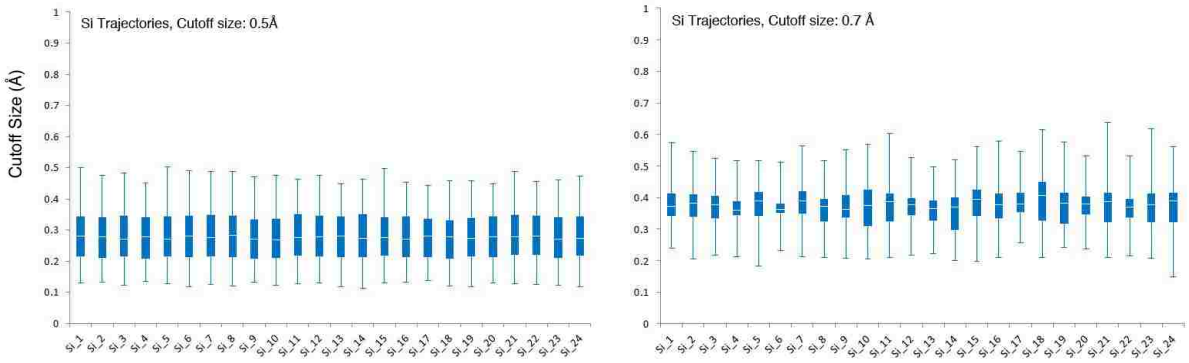


Figure 2.28: Box plots of RMSD along 24 simplified Si trajectories in silica liquid; weak merging (left) and strong merging (right).

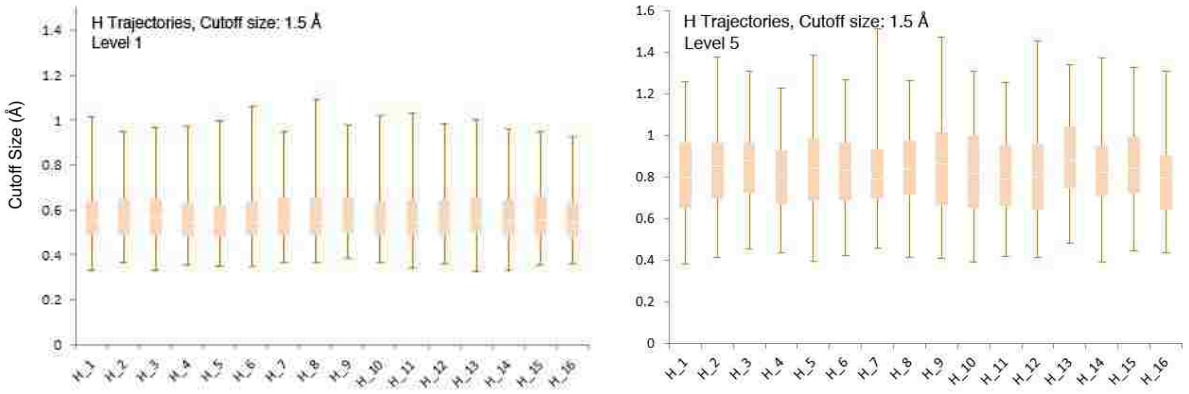


Figure 2.29: Box plots of RMSD along simplified H trajectories in hydrous silicate melt after first level merging (left), and fifth level and final merging (right) for a cutoff window of size 1.5Å.

after the merging saturates, is roughly the half of cut-off window size. A higher value of deviation indicates that the original trajectory is greatly simplified. In case of weak merging, box plots for measured deviation along simplified trajectories have less variation in shape and extent; whereas the extent of box plots varies notably for strong merging. Due to adaptive-hierarchical scheme, the maximum deviation after multiple levels merging gets closer to the cut-off distance, as represented by top whiskers in Figure 2.29.

Another metrics is the maximum deviation/variance at each position after the merging. In our implementation, maximum deviation is the distance of farthest original position (within cutoff window) from the merged position. For any cutoff window, the maximum possible value of maximum

deviation is twice the size of the cut-off window. Furthermore, either of these deviation metrics can be color coded along the simplified trajectory while rendering, as shown in Figure 2.30.

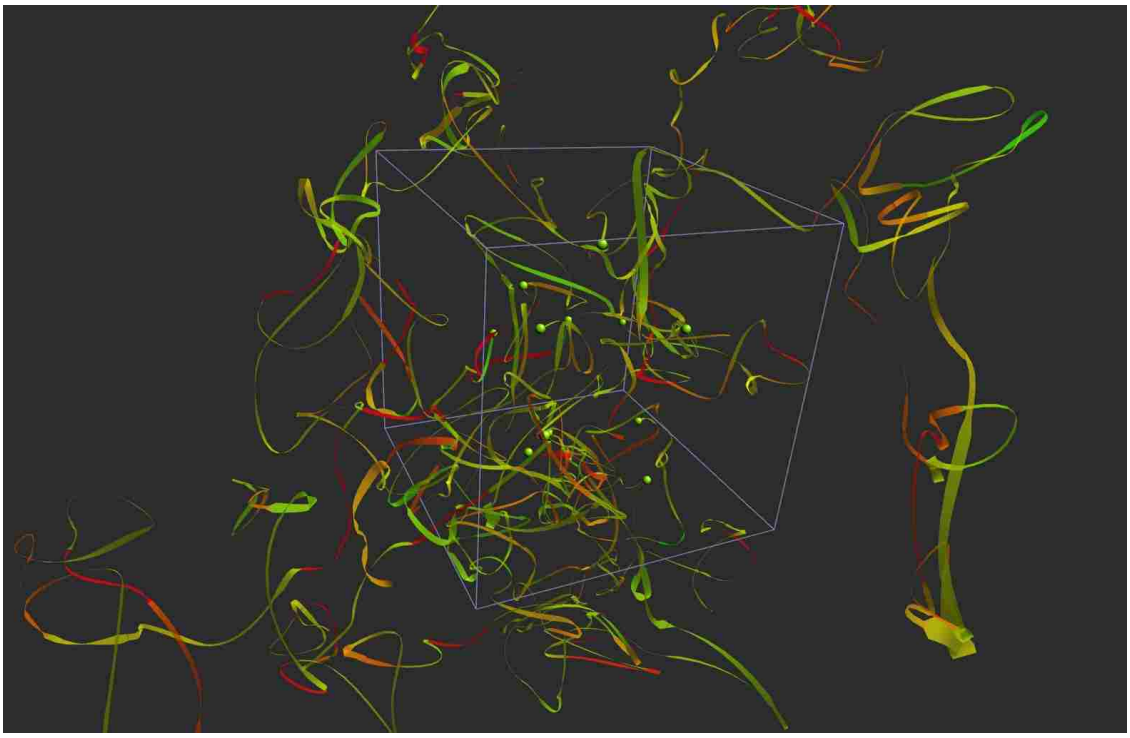


Figure 2.30: Trajectories of Mg atoms after adaptive merging rendered as Frenet ribbons, with merged positions count information encoded along the width of ribbon, and standard mean deviation value color coded: green(low) to red(high).

Chapter 3
Applications: Visualizing Simulated Silicate Melt

Silicate liquids are directly involved in all types of magmatic processes. It is, therefore, important to advance our knowledge about microscopic (atomistic) structures of component silicate liquids and how they control various relevant physical properties. Current information about melt structure is mostly based on the experimental studies of mostly glasses [33–36] and some melts [37, 38], which suggest that the structure exhibits short- to mid-range order. On the other hand, the computational approaches are now increasingly used in the study of non-crystalline geo-materials including silicate liquids [39–47]. Computations allow us to characterize the microscopic characteristics and the bulk properties of silicate liquids in a quantitative manner as well as in terms of underlying physics.

In this chapter, we present the detailed visualization-based analysis of atomic-position series data for model basalt melt for low (0 GPa) and high pressure (30.4 GPa). Moreover, hydrous magnesium silicate and silica liquid are two other simulated liquid systems that we have performed detailed visualization-based analysis prior to the model basalt visualization. The space-time multi resolution atomistic visualization system [5, 48] with subsequent improvement was used to gain insight into structural and dynamical information contained in the simulated data.

3.1 Simulation Data

The model basalt represents zero pressure eutectic composition, which is a mixture of 36 mol% anorthite ($CaAl_2Si_2O_8$) and 64 mol% diopside ($CaMgSi_2O_6$). The data for visualization was generated by simulating the model basalt liquid with the first principles molecular dynamics program VASP [49]. The simulation cell consists of 244 atoms (22 Ca, 14 Mg, 16 Al, 44 Si and 148 O) in a cubic volume of 3422.47 $Å^3$ (Figure 3.1, left), and for a simulation run of 50,000 time steps. The simulation schedule involved melting an initial atomic configuration at 6000 K, and then cooling isochorically first to 4000 K, then to 3000 K, and finally to 2200 K, and the calculated pressure is close to the ambient pressure. Further computational details can be found elsewhere [50, 51]. We perform visualization based analysis of model basalt melt for low (0 GPa) and high pressure (30.4 GPa; simulation cell of a cubic volume 2224.61 $Å^3$).

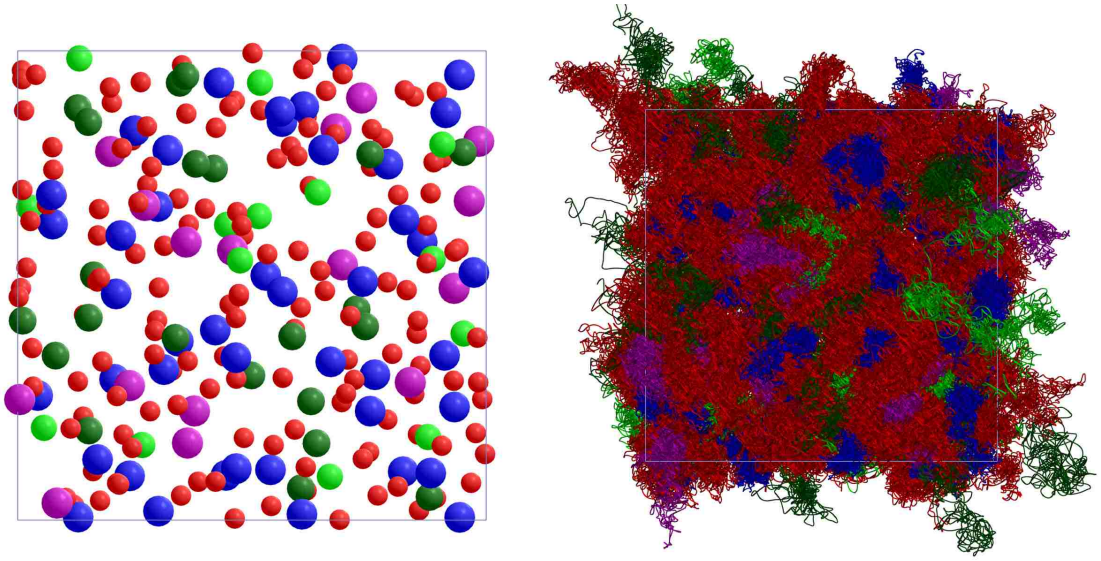


Figure 3.1: Rendering of the simulation data for model basalt liquid: an instantaneous snapshot of the constituent atoms as spheres on the left and a finite position-time series displayed as trajectories (for duration of 30 ps) on the right. The atomic species are represented by different colors: Ca (dark green), Mg (light green), Al (Magenta), Si (blue) and O (red).

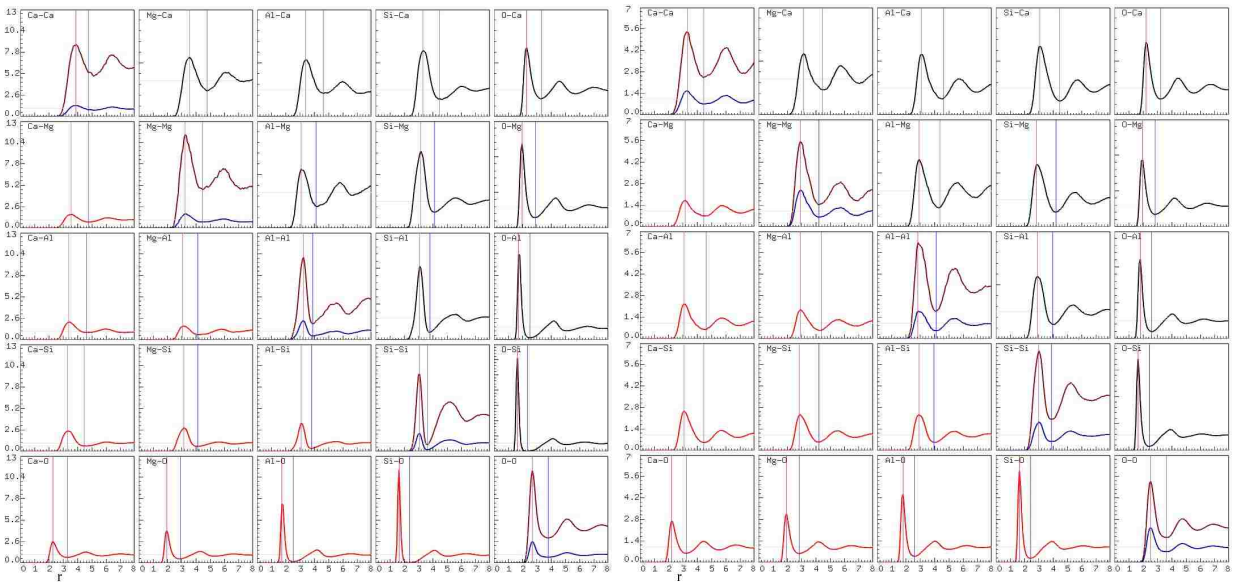
Here, we explore the atomic correlations and coordination environments by considering all atomic species to identify key structural units (such as Si-O polyhedra, non-bridging oxygen, bridg-

ing oxygen, over-coordinated cations, etc.) and explore their spatio-temporal behavior. Our main purpose here is to provide visual representations (microscopic pictures) of the model basalt melt structure. Figure 3.1 shows direct rendering of an instantaneous snapshot (one time step) as atomic spheres and a finite simulation run (30,000 time steps) as atomic trajectories.

3.2 Radial Distribution Function Matrix

Our structural analysis starts with the computation and rendering of the radial distribution function (RDF). The RDF matrix plot (Figure 3.2) allows us to compare all partial RDFs including like and unlike atomic pairs. It can be clearly seen that RDF varies remarkably in the shape between different atomic species pair types. Each type shows a characteristics feature that the size of the fluctuations after the first peak decreases rapidly with distance and the function approaches unity at larger distance indicating long-range disorder nature of the liquid state.

Each cation-anion (or equivalently, anion-cation) function is characterized by a sharp first peak followed by a well-defined minimum and then by a broad second peak. The Si-O RDF is perhaps best structured exhibiting the tallest and sharpest first peak and the value at the minimum after the first peak is almost zero. The Al-O function exhibits a shorter and broader first peak with both peak and minimum positions shifted to longer distances, compared to the Si-O function. Both cations (Al/Si) are considered as network-formers often referred to as tetrahedral cations (T), and Al/Si-O bonding essentially controls the melt structure. However, the RDFs of Ca and Mg with respect to O are not so well structured. The Mg-O RDF shows somewhat sharper peak and shallower minimum both lying at shorter distances compared to the Ca-O RDF. Compared to the cation-anion cases, other RDFs show relatively broader and shorter first peak and, in some cases the minimum and the second peak are poorly defined. Also, the peaks are located at relatively larger distances, for example, the first peaks of cation-cation distributions lie within a range of 3-4Å whereas the O-O peak position lies at 2.5Å. The first peaks show varying degree of asymmetry with the right side extending more than the left side and the minimum values being greater than zero. The fact that all



(a) Zero Pressure, 2200K (b) High (30.4 GPa) Pressure, 3000K

Figure 3.2: RDF matrix (symmetric) plot for model basalt melt. In each entry, the partial radial distribution function (in arbitrary units) for a given atomic species pair (α, β) is plotted as a function of distance (r, in Å). The lower triangle (off diagonal) entries show the partial RDF curves using the global (same: 0-10) vertical scale, whereas the upper triangle entries use the local (different) vertical scales. The diagonal entires show the RDF curves of alike species pair using both global and local scales.

