VISUS

Bachelorarbeit

Visual Analysis of Injection Processes

Alex Heller

Course of Study:	Informatik
Examiner:	Prof. Dr. Thomas Ertl
Supervisor:	Dr. Guido Reina, Alexander Straub, M.Sc. Moritz Heinemann, M.Sc. Patrick Gralka, M.Sc.
Commenced:	May 23, 2022
Completed:	November 23, 2022

Abstract

The injection of fluid into a vacuum can be simulated with different simulation architectures. This thesis tries to establish a link between particle and grid-based simulations based on the fluid droplets formed in such simulations. For this, a shared data representation is introduced, which enables the use of the same analysis and visualization methods for either particle or grid-based data.

As such, techniques were implemented to extract droplets and their corresponding descriptions from the simulation data. Moreover, these droplets are also tracked over time to capture their evolution throughout the simulation. The resulting droplets and their trajectories are then visualized in a way that showcases their behavior across the entire simulation time. These methods were implemented as generalized MegaMol modules and are available for further use as open source software.

Furthermore, the droplet trajectories and the interactions between them were analyzed for a sample particle simulation. This is done once in a standalone manner and once in a comparison between grid and particle based data. The comparison was performed by creating grid data from particle data and applying the tracking and analysis techniques to both, to test if the results are comparable. It was found that the comparison between two different simulation architectures is possible, but still poses some challenges when trying to relate the extracted droplet behavior.

Kurzfassung

Einspritzverfahren von Flüssigkeiten in ein Vakuum können durch unterschiedliche Metodiken simuliert werden. Diese Arbeit versucht eine Verbindung zwischen den häufig eingesetzten Partikel- und Gitter basierten Simulationsdaten herzustellen. Dafür wird eine gemeinsame Datenräpresentation eingeführt, welche die Nutzung der selben Analyse- und Visualisierungsmethoden für beiderlei Daten ermöglicht.

Dazu wurden Techniken implementiert, um Tropfen und ihre beschreibenden Metriken aus den Simulationsdaten zu extrahieren. Darüber hinaus werden diese Tröpfchen auch über die Simulationszeit verfolgt, um ihre Entwicklung über die Zeit hinweg zu erfassen. Die resultierenden Tröpfchen und ihre Flugbahnen werden dann so visualisiert, dass ihr Verhalten über die gesamte Simulationszeit sichtbar wird. Diese Methoden wurden als allgemeine einzelnstehende MegaMol-Module implementiert und werden als Open Source Software zur Verfügung gestellt.

Weiterhin wurden die Tröpfchenbahnen und die Wechselwirkungen zwischen ihnen für eine gegebene Partikelsimulation analysiert. Dies erfolgt zum einen für eine einzeln stehende Simulation an sich und einmal im Vergleich zwischen Gitter- und Partikelbasierten Daten. Der Vergleich wurde durchgeführt, indem die Rasterdaten aus Partikeldaten erstellt und die eingeführten Verfolgungs- und Analysetechniken auf beide angewendet wurden, um zu testen, ob die Ergebnisse vergleichbar sind. Es wurde dabei festgestellt, dass der Vergleich zwischen zwei verschiedenen Simulationsarchitekturen zwar möglich ist, aber dennoch einige Herausforderungen mit sich bringt, wenn versucht wird, das extrahierte Verhalten der Tropfen in Beziehung zu setzen.

Contents

List of Figures

List of Tables

1 Introduction

Injection processes are used in many different fields, such as medicine, chemistry, and manufacturing. In these fields it is common to simulate such injections, to determine good injection parameters. One such analysis was done by Tao et al. [1] to analyze the injection process for micro-droplet formation in electrohydrodynamic micro/nano scale 3D printing.

This work focuses on analyzing simulations of a subset of fluid injection processes, wherein the fluid is introduced into a vacuum through a nozzle. The fluid streams are often very complex and difficult to visualize and thus also hard to analyze. This is mainly due to the fact that the raw data volume generated by such simulations can be large, e.g. ranging somewhere from 5 to 60 GB per simulation analyzed in this thesis. Such verboseness often times hinders the analysis of the data, making it infeasible to analyze the data in its raw form.

As such, higher level descriptions are useful for reducing the complexity of the data prior to its analysis. Abstractions of the simulations no longer need to perfectly retain all of the information contained by the original raw data. This enables the analyst to compare simulations of the same process with similar setups, highlighting the differences between them. Such differences can occur due to variations in the parameters of the simulation, or as result of the stochastic nature of the simulation.

Furthermore, the fundamental injection model can be either simulated within a discrete *particle* model, where each particle represents a small unit of fluid moving through the domain, or via a *continuous* fluid model where the fluid is represented by a continuous field in which the governing fluid equation is computed on discrete grid points within the simulation domain.

To still be able to compare the results of both models, the data needs to be abstracted in a way which is independent of the underlying simulation architecture. Droplet formation within injection processes is an inherent fluid behavior, it can occur regardless of whether it was simulated by particle or continuum methods. This is the reason for choosing the behavior of these droplets as the abstraction level to perform our analysis on. One example of such a behavior is the scattering of the droplets, which may change drastically by modifying the simulation parameters.

These droplets are analyzable in terms of their size, shape, position and velocity, and their evolution over time. To be able to perform the same analysis methods on either model, the data is first converted to a joint representation, which is then used to perform the analysis. This joint representation is called *droplet graph* and is described in detail in section 2.1.

The tooling to perform such an analysis was developed in the context of this thesis. It builds upon the MegaMol visualization framework [2] and is available as open source software. Furthermore the developed tools were applied in a case study comparing particle simulations originating from Heinen and Vrabec [3]. This was done as a verification step, as volumetric data could be generated from the provided particle simulation. On both of which then an analysis could be performed, to check if the results are similar.

1.1 Related Work

A common occurrence in scientific visualization is the issue of information abundance, as the data sets that are to be analyzed and visualized are often large in size, or have a high dimensionality. This applies here as well, since fluid simulations are very feature rich even after their data has been reduced by preprocessing steps. To tackle this issue Potter et al. [4] outline the methodology of so called *Ensemble-VIS*, in which *multi-dimensional* and *multi-variate* weather simulations pose a similar challenge, due to the large amount of data that is available. It bears resemblance to the droplet data extracted from the simulations in this thesis, as each droplet can be viewed as an ensemble member. As such the visualizations options such as plume or quartile charts can be applied to the here analyzed fluid simulations as well.

Tsuji et al. [5] describe a discrete particle model for numerical simulation of *fluidized beds*, which is then used to attempt reproduction of previous results from a continuum model. The authors then go on to quantitatively compare the results of both models according to the clustering behavior of the simulation. They focus on global metrics and still frames for comparison without further investigation of individual cluster behavior. Futher, the underlying analyzed simulation differs strongly from the here observed injection processes.

Knuth and Henne [6] analyze droplet size with regard to their mean and distribution for a fluid expansion into a vacuum, with the goal of finding an expression for the distribution of the size of the droplets only. Similarly, Alred et al. [7] implements a method to simulate a water jet with injection into a vacuum was generated. It was subsequently analyzed regarding the impact of the parameters on the droplets behavior indicated by its equations of motion. These investigations were devoid of any visualization or metric regarding the individual droplets. Similarities to this thesis however are the high level comparisons and analyses according to the droplets size, shape or velocity.

More closely related to this work is [8], in which droplets within volumetric fluid simulations were tracked and analyzed according to their behavior over time, as well as had their phase interface visualized. A space-time connectivity graph of the droplets between frames is obtained by transporting particles through the volume data. The graph is then used for visualization of the droplet behavior, where care was taken to layout and portray the extracted graph.

While truly similar to the work achieved in this paper, we additionally introduce a way to extracting a similar space-time graph from particle data. Furthermore, we propose methods to further reduce the information stored within the graph via plotting the values contained within against each other. This is done in prospect of creating a comparison method between fluid simulations.

2 Methodology

After the first visualization of the simulation output of the fluid simulations, the formation of droplets is quickly observable. This droplet generation is a feature of the governing fluid equations. Thus the droplets should be able to emerge in both particle and grid-based simulated fluid injections.

As such there exists an interest in analyzing these droplets and their behavior over time in further detail, as that could provide insights in cross simulation type comparisons. For this the droplets need to be extracted from the raw simulation data. After extraction of the droplets from a single frame, they still need to be tracked between frames, such that their evolution can be observed. These stages preforming these computations are described in section 2.3.1 and 2.3.2.

A data structure was needed to store and pass around the extracted droplets and their evolution over time. The chosen representation for this task called *droplet graph* is outlined in section 2.1.

The then available extracted data can thus be used for visualizations within MegaMol to provide initial overviews of the data contained in the extracted droplets. This visualization is described in section 2.4, alongside which the surface extraction for additional droplet information can also be performed section 2.4.1. The droplets can instead also be exported for further analysis in external tools. This capability is demonstrated here by section 2.5.1 where a sample python framework was built to showcase it.

To actually apply the methods for testing their feasibility some simulation data was required. The data actually used for this thesis is outlined in section 2.2. To confirm that the developed methods are actually applicable to both simulation types the splatting method in section 2.2.1 is employed to generate volumetric from the particle data. This generated volumetric data should in turn definitively be comparable to the original particle data, as it stems from the same origin.

To actually check the implementation and test the feasibility of such a droplet based analysis, tests were performed which try to extract information from the droplet trajectories. These tests were performed by only observing one simulation at a time, comparing the same data for droplet and grid-based technologies. Lastly the potential speed improvement of splatting particle data prior to droplet extraction and tracking is considered.

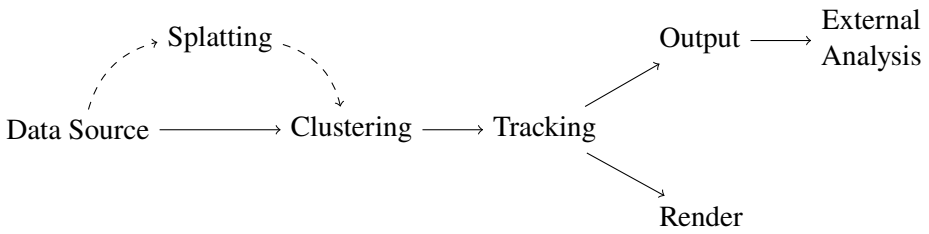


Figure 2.1: Graph outlining the data processing stages of this work. The dotted line resembles an optional stage transferring particle to volume data.

2.1 Data Representation

The first stage of this work was to develop a shared representation of the simulation results, which can be generated from either volume or particle based simulations. As the droplets that are extracted later on contain some sort of descriptive information for each frame a way to store this information is required. The droplets can contain general descriptive statistics such as their current center of mass, total mass (characterized either by contained particle count or aggregate density), velocity or bounding box. Additionally, information about the droplet's interface such as the surface area, its volume, or anisotropy measures (see section 2.4.1) should also be able to be represented.

Furthermore, since the droplets are to be analyzed with regards to their evolution over time as well, a way was needed to denote the temporal connection between the droplets extracted in different frames. For these temporal connections additional information should be able to be stored, such as the mass amount that is actually matched between the droplet frames.

From these two information types a graph like structure naturally emerges. The nodes hereby represent the droplets in each individual time frame and the edges depict their temporal connections. This structure is called *droplet graph* and is illustratively depicted in figure 2.2 without any of the additional node / edge information.

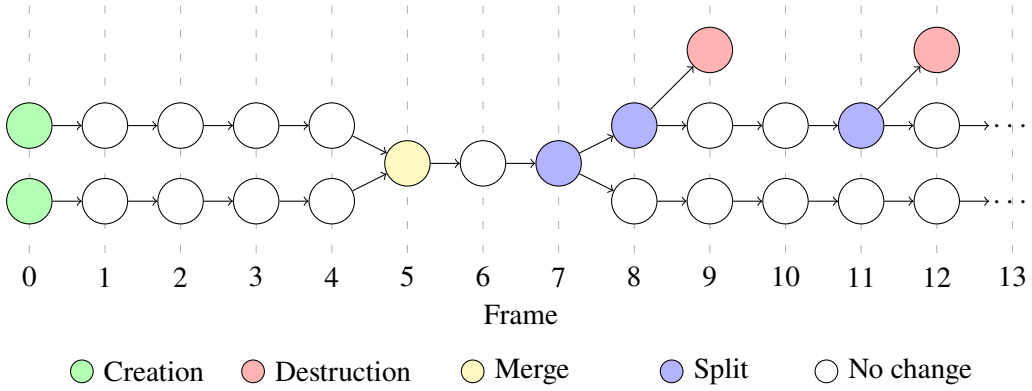


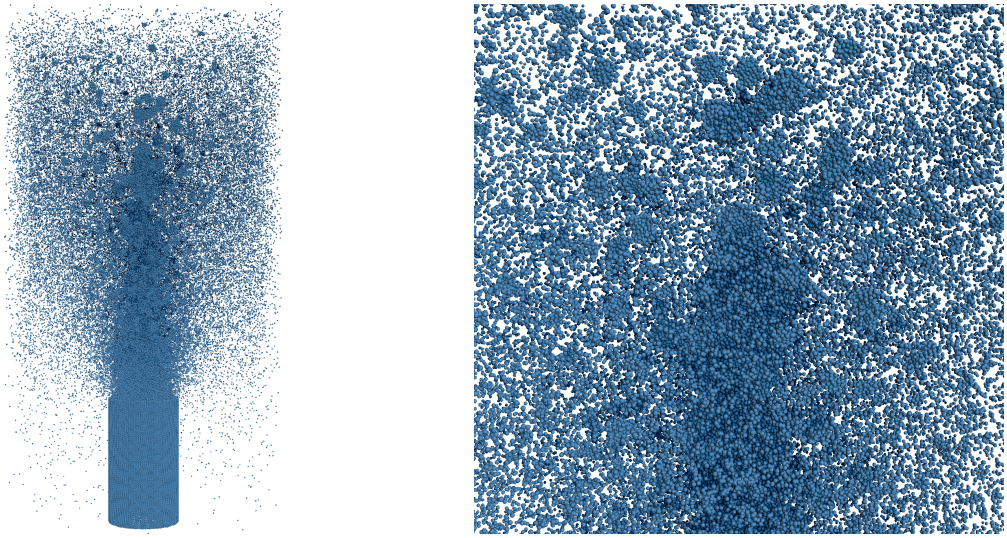
Figure 2.2: Example graph showcasing droplet merging and splitting events of 2 droplets over time.

2.2 Data Source

The raw data can originate from either volumetric or particle based simulations. Hereby, the volume based simulation is represented by a 3D grid of voxels, while the particle based simulation is represented by a set of particles.

The simulations which were used for the evaluation of the methods introduced by this thesis were generated in the context of Heinen and Vrabec [3]. They were available in the form of a set of ADIOS files, which contain the raw data of the simulation for some time frame.

The simulations are molecular dynamic (MD) simulations of a fluid, which is a particle based method. These particles are introduced into the simulation domain through a nozzle with the diameter of 50 particles, with an initial temperature T_{in} of 0.95. The varying parameters of the



(a) The particle data drawn as spheres.

(b) Closeup displaying the particle clusters forming droplets in the simulation.

Figure 2.3: A single frame of the particle simulation showing raw particle data. $T_{in} = 0.95$, $T_{out} = 1.90$, $v_{feed} = 0.004$ at the frame 200.

simulations are the feed velocity v_{feed} assuming values 0.001, 0.002 and 0.004, as well as the outgoing temperature T_{out} which was assigned the values 0.8, 0.97, 1.05 and 1.90. The boundary conditions of the sides was set as repeating, while top and bottom e.g. the ones in direction of the injection, were set to destroy exiting particles. Each of the simulations had data provided for 500 frames, with a timestep between them of 0.912. These values were given as unit less measures used for the parameterization of the simulation.

A single frame of the simulation is depicted in figure 2.3.

2.2.1 Splatting

The splatting stage is utilized to transfer particle based simulation data into a volumetric representation. This is done by projecting the particles onto a regular grid, which is then used to generate a volume. The result contains both the density of particles resolved into the voxels, as well as their weighted average velocity in these cells. While methods were implemented to support both volume and particle data in the clustering and tracking stages, and splatting thus not being strictly necessary for generating the droplet graph later on, it can be used for checking the comparison feasibility of the droplet tracking algorithms.

This is due to the fact that the splatting stage can be used to generate a volume from the particle based simulation, which can in turn be used to generate a droplet graph for both volume and grid based data types. As the underlying data is known to be the same for both types, the droplet graphs generated for both types can then be compared to check for differences.

Additionally, splatting the data instead of performing the computations on the raw particle data gives rise to performance benefits, as demonstrated later on in section 4.5.

2.3 Droplet extraction and matching

This stage is used to populate a droplet graph with data from the simulation. It is split into two parts, the *clustering* and the *tracking* stage.

These parts are the most computationally intensive parts of the whole pipeline, as they are responsible for the actual droplet extraction and matching. This means they need to iterate over all simulation frames of interest and perform the computations on each frame individually, combining the results of timestep into the entire graph.

The computations of this stage are therefore done once offline, storing the resulting droplet graph to disk. The extracted droplet graph is then used by the visualization and analysis stages to render the overview and provide the comparative figures.

2.3.1 Clustering

This part of the pipeline is responsible for extracting the droplets from single frames of the given data. It is the first step in the process of creating the droplet tracking graph.

By receiving either volume or particle based simulation data as input, it is responsible for extracting the droplets from the data. This is done by determining areas of high density and assigning them a label. While this label will be unique for each droplet within a single frame, it is not guaranteed to be unique nor consistent across frames.

2.3.2 Tracking

By receiving both the original simulation data and the labels assigned to the droplet in the clustering step (see 2.3.1), this part of the pipeline is responsible for matching the droplets across multiple frames. This is needed, as the labels assigned in the clustering step are not guaranteed to be consistent across frames. As such the tracking step is responsible for finding the correct matching between the droplets in the different frames.

This stage is what enables the analysis of the droplet evolution over time, as it connects the individual frames together. In whole it is responsible for determining the connections between the nodes in the droplet tracking graph.

2.4 Visualization

2.4.1 Meshing

By performing a meshing algorithm on the density data, the liquid-gas-interface in individual frames can be extracted. This interface can then be utilized as another source of information on the droplets, as it can be used to determine the volume of the droplet, as well as its surface area. Furthermore, it enables the possibility of determining statistics on the droplet's shape, such as its aspect ratio.

2.4.2 Rendering

This stage is responsible for an interactive visualization of the precomputed droplet graph. It is run directly within MegaMol without the need for any external tools.

Moreover, it can be combined with preexisting visualization modules, enhancing the possibilities of the visualization. For example the volume could be rendered by a preexisting volume renderer, which is uninformed of the droplet information, while also rendering the bounds of the droplets that were recognized by the droplet tracking pipeline.

This helps validate the selected parameters for the different stages of the pipeline.

2.5 Analysis

2.5.1 Interaction with external tooling

The droplet graph can be saved to disk and subsequently be loaded into an external tool. This can be used to perform further analysis on the droplet graph with preexisting libraries and tools, which might not be available within MegaMol.

The external analysis was demonstrated through the creation of a small python framework. It provides additional insights into the data contained within the droplet graph. This is done by generating summarizing graphics of one or multiple simulations at once. Such graphics for example include plotting the droplets mass, lifetime, velocity, etc. against the simulation time. The initial droplet trajectory renderer was also first implemented in python, before it was transferred to MegaMol.

2.5.2 Verification

To confirm that the clustering and tracking stages of the pipeline are working correctly, the results of the droplet tracking pipeline are compared to the results of the volumetric tracking pipeline. By splatting the particle data into a volume the volumetric tracking pipeline can be used on the volume data originating from the particle data. This allows the comparison between the results of the two pipelines, as that the underlying data is known to be the same, since it originates from the same simulation.

Within this work this was done using the molecular particle data from Heinen and Vrabec [3] with the feed velocity of 0.004 and the outgoing temperature of 1.90.

The splatting was done using a kernel radius of 2.5 cells. Further more the volume generated had the resolution of $200 \times 400 \times 200$ cells.

2.5.3 Performance

Another interest was whether speedups would be able to be obtained by using the splatted data instead of the raw particle data. For this the time each approach took to generate the droplet graph was recorded, for each simulation data set. The results of this can be seen in section 4.5.

3 Implementation

Many parts of this project were created as new modules in the project MegaMol, written in C++. Concretely this applies to the droplet tracking pipeline, the surface extraction, and the interactive visualization of the droplet graph.

For further and quicker data analysis iteration times a small python framework was created. Alongside it a jupyter notebook that showcases the frameworks capabilities. This allows the use of short and concise python scripting and the benefits this entails on iteration time, before actually transferring the observations made there to new MegaMol modules.

3.1 Splatting

To transfer particle data into a volume consisting of voxels splatting is performed. The transferred data contains both the estimated density at each voxel as well as the average velocity of the particles. The module supports selection of the desired voxel size, as well as the splatting method.

3.1.1 Splatting methods

The splatting module supports three different methods for splatting the particles onto the grid.

Nearest neighbor assigns the particle to the voxel containing it. This yields integer densities, alongside the average velocity of the particles in the voxel. Since the likelihood that two particles land within the same voxel decreases with smaller voxel sizes, this method is not suited for high resolution splatting, as the resulting data can be very noisy in that case.

Kernel based applies a *radial basis function* (RBF) on the location of the individual particles to project the particle data into the local volume neighborhood. This projection is weighted according to the distance between the particle and the cell center together with the shape of the RBF. This algorithm is outlined visually within figure 3.1 where it is applied to the simpler 1D case. However in the actual implementation the algorithm iterates over all the cells contained within the splat radius and computes the contribution of the particle to each cell in the 3D neighborhood. By writing to multiple cells a smoothing behavior is obtained.

The RBF implemented is the so called bump function:

$$\varphi(r) = \begin{cases} \exp\left(\frac{-1}{1-(\varepsilon r)^2}\right) & \text{for } r < \frac{1}{\varepsilon}, \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Where r is the distance between particle position and the cell, and ε is the reciprocal kernel radius.

That these weights sum to 1 is guaranteed by normalizing them before committing the values to the volume, so that the values splat into the volume relate to the actual particle density. This cell-by-cell density then can be used to further accumulate other data, such as the velocity, by simply weighting the original values by the splatted density.

By specifying the kernel radius, the size of the neighborhood can be adjusted, and thus the smoothing behavior. This is done in world space, meaning that the radius is the same independent of the voxel size.

Performing this method however comes at the cost of increased computation time over nearest neighbor splatting, as the kernel needs to be evaluated for each cell in the neighborhood, instead of just modifying the nearest cell.

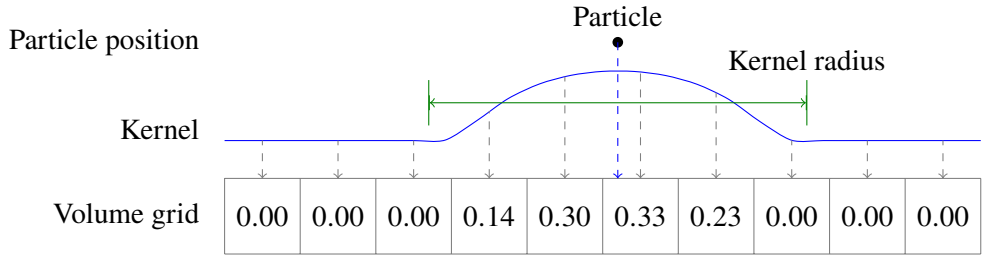


Figure 3.1: Splatting example in a 1D case with the bump function $\varepsilon = 1/2.5$

Natural neighbor: By using a natural neighbor interpolation, an even smoother result which fits the "true" underlying density function even better can be obtained. This is done by computing a *Voronoi tessellation* of the particles. Such a tessellation is a partitioning of space into regions based on the distance to a set of points S which are the particles in this case. A region (or cell) is defined by the set of points that are closer to a specific point $x \in S$ than to any other point p in the set S.