RDF types do have the first peaks with their positions well spread in the distance range of 1.6-4Å suggests that the liquid contains short- to medium-range structural features.

We observe that at high pressure, the peaks get shorter, broader, and shift to shorter distances, as shown in Figure 3.3. In particular, the cation-anion peaks are significantly shorter for high pressure melts compared to the ambient pressure. For example, the Si-O first peak drops to less than half in height for high pressure. The RDF plot also allows us to interactively select the critical distances such as the distance to the first minimum, r_{min}, for subsequent analyses (note two vertical lines in each entry of Figure 3.2). For instance, the bonds exist between the cation-anion (e.g., Si-O) pairs of all interatomic distances covered by the first peak of the corresponding partial RDF. The calculated position of the first peaks in cation-anion functions compares favorably with available experimental data on silicate melts. For instance, the calculated Si-O peak distance is 1.625 Å, compared to the measured value (1.62 Å) for $MgSiO_3$ melt [38].

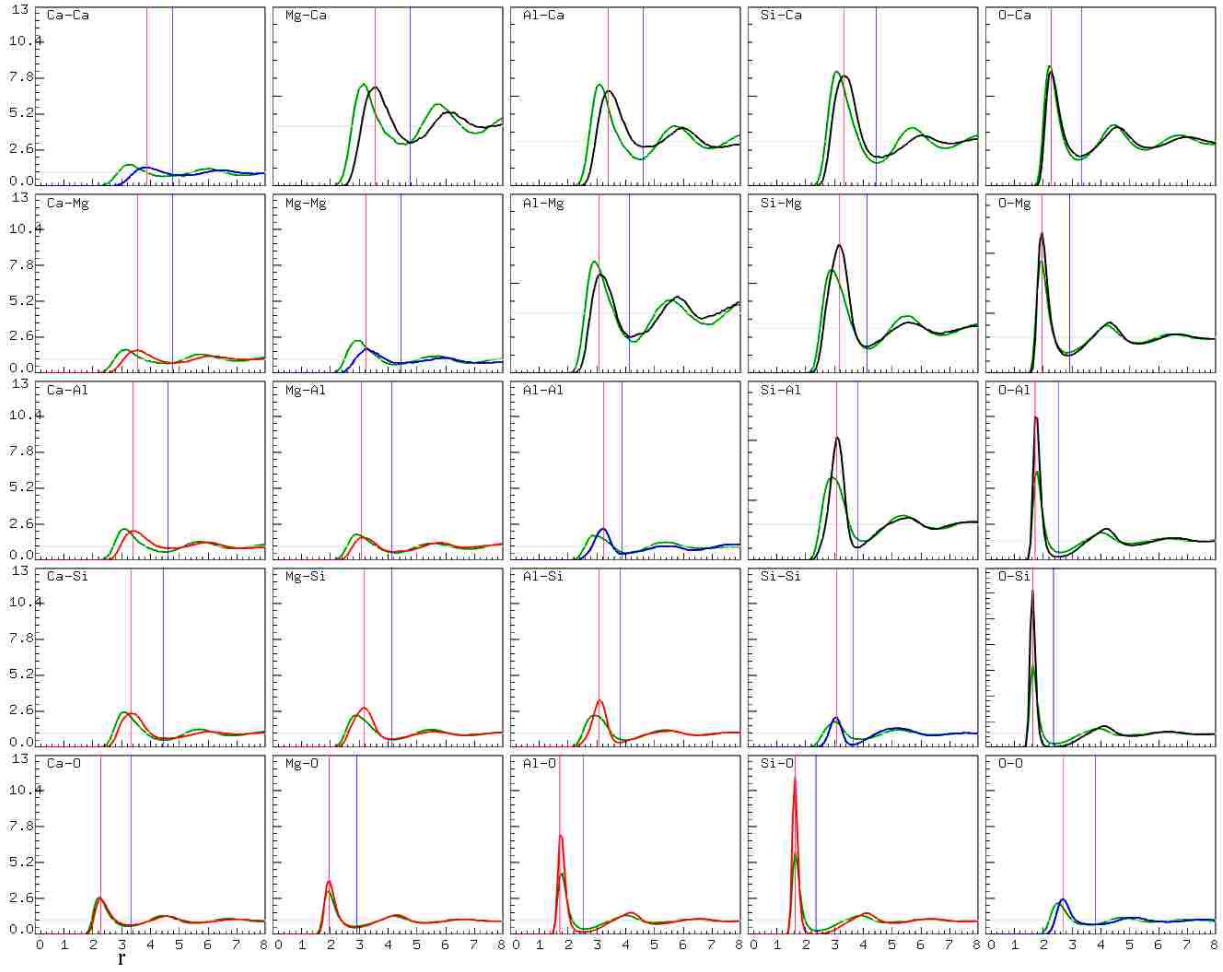


Figure 3.3: RDF matrix (symmetric) plot for model basalt melt, show radial distribution functions for low pressure overlapping the distribution functions for high pressure condition (green curves).

3.3 Mean Coordination Plot

The atomic coordination, which is often used to characterize the local structure, can be calculated for a given species α with respect to another species β using

$$C_{\alpha\beta} = 4\pi\rho_{\beta} \int_{0}^{r_{min}} r^2 g_{\alpha\beta}(r)\ dr \tag{3.1}$$

This nearest neighbor coordination ($Z_{\alpha\beta}$) is the number of contributing atoms (of species β), which lie within a spherical region centered at atom of species α and of radius defined by the

corresponding r_{min} value. The model basalt liquid shows 25 types of coordination as represented by an asymmetric square matrix.

Table 3.1: Mean coordination number matrices of model basalt liquid with 25 possible types of coordination. Two matrices compare the mean coordination number of possible coordination types for model basalt liquid at different pressure condition: low (left matrix) and high (right matrix).

	Ca	Mg	Al	Si	O	Ca	Mg	Al	Si	O
Ca	3.1	1.8	1.9	6.3	7.1	3.0	2.0	2.7	7.6	9.6
Mg	2.9	1.0	1.9	4.7	5.3	3.2	2.2	2.1	6.4	6.9
Al	2.6	1.6	1.7	3.4	4.8	3.7	1.8	1.7	5.6	5.9
Si	3.1	1.5	1.2	2.2	4.5	3.8	2.0	2.0	4.2	5.2
O	1.0	0.5	0.5	1.3	9.8	1.4	0.6	0.6	1.5	12.7

Each mean coordination type is expressed as a single number. The diagonal terms represent like-atom coordination whereas the off-diagonal terms represent unlike-atom coordination. The mean coordination numbers span a wide range from less than 1 to more than 9. All mean coordination numbers at high pressure condition are systematically higher than those at zero pressure, as seen in Table 3.1. The mean Ca-O, Mg-O, Al-O and Si-O coordination numbers follow the order of the cationic sizes. The larger is the cation ion, the larger is the mean coordination of that ion with respect to O (Figure 3.4).

On the other hand, the opposite coordination (i.e., anion-cation) depends on the number of cation under consideration more than on the cation size. For instance, C_{OSi} and C_{OMg} are the maximum and minimum, respectively. Similarly, cation-cation coordination values depend on both the number and size of relevant cations. The mean coordination of each cation with respect to Si is higher than that with respect to any other cation. Finally, the mean anion-anion coordination number takes the highest value of around 10 being distinctly higher than any other coordination number.

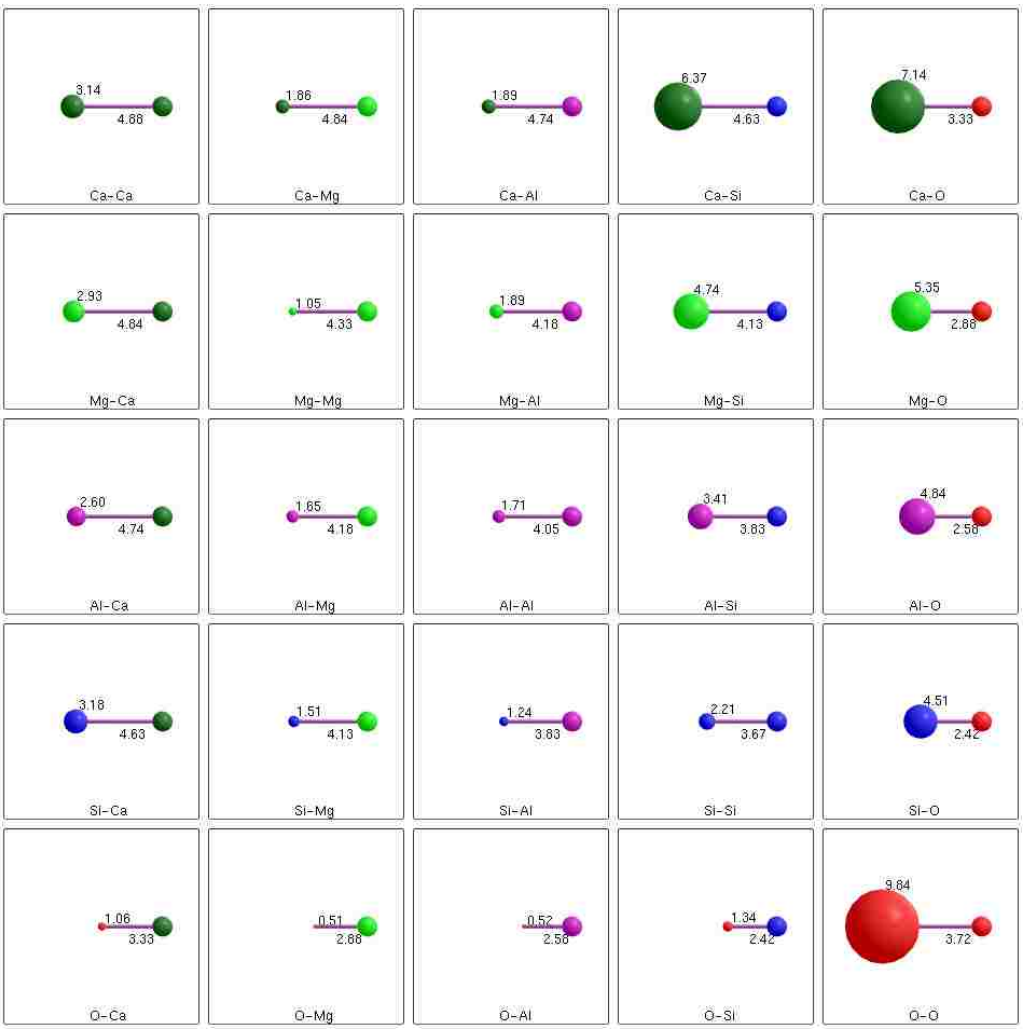


Figure 3.4: Mean coordination matrix plot showing 25 different coordination types. The coordination number is encoded by the size of the first sphere and r_{min} is encoded by the length of the line connecting two spheres. Color represents the atomic specs which are coordinated.

We can consider Si and Al together as tetrahedral (T) atoms, and Mg and Ca together as structure modifier (M) atoms. This reduces the types of coordination environments from 25 to 9. It is evident that C_{TO} and C_{MO} represent the weighted averages of the corresponding cation-anion coordination numbers whereas C_{OT} and C_{OM} represent the sums of the corresponding anion-cation coordination numbers. Unlike individual C_{OSi} and C_{OAl}, their combined (C_{OT}) value is almost equal to 2, which means that Al and Si together form a nearly complete network via bridging oxygen. This is consistent with the value of C_{TT} being higher than 4.

3.4 Coordination Environment

The atomic coordination (or nearest neighbor coordination) is the number of contributing atoms (of species β), which lie within a spherical region centered at atom of species α and of radius defined by the corresponding r_{min} value. The model basalt melt shows 25 pairs of coordination environments as shown in Figure 3.5.

Coordination Matrix Plot

The coordination data can be visualized in the form of spheres or polyhedra with their values encoded using a color map. Figure 3.5 renders the coordination states at the individual atom level for all 25 environments arranged in the matrix form.

The coordination environments involving Si/Al and O provide the information associated with the degree of polymerization in a silicate melt. A mixture of various coordination species (from 2- to 8-fold states) is present. The Si-O coordination is much more tetrahedral (${}_4\mathrm{C}_{SiO}$ = 96%) than the Al-O coordination (${}_4\mathrm{C}_{AlO}$ = 78%), which also consists of pentahedra in large proportion (${}_5\mathrm{C}_{AlO}$ = 20%, compared to ${}_5\mathrm{C}_{SiO}$ = 4%). Five-fold coordinated Si and Al have been detected experimentally in silicate glasses [34, 36, 52]. The O atoms are less coordinated (singly or doubly) with Al than they are coordinated with Si, and as such, more O atoms are not bonded to any Al (58%) than to any Si (5%). The large difference between the O-Si and O-Al coordination suggests the important role of Si-O bonding in melt structure. Their structural relevance becomes more insightful when Si and Al atoms considered together. Almost all O atoms are now coordinated with one or more T atoms. Unlike tetrahedral atoms, coordination environments for Ca and Mg atoms involve high-coordination species including 5-, 6-, and 7- fold states existing at roughly equal proportions. Some of these states have been suggested based on the experimental studies [53–55]. One can combine two or more viewports to render mixed environment such as showing T-O and O-T coordination spheres together (Figure 3.7). All coordination environments for high pressure have higher atomic coordination states than those at low pressure, as seen in Figure 3.6 compared to Figure 3.5.