These Voronoi cells are then used to determine interpolating weights for each volume cell point by inserting a virtual particle at the volume grid point. From this virtual particle one can determine the Voronoi cell it would create when added to the set of all points. The overlap between this virtual cell and the original adjacent Voronoi cells then yields the interpolation weights. Here the *Laplacian / Non-Sibsonian* weights [9] are used and can be computed as follows:

$$w_i = \frac{\frac{A_i}{d_i}}{\sum_{j=1}^{n} \frac{A_j}{d_j}} \tag{3.2}$$

Herein A_i is the face area between the virtual Voronoi cell and the Voronoi cell of the i-th particle, d_i is the distance between the virtual particle and the i-th particle, and n is the amount of particles adjacent to the virtual Voronoi cell.

Using these weights the interpolated density and velocity for each volume cell can be computed as following:

$$\phi(x) = \sum_{j=1}^{n} w_j \phi_j \tag{3.3}$$

Where ϕ_j is either the velocity of the particles or the density as computed by the shaping kernel function and $\phi(x)$ is the interpolated density or velocity at the volume cell point x.

For the computation of the required Voronoi tessellation and virtual Voronoi cells the *VORO++* [10] library was used. This method however is computationally even more expensive than the plain kernel based method, as it requires the computation of the tessellation for each frame. Computing that is expensive as it has to organize the unstructured point data first which is an overhead the regularly structured volume methods do not have.

3.1.2 GPU acceleration

Due to the high computational cost of the splatting process, it was decided to implement the splatting on the GPU. This version only supports the plain kernel based splatting method, as the natural neighbor method requires the computation of the Voronoi tessellation. This seemed to be a larger task to implement on the GPU, as more intricate datastructures are required for natural neighbor interpolations.

The plain iteration over the neighboring cells of each particle is however easily parallelizable, as each particle only affects a small neighborhood of cells. Thus the implemented acceleration can utilize the capabilities of newer GPUs to perform atomic operations on a buffer, which was applied here to atomically increase the density and velocity fields while iterating over all particles within a compute shader.

3.2 Volume Clustering

To identify individual droplets within volumetric data, the region grow algorithm [11] is run on the density map of the volume data. The algorithm performs following steps on each volumetric cell.

If a cell is found which crosses the given threshold, a region is initialized to grow from there. The cell is marked with the new clusters id. Afterwards the cells in each cardinal direction of the current cell are observed. Should the neighboring cell also cross the threshold and not be assigned an id yet, the region is grown recursively from the neighbor cell assigning it the same id as was assigned by the initial region grow call.

After running this algorithm until every cell has been visited once at least, a unique id for each connected region of voxels within the volume is obtained. These regions correspond to the droplets of the simulation data in our case. The ids are then emitted from this module in a volume, where each cell contains the id of the droplet it belongs to.

3.3 Volume Tracking

The region grown id mappings generated by the volume clustering (see section 3.2) of 2 consecutive frames, are used to track droplets within volumetric data.

As the generated ids are not guaranteed to be consistent for 1 droplet across multiple frames the unique id's within the current frame thus require a matching to the droplet id of the next frame. For this the velocity stored within the simulation voxels of the fluid is made use of.

The connection is computed by creating a virtual particle within each cell center and integrating it forward one timestep along its corresponding velocity. This is due to the fact that the spawned virtual particle then can carry the cluster id from its starting frame and can be compared to the id of the cell it lands in after the integration step.

Figure 3.2 visualizes this process. Different colored grid cells represent the different ids generated by the volume clustering for the 2 timesteps. The spawned virtual particles are colored according to the id they carry. Here the matching of the ids ● and ● can be seen.

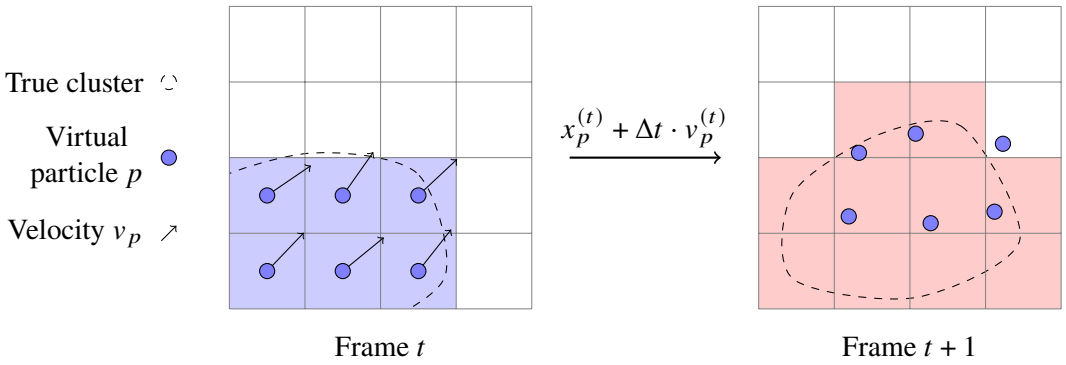


Figure 3.2: Connection creation between volume cluster ids during volume tracking. The virtual particle transfers the id from one frame to another by performing an euler step. Colors represent different cluster ids.

Thus a connection between consecutive time frames can be established. These connections can then be returned within a `GraphCall` with the nodes carrying the droplet's metadata and the edges transporting the cross frame connectivity of the droplets.

The metadata that these nodes carry, contains information about the droplet, such as its bounding box, center of mass, "total mass" (contained particles / density), velocity, ids. This data is also determined within this module, as all the incoming data does not yet contain the concept of a droplet. So it aggregates the aforementioned data according to the given id-map.

3.4 Particle Clustering

Prior to the droplet extraction on particle data a method to prefilter the data according to its thermodynamic properties was employed. This was done by the preexisting module `ParticleThermodyn` which computed the thermodynamic properties of the particles, such as temperature, and density, and classified them as either fluid or gas. This classification could thereafter be utilized to filter out the gas particles, to denoise the data and to improve the clustering results.

To then determine the clusters within the particle space one has to employ algorithms which work on unstructured data. In this case the *DBSCAN* algorithm [12] was used to extract clusters with higher density directly from the particle data, which as previously discussed represent the droplets of our fluid simulation.

The preexisting module `IColClustering` already implemented this within MegaMol and could be reused for this purpose. It performs the clustering based on the particles position, along with their stored color intensity. The particle color intensity contribution can be set to 0 for it to only consider the particles position. In the end the resulting cluster id is written to the particles color intensity.

3.5 Particle Tracking

This stage works by using two different ids. The first one is the *unique* id of the particle which is emitted by the simulation and persists across all frames. It can be used to identify a specific particle throughout the entire simulation. The second one is the id of the cluster the particle belongs to in the current frame, which is computed by the particle clustering module (see section 3.4).

By storing the cluster id for each encountered particle the connection between consecutive frames can be established. This is done by comparing the cluster id of the particle in the current frame to the cluster id of the particle in the previous frame.

Consider following example: The particles with global ids $\{1, 2, 5\}$ are clustered into the cluster A in the first frame. In the second frame the particles with ids $\{1, 2, 3, 4\}$ are clustered into the cluster B. As particles 1 and 2 are part of both clusters, cluster B is eligible to be considered as the continuation of A. In practice this can be further refined by requiring that a certain amount of particles are part of both clusters, before considering them as a match. The amount of particles verifying this connection is also stored along the connecting edge in the droplet graph.

3.6 Volume Meshing

To handle the fluid surface instead of the volumetric data itself a way for surface extraction was created.

The dual contouring approach [13] is used to extract the liquid-gas-interface. It generates surfaces based on the density at 0 crossings in the data along cell edges, as well as the density gradient as the estimated normal of the surface at these positions. Since a supporting position and a normal vector describe a plane, a local description of the surface is obtained. This local description needs to be joined with the other planes found on the other edges with zero crossings. This joining is done by determining a shared vertex within the cell which fits all previously extracted planes best.

In the optimal case the vertex would be described uniquely be the intersection of all the planes. This yields a linear system of equations of the form:

$$\vec{n}_i^{\top}(\vec{x} - \vec{p}_i) = 0$$

Where $\vec{p}_i$ is the point of the zero crossing along the edge and $\vec{n}_i$ is the surface normal at that point. The value to be calculated would then be $\vec{x}$. However in reality this system of equations is often over determined since there can be more than 3 zero crossing faces to intersect, requiring to find a "best" solution instead. This best fit is determined using the least squares error, as it yields a closed form solution.

$$\hat{\beta} = \left(X^{\top}X\right)^{-1} X^{\top}Y$$

Here X is the matrix containing all the normal vectors and Y a column vector containing $\vec{n}_i^\top \vec{p}_i$. Solving this equation is done by using the *Eigen* library [14].

This optimal vertex position is computed for every cell, where the isolevel is passed on at least one edge. Afterwards all the edges which contain a 0-crossing generate a quad perpendicular to the edge direction, and set its corner vertices to the previously computed best fitting vertices. By sharing the adjacent vertices the algorithm always yields watertight meshes, which works in our favor, as this is an apt description of a fluid interface.

Lastly, the amount of volume this algorithm captures can be determined by a threshold factor. It shifts the 0 crossing up or down to enable the extraction of any arbitrary isosurface height.

3.7 Mesh Segmentation

To determine the droplets within the mesh for further computations on a droplet basis, the so called *union-find* algorithm [15] is applied for efficient set queries and merge operations. This is required as the mesh only consists of the vertex positions and edge indices connecting the vertices. To keep the process as modular as possible no assumptions on the layout of this data were made, being able to be implemented efficiently using the previously mentioned algorithm.

It efficiently performs set merges and lookups on the data. Each vertex id is assigned to a set. By iterating over all triangles and merging the sets of the vertices of each triangle, the sets are then merged. Afterwards all indices are collected according to the set they belong to, thus yielding the connected components of the mesh.

The resulting index sets are then emitted as segments, which offset into the underlying vertex data. This can then be utilized by the mesh analysis to compute properties for each droplet surface.

3.8 Mesh Analysis

For determining some further metrics describing the droplets interface, a way for analyzing the previously generated mesh segments was implemented. It yields general metrics such as the *bounding box* of each mesh segment.

Furthermore the *surface area* is computed by aggregating the area of all triangles within the segment. The *volume* is computed by computing the signed volume of a tetrahedron for each triangle and a fixed point, summed over all triangles. These tetrahedrons are oriented, e.g. if the triangle is oriented, such that the normal points away from the enclosed volume, the volume is considered negative, otherwise it is positive. This method for determining the volume of a mesh was shown by [16].

Lastly some shape descriptors where implemented. For this the singular value decomposition (SVD) of the vertex positions of the surface mesh is computed to determine the droplets *principal axes* and their *singular values*. This SVD computation is done by the `Eigen` library [14]. The resulting

singular values σ_i can be used to compute the eigenvalues λ_i of the covariance matrix of the vertex positions, which in turn can be used to compute the anisotropy values c_l, c_p, c_s. [1]

$$c_s = \frac{3\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3} \qquad c_p = \frac{2(\lambda_2 - \lambda_3)}{\lambda_1 + \lambda_2 + \lambda_3} \qquad c_l = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3}$$

These in turn describe how similar the shape is compared to a sphere ($c_s = 1$), a plane ($c_p = 1$) or a line ($c_l = 1$), while they sum to 1 for any shape.

The sphericity Ψ similar to the aforementioned c_s value measures how much like a sphere the droplet is. It was originally introduced by Wadell [17] as a shape measure for grains of quartz particles, but can be applied to any geometrical shape with a surface area and volume. However instead of relying on the SVD this time it is a normalized measure of the surface area A_p to volume V_p ratio. The normalization is required as the simple ratio $\frac{V_p}{A_p}$ changes if the droplet is scaled uniformly up or down, due to the different scaling laws for volumes (cubic) and areas (quadratic).

It is computed by relating the surface area to the $\sqrt[3]{V_p^2}$, as this measure now also grows quadratically when scaling, cancelling out the scaling of the volume. Furthermore to limit this measure to the range $[0, 1]$ it is normalized by the maximum possible value for a sphere, which is $\sqrt[3]{36\pi}$, as all shapes other to a sphere have a higher surface to volume ratio, thus yielding a value less than 1.

$$\Psi = \frac{\sqrt[3]{36\pi V_p^2}}{A_p}$$

The closer this value is to 1, the higher the objects similarity is to a sphere. Another way to look at this is that the sphericity is the ratio of the area of an optimal sphere with the same volume to the actual surface area of the object.