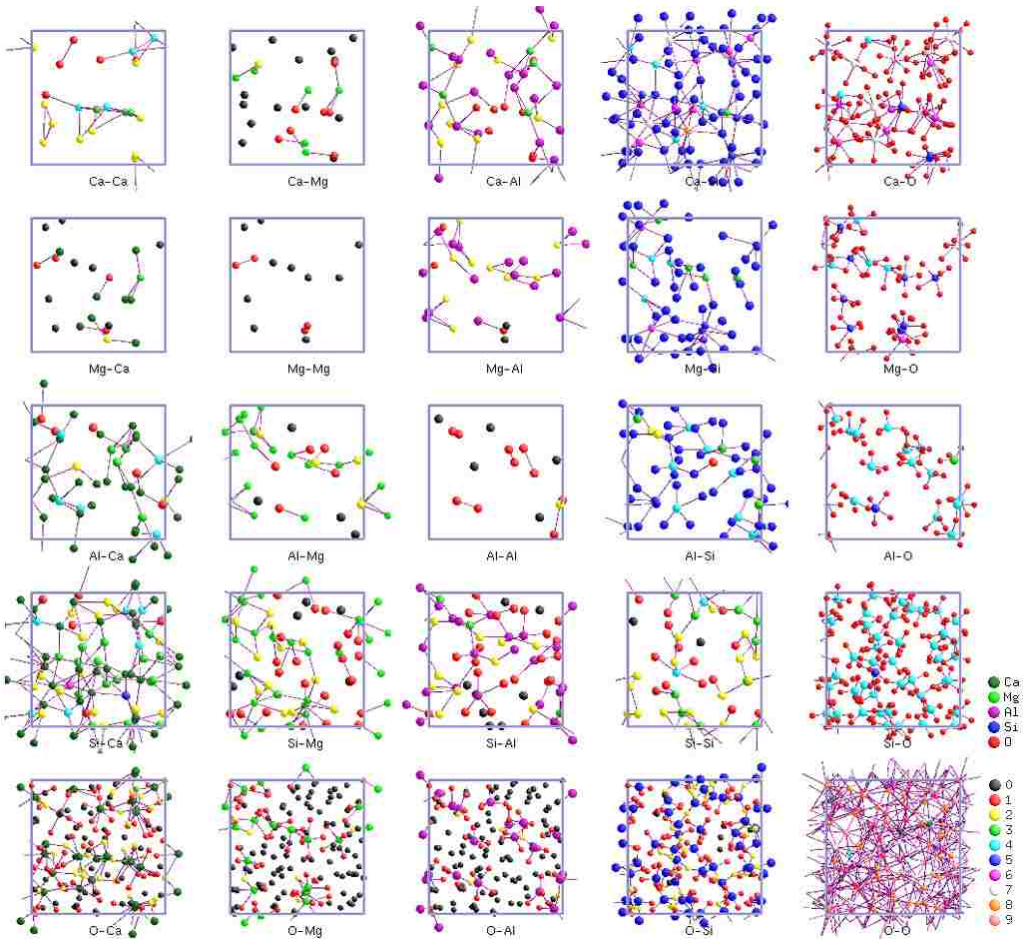


Figure 3.5: Per-atom coordination matrix plot of model basalt melt. The spheres, in each entry, represent the atoms of two species (α and β) forming coordination environment. Color of the centered spheres (of species α) encodes the respective coordination numbers with respect to atoms of species β.

Coordination State Changes

It is interesting to see how an instantaneous structure of the melt changes with time. An animation can be performed to show how and when coordination species appear and disappear as time evolves, that is, high coordination species are formed from low coordination species (for example, the appearance of 5-fold defects in a tetrahedral environment) and vice versa (S2). Visualization suggests 3 relevant mechanisms for coordination changes involving T and O atoms (Figure 3.7). Four states of oxygen coordination with respect to T are relevant: free oxygen (FO) which is not bonded to any T atom, non-bridging oxygen (NBO) which is singly bonded with one T atom, bridg-

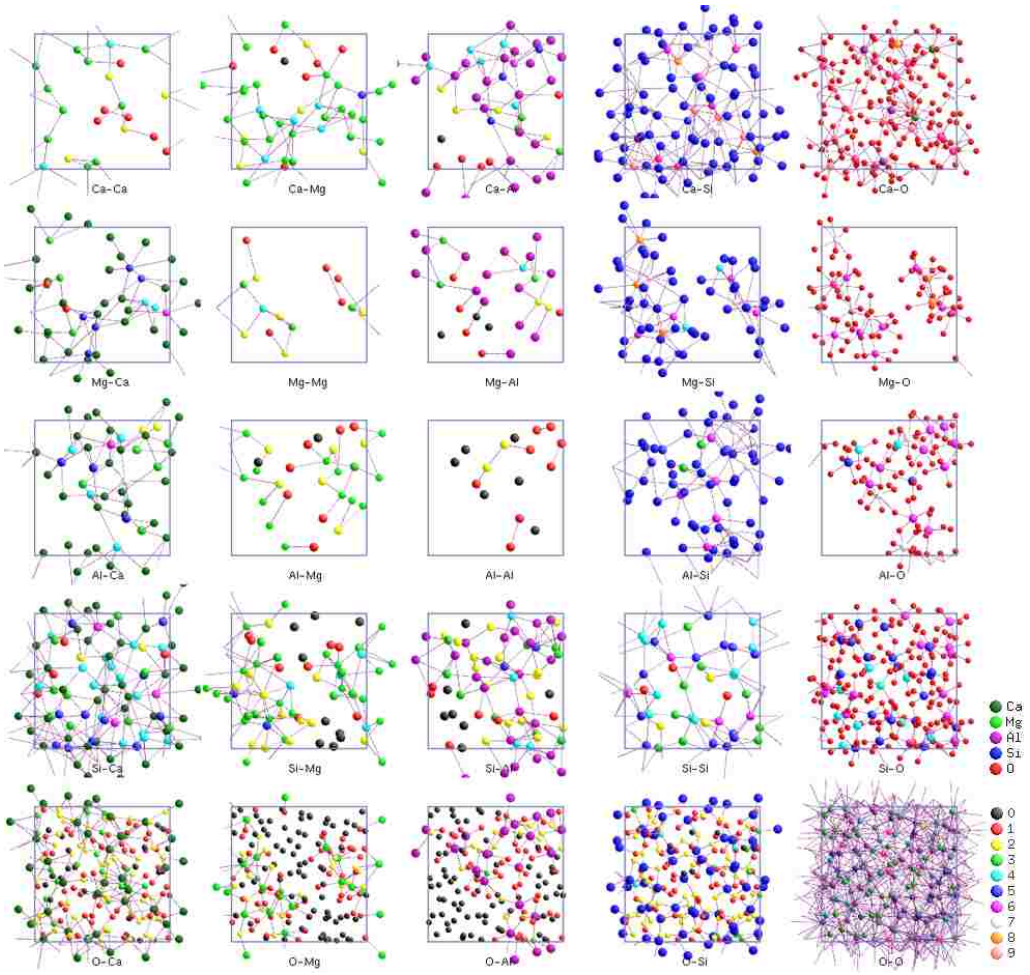


Figure 3.6: Per-atom coordination matrix plot of model basalt melt for high pressure. The spheres, in each entry, represent the atoms of two species (α and β) forming coordination environment.

ing oxygen (BO) which is bonded with two T atoms, and oxygen-tri-cluster (O3) which is bonded to 3 T atoms. The presence of NBO and O3 has been suggested by the experimental studies of silicate glasses [33, 56].

In the first mechanism, FO gets bonded with T atom to form a NBO, the state (Z) of T coordination with respect to O increases. If the bond (between NBO and T) breaks, FO appears and Z decreases. Thus, we have the following reaction:

$$FO +_z T^n \leftrightarrow NBO +_{z+1} T^n \tag{3.2}$$

where n is the number of O atoms bridging two T atoms and remains unchanged. In Figure 3.7 (upper panel), FO (black small sphere) and 4-fold coordinated T atom (cyan large sphere) turn to NBO (red small sphere) and 5-fold coordinated T atom (blue large sphere), respectively.

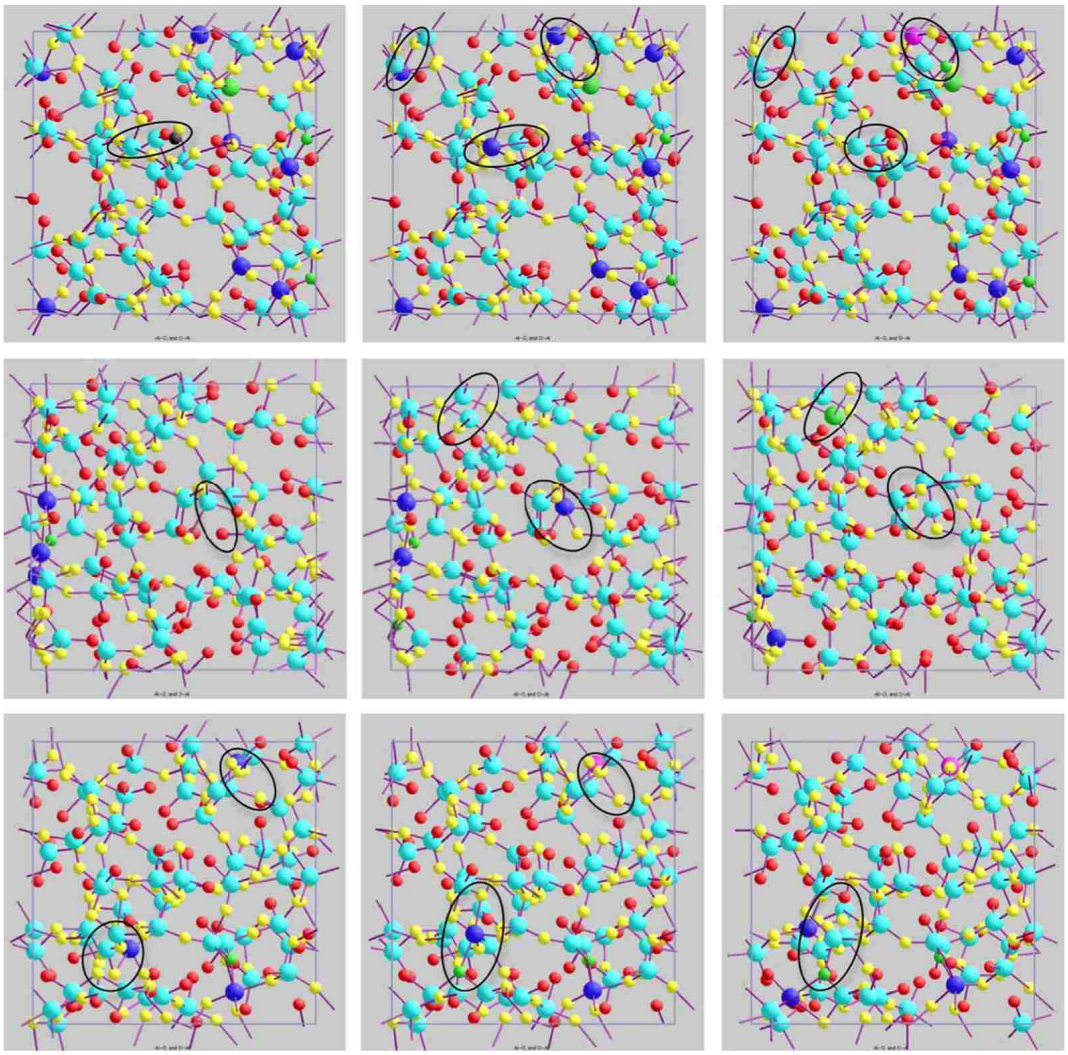


Figure 3.7: Changing T-O and O-T coordination states during simulation as marked by ellipses. Tetrahedral (T) atom represents both Si and Al species. The 3-fold, tetrahedral, pentahedral and octahedral coordination states of T atoms with respect to O are shown by green, cyan, blue and magenta (large) spheres, respectively. The free (FO), nonbridging (NBO), bridging (BO) and tri-clustered (O3) coordination states of O atoms with respect to T are displayed as black, red, yellow and green (small) spheres, respectively. Note the color changes of spheres and the bond formation/breakage between snapshots in the marked regions.

In the second mechanism, NBO gets bonded with neighboring T so both the atoms undergo coordination increase. They are reverted back to their original states when the new bond is broken.

This reaction can be expressed as:

$$NBO +_Z T^n \leftrightarrow BO +_{Z+1} T^{n+1} \tag{3.3}$$

here both Z and n are affected. In the middle panel of Figure 3.7, NBO (red small sphere) and 4-fold coordinated T atom (cyan large sphere) turn into BO (yellow small sphere) and 5-fold coordinated T atom (blue large sphere), respectively. At a later time, a different BO turns itself into NBO thereby restoring the 4-fold T coordination state. Thus, one O enters into and the other O leaves the coordination shell. Also, note an event where a 3-fold T coordination state (green large sphere) appears when BO turns in to NBO.

The third reaction involves bridging oxygen and oxygen tri-cluster (O3):

$$BO +_Z T^n \leftrightarrow O3 +_{Z+1} T^{n+1} \tag{3.4}$$

When the T atom is bonded with BO atom that already bridges two other T atoms, both Z and n increase. In Figure 3.7 (bottom panel), a bond forms between BO (yellow small sphere) and 4-fold coordinated T atom (cyan large sphere) thereby converting them to O3 (green small sphere) and 5-fold state (blue large sphere), respectively. Later the 4-fold state is restored when BO turns into NBO. These mechanisms were previously suggested based on experimental studies of sodium silicates to explain the pressure-induced coordination increases [35, 57, 58].

Coordination Clusters

All β-type atoms contributing to coordination environment of a given α-type atom during the entire simulation time can be viewed collectively as a cluster. We visualize different types of cation-anion coordination clusters. For each cation, the cluster consists of all O atoms, which are bonded to the cation at some point of time during a finite simulation period. The clusters differ in size and shape within each coordination type as well as between different coordination types. The Ca-O clusters are largest whereas the Si-O clusters are smallest (Figure 3.8). Ca atoms are

most mobile, and as they diffuse through the liquid, they get bonded with many more O atoms over the time. Alternatively, because of weakest Ca-O bonds, O atoms can easily escape from the local neighborhood. This is supported by the widest and largest O-Ca clusters compared to other anion-cation clusters (Figure 3.9). On the other hand, the Si-O coordination cluster is very small with only a few O atoms lying at very long distances, and so is the O-Si cluster.

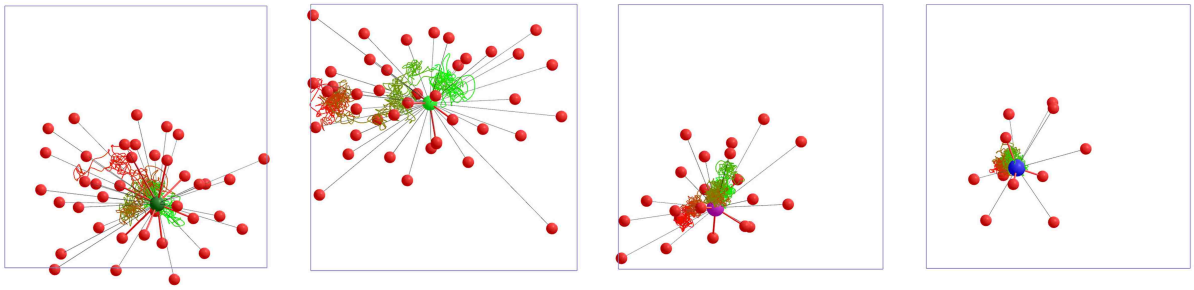


Figure 3.8: Coordination clusters of one selected cation for each type. The clusters represent the coordination of Ca, Mg, Al, and Si atoms (central spheres, from left to right) with respect to O atoms (small red spheres) over the simulation period of 30 ps. The current cation-anion bonds are shown in each case by the red lines and their counts represent the current coordination numbers of the respective cations. The gray lines connect the central atom to all O atoms, which are also bonded at different times. The trajectories of the central atoms are time-encoded (green to red).

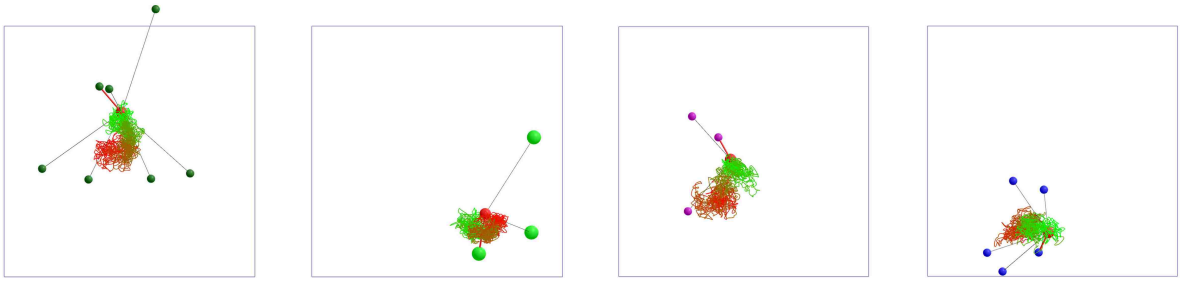


Figure 3.9: Coordination clusters of one selected O atom (central red sphere) with respect to each cation during the simulation period of 30 ps. The current anion- cation bonds are shown in each case and their count represents the current coordination state of the corresponding anion.