3.9 Tracking Graph Renderer

For displaying the droplet graph data generated by the tracking modules the data is uploaded to the GPU. This is done by passing the droplets (nodes) as vertices and the inter frame connections (edges) as line primitives to the GPU. The droplets properties are also passed along as vertex attributes.

From there on custom shaders were employed to render the data in interesting ways. One view that can be shown are the trajectories of the droplets by rendering the lines between the center of mass of each droplet in each frame. There additional information can be displayed by coloring the lines based on the droplet data using a transfer function. The transfer function maps from the scalar droplet properties to different colors for better differentiation. For example the lines could be tinted based on the velocity direction, or the relative mass (either number of particles or density) of the droplet at the corresponding time frame it is alive in.

Another view implemented is showing animated bounding boxes around the droplets. This can be rendered in real time, thanks to the precomputation done in the tracking stage. The compact representation of the droplet evolution then allows their bounding boxes to be animated smoothly in real time.

[1] The singular values and eigenvalues are sorted in descending order, thus $\sigma_1 \geq \sigma_2 \geq \sigma_3$.

Lastly a filtering method was added as a proof of concept. It works by discarding any vertex that falls below a given mass threshold. Through this smaller droplets which might be discretization errors can be filtered out.

3.10 External analysis

External analysis was performed within a small python framework. It was used to generate many of the plots shown in chapter 4.

It consists of 3 main parts:

- A `DropletGraph` which loads the data exported from MegaMol.
- A `TrackExtractor` which extracts the droplet life cycles from the graph data.
- A `GraphGenerator` which generates the plots from the data.

The `DropletGraph` reader supports parsing the data fresh from the MegaMol graph data exported files. Furthermore it caches the data and stores it on the disk for faster access times.

The `TrackExtractor` is used to extract the droplet life cycles from the graph data. This means it tries to find the paths in the graph, which are temporally "most consistent", i.e. the longest paths, along with the events where the paths split and merge with other paths. Whenever there is a split along a path this means that a new droplet has split off from the original one.

Likewise whenever a merge occurs within the graph, it is know that multiple droplets have merged into one. Furthermore the matching between the droplets in such an event are also computed, to be able to continue tracking the droplets consistently. E.g. if a droplet splits into 2 new ones, the larger of the droplets is matched to the original droplet, while a new life cycle is generated for the smaller droplet.

The `GraphGenerator` is used to generate the plots from the data. This is done by utilizing the `matplotlib` [18] library. It can for example plot the number of droplets over time, show the droplet lifetimes as a histogram, plot the droplets size over time as a functional boxplot, or the distance the droplet traveled over their lifetime.

This is all then bundled within the `analysis.ipynb` python notebook. It simultaneously provides the previous analysis and graphing functionality within an interactive environment while also adding some additional functionality, such as the ability to pre-filter the data to only show a subset of the droplets (e.g. filter by min volume to only show the larger ones). Lastly this is also where the comparative plots are generated, in which data from the different simulations is overlaid. The sections 4.3 and 4.4 go further into detail of the analysis methods implemented by this notebook.

4 Results

The following visualizations have been generated within MegaMol. They showcase the extracted liquid gas interface in section 4.1 and the metrics computed upon those surfaces. Secondly section 4.2 presents visualizations of the droplet trajectories and their stored meta descriptions.

These visualizations are based upon the molecular dynamics particle simulation data from Heinen and Vrabec [3]. Specifically the simulations with parameters $T_{in} = 0.95$, $T_{out} = 1.90$ and $v_{feed} = 0.002$. Further detail on the data is given in section 2.2.

As at least the surface extraction algorithm requires a volumetric density field to work on, this field first needs to be extracted from the given particle data. This is done by utilizing the splatting stage described in section 3.1, with the chosen parameters visible in table 4.1.

Parameter	Value
Kernel radius r	2.5
Kernel function φ	Bump function see eq. 3.1
Grid cell size h	1
Volume dimensions	$200 \times 400 \times 200$

Table 4.1: Splatting parameters used for the results.

Transfer functions were used to color various entities throughout the visualizations. The ones used in this work are *viridis* and *plasma* shown in figure 4.1.

(a) Viridis **(b)** Plasma

Figure 4.1: Transfer functions used in the visualizations.

The simulation with higher temperature T_{out} was chosen to observe a plethora of droplets. Those with lower T_{out} have a much lower droplet count, which makes it harder to extract any meaningful droplets. This issue is visible when observing 3 simulations differing only in T_{out}.

Figure 4.2 showcases the deficit of large droplets within simulations with lower T_{out}, which are relevant to the analysis methods implemented in this thesis. As only T_{out} varies, $v_{feed} = 0.002$ and $T_{in} = 0.95$ are kept constant in this comparison. Every simulation does indeed generate droplets as can be seen in that figure, however the droplets are much smaller in size and thus less consistent during tracking.

(a) $T_{out} = 0.97$ **(b)** $T_{out} = 1.05$ **(c)** $T_{out} = 1.90$

Figure 4.2: Extracted liquid gas interface for different T_{out} values.

4.1 Surface visualization

After obtaining the density grid the surface can be extracted from it. This is done by utilizing the dual contouring method described in section 3.6 with the isolevel set to 10% between the minimum and maximum density value.

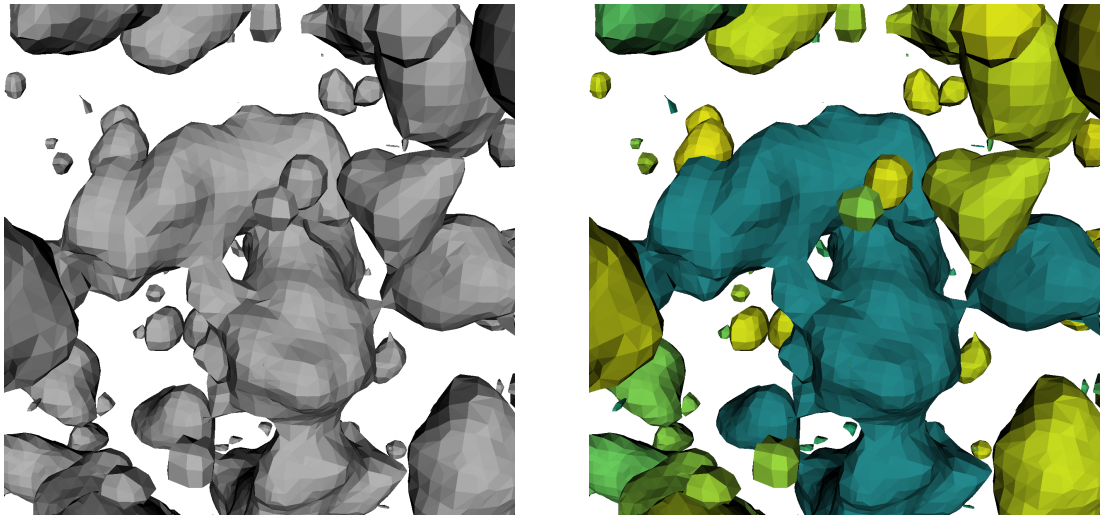
Since this mesh does not contain any information about potential ids of the density field and the therein embedded droplets such information still needs to be acquired. This is done by determining the connected components of the mesh, as described in section 3.7.

From these mesh segments one can then employ the methods described in section 3.8 to compute descriptive metrics for each segments shape, specifically the droplets linear, planar and spherical anisotropy values c_l, c_p, c_s or sphericity Ψ. These metrics are then sent to the preexisting `TriangleMeshRenderer3D` together with a transfer function to be rendered as colored meshes.

As the name implies this renderer is capable of rendering any mesh in a 3D scene, like the droplet surface in this case. It achieves this by receiving the mesh vertices, face indices and optionally normals and scalar values with a transfer function. The renderer displays the mesh using OpenGL, rendering the faces colored by the transfer function and scalar values provided. Moreover it also supports computing the normals of the mesh, front and back face culling and a wireframe view. In this case the transfer function chosen was the *viridis* color map (see figure 4.1a).

As the dual contouring method generates a vertex per grid cell, the resulting mesh is very dense. In this case the generated mesh contains approximately 291k triangles. Figure 4.3a shows a closer look of the surface, and its resolution. This mesh resolution results in an apparent smoothness of the surface, even if no additional smoothing is applied to the normals prior to shading.

If the surface is observed without any additional coloring, another challenge becomes evident: The separate droplets are hard to distinguish from one another. To remedy this the surface can be colored based on the computed metrics, as demonstrated in figure 4.3b, where the same zoomed in data was tinted according to the droplets' sphericity. The closer the droplets surface area is to a



(a) Surface mesh from volume.

(b) Surface colored by droplet sphericity. Coloring according to *viridis*

Figure 4.3: The extracted droplets shown as a mesh, unique colors assigned by the mesh analysis.

sphere of same volume, the more the sphericity metric approaches 1, which results in a more yellow tint. Straying away from the optimal volume to area ratio decreases the metric, leading to colder blueish green hues. Once colored, the connected components are much easier to distinguish from one another.

(a) Linear anisotropy **(b)** Planar anisotropy **(c)** Spherical anisotropy **(d)** Sphericity

Figure 4.4: Extracted liquid surface mesh colored by different droplet shape properties. The applied color map is *viridis*.

By computing the segment analysis metrics, the surface can be colored according to the cluster id, the surface area and volume, as well as the sphericity and anisotropy values. This is demonstrated in figure 4.4, where both sphericity and anisotropy are shown. As seen in section 3.8 the anisotropy

values represent interpolating factors between the extrema shapes of either completely linear ($c_l = 1$), planar ($c_p = 1$) or spherical ($c_s = 1$) form. Once again the yellowish hues indicate higher, while more blueish hues show a lower evaluation result of the respective metric.

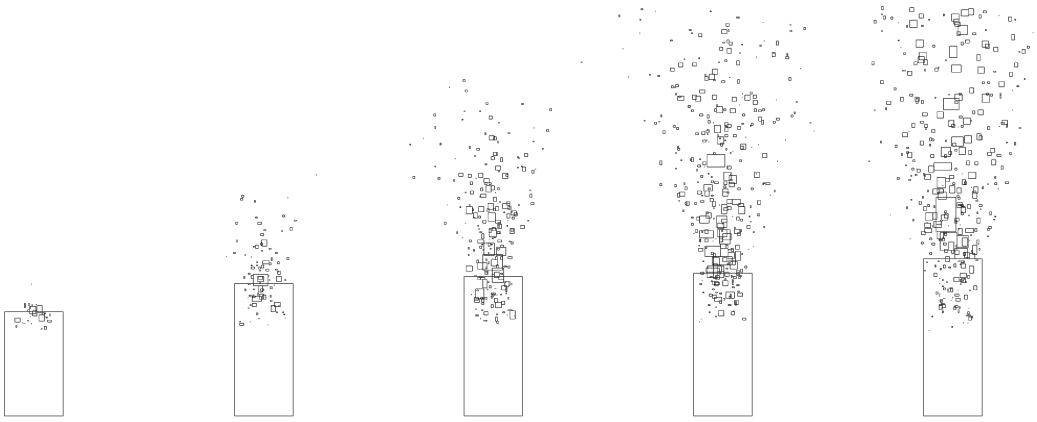
In these images multiple observations can be made. Firstly, the droplets tend to be more spherical the longer they are in the simulation, as is visible by the increasing sphericity and spherical anisotropy values distance from the inlet.

Secondly, the behavior at the inlet is very chaotic, which is visible as the droplets located there are not considered as separate entities but rather determined to be one single cluster by the surfacing algorithm. This single blob however sports a high amount of visible surface roughness in that area. This suggests that the fluid does not have distinct droplets in that zone, but rather that there the fluid vehemently interacts with itself.

4.2 Trajectory visualization

As the evolution of the droplets was of interest as well, the droplet graph was computed. It contains the entire simulation with the droplet data simplified to a few metrics per frame. To visually confirm the validity of the tracked graph methods for displaying the contained data were developed.

This is shown by computing the particle trajectories with the methods described in section 3.2 and section 3.3. After these modules precomputed the droplet graph, the renderer described in section 3.9 was used to visualize the graph.

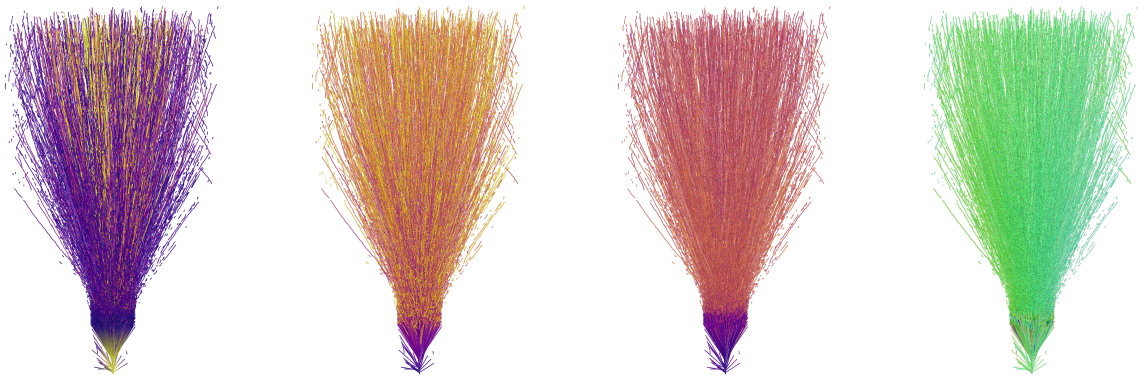


(a) Frame 150 **(b)** Frame 200 **(c)** Frame 250 **(d)** Frame 300 **(e)** Frame 400

Figure 4.5: Bounding boxes of extracted droplets at different timesteps.

One such rendering method is to display the axis aligned bounding boxes of the droplets animated and interpolated over time. Some still frames of such an animation are shown in figure 4.5. They showcase various stages of the fluid simulation. Frame 150 shows the first generation of droplets, frame 250 and 300 display the initial smaller droplets spreading out. Lastly frame 400 then reveals larger droplet conglomerates emerging while still emitting the approximately same number of particles.

The seaming increase in size of the bounding box surrounding the nozzle from frame 150 to 200 and onwards might appear weird at first. However as already seen within the meshing section 2.4.1 the neighborhood surrounding the nozzle exit gets clustered as one blob. This cluster is joint with the particles defining the nozzle. Since the fluid just about starts to exit the nozzle at frame 150, this region is not yet fully established, but grows with further simulation time. This added region explains the increase in size there.



(a) Mass **(b)** Frame **(c)** Velocity magnitude **(d)** Velocity direction

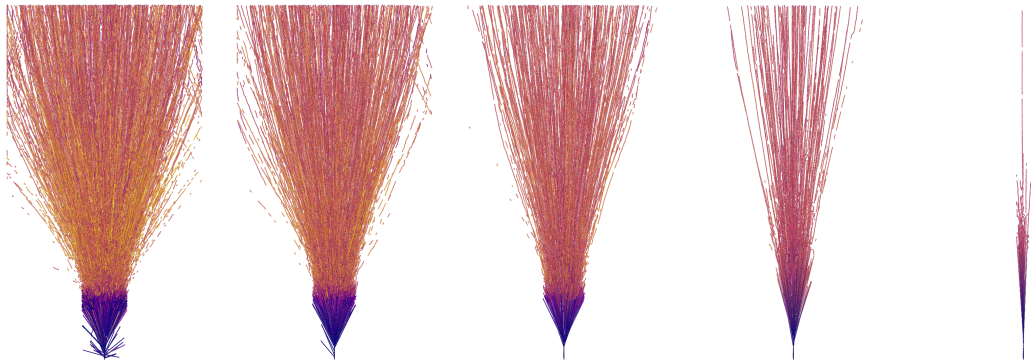
Figure 4.6: Droplet trajectories, (a) - (c) are colored using the *plasma* transfer function. (d) is colored by placing the unit sphere of directions into the $[0, 1]^3$ unit cube, interpreting the coordinates as rgb.

To visualize the droplets evolution over time, the entire trajectory of each droplet can be rendered as a line strip. This results in something akin to a 3D plume chart. The traces can be colored either by the droplet id, the frame the droplet lies on the specific point on the trajectory, the mass of the droplet, or by the droplet's velocity magnitude or direction. This is shown in figure 4.6.

These colorations are especially useful when exploring the data interactively in 3D, as they help to distinguish different trajectories within these plumes. One such example is the observation that droplet mass seems to influence the deflection angle, visible in figure 4.6a. There the trajectory is colored according to the droplet mass using the *plasma* color map, indicating heavy droplets in bright yellow and light droplets in bluish purple colors. As the core of the plume is colored brighter then the surrounding trajectories, it is indicated that the heavier droplets stay more inline with the injection direction than the lighter ones.

Another feature is the filtration of the trajectories regarding droplet mass, e.g. the size of the droplet, which is currently the only implemented filter, demonstrating the application of real time filtering capabilities. This is shown in figure 4.7 where the color is assigned according to the velocity magnitude and the *plasma* color map. Furthermore the lower threshold for the droplet mass T_m is set to either 0, 10, 25, 100, or 1000. By doing this the amount of data rendered is reduced, allowing to observe previously obstructed portions.

A relationship between droplet mass and the amount of deflection away from the injection direction can be observed once again. The smaller droplets are more likely to be deflected, while the large droplets seem to follow the injection direction more closely. This should be a sensible conclusion, as a higher droplet mass should correspond to larger moments of inertia and thus a greater resistance to velocity changes, compared to the velocity the fluid has within the nozzle.



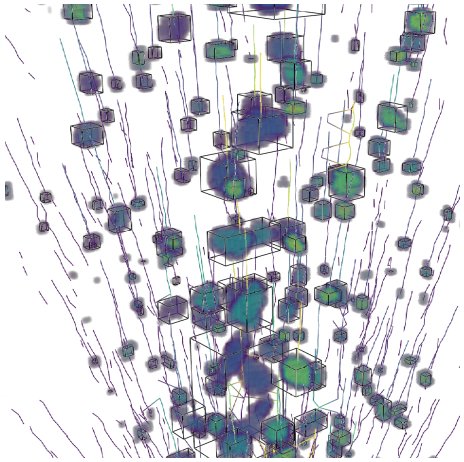
(a) $T_m = 0$ **(b)** $T_m = 10$ **(c)** $T_m = 25$ **(d)** $T_m = 100$ **(e)** $T_m = 1000$

Figure 4.7: Droplet trajectories, filtered by lower mass threshold T_m. Coloring uses the *plasma* color map on the velocity magnitude, compare figure 4.6c.

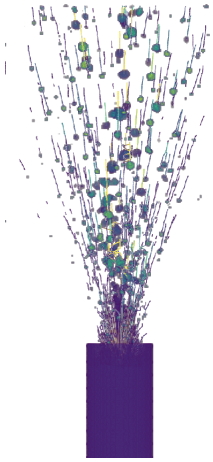
Another method for reducing the amount of data is to partially render the trajectories. This leads to an estimate of where the particles will be some short time in the future, and how they interact in that time interval.

Here this was shown by tracking just 10 frames, from frame 400 to 410. Furthermore the potential of combining the rendering methods is shown by coloring the trajectories according to the velocity magnitude, and rendering the bounding boxes of the droplets, while simultaneously using the preexisting `RaycastVolumeRenderer` to render the underlying volumetric data.

The `RaycastVolumeRenderer` takes any provided volumetric density map and renders its contents using raycasting. It steps through the volume along rays originating from the camera to raycast the volumetric density map. Along that ray it samples the volume, passes the sampled value through a transfer function and accumulates the resulting color, either until the color is opaque or the maximum marching distance is reached. An example of this is shown in figure 4.8. The transfer function applied to the volume renderer in this instance was again the *viridis* color map with transparency set so that the empty regions are ignored. In this image the volume rendered was the voxel ids assigned by the volume clustering algorithm described in section 3.2. This tinted the voxels of the found droplets according to the order they were found in.



(a) Bounding boxes, trajectories and volume rendering



(b) Full view of trajectories and raw volume

Figure 4.8: Extracted droplets displayed with their volume data. The volume is colored by mapping the ids from the volume clustering via the *viridis* color map.

4.3 Standalone graph analysis

The droplet graph can be exported into a standalone graph format. This allows for further analysis of the graph in other tools. To demonstrate this the graph was exported into the custom graph format and subsequently imported into a custom python framework. This framework is capable of analyzing the graph, extracting meta statistics and visualizing the attributes of the graph in different manners. It builds upon the `matplotlib` library, which is a python library for plotting and visualization, as well as `pandas`, which is a python library for data analysis.

4.3.1 Prefiltering

One of the first visualizations that was created during the course of this project was a way to visualize the droplets' trajectories in a 3D plot. This initial visualization was created to get an initial feel for the data. It was prototyped in the custom python framework using the 3D capabilities of `matplotlib` [18] and was partially transferred to MegaMol afterwards. This was done to achieve a more interactive version and was previously described in section 4.2, which showcases functionality of the `TrackingGraphRenderer` in MegaMol.

Within this 3D visualization the droplet paths were marked according to special events occurring at specific times in the trajectory. These events are categorized based on the droplet graph structure. In particular it depends on the number of incoming d_{in} and outgoing d_{out} edges per node. They determine whether the node is classified as an appearance, disappearance, split or merge event. The event assignment decision is outlined in table 4.2.

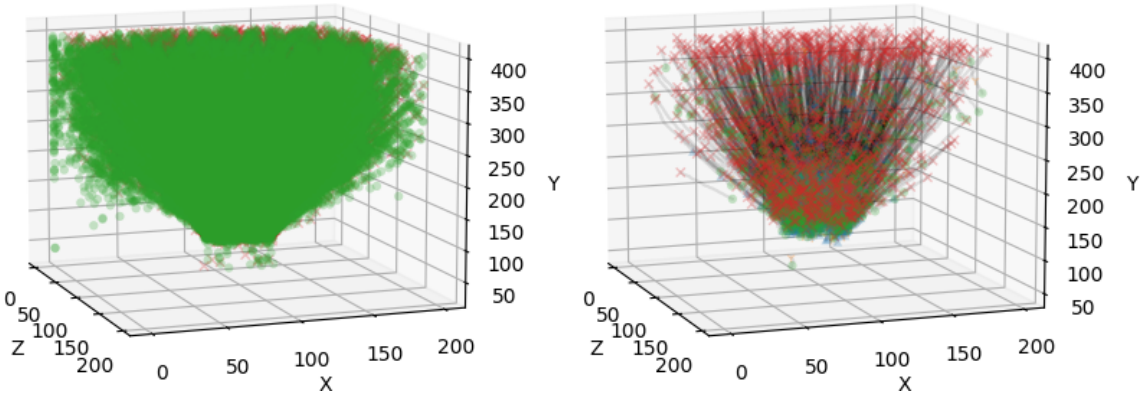
Here the split and merge events are based on the underlying idea of the fluid simulation, in which droplets may separate or combine when interacting between one another. The appearance and disappearance events however were conceptualized as means of measuring the tracking quality. They take note whenever a droplet pops in or out of the analysis, with no prior or following droplet

attached in the graph. In these metrics the first and last frame are exempt edge-cases. In those instances it is expected for droplets to not have a prior or following droplet, as there is no further information available from the simulation data.

Event	Condition
Appearance	$d_{in} = 0$
Disappearance	$d_{out} = 0$
Split	$d_{out} > 1$
Merge	$d_{in} > 1$

Table 4.2: Event classification

Figure 4.9a shows the initial result of this 3D visualization. It reveals a crucial issue with the raw tracking data generated. Many of the traces seem to pop in and out of existence. This is visible at the outer edges of the plume where many short trajectories exist. They are marked with green and red dots on either end, e.g. the appearance and disappearance markers. These are the trajectories of very small droplets, which are inconsistent in their tracking. Small droplets fluctuate around the cutoff threshold during tracking, yielding cut tracks whenever they fall below the threshold.



(a) Unfiltered trajectories

(b) Filtered trajectories. Removed short lived (< 10 frames) and light (always < 10 mass) droplets

Figure 4.9: 3D Trajectories of droplets in the simulation.