At high pressure, all coordination clusters, both cation-anion and anion-cation clusters, are systematically bigger that the clusters for low pressure condition. This is consistent with the increase in average coordination numbers for high pressure system, implying more atoms of type β are in coordination state with α-type atom for longer time.

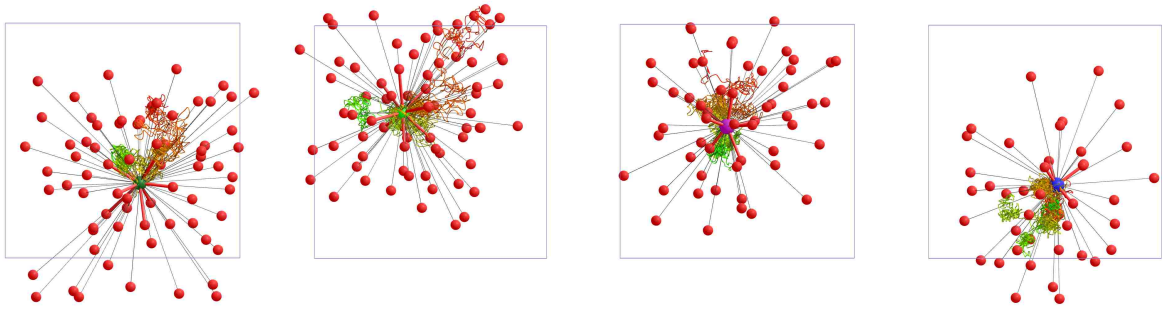


Figure 3.10: Coordination clusters represent the coordination of Ca, Mg, Al, and Si atoms (central spheres, from left to right) with respect to O atoms (small red spheres) for high pressure (30.4 GPa) condition at 3000 K. The current cation-anion bonds are shown in each case by the red lines and their counts represent the current coordination numbers of the respective cations. The gray lines connect the central atom to all O atoms, which are also bonded at different times. The trajectories of the central atoms are time-encoded (green to red).

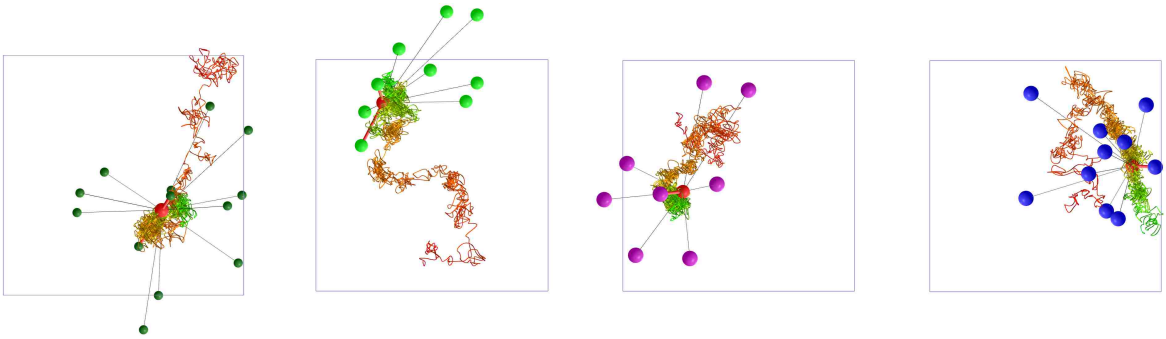


Figure 3.11: Coordination clusters of one selected O atom (central red sphere) with respect to each cation for high pressure (30.4 GPa) condition at 3000 K. The current anion-cation bonds are shown in each case and their count represents the current coordination state of the corresponding anion. The trajectories of central O atom are time-encoded (green to red).

Atomic Trajectories

The atom trajectories (Figure 3.12) can be used to encode the per- atom coordination information at multiple time steps, i.e., as a function of time [48]. As can be seen, the Si-O coordination is mostly 4-fold (long cyan portions) with 3- and 5-fold coordination states appearing for short durations (short green and blue portions). The Al trajectories also contain relatively long blue portions (5-fold) and also short magenta portions (6-fold). This means that highly stable 4-fold Si/Al-O coordination species dominate the melt structure. On the other hand, Ca and Mg trajectories are

composed of blue (5-fold), magenta (6-fold) and white (5-fold) segments of similar lengths thereby suggesting comparable stabilities of these high coordination states. Moreover, consecutive line segments with same coordination information can be simplified by merging consecutive positions (with same coordination value) along the trajectory (Section 2.5.5).

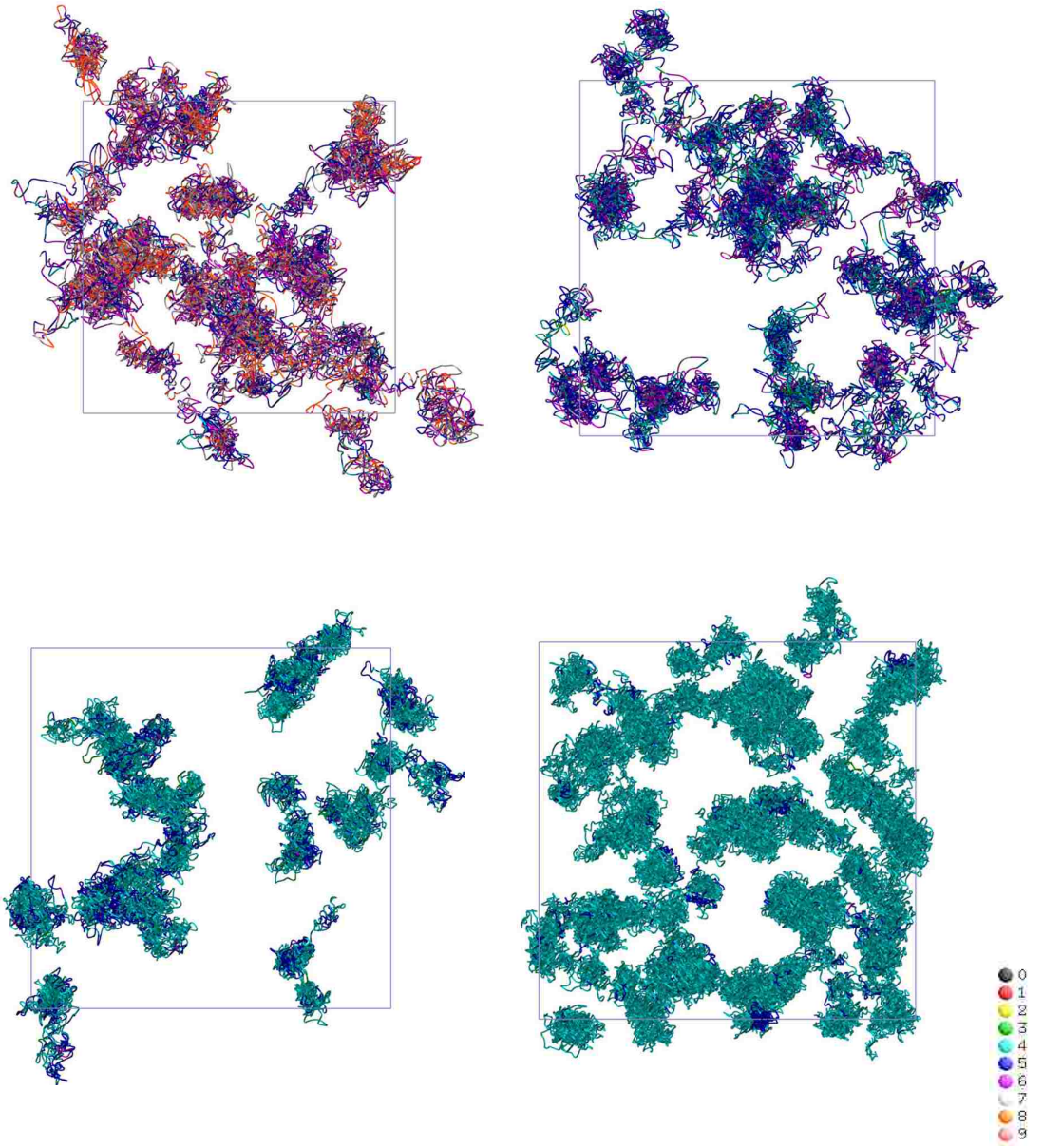


Figure 3.12: Trajectories of Ca (top left), Mg (top right), Al (bottom left), and Si (bottom right) atoms for 30 ps, encoding corresponding cation coordination with respect to O as per the color map shown.

At high pressure, the coordination states for cation-anion pairs are of higher-order than for low pressure. The Ca-O coordination is mostly 8-fold (orange) and 9-fold (pink) states. Similarly, the Mg-O coordination contain 5-, 6-, 7-, and 8-fold states all color coded along the Mg trajectories. And the Al and Si trajectories contain relatively long blue (5-fold) and magenta (6-fold), and short portions of cyan (4-fold) line segments (Figure 3.13).

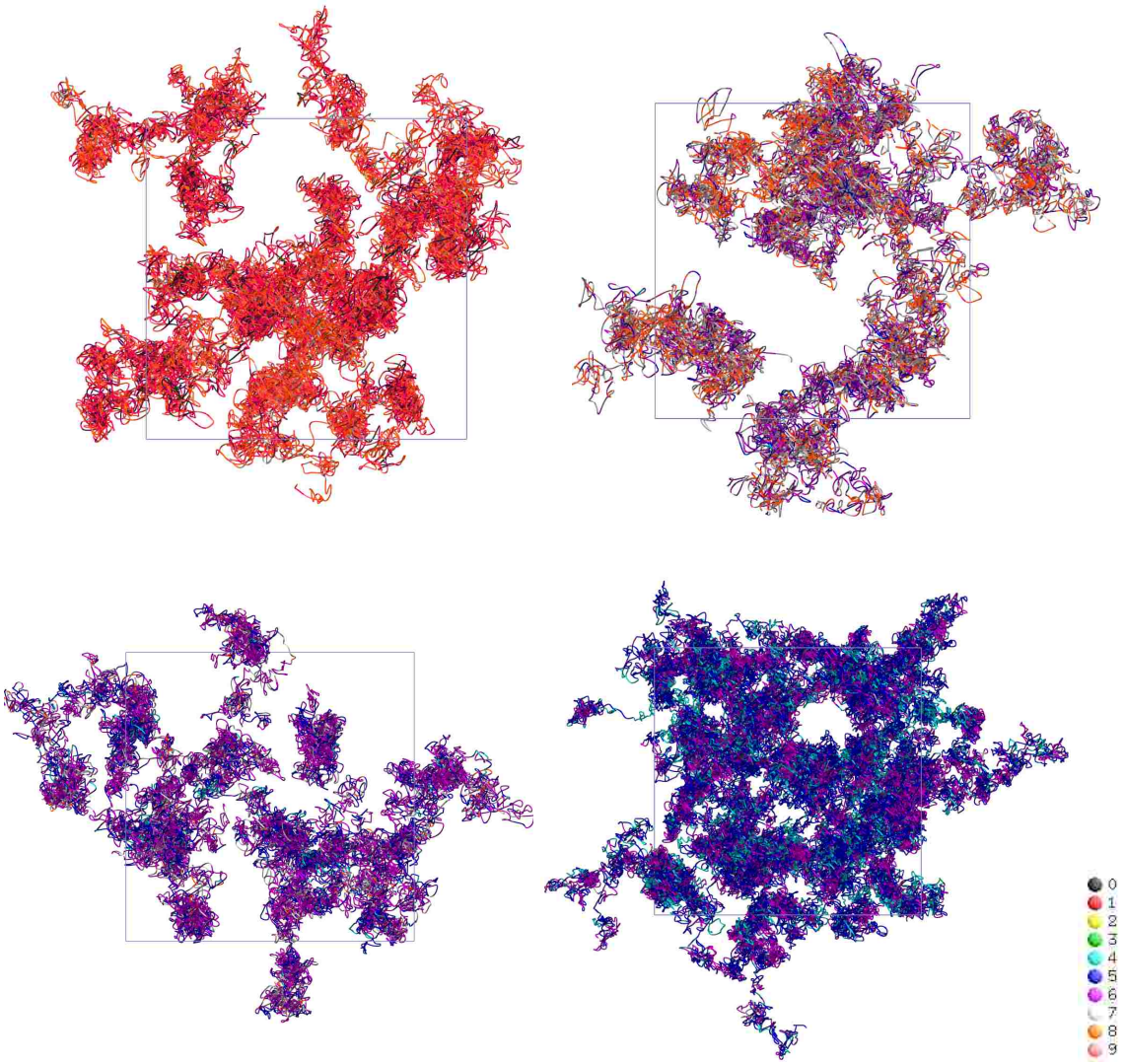


Figure 3.13: Trajectories of Ca (top left), Mg (top right), Al (bottom left), and Si (bottom right) atoms for melt at 30.4 GPa and 3000 K, encoding corresponding cation coordination state with respect to O, as per the color map shown.

3.5 Visual Melt

Our detailed analysis of the simulation data suggests that the model basalt melt (or any silicate melt) can be viewed as composed of Al/Si-O polyhedra, which are primarily Si-O tetrahedra, and Al-O tetrahedra and pentahedra. They rarely appear in isolation, rather they are mostly connected to each other via oxygen (BO) with many non-shared corners (NBO) present so the network is not fully complete. In such a polyhedral network formed by Si/Al and O atoms, other two cations (Ca and Mg) are relatively mobile and non-homogeneously scattered. Figure 3.14 visualizes the simulated model basalt melt (at 2200 and 3000 K) in the form of Al/Si-O coordination polyhedra with Ca and Mg atoms shown as coordination spheres. The Ca and Mg trajectories are also shown. At the higher temperature, the melt shows more pentahedral coordination states (blue polyhedra) and longer trajectories.