Thus the first step in the analysis of the graph is filtering out these small, short lived droplets. This is attempted in multiple ways. First of all the droplets are filtered by their mass. In this case a threshold of about 10 was chosen, where one unit corresponds to one particle of the underlying particle data. If the droplets mass never exceeds this threshold in its lifetime, it is discarded from further analysis. Additionally the droplets are filtered by their life time. A threshold of 10 frames was used to filter out all droplets which have a life shorter than 10 frames.

The steep disconnect between the individual droplets and the central jet was another challenge that arose during analysis of the droplet graph with regard to the droplets. This is due to the fact that droplets and the stream operate on wildly different scales. As the droplets are the subjects of interest in this analysis, the stream was discarded.

After performing these filtration steps, the graph seems cleaner, as can be seen in figure 4.9b. There the trajectories of the droplets are actually visible, compared to the noisy unfiltered data.

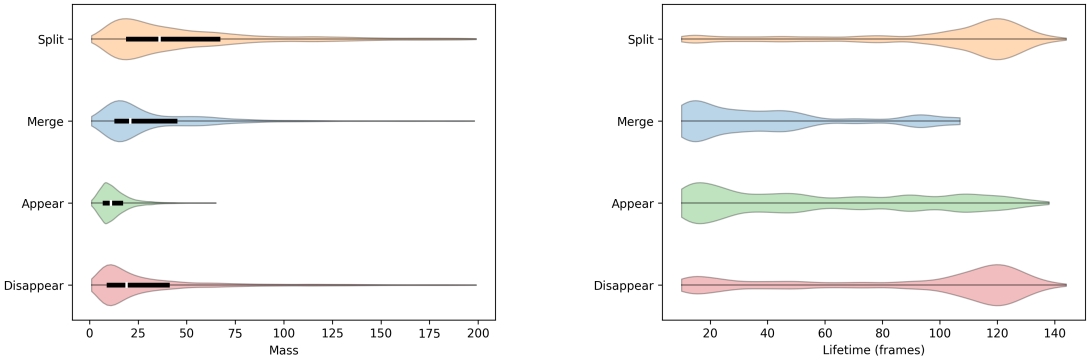
4.3.2 Statistic measures of the simulation

	mean	std	min	25%	50%	75%	max
# Droplets	219.76	130.73	1.00	78.00	308.00	321.00	354.00
# Appearances	78.61	45.92	0.00	41.00	93.00	113.75	147.00
# Disappearances	172.93	115.60	0.00	54.25	185.00	288.00	336.00
# Splits	141.14	89.89	0.00	36.75	187.00	221.75	241.00
# Merges	10.88	6.56	0.00	5.00	13.00	16.00	21.00
Life times	83.48	41.08	10.00	45.00	100.00	119.00	144.00
Droplet mass	13130.87	9383.91	2.00	1994.25	15805.50	22057.50	26107.00
Velocity	2.38	0.22	0.80	2.23	2.36	2.51	3.59

Table 4.3: Statistics of the droplet graph computed over all frames.

Table 4.3 shows descriptive statistics of the droplet graph. For one it summarizes the droplets metrics over the entire simulation. This allows for a quick overview of the simulation. For example the average mass, velocity and count of the droplets each frame can be determined. Furthermore it shows the expected number of frames a droplet lives for.

Lastly it contains some statistics on the graph structure itself. In this case those are the amount of the special events described in section 4.3.1. The number of appearance, disappearance, split and merge events are counted and averaged per frame. Furthermore the frequency of these events is also visualized in figure 4.10 according to different droplet measures.



(a) Event frequency by droplet mass at event

(b) Event frequency by droplet lifetime

Figure 4.10: Frequency of graph events by droplet mass and lifetime.

In figure 4.10a the event occurrence frequency is given according to the droplet mass at the events timestep. The shown masses do not encompass the entirety of the droplets contained within the data, they where constrained to a maximum of 200 mass. This was done due to a prior observation

that the distribution is mainly made up by the lower smaller droplet sizes, while larger droplets contribute less. The appearance and disappearance events are noticeably most frequent for smaller droplets, while the split and merge events are more common on slightly larger droplets.

Another perspective is the frequency of droplet events against the droplets lifetime, as shown in figure 4.10b. The split and disappearance event distributions seam to have a similar shape, with a peak around the lifetime of 120 frames. This peak however does not manifest in the frequencies of the merge and appearance events.

From this it can be concluded that split off droplets which live long are more likely to disappear (for example due to them hitting the simulation boundaries), than interact with other droplets again. Otherwise the merge frequency would sport a similar distribution.

Another global statistic is the droplet lifetime within the tracked data. This is visualized via a histogram, as shown in figure 4.11. One observation is that the droplets which live for a very short time are still in the majority over longer lived ones. This is the case even after prefiltering the data according to section 4.3.1 and removing plenty of short lived and small traces.

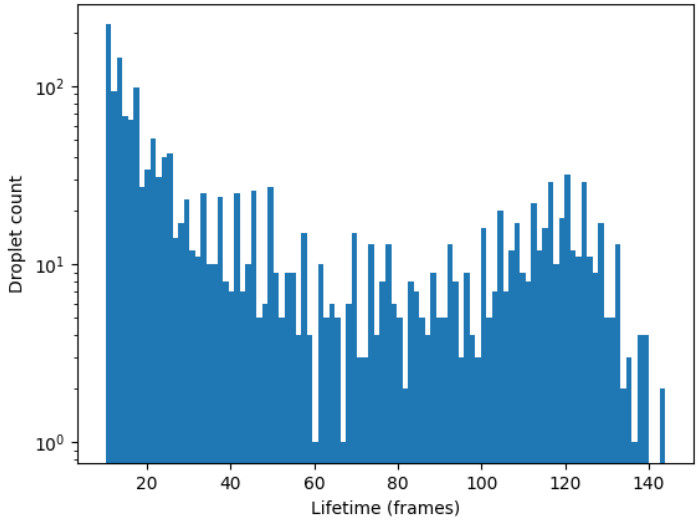


Figure 4.11: Histogram of droplet count by lifetime.

4.3.3 Statistic measures per cluster

Some of the previously computed statistics such as velocity and droplet mass can be determined per cluster instead of per frame as done previously in table 4.3. The distribution of average droplet mass within the clusters is such a per droplet measure and can be represented by a histogram. Such a histogram is provided in figure 4.12.

Firstly, the impact that the prefiltering steps have on the distribution can noticed as the distribution sharply falls off before the 10 unit threshold. This is due to the fact that by filtering out the droplets which never reach a mass of 10 the ones that reach the threshold are more likely to be larger than the

threshold, at least on average. Moreover, it can also be observed that as the droplet size increases, the amount of them decreases. Note however the logarithmic x-axis scaling, as the distribution is heavily skewed towards the smaller droplets.

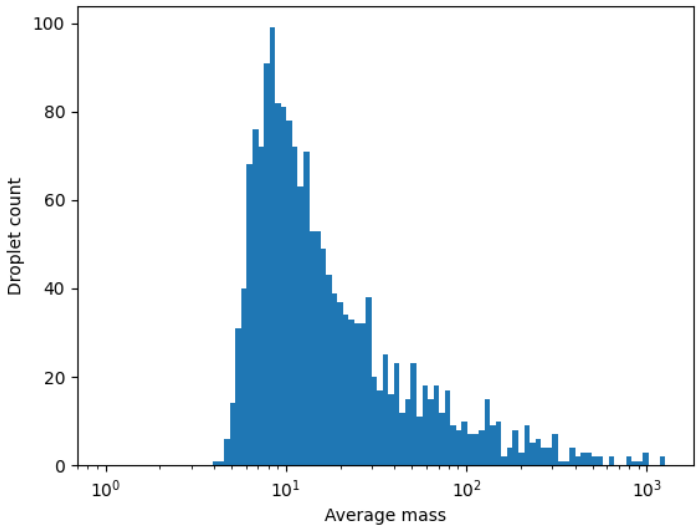


Figure 4.12: Histogram of average droplet mass.

Another metric to analyze would be the droplet velocities. Due to the radial symmetry of the injection process setup, the rotational component along the injection direction can be considered irrelevant and thus subject to dimension reduction. To visualize the 3D velocity distribution a 2D scatter plot was used, in which the velocity component along the injection direction is plotted against the component perpendicular to it. This is exemplified in figure 4.13. This provides a more in depth feature description of the velocity distribution than the average and standard deviation provided by table 4.3.

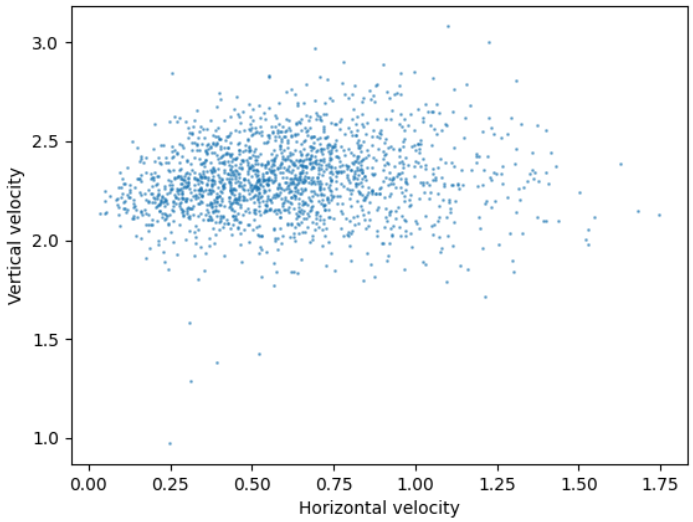


Figure 4.13: Droplet velocity components along and perpendicular to injection direction.

Lastly as mentioned in section 4.2 the droplet mass seems to at least visually relate to the droplet deflection angle. To quantify this the droplet mass is plotted against the estimated deflection angle in a scatter plot. The deflection angle here is determined by the angle between the injection direction (here the y-axis) and the path the droplet took, which is seemingly linear (compare section 4.2) thus making this angle well defined. This is illustrated in figure 4.14. Within this figure the droplet mass seems again to relate to the maximal deflection angle, as the droplets with a higher mass tend to deflect less. However this is not a linear relationship, as the droplets' mass is ordered in logarithmic fashion.

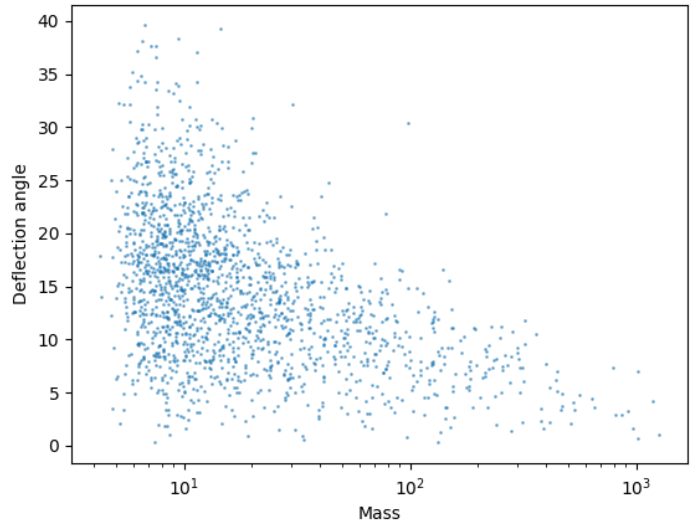


Figure 4.14: Droplet mass vs. deflection angle scatter plot.

4.3.4 Time-Series Plots

The time series plots are used to visualize the evolution of the droplet graph over time in reduced from. This is done by generating 2D plots to capture some of the underlying time dependencies in the data.

One such plot is the number of droplets over time, as demonstrated in figure 4.15. There it can be observed, that the droplet count is initially very low, but at about frame 150 increases rapidly. After rising to a maximum of about 300 - 350 droplets, the simulation domain is saturated, and the rate at which the droplets die off is about the same as the rate at which new droplets are created. This is also visible in the fact that the droplet count plateaus around frame 300.

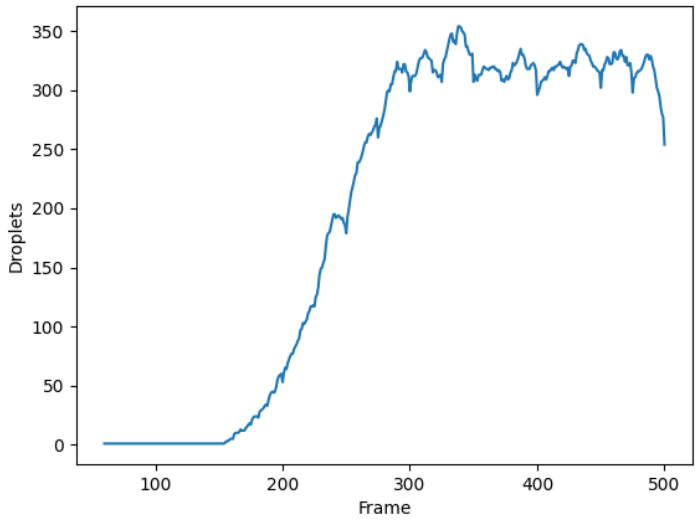


Figure 4.15: Droplet count over time.

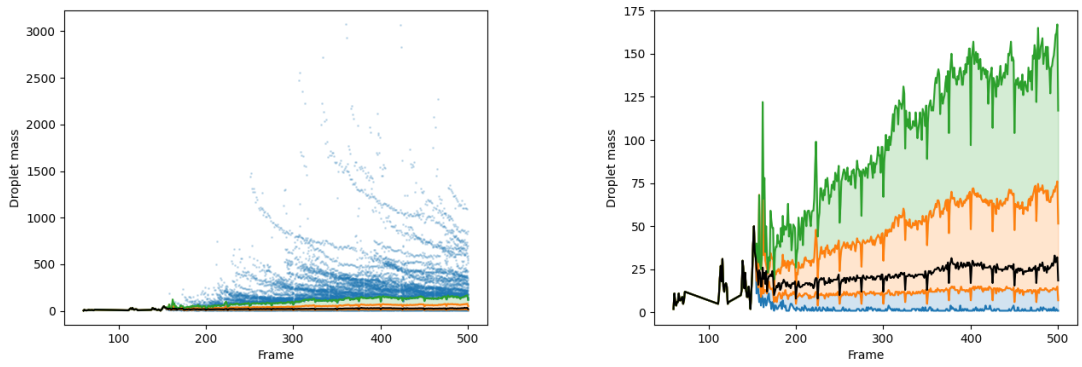
Another metric to consider would be the droplet mass over the simulation time. This is demonstrated in figure 4.16. It is plotted as a functional boxplot to visualize how the droplet masses are distributed over the droplets across time. The functional boxplot is a plot of the regions a normal boxplot would cover, e.g the lower and upper quartile, the median and the whisker range. However the functional boxplot does not just cover 1 frame, but computes these descriptive values for each frame, and joins them with a line.

This means in the given example, the lower whisker region is colored blue, the inter quartile region is colored orange, and the upper whisker region is colored green. The median is colored black and layered on top of the other regions. When the outliers are plotted, the rest of the graph is compressed noticeably, compare figure 4.16a with figure 4.16b.

The visible periodic dips in figure 4.16b occur at the same interval as the periodic batched addition of new fluid particles to the simulation. These periodic batches then reach the end of the nozzle after a fixed amount of time since the particle batch creation. As the batch reaches the nozzle together this leads to the new particles suddenly being able to split into droplets at the the same time.

This creates an influx of new small droplets. In the next frame however they already had enough time to then recombine and interact in the nozzle exit to average out the data again. However this sudden influx is visible in the data, as many small droplets are created at the same time, causing the quartiles and median to drop temporarily. This behavior was also present when observing the simulation animated with the MegaMol bounding box renderer from section 4.2.

Additionally the graphs only start around frame 150, as there do not exist enough droplets to create any meaningful statistics before. This also causes the noisy start of figure 4.16b.



(a) With outliers

(b) Without outliers

Figure 4.16: Droplet mass over time in functional boxplot. Whiskers are colored ●/●, inter quartile range ●, and the median ●

Lastly there are metrics which can be computed over the lifetime of the droplets. This is demonstrated by figure 4.17, which shows the distance traveled by the droplets over their lifetime. It is created by integrating the droplet velocity over the droplets lifetime. When rendered it shows a succinct shape, unique to the simulation parameters.

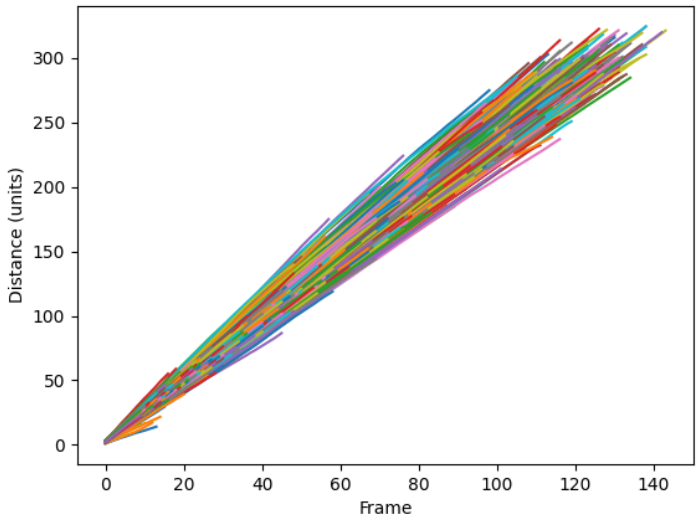


Figure 4.17: Droplet distance traveled over lifetime. Each droplet trajectory is assigned an individual color.

4.4 Comparison of cross domain simulations

Some of the previously mentioned graphing techniques can be used to compare multiple simulations. This is done by creating a plot for each simulation, and then overlaying them on top of each other. One simulation will be assigned one color each, and their graphs will be plotted in that color.

For this example the simulation parameters chosen are the same as was introduced in chapter 4, except the feed velocity being set to $v_{feed} = 0.004$ instead. Then the simulation is tracked for 500 frames. Once in the particle domain, and once in the volumetric domain. In the following the red color ● is used to represent the particle domain. The blue color ● is used to represent the volumetric domain.

One feature that can be compared is the count of different droplet lifetimes within the simulation. This is shown in figure 4.18. It shows that while the simulations behave in approximately the same way, less particles are actually generated in the volumetric domain.

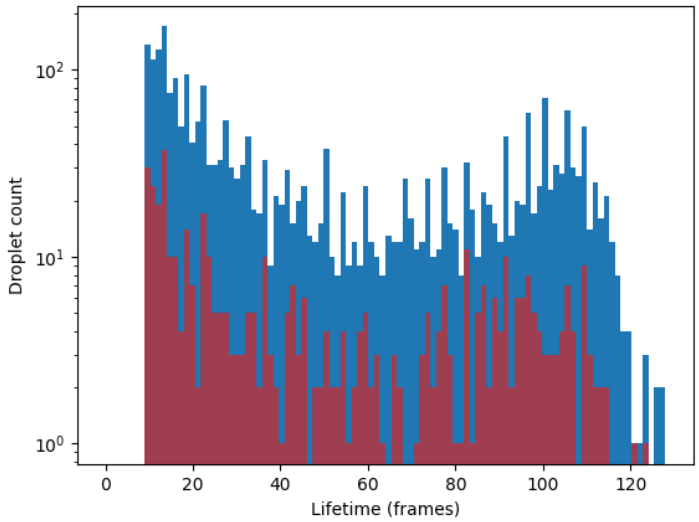


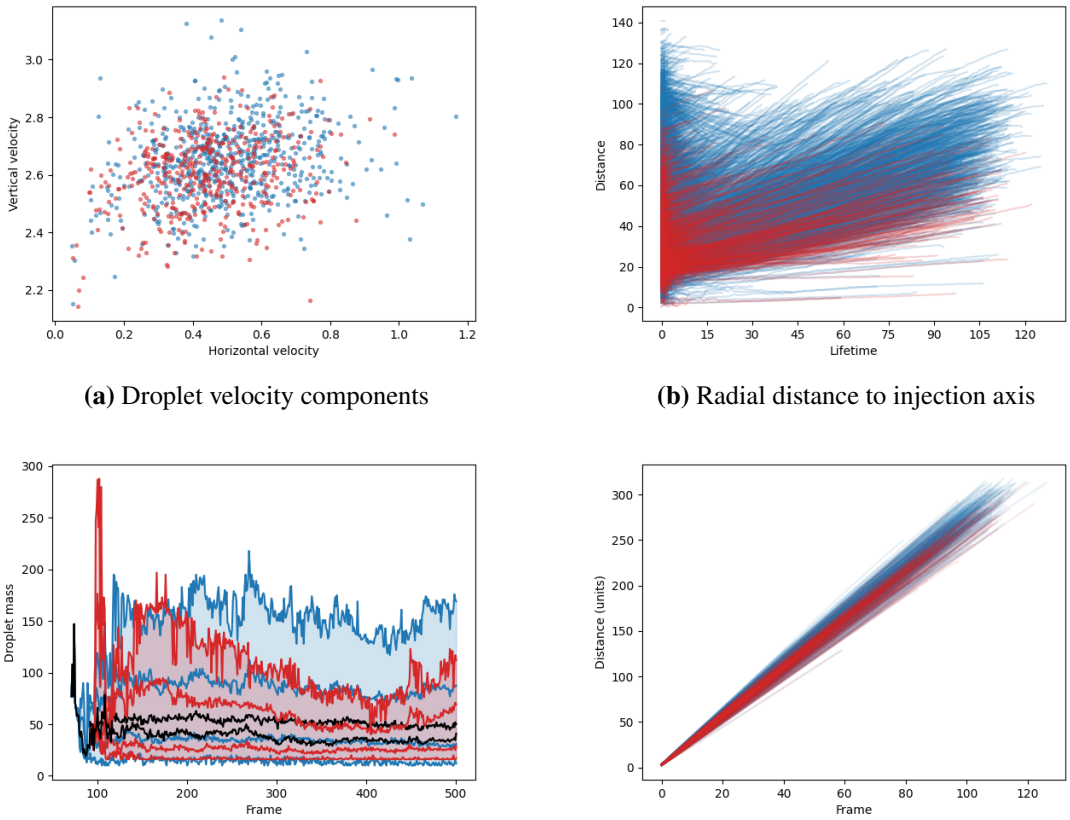
Figure 4.18: Comparison of droplet lifetime count. ● Volumetric vs. ● particle based results.

Another comparison point is the radial distance of the droplets to the injection axis over time. It is measured as the euclidean distance of the x and z components of the droplets center of mass, to the as the axis through the center of the injection nozzle. There we see the impact that the magnitude of difference in droplet count comes into play. The volumetric domain has a much higher amount of droplets, and thus more trajectories plotted than the particle domain we wish to compare it to. This is even the case after prefiltering the data as described in section 4.3.1.

This raises the issue on selection of filtering parameters before the actual comparison between the results. This is a challenge, as when data is filtered, information can be lost in different manner between the tracked results. This can lead to the comparison becoming more brittle and less meaningful, as differences between the simulations might get discarded.

If we observe the plots some of them seem to indicate that the compared simulations are indeed similar. This for example applies to the velocity component scatter plots in figure 4.19a where the velocities seem to be distributed in same fashion, or the paths which are visible in the lower region of the radial distance to the injection axis plotted against the droplets lifetime in figure 4.19b. Lastly figure 4.19d also showcases a great likeness between the data obtained by the particle and volume data.

These plots at least seem to portray at least somewhat similar, if not the same data. To that some noise is seemingly added on to the volumetric data, for one due to the selection of parameters yielding more smaller droplets, as well as the quantization errors produced by the splatting step.



(a) Droplet velocity components

(b) Radial distance to injection axis

(c) Droplet mass over time

(d) Traveled distance over time

Figure 4.19: Comparative figures between ● volume and ● particle domain.

Other parts of the views however seem to deter from the fact that the same data is the basis for both representations. This is the case for the droplet mass over time in figure 4.19c. There the data ranges do not seem to correlate in a visible way, as the particle based data exhibits a considerable dip in the range surrounding frame 400, while the volumetric data does not. Another discrepancy is visible in the upper trajectories of the radial distance figure 4.19b. It seems as though droplets exist in the volume which can travel farther than those extracted from the particle data.