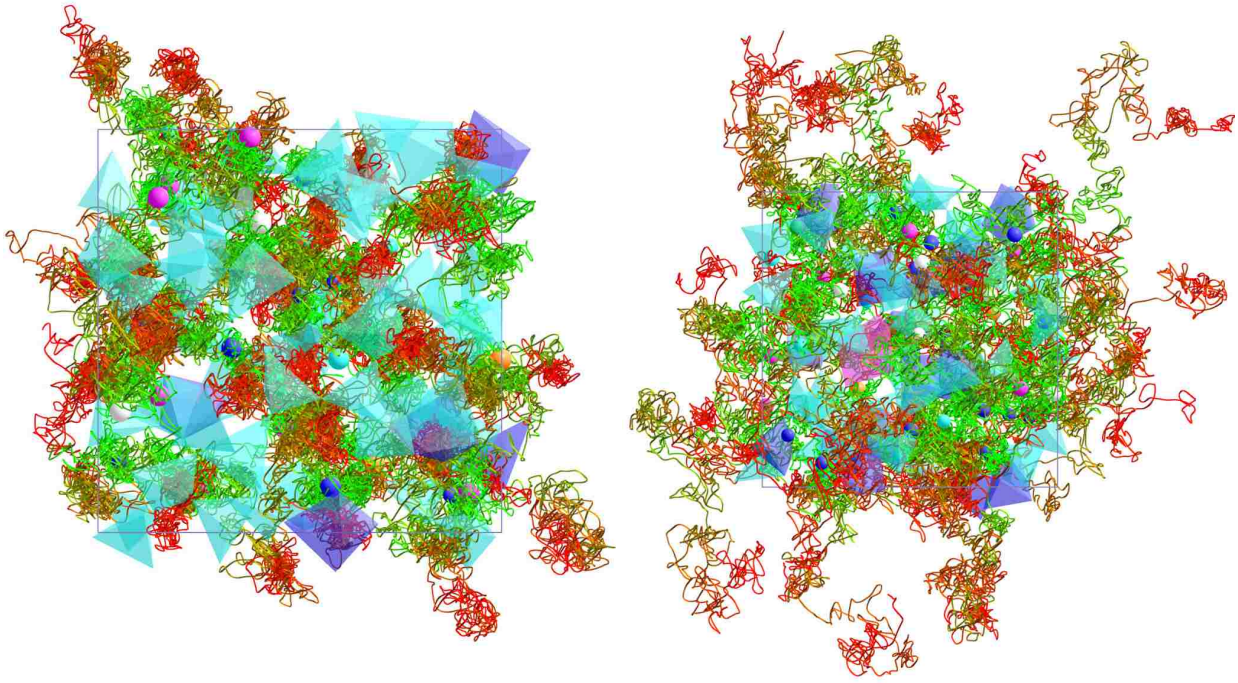


Figure 3.14: Visualizing a model basalt melt at 2200 K (left), and 3000 K (right). The Si/Al-O polyhedra and Ca/Mg-O spheres encode the respective coordination information with respect to oxygen. Also rendered are the trajectories (time encoded from green to red) of Ca and Mg atoms.

Visualization of the melt for high pressure condition of 30.4 GPa at 3000 K, shows high-order coordination polyhedras (blue, pink and white) representing Al/Si-O coordinations at 5-, 6-, 7-fold states and more confined Mg and Ca trajectories (Figure 3.15).

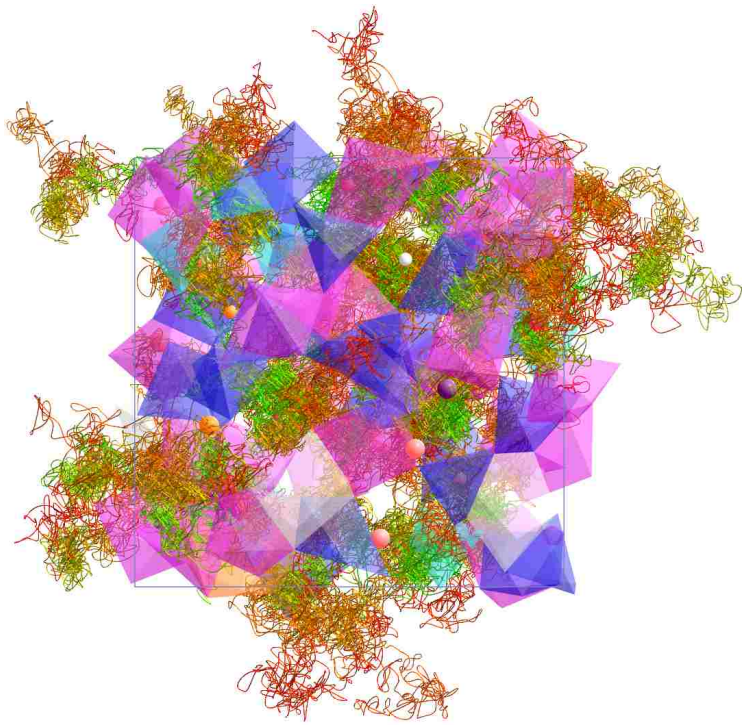


Figure 3.15: Visualizing a model basalt melt at 3000 K for high pressure of 30.4 GPa. The Si/Al-O polyhedra and Ca/Mg-O spheres encode the respective coordination information with respect to oxygen. Also rendered are the trajectories (time encoded from green to red) of Ca and Mg atoms.

Chapter 4
Fluid Dynamics Visualization : Evolving Time Surfaces

Computational Fluid Dynamics (CFD) is a computationally based design and analysis technique for the study of fluid flow. CFD can provide high fidelity temporally and spatially resolved numerical data that can range to several hundred thousand time steps and be of sizes in the order of terabytes. Therefore, a key challenge here is the ability to easily mine the time dependent CFD data, extract key features of the flow field, and display these spatially evolving features in the space-time domain of interest. We present an interdisciplinary effort to generate and visualize time surfaces of the fluid flow from the time dependent CFD data. The implementation of time surfaces, such as an evolving surface of a sphere, for analyzing the flow field is more relevant in context of a stirred tank system. The integration of surfaces over time generates an evolving surface that can illustrate key flow characteristics such as how matter injected in a stirred tank disperses, and in what regions of the tank is the turbulence high. Such observations are crucial to identifying the best conditions for optimal mixing.

The CFD dataset was obtained from a large eddy simulation (LES) of flow inside a stirred tank reactor. Stirred tanks are the most commonly used mixing device in chemical and processing industries. Improvements in the design of stirred tanks can translate into several billion dollar annual profit. However, better designs of stirred tanks require detailed understanding of flow and mixing behavior inside the tank. The curvilinear grid representing the tank geometry is distributed over 2088 blocks and comprised of 3.1 million cells in total. Flow variables like velocity and pressure are defined at the center of each cell and computed for each time step over a total of

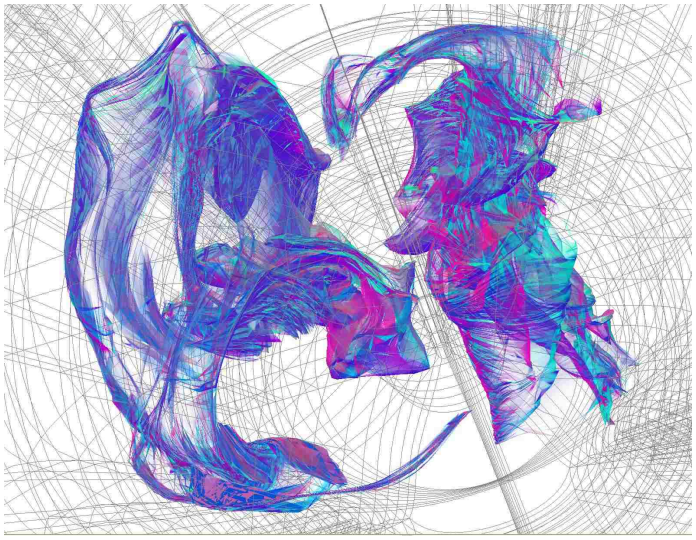


Figure 4.1: Two evolving spheres visualized just before their mixing in the stirred tank simulation system.

5700 time steps representing 25 complete rotations of the impeller. The handling and processing of these voluminous, multi-block, non-uniform curvilinear datasets to generate time surfaces and track a set of particles in the fluid flow is the main challenge.

4.1 Related Work

One of the earliest works related to this problem is the generation of stream surfaces, in particular Hultquist's attempt to generate a triangular mesh representation of stream surfaces. Hultquist introduced an algorithm that constructs stream surfaces by generating triangular tiles of adjacent streamlines or stream ribbons. In Hultquist's algorithm, tiling is done in a greedy fashion. When forming the next triangle, the shortest leading edge is selected out of the two possible trailing triangles and appended to the ribbon. Each ribbon forming the stream surface is advanced until it is of equivalent length to its neighboring ribbon along the curve they share [59]. Particles are added to the trail of the stream surface by splitting wide ribbons, and particles are removed from the stream surface by merging two narrow (and adjacent) ribbons into one. Note that Hultquist's algorithm was developed for steady flows. Also, advancing the front of the stream surface requires examining all the trailing ribbons.

Along the same lines, Schafhitzel et al. [60] adopted the Hultquist criteria to define when particles are removed or added, but they derived a point-based algorithm that is designed for GPU implementation. In addition to rendering a stream surface, they applied line integral convolution to show the flow field patterns along the surface.

Rather than remeshing a stream surface when the surface becomes highly distorted, von Funck et al [61] introduced a new representation of smoke in a flow as a semi transparent surface by adjusting opacity of triangles that get highly distorted and making them fade. Throughout the evolution of the smoke surface, they do not change the mesh, but rather use the optical model of smoke as smoke tends to fade in high divergent areas [61]. However, the authors report that this method does not work well if the seeding structure is a volume structure instead of a line structure.

4.2 Mathematical Background

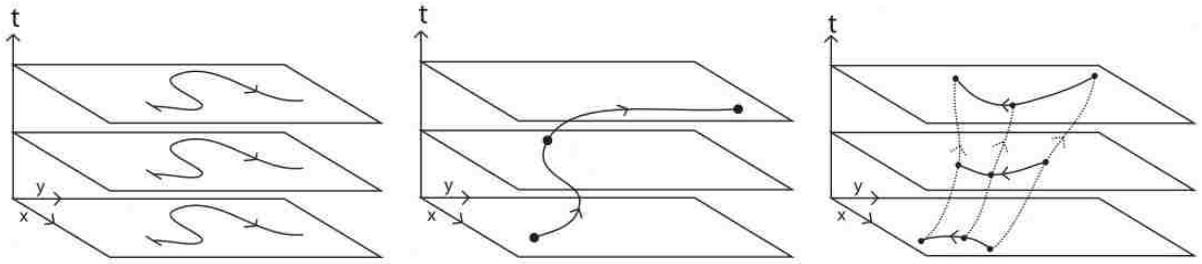
In the domain of computer graphics one distinguishes four categories of integration lines $q \subset M$ that can be computed from a time-dependent vector field $v \in \mathcal{T}(M)$, mathematically a section of the tangent bundle $T(M)$ on a manifold M describing spacetime: *path lines*, *stream lines*, *streak lines* and *material line*s. Each category represents a different aspect of the vector field:

Path lines (also called *trajectories*) follow the evolution of a test particle as it is dragged around by the vector field over time.

Stream lines (also called *field lines*) represent the instantaneous direction of the vector field. They are identical to path lines if the vector field is constant over time.

Streak lines represent the trace of repeatedly emitted particles from the same location, such as a trail of smoke.

Material lines (also called time lines) depict the location of a set of particles, initially positioned along a seed line, under the flow of the vector field.



(a) (b) (c)

Figure 4.2: Representation of different kinds of integral lines: (a) stream lines, (b) path lines, (c) time lines. The xy-plane is the manifold hosting the integral curves; time-evolution is shown in the t-axis direction [2].

Each of these lines comes with different characteristics: stream lines and path lines are integration lines that are tangential to the vector field at each point

$$\dot{q} \equiv \frac{d}{ds} q(s) = v(q(s)) \tag{4.1}$$

Since the differential equation is of first order, the solution can be determined by specifying the initial condition $q(0) = q_0$ by a seed point $q_0 \in M$ in space-time. In contrast to stream- and path lines, streak- and material lines are one-dimensional cuts of two-dimensional integration surfaces $S \subset M$, $dim(S) = 2$. The surface is constructed from all integral lines that pass through an event on the initial seed line $q_0(\tau)$:

$$S = \{q : \mathbb{R} \to M, \dot{q}(s) = v(q(s)), q(0) = q_0(\tau)\} \tag{4.2}$$

The resulting surface contains a natural parametrization $S(s, \tau)$ by the initial seed parameter τ and the integration parameter s. If the integration parameter is chosen to be proportional to the time $s \propto t$, for instance when performing Euler steps, then the original seed line parameter τ provides a natural parameter for the resulting lines, i.e. each point along a time line is advanced by the same time difference dt.

Refinement of lines by introducing new integration points is mandatory to sustain numerical accuracy of the results. The ideas of the Hultquist algorithm [62] and its improvements by Stalling [63] could also be applied to the spatio-temporal case; however, such would result in the requirement to perform time-like interpolation of the vector field.

A time surface (Figure 4.3) is the two-dimensional generalization of a time line, a volumetric object in space-time. The Hultquist algorithm, if applied to a spatio-temporal surface, discusses criteria on refining one edge, whereas we have a much richer set of possible surface characteristics that may trigger creation or deletion of integration points. Some options are to refine a surface at locations where a triangle's edge, area, curvature, or triangle degeneration (stretching) becomes larger than a certain threshold. Section 4.5.1 reviews our results experimenting with different such criteria.

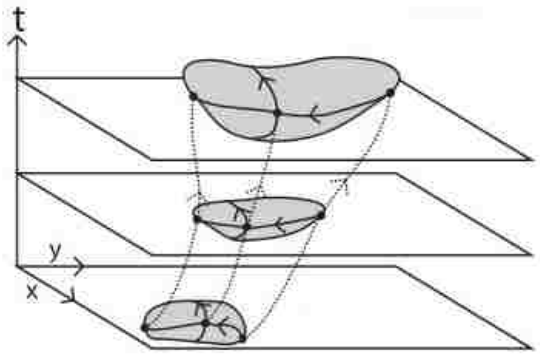


Figure 4.3: Evolution of time surface in space-time manifold [2].

4.3 Desing and Implementation

4.3.1 Data Model

We used VISH [64] visualization shell as our implementation platform. It supports the concept of fiber bundles [65] for the data model. The data model consists of seven levels, each of which is comprised of compatible arrays that represent a certain property of the dataset [66]. These levels, which constitute a Bundle, are Slice, Grid, Skeleton, Representation, Field, Fragment and

Compound. The Field represents arrays of primitive data types, such as int, double, bool, etc., and the collection of Fields describes the entire Grid. The Grid objects for different time slices are bundled together and are represented as a Bundle. As an example of our implementation, each Field contains values of a property such as coordinates, connectivity information, velocity, etc. The collection of these Fields is a Grid object, and the collection of Grid objects for all time slices is the Bundle of the entire dataset.

The dataset used for visualizing the features of fluid flow contains numerical data for 2088 curvilinear blocks constituting the virtual stirred tank. The input vector field is fragmented and these fragments are the blocks of the curvilinear grid. The input dataset for each time slice consists of coordinate location, pressure and fluid velocity for each grid point in the entire 2088 blocks. These properties are stored as Fields in the Grid object for each time slice, and these Grid objects are then combined into a Bundle.

4.3.2 Out-of-Core Memory Management

The original approach taken while visualizing the features of fluid flow is to keep the entire vector field data in the main memory and integrate over the vector field to extract the features. However, with the necessity of visualizing the time-dependent 3D vector field, the original approach has restrictions such as the size of the time-dependent data can easily exceed the memory capacity of even the state of the art workstations. In [25], the authors present the concept of an out-of-core data handling strategy to process the large time dependent dataset by only loading parts of the dataset and processing it. Two major strategies presented for out-of-core data handling are block-wise random access and slice-wise sequential access. The authors emphasize the slice-wise sequential access strategy for handling the data in time slices; however, we have implemented both block-wise access and slice-wise access of time-dependent data while generating the time surfaces for visualizing the fluid flow.

The virtual stirred tank system has 2088 blocks, and each block has vector field data for every time slice. The data for each time slice is accessed only once as a Grid object from the input

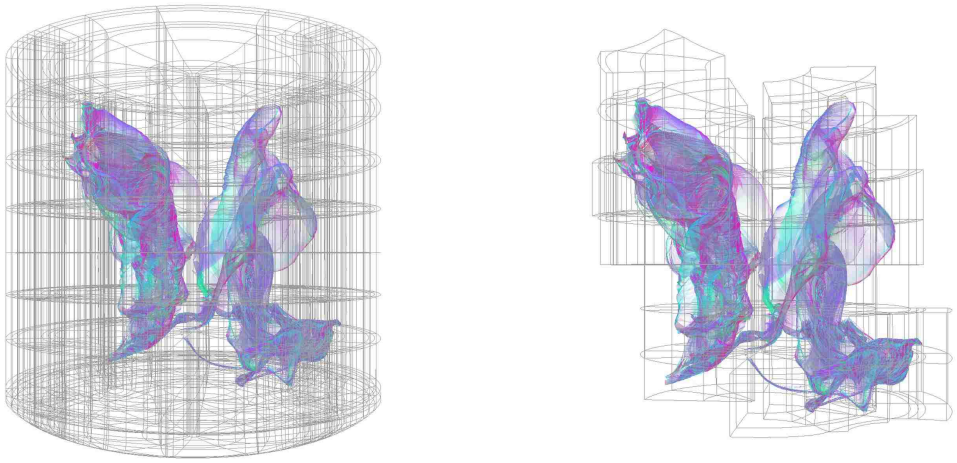


Figure 4.4: Time surface computed from a vector field given in 2088 fragments (curvilinear blocks) covering the Stirred Tank Grid (left). Only those fragments that affect the evolution of the time surface (right) are actually loaded into memory.

Bundle and processed to generate the time surface at that particular time. The integration of the time surfaces do not process all blocks but instead only the blocks that are touched at the given time slice are loaded and processed.

At every time slice two Grid objects are handled, one containing the input data of the vector field and the other containing seed points and connectivity information among these points. The connectivity information is used to generate the triangle mesh for surface generation. In case of no surface refinement, the connectivity information is constant throughout the time slices and is stored once and used multiple times. This conserves the memory and also reduces the memory access. However, with surface refinement the number of points and their connectivity changes over time resulting in an increase in memory usage.

4.3.3 Particle Seeding and Advection

Our set of particle seeds q_{i,t_0} for $i = 0, ..., n-1$, lie on a sphere. At any given time $t > t_0$, the time surface is represented as a triangular mesh formed by the particles $q_{i,t}$ that have been advected using equation 4.1. Figure 4.5 illustrates the difference between our seeding approach

versus Hultquist's where we are evolving a surface element (a triangle) over time as opposed to spanning a surface out of a line segment element.

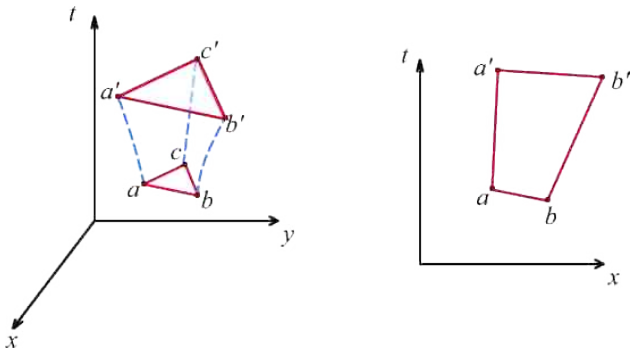


Figure 4.5: Particle advection of a 2-dimensional element vs. a 1-dimensional element. In our case, the surface element is in 3-dimensional space spanned over time.

4.3.4 Triangular Mesh Refinements

As time elapses, the triangular mesh of particles enlarges and twists according to the flow field. To preserve the quality of the mesh, we refine it by adding new particles and advecting them while updating the mesh connectivity. Of the possible refinements criteria mentioned above, we have implemented the following:

Edge length: If the distance between pairwise particles of a triangle is larger than a threshold edge length, we insert a new midpoint and subdivide the triangle accordingly.

Triangle area: If the area of the triangle formed by the new positions of the particle triplet is larger than a threshold area, we insert three midpoints and subdivide the triangle to a new set of four triangles.

4.4 Time-Curvature and Time-Torsion as Fluid Mixing Indicators

When two mutually soluble liquids are mixed together two things happen. First, the liquids are broken up into intermingled clumps. The shapes of these clumps vary depending upon the mixing process. But the average size of the clumps decreases up to a certain range as mixing is continued.

This is the macro aspect of mixing which is mainly driven by the mechanical forces. The second aspect is micro-mixing, i.e. molecular inter-diffusion of the fluids across the boundary of their clumps. This process is spontaneous and continues even after the mechanical mixing is stopped and eventually the mixture of two soluble liquids becomes completely uniform after a substantial amount of time. However, this molecular mixing is also restricted by the macro mixing process as the break-up of the clump, rate of generation of the surface area of each clump and the topological qualities of the surfaces govern the rate and process of inter-diffusion of the liquids [67]. In order to develop an understanding of the mechanical macro mixing process, we investigate motion of the fluid particles in the flow domain.

4.4.1 Curvature and Torsion Measures of Flow Field

The flow domain considered is a mechanically agitated turbulent Stirred tank. Water is considered as the working fluid in which a blob of a tracer is injected at a particular location. We look into the pathlines of a number of points associated with the injected tracer. These pathlines show exactly how tracer particles move inside the stirred tank volume. A good macro-mixing will show the entire volume being filled-up by the pathlines. From the pathline visualizations, we try to qualitatively understand the stretching and distortions on the fluid elements by calculating curvature at different points on the pathline (Figure 4.6). A higher value of the curvature indicates straining of the fluid element associated with the pathline and also higher residence time of the fluid particle in that particular region resulting into prolonged inter-diffusion of the fluids promoting micro level mixing [68]. Therefore, the curvatures of the pathlines are of prime interest in order to comment on the potential of better mixing at different locations of the tank over the computed time-interval.

Another interesting geometric property of the pathline is its torsion which basically measures its deviation from the osculating plane (Figure 4.7). Torsions show the three-dimensionality of the motion of the fluid particles and indicate on the mixing process away from its plane of motion. Torsion is also a measure of twisting strains on the fluid elements.

Curvature

In differential geometry, a curve q is a function that maps a real-valued parameter s to a point $q(s)$ within a differential manifold M. Variation of the curve parameter s defines the tangential vector $\dot{q} = \mathrm{d}q/\mathrm{d}s$ along the curve. The second derivative $\ddot{q} = \mathrm{d}^2q/\mathrm{d}s^2$ determines the radius of curvature of the curve, usually expressed via the curvature κ parameter given by

$$\kappa = \frac{|\ddot{q} \times \dot{q}|}{|\dot{q}|^3} \tag{4.3}$$

Torsion

The third derivative $\dddot{q} = \mathrm{d}^3q/\mathrm{d}s^3$ is expressed by the torsion parameter, given by

$$\tau = \frac{\dddot{q} \cdot (\ddot{q} \times \dot{q})}{|\ddot{q} \times \dot{q}|^2} \tag{4.4}$$

It describes the deviation of the curve from the plane defined by its tangential and acceleration vector.

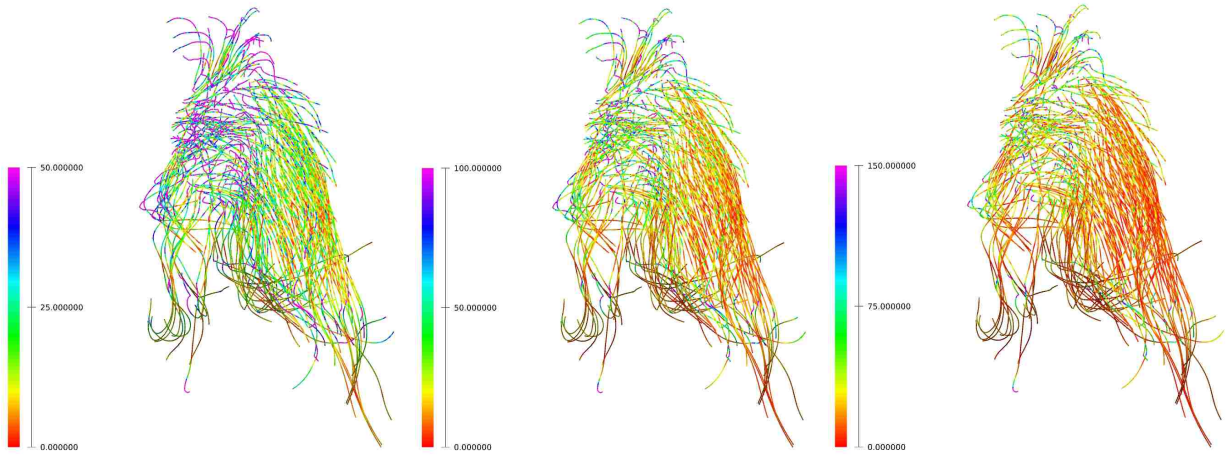


Figure 4.6: Color coded representation of curvature measure on pathlines for 200 time steps. The maximum curvature value represented is 50, 100, and 150, respectively from left to right image.

Both curvature and torsion can be computed along streamlines and pathlines of a vector field v where the tangential vector is identical to the vector field $v(q(s)) = \dot{q}(s)$. These scalar measures

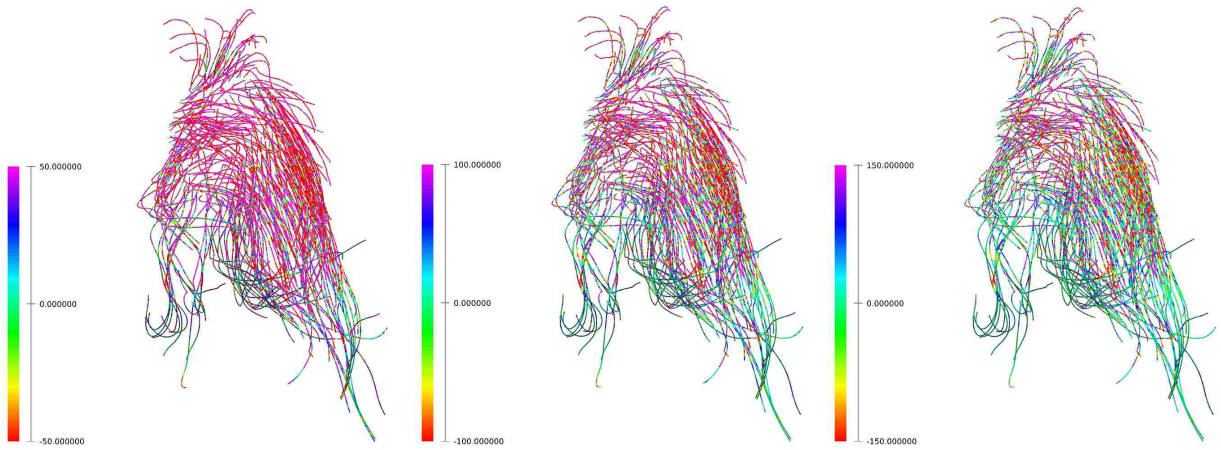


Figure 4.7: Color coded representation of torsion measure on pathlines for 200 time steps. The range of torsion value represented is [-50, 50], [-100, 100], and [-150, 150], respectively from left to right image.

are thus indicators of the first and second derivatives of the vector field itself. Given the first and second order derivatives, it is also possible to directly compute the curvature and torsion field from a vector field [69].

However, since the first derivative of the vector field requires three times as much storage space in memory as the original vector field and its derivatives nine times as much, it is unrealistic to output these quantities by the simulation code itself, as the vector field itself already occupies 500GB. On the other hand, computation on real time is hindered by the complexity of the dataset that is constructed by curvilinear coordinates distributed across multiple blocks with non-regular topology at their boundaries. Furthermore, the curvature and torsion are quantities referring to each time-step; whereas, we are interested in quantities describing the time-dependency of the vector field. Thus in our approach, we first concentrate on computing pathlines from a vector field, based on user-specified initially defined seeding points, and then computing curvature and torsion quantities along this subset of the vector field as a post-processing step on the pathlines.

A better understanding on scalar mixing can be accomplished by studying the area properties [67] of the tracer blob volume, so the specific rate of generation of the area can give a measure of mixing efficiency. Also different topological properties of the surface area serve as a metric of

mixing inside the vessel, such as curvature and local undulations and depressions of the surface area can be used to identify the roll-up or spread-out of the unmixed fluid sheet. Moreover, curved area can form horseshoe maps resulting into fluid exchange and mixing of one fluid into another. The initial tracer volume breaks up into clumps of smaller volume and disperses through the entire fluid domain. The break-up of the blobs assume an extremely important role in the overall mixing process by diminishing sizes of the unmixed tracers, increasing rate of mixing and providing higher area for molecular mixing. However, the point of break-up of a tracer blob must be a critical point in the flow domain. Hence, a visualization of the flow topology (not the area topology discussed earlier) via velocity vector field can be utilized to identify the regions at which the tracer clumps break down. Hence, the flow feature extraction strategies can be combined with the Lagrangian mixing study to obtain a more vivid description of the mixing process.

4.5 Results and Analysis

4.5.1 Surface Refinement

We benchmarked our implementation with a 30 time-steps subset (85 MB per timestep) of the stirred tank data and on a 64-bit dual core (2GHz each) Pentium laptop machine with 4GB of memory. We advected one sphere for the first 30 time-steps of the simulation. Due to the small size of our test data, we could not notice a difference in time surface meshing quality from the visualization itself, but from the data in Tables 4.1 and 4.2, we notice a slight performance improvement of the area criteria over the edge length criteria. Though the number of particles is slightly higher in the second case, this suggests that the quality of the surface with the area criterion is better.

Table 4.1: Timing Analysis (in seconds) for the Edge Length Criteria

threshold	tot points	avg time/slice	tot time
0.005	4269	6.480	200.868
0.01	822	1.519	47.1
0.02	258	1.165	36.101

Table 4.2: Timing Analysis (in seconds) for the Triangle Area Criteria

threshold	tot points	avg time/slice	tot time
0.005	4269	6.864	212.785
0.01	837	1.454	45.08
0.02	258	1.150	35.646

From either Tables 4.1 or 4.2, picking a threshold too small compared to the characteristic of the triangle being examined results in maximum refinement, while a large enough threshold leads to no refinement at all.

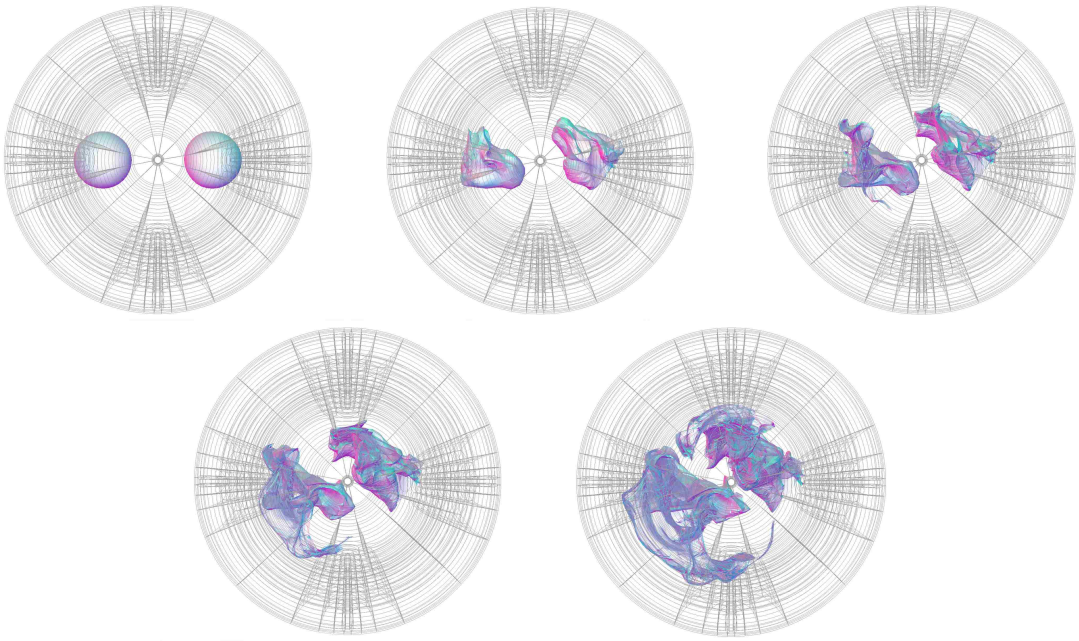


Figure 4.8: Images showing evolution of two spheres at time slices 0, 50, 100, 125 and 150, respectively from left-top to bottom, as seen top-view of the stirred tank. First image shows the seed spheres, and the last image shows two sphere just before the surfaces are about to mix.

4.5.2 Timing Analysis

For the overall integration and refinement of the time surface, we used a larger dataset of size 12GB with 150 time-steps. We ran the implementation on a 64 bit quad-core workstation with 64 GB of memory. We used the edge length criterion with a threshold of 0.01.

Table 4.3: Timing Analysis for Threshold=0.01

time	no. of points	time/slice(s)	time/point(ms)
0	516	0.4	7.0
50	3468	2.0	5.9
100	15822	7.4	4.8
125	41574	18.8	4.7
150	129939	49.7	4.0

Figure 4.9: Left and middle graphs show the increase in no. of points and thus increase in processing time per slice over the time. Right graph shows the decrease in time per point as number of point increases.

The listing in the Table 4.3 is for 12 GB of input data from an first time-step to a final time-step of 150. Initially the number of points is 516, which increases over time as more points are generated for surface refinement. As the number of points increases, the computation time for the next time slice increases. However, the time per point seems to be slowly decreasing, as seen in third graph of Figure 4.9. This may be because more and more points tend to locate in the same block and the data of one block is shared by many points, resulting in less memory access per point.

4.5.3 Fluid Mixing Indicators

Curvature is computed by accessing the vector fields at the previous and the current time slices, hence it is computed as the first order derivative of the vector field. Likewise, torsion is computed as the second order derivative of vector field over time. Torsion values along the pathline is computed

by accessing vector field for one more time slice than the curvature measure. Once the vector field is interpolated at each integration point, the curvature and torsion measures for pathline are computed from the interpolated vector field along the pathline.

Since the computation for large time-dependent data will be computationally (time and resource) intensive, we only process the reduced set of the vector field data for the computation of curvatures along the pathline. Curvature of a line in space indicates a measure of change in direction of a particle moving along the line and as the curvature increases the particle traverses more in space as it moves between two points along its trajectory. The pathlines map the trajectories of tracer particles, and the curvatures of these pathlines give estimate of interaction of tracer particles with its surrounding fluid. For a pathline with a higher level of curvature at a particular location of the stirred tank, the corresponding particle stays a longer time and interacts with a higher amount of surrounding fluid implying high mixing at that region of the tank. Figure 4.6 shows individual pathlines and their curvatures over the flow field. After being seeded on the surface of a small circular blob, the tracer particles encompass a substantial area of the tank allowing these particles to mix with neighboring fluid. We can see that in a range of 0-100, most of the pathlines show a below 50 curvature value; whereas, the curvature is as high as 100 at some locations, indicating that better mixing has been obtained as the tracer particles resided there. However, the pathlines show map of the particle movement both in time and space, so the high curvature values (i.e., better local mixing) may occur at different locations and disjoint time instants. In order to realize the mixing behavior in time and space, the time surface of a group of particles is drawn for different time instants. These time surfaces show the growth of the injected tracer blob inside the tank over a time period. In addition, the time surfaces are contoured with the curvature levels of the pathlines passing through it. Figure 4.10 shows such a contoured time surface for two different ranges of curvature measure. For left image with curvature range 0-100, at a particular instant, the pathlines in the lower portion of the time-surface show higher curvature indicating that particles residing at those locations are better mixing with the surrounding fluid.

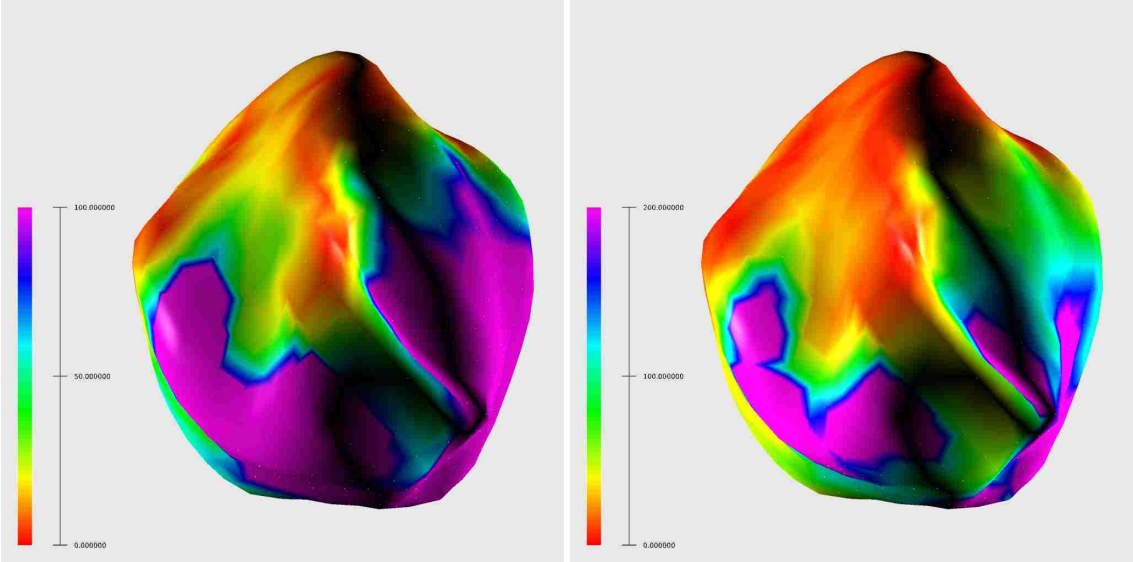


Figure 4.10: Curvature measures along pathlines is color coded on the time surfaces. Curvature values are of points on pathlines at time step 25, and are in range of [0-100] (left) and [0-200] (right).

4.6 Conclusion

While most of the previous visualization techniques for fluid flow have concentrated on flow streamlines and pathlines, our approach has been directed towards generating the time surfaces of the flow. The interdependencies of integration over a vector field require random access to amounts of data beyond a single workstation's capabilities, while at the same time requiring shared memory for required refinements. The evolution of a seed surface required refinement of its corresponding triangular mesh to preserve the quality of the time surface over time. From the results we noticed a slight superior quality of the area refinement criterion over the edge length criterion.

Another approach for flow visualization is concentrated in computing pathlines from a vector field, based on user-specified initially defined seeding points, and then on computing curvature and torsion measures along the pathlines (subset of the vector field). The computed curve measures are mapped onto the time surfaces generated over the time dependent vector field. The curvatures of the pathlines are of prime interest in order to comment on the potential of better mixing at different locations of the tank over the computed time-interval. Other interesting geometric property of the

pathline is its torsion which basically measures its deviation from the osculating plane. However, the measure of curvature and torsion values of a vector field highly depend upon the seeding position and geometry, as the seed spheres at different places result in different curvature measures for the same time slice. Our approach is not a comprehensive method to find all features and measures of total mixing, but to give the indicative behavior of the fluid mixing. This may result in some missing of key features of flow during the visualization and analysis, as it mostly depends upon the seeding geometry and position.

Chapter 5
Conclusions and Future Works

We present the visualization of spatially evolving features of a flow field in the stirred tank as evolving surfaces, called time surfaces of the flow. Another approach for flow visualization is concentrated in generating pathlines from the vector field of the flow and evaluating the torsion and curvature along pathlines as a measure of fluid mixing. However, the measure of curvature and torsion depends upon the seeding location and geometry for computing pathlines, thus may not be the comprehensive method to measure total mixing of fluids.

A meaningful representation of spatio-temporal behaviour of atomic systems is a tremendous challenge in visualization of molecular dynamics. Of particular interest is an approach to render atomic trajectories (that provide a complete representation of spatial and temporal information) and encode useful information along them. The associated cluttering/occlusion limits the usefulness of these trajectories. Primarily based on the idea of constraining the rendering of trajectories in time and space, we have presented various approaches to reduce the number of rendered line segments which constitute the trajectories. We have demonstrated the effectiveness of our approaches by rendering atomic trajectories for simulated hydrous silicate and silica liquid data. To reduce the clutter caused by the crowded trajectories, we perform adaptive hierarchical merging of multiple positions along the trajectories. According to our approach, the multiple positions to be merged at each level are picked up using a proximity window, which is defined in terms of space window (distance cutoff). Our analysis shows that the number of effective positions needed to render the trajectories decreases dramatically under merging (with/without information constraint), and the processed (reduced) trajectories show significantly reduced clutter. We could further enhance the

visualization process by encoding additional information (time, 3D position, coordination number, deviation, and merge count) along the trajectories. Improved trajectories allow us to better assess the nature and extent of the corresponding atomic movements. In particular, they suggest that atoms move via discrete jumps (hopping-like motion) in addition to continuous forward motion. More importantly, the underlying atomic structures become visible with all trajectories rendered. Furthermore, we examined the structural behavior of atomic systems by evaluating and rendering radial distribution function, coordination environment, and coordination clusters of the simulated atomic system.

While moderate-size data sets containing several millions of data points (atomic positions) were considered in this study, we anticipate to extend the proposed position merging to larger data sets produced by large-scale molecular dynamics simulations. Making parallel the compute-intensive analysis algorithms, such as calculation of RDF, computing coordination stability, bond life, adaptive hierarchical scheme for merging,etc., to scale across heterogeneous computing resources is an effective way to deal with the large scale data sets. However, a challenge is to implement data parallelism (or task parallelism) across multiple compute nodes. One objective of parallelization can be minimizing the computation time of an algorithm so as to keep the visualization system interactive. In addition, we plan on integrating data mining with visualization application, so that a subset of data identified by data mining analysis can be further studied by detailed visualization-based analysis. Our goal is to make the application scalable, efficient and effective for the domain-specific visualization of MD data.

Bibliography

[1] Sébastien Le Roux and Valeri Petkov. ISAACS – interactive structure analysis of amorphous and crystalline systems. *Journal of Applied Crystallography*, 43(1):181–185, Feb 2010. doi: 10.1107/S0021889809051929. URL http://dx.doi.org/10.1107/S0021889809051929.

[2] Marcel Ritter, Bidur Bohara, and Werner Benger. A framework for computing integral geometries in vish using template meta programming. *In 6th High-End Visualization Workshop*, December 2010.

[3] B. H. McCormick. Visualization in scientific computing. *SIGBIO Newsl.*, 10(1):15–21, March 1988. ISSN 0163-5697. doi: 10.1145/43965.43966. URL http://doi.acm.org/10.1145/43965.43966.

[4] Theresa-Marie Rhyne and Min Chen. Cutting-edge research in visualization. *Computer*, 46(5):22–24, 2013. ISSN 0018-9162. doi: http://doi.ieeecomputersociety.org/10.1109/MC.2013.166.

[5] D. Bhattarai and B.B. Karki. Atomistic visualization: Space-time multiresolution integration of data analysis and rendering. . *Journal of Molecular Graphics and Modelling*, 27:951:33–38, 2009.

[6] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, USA, June 1989.

[7] Liu Peng, Manaschai Kunaseth, Hikmet Dursun, Ken-ichi Nomura, Weiqiang Wang, RajivK. Kalia, Aiichiro Nakano, and Priya Vashishta. Exploiting hierarchical parallelisms for molecular dynamics simulation on multicore clusters. *The Journal of Supercomputing*, 57(1):20–33, 2011. URL http://dx.doi.org/10.1007/s11227-011-0560-1.

[8] Ju Li. *Atomistic Visualization*, pages 1051–1068. Springer Netherlands, 2005. URL http://dx.doi.org/10.1007/978-1-4020-3286-8_52.

[9] W. Humphrey, A. Dalke, and K. Schulten. VMD: Visual molecular dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[10] Sebastian Grottel, Guido Reina, Carsten Dachsbacher, and Thomas Ertl. Coherent culling and shading for large molecular dynamics visualization. In *Proceedings of the 12th Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'10, pages 953–962, Aire-la-Ville,

Switzerland, Switzerland, 2010. Eurographics Association. doi: 10.1111/j.1467-8659.2009.01698.x. URL http://dx.doi.org/10.1111/j.1467-8659.2009.01698.x.

[11] John E. Stone, Kirby L. Vandivort, and Klaus Schulten. Immersive out-of-core visualization of large-size and long-timescale molecular dynamics trajectories. In *Proceedings of the 7th International Conference on Advances in Visual Computing - Volume Part II*, ISVC'11, pages 1–12, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-24030-0. URL http://dl.acm.org/citation.cfm?id=2045195.2045197.

[12] Cheng Zhang, Scott Callaghan, Thomas Jordan, Rajiv K. Kalia, Aiichiro Nakano, and Priya Vashishta. Paraviz: A spatially decomposed parallel visualization algorithm using hierarchical visibility ordering, 2007.

[13] P. J. Kraulis. *MOLSCRIPT*: a program to produce both detailed and schematic plots of protein structures. *Journal of Applied Crystallography*, 24(5):946–950, Oct 1991. doi: 10.1107/S0021889891004399. URL http://dx.doi.org/10.1107/S0021889891004399.

[14] Anton Kokalj. Xcrysden—a new program for displaying crystalline structures and electron densities. *Journal of Molecular Graphics and Modelling*, 17(3–4):176 – 179, 1999. ISSN 1093-3263. doi: http://dx.doi.org/10.1016/S1093-3263(99)00028-5. URL http://www.sciencedirect.com/science/article/pii/S1093326399000285.

[15] Ju Li. Atomeye: an efficient atomistic configuration viewer. *Modelling and Simulation in Materials Science and Engineering*, 11(2):173, 2003. URL http://stacks.iop.org/0965-0393/11/i=2/a=305.

[16] Kevin Buchin, Urska Demsar, Aidan Slingsby, and Erik P. Willems. Results of the breakout group: Visualisation. In *Representation, Analysis and Visualization of Moving Objects*, Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2011.

[17] N. Adrienko and G. Adrienko. Spatial generalization and aggregation of massive movement data. *Visualization and Computer Graphics, IEEE Transactions on*, 17(2):205–219, 2011. ISSN 1077-2626. doi: 10.1109/TVCG.2010.44.

[18] O.M. Borcan. Improving visualization of trajectories by dataset reduction and line simplification. Master's thesis, Utrecht University, 2012.

[19] Katrin Bidmon, Sebastian Grottel, Fabian Bös, Jürgen Pleiss, and Thomas Ertl. Visual abstractions of solvent pathlines near protein cavities. *Computer Graphics Forum*, 27 (3):935–942, 2008. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2008.01227.x. URL http://dx.doi.org/10.1111/j.1467-8659.2008.01227.x.

[20] Kuangyu Shi, Holger Theisel, Helwig Hauser, Tino Weinkauf, Kresimir Matkovic, Hans-Christian Hege, and Hans-Peter Seidel. *Path Line Attributes - an Information Visualization Approach to Analyzing the Dynamic Behavior of 3D Time-Dependent Flow Fields*, pages 75–88. Springer Berlin Heidelberg, 2009. URL http://dx.doi.org/10.1007/978-3-540-88606-8_6.

[21] M. Zockler, D. Stalling, and H.-C. Hege. Interactive visualization of 3d-vector fields using illuminated stream lines. In *Visualization '96. Proceedings.*, pages 107–113, 1996. doi: 10.1109/VISUAL.1996.567777.

[22] Marc Schirski, Torsten Kuhlen, Martin Hopp, Philipp Adomeit, Stefan Pischinger, and Christian Bischof. Efficient visualization of large amounts of particle trajectories in virtual environments using virtual tubelets. In *Proceedings of the 2004 ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry*, VRCAI '04, pages 141–147, New York, NY, USA, 2004. ACM. ISBN 1-58113-884-9. doi: 10.1145/1044588.1044615. URL http://doi.acm.org/10.1145/1044588.1044615.

[23] Chad Jones, Kwan liu Ma, Stephane Ethier, and Wei li Lee. An integrated visual exploration approach to particle data analysis. Technical report, University of California at Davis, 2007.

[24] Frits H Post, Benjamin Vrolijk, Helwig Hauser, Robert S Laramee, and Helmut Doleisch. Feature extraction and visualization of flow fields. *Eurographics 2002 State-of-the-Art Reports*, 1:69–100, 2002.

[25] T. Weinkauf, H. Theisel, H.-C. Hege, and H.-P. Seidel. Feature flow fields in out-of-core settings. In H. Hauser, H. Hagen, and H. Theisel, editors, *Topology-based Methods in Visualization*, Mathematics and Visualization, pages 51–64. Springer, 2007. ISBN 978-3-540-70822-3. URL http://tinoweinkauf.net/. Topo-In-Vis 2005, Budmerice, Slovakia, Sept. 29 - 30.

[26] Bidur Bohara, Farid Harhad, Werner Benger, Nathan Brener, Sitharama Iyengar, Marcel Ritter, Kexi Liu, Brygg Ullmer, Nikhil Shetty, Vignesh Natesan, Carolina Cruz-Neira, Sumanta Acharya, Somnath Roy, and Bijaya Karki. Evolving time surfaces in a virtual stirred tank. In Vaclav Skala, editor, *18th International Conference on Computer Graphics, Visualization and Computer Vision'2010*, 2010.

[27] J. Chandler, H. Obermaier, and K.I. Joy. Interpolation-based pathline tracing in particle-based flow visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 21 (1):68–80, Jan 2015. ISSN 1077-2626. doi: 10.1109/TVCG.2014.2325043.

[28] Aiichiro Nakano, Rajiv K. Kalia, Ken ichi Nomura, Ashish Sharma, Priya Vashishta, Fuyuki Shimojo, Adri C.T. van Duin, William A. Goddard, Rupak Biswas, and Deepak Srivastava. A divide-and-conquer/cellular-decomposition framework for million-to-billion atom simulations of chemical reactions. *Computational Materials Science*, 38(4):642 – 652, 2007. ISSN 0927-0256. doi: http://dx.doi.org/10.1016/j.commatsci.2006.04.012. URL http://www.sciencedirect.com/science/article/pii/S0927025606001054.

[29] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, New York, NY, USA, 1996. ISBN 0521445612.

[30] A. Rahman. Correlations in the motion of atoms in liquid argon. *Phys. Rev.*, 136:A405–A411, Oct 1964. doi: 10.1103/PhysRev.136.A405. URL http://link.aps.org/doi/10.1103/PhysRev.136.A405.

[31] Christian Sigg, Tim Weyrich, Mario Botsch, and Markus Gross. Gpu-based ray-casting of quadratic surfaces. In *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, SPBG'06, pages 59–65, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association. ISBN 3-905673-32-0. doi: 10.2312/SPBG/SPBG06/059-065. URL http://dx.doi.org/10.2312/SPBG/SPBG06/059-065.

[32] William J. Gordon and Richard F. Riesenfeld. Bernstein-bézier methods for the computer-aided design of free-form curves and surfaces. *J. ACM*, 21(2):293–310, April 1974. ISSN 0004-5411. doi: 10.1145/321812.321824. URL http://doi.acm.org/10.1145/321812.321824.

[33] Sung Keun Lee, Jung-Fu Lin, Yong Q. Cai, Nozomu Hiraoka, Peter J. Eng, Takuo Okuchi, Ho-kwang Mao, Yue Meng, Michael Y. Hu, Paul Chow, Jinfu Shu, Baosheng Li, Hiroshi Fukui, Bum Han Lee, Hyun Na Kim, and Choong-Shik Yoo. X-ray raman scattering study of mgsio3 glass at high pressure: Implication for triclustered mgsio3 melt in earth's mantle. *Proceedings of the National Academy of Sciences*, 105(23):7925–7929, 2008. doi: 10.1073/pnas.0802667105. URL http://www.pnas.org/content/105/23/7925.abstract.

[34] Jonathan F. Stebbins. Nmr evidence for five-coordinated silicon in a silicate glass at atmospheric pressure. *Nature*, 351(6328):638–639, 06 1991. URL http://dx.doi.org/10.1038/351638a0.

[35] D.L. Farber and Q. Williams. An in-situ raman spectraoscopic study of na2si2o5 at high-pressures and temperatures-structures of compressed liquids and glasses. *The American Mineralogist*, 81(3-4):273–283, 1996.

[36] Michael J. Toplis, Simon C. Kohn, Mark E. Smith, and Iain J.F. Poplett. Fivefold-coordinated aluminum in tectosilicate glasses observed by triple quantum mas nmr. *American Mineralogist*, 85(10):1556–1560, 2000. URL http://ammin.geoscienceworld.org/content/85/10/1556.abstract.

[37] Nobumasa Funamori, Shino Yamamoto, Takehiko Yagi, and Takumi Kikegawa. Exploratory studies of silicate melt structure at high pressures and temperatures by in situ x-ray diffraction. *Journal of Geophysical Research: Solid Earth*, 109(B3):B03203, 2004. doi: 10.1029/2003JB002650. URL http://dx.doi.org/10.1029/2003JB002650.

[38] Akihiro Yamada, Toru Inoue, Satoru Urakawa, Ken-ichi Funakoshi, Nobumasa Funamori, Takumi Kikegawa, Hiroaki Ohfuji, and Tetsuo Irifune. In situ x-ray experiment on the structure of hydrous mg-silicate melt under high pressure and high temperature. *Geophysical Research Letters*, 34(10):L10303, 2007. doi: 10.1029/2006GL028823. URL http://dx.doi.org/10.1029/2006GL028823.

[39] J.D. Kubicki and A.C. Lasaga. Molecular dynamics simulations of pressure and temperature effects on mgsio3 and mg2sio4 melts and glasses. *Physics and Chemistry of Minerals*, 17(8):661–673, 1991. URL http://dx.doi.org/10.1007/BF00202236.

[40] James R. Rustad, David A. Yuen, and Frank J. Spera. Molecular dynamics of liquid sio_2 under high pressure. *Phys. Rev. A*, 42:2081–2089, Aug 1990. doi: 10.1103/PhysRevA.42.2081. URL http://link.aps.org/doi/10.1103/PhysRevA.42.2081.

[41] Lars Stixrude and Bijaya Karki. Structure and freezing of mgsio3 liquid in earth's lower mantle. *Science*, 310(5746):297–299, 2005. doi: 10.1126/science.1116952. URL http://www.sciencemag.org/content/310/5746/297.abstract.

[42] Mainak Mookherjee, Lars Stixrude, and Bijaya Karki. Hydrous silicate melt at high pressure. *Nature*, 452(7190):983–986, 04 2008. URL http://dx.doi.org/10.1038/nature06918.

[43] Rodolphe Vuilleumier, Nicolas Sator, and Bertrand Guillot. Computer modeling of natural silicate melts: What can we learn from ab initio simulations. *Geochimica et Cosmochimica Acta*, 73(20):6313 – 6339, 2009. ISSN 0016-7037. doi: http://dx.doi.org/10.1016/j.gca.2009.07.013. URL http://www.sciencedirect.com/science/article/pii/S0016703709004530.

[44] Frank J. Spera, Dean Nevins, Mark Ghiorso, and Ian Cutler. Structure, thermodynamic and transport properties of caal2si2o8 liquid. part i: Molecular dynamics simulations. *Geochimica et Cosmochimica Acta*, 73(22):6918 – 6936, 2009. ISSN 0016-7037. doi: http://dx.doi.org/10.1016/j.gca.2009.08.011. URL http://www.sciencedirect.com/science/article/pii/S0016703709005237.

[45] BijayaB. Karki, Dipesh Bhattarai, Mainak Mookherjee, and Lars Stixrude. Visualization-based analysis of structural and dynamical properties of simulated hydrous silicate melt. *Physics and Chemistry of Minerals*, 37(2):103–117, 2010.

[46] Nevins D. and Spera F. Molecular dynamics simulations of molten caal2si2o8: Dependence of structure and properties on pressure. *American Mineralogist*, 83:1220–1230, 1998.

[47] Andrea Trave, Paul Tangney, Sandro Scandolo, Alfredo Pasquarello, and Roberto Car. Pressure-induced structural changes in liquid Sio_2 from ab initio simulations. *Phys. Rev. Lett.*, 89:245504, Nov 2002. doi: 10.1103/PhysRevLett.89.245504. URL http://link.aps.org/doi/10.1103/PhysRevLett.89.245504.

[48] B. Bohara and B.B. Karki. Rendering particle trajectories with color-coded information for atomistic visualization. In *Proceedings of the IASTED International Conference on Visualization, Image and Image Processing (VIIP 2012)*, pages 35–42, 2012.

[49] G. Kresse and J. Furthmüller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Computational Materials Science*, 6(1):15 – 50, 1996. ISSN 0927-0256. doi: http://dx.doi.org/10.1016/0927-0256(96)00008-0. URL http://www.sciencedirect.com/science/article/pii/0927025696000080.

[50] Bijaya B. Karki. First-principles molecular dynamics simulations of silicate melts: Structural and dynamical properties. *Reviews in Mineralogy and Geochemistry*, 71(1):355–389, 2010. doi: 10.2138/rmg.2010.71.17. URL http://rimg.geoscienceworld.org/content/71/1/355.abstract.

[51] Nico de Koker. Structure, thermodynamics, and diffusion in caal2si2o8 liquid from first-principles molecular dynamics. *Geochimica et Cosmochimica Acta*, 74(19):5657 – 5671, 2010. ISSN 0016-7037. doi: http://dx.doi.org/10.1016/j.gca.2010.02.024. URL http://www.sciencedirect.com/science/article/pii/S0016703710000943.

[52] Jonathan F. Stebbins and Paul McMillan. Compositional and temperature effects on five-coordinated silicon in ambient pressure silicate glasses. *Journal of Non-Crystalline Solids*, 160(1–2):116 – 125, 1993. ISSN 0022-3093. doi: http://dx.doi.org/10.1016/0022-3093(93)90292-6. URL http://www.sciencedirect.com/science/article/pii/0022309393902926.

[53] Scott Kroeker and Jonathan F. Stebbins. Magnesium coordination environments in glasses and minerals: New insight from high-field magnesium-25 mas nmr. *American Mineralogist*, 85(10):1459–1464, 2000. URL http://ammin.geoscienceworld.org/content/85/10/1459.abstract.

[54] Keiji Shimoda, Yasuhiro Tobu, Moriaki Hatakeyama, Takahiro Nemoto, and Koji Saito. Structural investigation of mg local environments in silicate glasses by ultra-high field 25mg 3qmas nmr spectroscopy. *American Mineralogist*, 92(4):695–698, 2007. doi: 10.2138/am.2007.2535. URL http://ammin.geoscienceworld.org/content/92/4/695.abstract.

[55] L. Cormier and G. J. Cuello. Mg coordination in a mgsio$_3$ glass using neutron diffraction coupled with isotopic substitution. *Phys. Rev. B*, 83:224204, Jun 2011. doi: 10.1103/PhysRevB.83.224204. URL http://link.aps.org/doi/10.1103/PhysRevB.83.224204.

[56] Jonathan F. Stebbins and Zhi Xu. Nmr evidence for excess non-bridging oxygen in an aluminosilicate glass. *Nature*, 390(6655):60–62, 11 1997. URL http://dx.doi.org/10.1038/36312.

[57] QUENTIN WILLIAMS and RAYMOND JEANLOZ. Spectroscopic evidence for pressure-induced coordination changes in silicate glasses and melts. *Science*, 239(4842):902–905, 1988. doi: 10.1126/science.239.4842.902. URL http://www.sciencemag.org/content/239/4842/902.abstract.

[58] M. C. Wilding, C. J. Benmore, J. A. Tangeman, and S. Sampath. Coordination changes in magnesium silicate glasses. *EPL (Europhysics Letters)*, 67(2):212, 2004. URL http://stacks.iop.org/0295-5075/67/i=2/a=212.

[59] J. P. M. Hultquist. Constructing stream surfaces in steady 3d vector fields. In *VIS '92: Proceedings of the 3rd conference on Visualization '92*, pages 171–178, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press. ISBN 0-8186-2896-0.

[60] Tobias Schafhitzel, Eduardo Tejada, Daniel Weiskopf, and Thomas Ertl. Point-based stream surfaces and path surfaces. In *GI '07: Proceedings of Graphics Interface 2007*, pages 289–296, New York, NY, USA, 2007. ACM. ISBN 978-1-56881-337-0. doi: http://doi.acm.org/10.1145/1268517.1268564.

[61] W. von Funck, T. Weinkauf, H. Theisel, and H.-P. Seidel. Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization 2008)*, 14(6):1396–1403, November - December 2008. URL `http://tinoweinkauf.net/`.

[62] Jeff P.M. Hultquist. Constructing stream surfaces in steady 3d vector fields. In *Visualization '92*, pages 171–178. IEEE Computer Society, 1992.

[63] D. Stalling. *Fast Texture-Based Algorithms for Vector Field Visualization*. PhD thesis, Free University Berlin, 1998.

[64] Werner Benger, Georg Ritter, and René Heinzl. The Concepts of VISH. In 4^{th} *High-End Visualization Workshop, Obergurgl, Tyrol, Austria, June 18-21, 2007*, pages 26–39. Berlin, Lehmanns Media-LOB.de, 2007. ISBN 978-3-86541-216-4.

[65] David M. Butler and M. H. Pendley. A visualization model based on the mathematics of fiber bundles. *Computers in Physics*, 3(5):45–51, sep/oct 1989.

[66] Werner Benger, Marcel Ritter, Sumanta Acharya, Somnath Roy, and Feng Jijao. Fiberbundle-based visualization of a stir tank fluid. In *WSCG 2009, Plzen*, 2009.

[67] Julio M Ottino. *The kinematics of mixing: stretching, chaos, and transport*, volume 3. Cambridge university press, 1989.

[68] E Bruce Nauman. Residence time distributions. *Handbook of industrial mixing: science and practice*, pages 1–17, 2004.

[69] Tino Weinkauf and Holger Theisel. Curvature measures of 3d vector fields and their applications. 2002.

Vita

Bidur Bohara was born on January, 1984, in Kathmandu, Nepal. He holds a Bachelor in Computer Engineering from Institute of Engineering at Tribhuvan University, Nepal. In the fall of 2008, he was accepted to the graduate program in computer science at Louisiana State University. He worked as a research assistant to Dr. Bijaya B. Karki while working toward the doctoral degree in computer science. During his doctoral studies, he has co-authored eight refereed conference and journal publications. His primary interests include computer graphics and scientific visualization.