This is most likely caused by wrongly selected parameters during the tracking stage, or issues arising during data filtration which only badly affects the tracks produced by tracking the particle data. Another reason could be that the trajectory data is simply not comparable between the two domains, which is however unlikely, as the other metrics do work for that purpose.

4.5 Performance benefits

For the analysis of particle data one can either utilize the particle based tracking methods, or indirectly via the volumetric tracking methods. The latter is done by first projecting the particle data onto a volume using the splatting methodology described in section 2.2.1 and then tracking the volume data.

Since the volumetric data is well structured it does not require the use of more advanced algorithms and data structures to handle the unstructured nature of the particle data. For querying the particle neighborhood one has to utilize spatial partitioning methods, such as octrees or kd-trees, while for the neighborhood queries on the volume data one can simply use an offset into the underlying voxel array.

The resulting performance boost is visible in the time it takes to perform the tracking show in figure 4.20. There a increase is noticeable somewhere between $2.5 - 5.4\times$ of the time required to perform the tracking on the particle data directly.

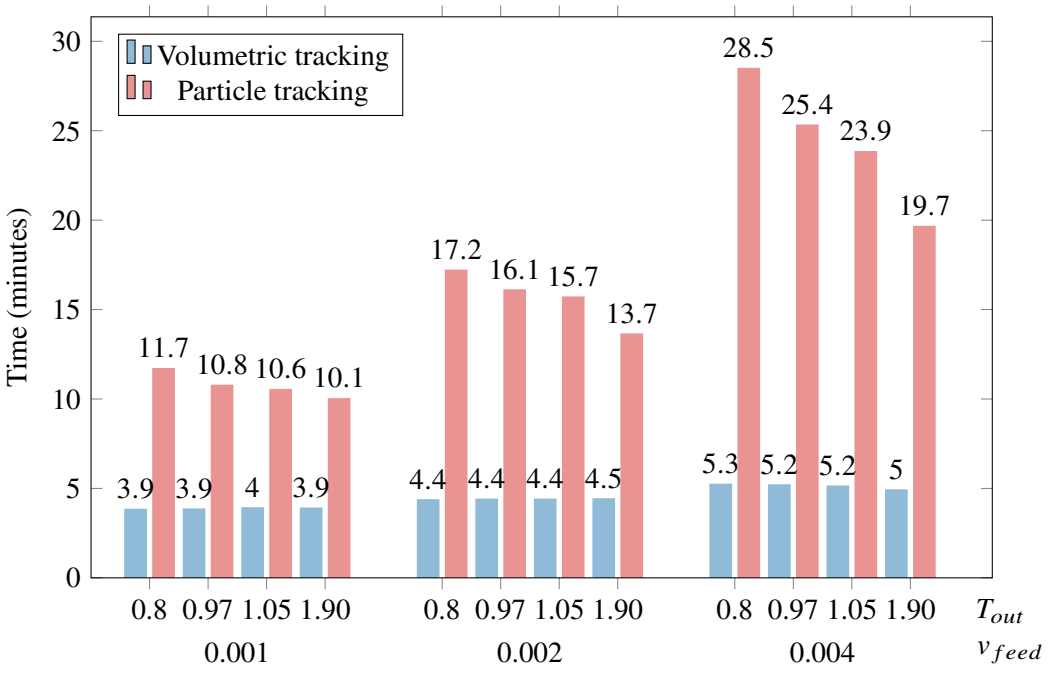


Figure 4.20: Time it takes to track the same simulation data for the different tracking methods.

5 Conclusion and Outlook

In this thesis a droplet-centric approach was presented to perform visual analysis of multiphase flow simulations. The goal of this approach is to provide tooling to the user to analyze and compare fluid simulations in a manner that abstracts away the raw underlying simulation data.

The decision to focus on the droplets as fundamental units within the simulation stems from the of droplet formation in injection processes. These processes were the chosen type of fluid simulation to be analyzed within this work. For this, methods were implemented to try and extract the droplets and their behavior over time from the raw simulation data.

After extracting the droplets some analysis approaches were put in place to enable further investigation into the droplet properties. To analyze the shape of the droplets an algorithm for surface mesh generation was employed, alongside ways to classify the resulting shape in reduced manner.

To be able to study the droplets regarding their behavior and interaction over time, tracking algorithms were applied for matching the droplets between different simulation frames. The found cluster evolutions can than either be visualized within MegaMol, or exported for analysis with external tools. For this a python framework, which can further process the relationships between the individual frame droplets, and in turn generate descriptive statistics for the simulation.

By transferring the supplied molecular dynamics particle simulation of a chaotic injection process to volumetric data, the previously implemented methods for both simulation types were put to the test. In this manner the resulting metrics were expected to be easily comparable, as the same underlying simulation data should yield the same results, regardless of the type of data used as input. However the extracted data posed difficulties when trying to compare data between simulation types. This is due to the fact that the different algorithms are parameterized differently, and a suitable set of variable assignments for extracting droplet data in similar magnitudes was not yet found.

None the less some exemplary insights could be made when analyzing a simulation on its own. These are for example that the maximal angle a droplet is deflected is determined by its mass (compare figure 4.14), or that the droplets tend towards spherical shapes over time (see analysis of figure 4.4).

The methods to achieve 1. surface extraction, segmentation by loose parts and shape analysis 2. projection of the particle data into a volume 3. clustering a volume into regions 4. tracking clusters over time for both particle and volumetric data were all implemented as new modules for MegaMol, and are provided as free open source software. They should be general enough to not only be able to find use in this specific example, but other applications as well. For example the isosurface extraction can be applied whenever a discrete surface is required to be extracted from a volume.

Outlook

Further investigation is required into the subject of performing cross domain simulation comparisons. While this thesis tries to provide the analyst with tools required to perform such work, it fails at successfully applying the methods for a comparison. Additionally as similarly enough setup simulations did not exist for proper comparison, these methodologies are also still unproven against true cross domain analysis. In this regard a proper search for physically motivated thresholds for all the algorithms might show promise, when trying to apply the same methodologies as in this work.

Another point of interest would be optimizations of the deployed algorithms. As it stands they have quite a large run time, here at best about 400ms per analyzed frame, before the results can be viewed or further processed. This is fine when the required parameters are known, as the precomputation of the droplet graph is then only needed to be performed once. However due to high iteration times explorative users are hindered from tinkering with the parameters when trying to obtain a satisfying and complete but not over specified visualization.

Lastly due to time constraints not all of the views were yet linked. For example some of the analysis capabilities that exist within the python framework can still be transferred to MegaMol, such as the visualization of different droplet events like splitting and merging. On top of that, the mesh extraction pipeline is still missing connection to the actual droplet tracks. This means that the measures extracted from the surface such as those measuring the particle shape, are constrained for now to only the surface visualization. Moreover the particle tracking still can only output the older format, which while it has the same information content, is magnitudes slower to parse.

Bibliography

[1] X. Tao, Z. Qingwei, L. Zicheng, W. Houneng. "Numerical Simulation and Analysis of Electronic Jet Printing Injection Process". In: *2020 5th International Conference on Control, Robotics and Cybernetics (CRC)*. 2020, pp. 250–255. DOI: 10.1109/CRC51253.2020.9253474 (cit. on p. 11).

[2] P. Gralka, M. Becher, M. Braun, F. Frieß, C. Müller, T. Rau, K. Schatz, C. Schulz, M. Krone, G. Reina, T. Ertl. "MegaMol – A Comprehensive Prototyping Framework for Visualizations". In: *The European Physical Journal Special Topics* 227.14 (Mar. 2019), pp. 1817–1829. ISSN: 1951-6401. DOI: 10.1140/epjst/e2019-800167-5. URL: https://doi.org/10.1140/epjst/e2019-800167-5 (cit. on p. 11).

[3] M. Heinen, J. Vrabec. "Evaporation sampled by stationary molecular dynamics simulation". In: *The Journal of Chemical Physics* 151.4 (2019), p. 044704. DOI: 10.1063/1.5111759. eprint: https://doi.org/10.1063/1.5111759. URL: https://doi.org/10.1063/1.5111759 (cit. on pp. 11, 14, 17, 27).

[4] K. Potter, A. Wilson, P.-T. Bremer, D. Williams, C. Doutriaux, V. Pascucci, C. R. Johnson. "Ensemble-Vis: A Framework for the Statistical Visualization of Ensemble Data". In: *2009 IEEE International Conference on Data Mining Workshops*. 2009, pp. 233–240. DOI: 10.1109/ICDMW.2009.55 (cit. on p. 12).

[5] Y. Tsuji, T. Tanaka, S. Yonemura. "Cluster patterns in circulating fluidized beds predicted by numerical simulation (discrete particle model versus two-fluid model)". In: *Powder Technology* 95.3 (1998), pp. 254–264 (cit. on p. 12).

[6] E. L. Knuth, U. Henne. "Average size and size distribution of large droplets produced in a free-jet expansion of a liquid". In: *The Journal of Chemical Physics* 110.5 (1999), pp. 2664–2668. DOI: 10.1063/1.477988. eprint: https://doi.org/10.1063/1.477988. URL: https://doi.org/10.1063/1.477988 (cit. on p. 12).

[7] J. W. Alred, N. L. Smith, K. Wang, F. E. Lumpkin, S. M. Fitzgerald. "Modeling of Water Injection into a Vacuum". In: *Proceedings of the Eighth Annual Thermal and Fluids Analysis Workshop: Spacecraft Analysis and Design*. 1997 (cit. on p. 12).

[8] G. K. Karch, F. Beck, M. Ertl, C. Meister, K. Schulte, B. Weigand, T. Ertl, F. Sadlo. "Visual analysis of inclusion dynamics in two-phase flow". In: *IEEE Transactions on Visualization and Computer Graphics* 24.5 (2017), pp. 1841–1855 (cit. on p. 12).

[9] V. V. Belikov, A. Y. Semenov. "Non-Sibsonian interpolation on arbitrary system of points in Euclidean space and adaptive isolines generation". In: *Applied Numerical Mathematics* 32.4 (2000), pp. 371–387. ISSN: 0168-9274. DOI: https://doi.org/10.1016/S0168-9274(99)00058-6. URL: https://www.sciencedirect.com/science/article/pii/S0168927499000586 (cit. on p. 20).

[10] C. H. Rycroft. “VORO++: A three-dimensional Voronoi cell library in C++”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 19.4 (2009), p. 041111. DOI: 10.1063/1.3215722. eprint: https://doi.org/10.1063/1.3215722. URL: https://doi.org/10.1063/1.3215722 (cit. on p. 21).

[11] D. Marshall. *Region Growing*. Accessed: November 14, 2022. 1994. URL: https://users.cs.cf.ac.uk/Dave.Marshall/Vision_lecture/node35.html (cit. on p. 21).

[12] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In: *kdd*. Vol. 96. 34. 1996, pp. 226–231 (cit. on p. 22).

[13] T. Ju, F. Losasso, S. Schaefer, J. Warren. “Dual Contouring of Hermite Data”. In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’02. San Antonio, Texas: Association for Computing Machinery, 2002, pp. 339–346. ISBN: 1581135211. DOI: 10.1145/566570.566586. URL: https://doi.org/10.1145/566570.566586 (cit. on p. 23).

[14] G. Guennebaud, B. Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010 (cit. on p. 24).

[15] R. E. Tarjan. “Efficiency of a Good But Not Linear Set Union Algorithm”. In: *J. ACM* 22.2 (Apr. 1975), pp. 215–225. ISSN: 0004-5411. DOI: 10.1145/321879.321884. URL: https://doi.org/10.1145/321879.321884 (cit. on p. 24).

[16] C. Zhang, T. Chen. “Efficient feature extraction for 2D/3D objects in mesh representation”. In: *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*. Vol. 3. 2001, 935–938 vol.3. DOI: 10.1109/ICIP.2001.958278 (cit. on p. 24).

[17] H. Wadell. “Volume, Shape, and Roundness of Quartz Particles”. In: *The Journal of Geology* 43.3 (1935), pp. 250–280. DOI: 10.1086/624298. eprint: https://doi.org/10.1086/624298. URL: https://doi.org/10.1086/624298 (cit. on p. 25).

[18] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55 (cit. on pp. 26, 33).

All links were last followed on November 11, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature