Eurographics Symposium on Parallel Graphics and Visualization (2021)
M. Hadwiger, M. Larsen, F. Sadlo (Editors)

Interactive Selection on Calculated Attributes of Large-Scale Particle Data

B. Wollet[1,2], S. Reinhardt[1,2], D. Weiskopf[1], and B. Eberhardt[2]

[1]University of Stuttgart, Germany
[2]Hochschule der Medien, Germany

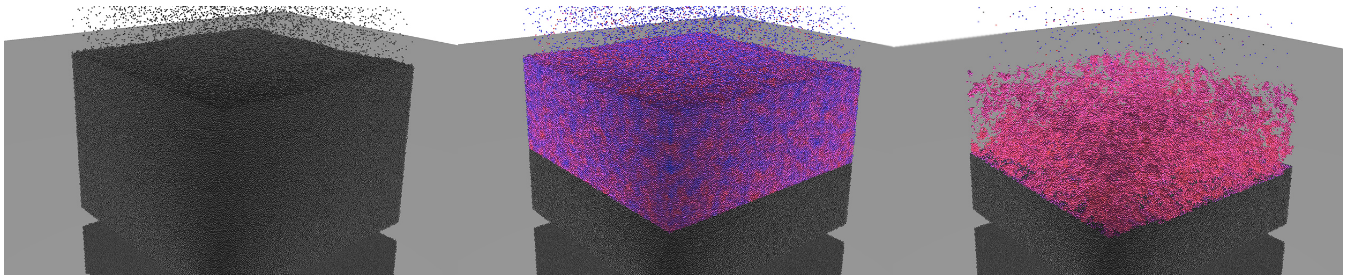

Figure 1: *Visualizations of the evaporation scenario we used to evaluate our method. On the left, the unselected dataset is shown. Selecting particles on the liquid-gas interface and extending the selection on the neighborhood temperature results in selecting hot spots on and beneath the interface area. The non-selected particles can be hidden to investigate hot spots within the liquid phase (on the right). The lazy evaluation, including color-coding (in the center image), enables interactive selection and context-sensitive region growing.*

Abstract

We present a GPU-based technique for efficient selection in interactive visualizations of large particle datasets. In particular, we address multiple attributes attached to particles, such as pressure, density, or surface tension. Unfortunately, such intermediate attributes are often available only during the simulation run. They are either not accessible during visualization or have to be saved as additional information along with the usual simulation data. The latter increases the size of the dataset significantly, and the required variables may not be known in advance. Therefore, we choose to compute intermediate attributes on the fly. In this way, we are even able to obtain attributes that were not calculated by the simulation but may be relevant for data analysis or debugging. We present an interactive selection technique designed for such attributes. It leverages spatial regions of the selection to efficiently compute attributes only where needed. This lazy evaluation also works for intelligent and data-driven selection, extending the region to include neighboring particles. Our technique is evaluated by measurements of performance scalability and case studies for typical usage examples.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ ***Visual analytics;*** • ***Human-centered computing*** $\rightarrow$ ***Visualization design and evaluation methods;*** *Visual analytics;*

1. Introduction

Particle-based simulations are widely used in many research disciplines, ranging, e.g., from molecular dynamics to astrophysics. They have also gained importance in the field of computer graphics to generate physically based simulations, e.g., of cloth or fluids. On the one hand, increased memory and computing power have led to more precise and faster simulations that can be calculated with an ever-increasing number of particles. On the other hand, this makes an efficient visualization and analysis of the dataset increasingly challenging.

There are many different techniques to visualize the particle positions and associated data, as well as intermediate attributes created during, and saved with, the simulation. For post-hoc analysis, this requires many attributes of the particle data to be stored on disk, which on the one hand increases the amount of disk space required, but, more importantly, also reduces the efficiency of high-

DOI: 10.2312/pgv.20211045

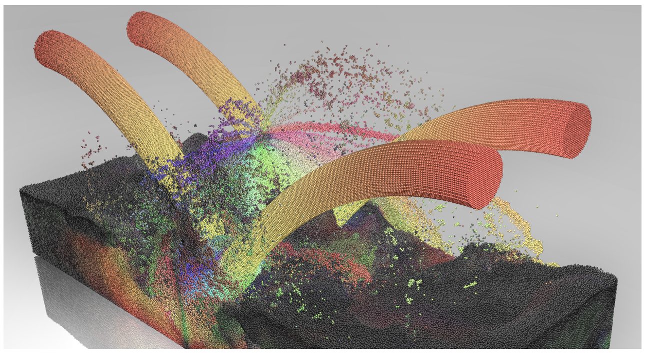

Figure 2: *The SPH scenario used to evaluate our selection technique. In this scenario, four inflow zones are defined that form the liquid jets filling the pool and hitting static and dynamic rigid boundary objects. It consists of up to 10M particles.*

performance rendering techniques. Moreover, insights gained only later during the analysis process may trigger interest in attributes that were not even computed during the simulation. To calculate these, either a rerun of the simulation or a modification of the simulation code would be necessary, whereby only the desired attributes would have to be calculated. Regardless of that, the desired attributes are often needed only in a particular area of interest, e.g., they are needed to find critical areas, such as local pressure maxima or shock waves in the dataset. Detecting irregularities within the simulation domain or evaluating new techniques often involves analyzing the temporal domain, i.e., critical areas must be tracked over time to analyze their development. Typical objectives of this kind are, for example, the analysis of compressibility, surface tension, or new boundary handling strategies.

In this paper, we combine brushing on a 3D-view with a data-driven selection-extension on lazily evaluated attributes. These attributes are calculated from spatial or temporal coherence (e.g., density, pressure, velocity, or local density variance) to perform intelligent brushing. In particular, attribute computations that require a neighbor search are supported. The extension of selected particles can be executed repeatedly to reach regions of bigger, smaller, or similar attribute values. They can then be isolated in combination with our rendering engine. After finishing a selection, these particles' temporal evolution can be tracked when using particle data with identifiers or consistent enumeration through several time steps, as shown in Fig. 3. Our method calculates the attributes required for extension upon each new extension step in a lazy fashion and allocates necessary memory only when needed. It is therefore designed to suit the analysis of large-scale datasets. Besides, our technique is conceptually split into different execution steps to offer easy integration for repetition cycles to create attributes based on other attributes or easily add additional calculation methods. The main technical contributions of this paper are:

- Intelligent brushing using attribute-based region growing
- Lazy evaluation of attributes within particle data triggered on demand by selection extension
- Dynamic memory allocation on evaluating attributes within subsets of large-scale data

We evaluate our technique by measuring scalability and through two use cases involving large-scale particle data. A simulation state of the first scenario is depicted in Fig. 2, containing particles created during an SPH-based fluid simulation. We analyze this scenario with regard to pressure shocks. The second scenario is shown in Figure 1. It is a molecular dynamics simulation of fluid evaporation. In this scenario, temperature hot spots in the vicinity of the interface are searched to investigate the temperature's influence on the fluid's evaporation behavior.

Our technique is implemented in CUDA and provided as a plugin for MegaMol [GKM*15], which enables basic user interaction. It can be easily adapted to work with other visualization frameworks using CUDA. Particle rendering of our use cases is performed using Nvidia OptiX [PBD*10] to provide high flexibility on possible rendering techniques while creating high-quality pictures.

2. Related Work

The importance of combining various aspects of multidimensional data within visualizations of particle-based data has been underlined and demonstrated by multiple visualization frameworks. The open-source rendering framework MegaMol [GKM*15] facilitates the visualization of raw particle data in real-time, presenting several 2D plots and 3D views. It provides modules for CPU and GPU-based rendering, relying on the work of Reina et al. [RE05] and optimizations of Grottel et al. [GRDE10]. Reinhardt et al. [RHD*17] presented additional possibilities to visualize already existing attributes for debugging purposes, using brushing-and-linking between 3D and 2D views, as well as dynamic color-coding for available simulation attributes. Price [Pri07] provides similar functionality for SPH simulation data in his interactive visualization tool SPLASH. With SPLASH it is possible to render a previously selected set of particles, tracking this set through multiple time steps and calculating additional quantities not dumped with the original dataset but limited to a particle's information scope. In contrast, we enable the computation of attributes that depend on the information of the neighboring particles. Linsen et al. [LMD*11] introduced SmoothViz as an interactive visual data analysis tool with increased performance in complex 3D plots leveraging OpenGL. They include cluster-extraction on densities and offer additional plots for multidimensional properties and coordinated views while depending on provided attributes saved during simulation time. SmoothViz partly supports multi-threaded computation on GPUs [MFR*13]. SmoothViz and SPLASH both provide isosurface representations and vector field visualizations.

Simulation runs produce increasingly larger amounts of data, and their sheer volume in combination with limited computational resources requires optimized memory usage to create scalable visualization solutions. Although region growing was originally developed for image segmentation [HK98] and later adapted for data classification in three-dimensional data [HM03], it is still relevant for extracting subsets of given datasets from given seed points. The majority of applications can still be found to classify medical data from CT or MRI [SBSG06, TLM05]. However, post-hoc extracting features of interest from simulation data is still under active research. In flow visualization and analysis, Sauer et al. [SYM13] used region growing within voxel data structures to classify and

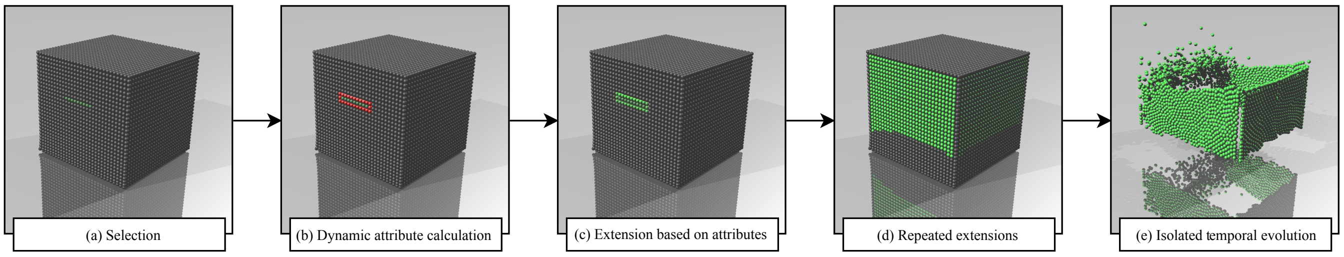
(a) Selection
(b) Dynamic attribute calculation
(c) Extension based on attributes
(d) Repeated extensions
(e) Isolated temporal evolution

Figure 3: *General workflow of our selection method. First, regions of interest manually are defined by selecting particles (a). Next, the needed attributes are calculated in the local neighborhood (b), and the selection is extended based on the result of the calculation (c). This process can be repeated either manually or automatically till no further particles are selected with the defined condition (d). The selection is consistent throughout different time steps. That enables the investigation of the temporal evolution of the selected regions (e).*

extract features. Sauer et al. [SYM14] extended the analysis of extracted features from joint particle and volume datasets into temporal correlations of recognized features to enable tracking through several timesteps. In contrast to our approach, they match both volumetric and particle data to extract features on the CPU.

Recent research picked up region growing for simulations in smoothed particle hydrodynamics. Jiang et al. [JSZ18] analyzed the behavior of separation during dissolution leveraging GPU-based region growing within their simulation. Additionally, selecting data on volumetric displays is using either brushing or adaption to lasso selections as described by Yu et al. [YEII12]. The extension of selected regions can then be done automatically, based on available contexts, as proposed by Sakou et al. [SWB15] within dense volumes. The extension process is conceptually similar to our solution. Still, it limits the decision of whether the selection should be extended to medians of potentially relevant, non-visual attributes present within the provided dataset.

The need for interactive use of features for selection within the visualizations of large datasets was discussed by Doleisch et al. [DGH03]. They presented a solution to use brushing within linked 2D-views of logically combined features to select data presented in a linked 3D-view for analysis. Their solution also allows for creating complex combinations of logical AND or OR selections of attributes existing within the dataset. Their research also confirms the need to consider the spatial context for selections and focus on the differentiation between selected and unselected contexts using different coloring. However, they limit selection possibilities to 2D-views and the combination of attributes to logical operations, not combining the characteristics of spatial-correlated elements to new ones. This limitation has been overcome by Jones et al. [JMEL08], who presented a visualization solution that uses linked views on physical and derived attributes for flows and uses temporal and spatial correlation as a basis for derived attributes. However, their solution does not calculate attributes lazily when required, nor does it allow for region growing of already selected attributes.

Derived field generation was introduced as a multi-core solution by Harrison et al. [HNM*12] and is included in today's production tools like ParaView [AGC05] and VisIt [CBW*12]. It has recently been expanded to the dynamic creation of logical combinations using just-in-time compilation systems by Ibrahim et al. [IHL20]. They provide a set of primitives that enable the user to define expressions to derive visualization attributes without the necessity to recompile the application. Our approach differs from theirs, as we improve performance by limiting the field derivation to a spatial subset and use region-growing on these subsets.

3. Data Characteristics and Design Goals

Especially particle-based simulations with a large number of particles generate very large datasets. The storage of additional "calculated, intermediary attributes" is not feasible due to the extra memory needed. Especially for very large-scale data, the possibility of saving intermediary attributes along with simulation results is limited, as it drastically increases the additional amount of data to be stored, transferred, or processed. This paper presents a solution for interactively selecting subsets of these datasets with or without intermediary attributes saved during simulation time. However, we restrict the investigations to restoring attributes that can entirely be restored using neighboring particles, their positions, and attributes. In the following, we first describe our input data's typical characteristics and develop design goals to match typical use cases for our solution.

Since we do want to support all kinds of renderers of particle-based data that support selection, the interfaces must be limited to common properties of particle-based simulations. Therefore, we assume to have particles consisting of three-dimensional position data, color information, and optional attribute data saved during simulation time as input parameters. When calculating attributes, typical operations, kernels, and all accessible data saved during the simulation need to be accepted as input values.

Using any available data as input adds the possibility of either reconstructing intermediary attributes used during simulation time or calculating additional attributes. The necessity of some possible attributes may not even be known during simulation time. It can solely be linked with the analysis that is intended to be performed on the dataset, i.e., downstream local variance analysis on attributes. Although not focusing on this aspect, this behavior adds the feature to use properties of a previous or following frame of time-dependent datasets to our solution, enabling the creation of additional time-dependent attributes.

Our interactive selection method on calculated attributes is intended to isolate particles of specific characteristics within a local neighborhood to explore regional evolutions through the dataset. These evolutions can be better investigated by either isolating the full subset of similar particles in one step or by observing the selection's progression through the data. Therefore, it is necessary to either stop on each extension step or have the full extension executed at once.

Aiming for improvements in the analysis of large-scale data, our solution needs to do be able to limit extension to a defined subspace. This additional restriction can be necessary because of small or reduced execution resources to achieve higher performance or intentionally limit the extension to a predefined area.

Based on the expected input data, usages, and addressed problems, we identify three main functional requirements (R1-R3) and two non-functional requirements (R4-R5):

R1 Processing 3D position and optional color data
R2 Lazy evaluation of attributes on data
- provided by the dataset
- calculated from provided attributes by the dataset
- calculated from calculated attributes

R3 Extension of the particle selection either step-by-step or to the full extent, selecting all matching neighbors' neighbors at once
R4 Restriction to manually created spatial subsets of input data
R5 Good scalability for large-scale data

4. Method

This section describes our technique for visual analysis of particle-based data on dynamically calculated particle attributes. Our interactive selection is started brushing on a 3D-view, as shown in Fig. 3(a). After this manual selection is performed, an attribute and a comparison, e.g., extend 'to smaller than' or extend 'to the maximum value' needs to be chosen. Based on this manual input, all necessary attributes for neighboring particles are dynamically calculated. These particles are highlighted in red color Fig. 3(b). Depending on the actual values and comparison chosen, the selection is then extended to the neighboring particles. This step can be repeated until the desired state is reached or no more new particles are selected. A fully extended selection of similar values of the calculated density is shown in Fig. 3(d). When using particle data with identifiers or consistent enumeration through time, the selected particles' temporal evolution can be examined. Fig. 3(e) illustrates the evolution within another frame, additionally isolating the selected particles. Particle isolation, i.e., not rendering unselected particles, is available with or without temporal evolution using our provided rendering engine.

We split our method into four main steps to address issues of scalability and compatibility with future attribute calculation or extension methods. An overview of the four execution steps is given in Fig. 5, illustrating the systematical data flow, including possible repetition cycles. The first step imports the data from a new manual (interactive) selection or previously saved particle selections into the general workflow. The second step is preparing a possible extension by calculating all necessary attributes for the following repetition cycle. As the extension is done in parallel within the third step, the last step consolidates the selection data. Fig. 5 illustrates this process for two selected particles and their neighbors in detail.

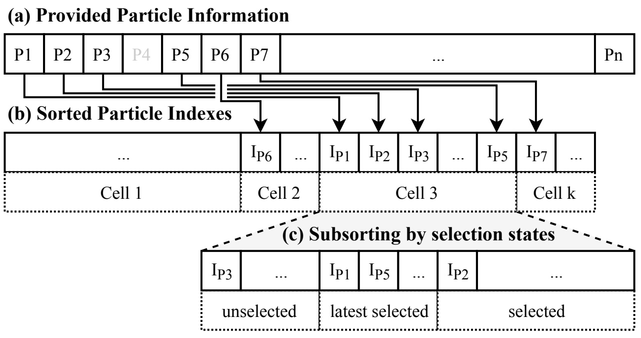

Figure 4: *Indexed Spatial Decomposition with particle indexes (I_{P1} to I_{P6}) being sorted into buckets of precalculated sizes by counting sort (b) and Subsorting with indexes being additionally sorted within a cell by their particles' selection states ((c), as discussed in Section 4.2.2)*

Throughout the process, we need to know the adjacent, i.e., neighboring particles to the selection. For this purpose, we use spatial decomposition, as described in the following section.

4.1. Spatial Decomposition

There is a wide variety of possible methods available to realize spatial decomposition necessary for neighborhood search. Our solution builds a regular grid either for the whole or a user-defined subspace and sorts all particle indexes in this space into the grid, applying a z-curve hash. Although we limit the evaluation to a regular grid, our method can be adapted to any index-based spatial decomposition, e.g., octrees or kd-trees, with an appropriate, and potentially more costly, neighbor search. Even though this approach creates lots of scattered read-operations during sort operations, it enables us to obtain all data, e.g., positions, or optional pre-calculated attributes, using only one single integer per particle as an index. We use counting sort as a sorting-mechanism because we expect the user to be interested in uniform areas of either higher particle densities or less dense areas with an increased search radius. For either case, we expect the possible number of particles to be much higher than the number of cells we sort them into. For selection extension in dense areas of generally huge sparse datasets, we offer the possibility to define a subspace of the grid to increase performance. Particles outside of the boundaries of the subspace will be omitted during the sort. This behavior addresses the requirement *R4 - Restriction to manually created spatial subsets of input data*, defined in Section 3 and is pictured for particle *P4* in Fig. 4.

Note that our non-destructive sorting process is similar to the counting-sort process described by Green [Gre10], but calculating the spatial hashing value on the fly, saving only the position within the original data instead of a tuple of spatial hashing value and index. Although this solution is inferior to counting-sort with included radix-sort in computation time, we favor this solution to reduce the necessity of having either additional spatial hashing

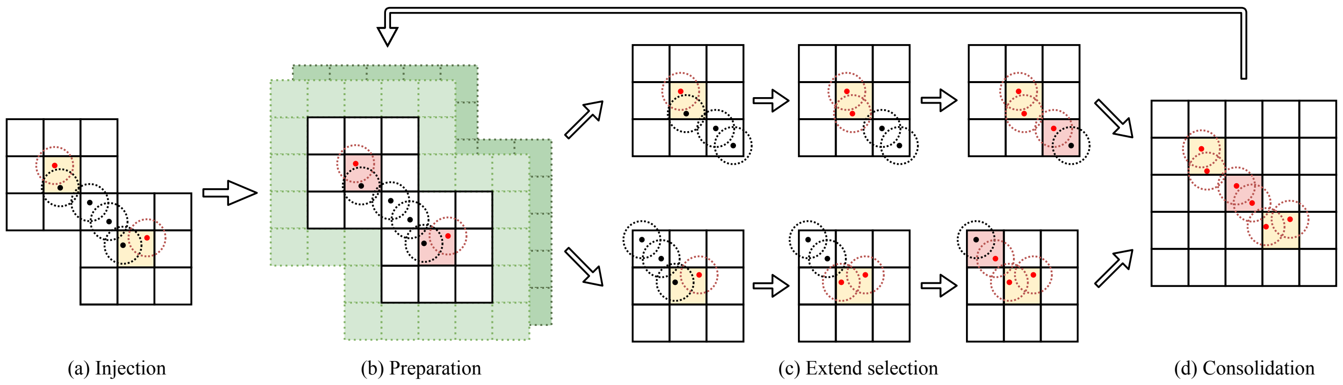

Figure 5: *Systematic flowchart of our three repeatedly executed computation steps and the data injection. Manually selected particles are imported into the workflow (a), necessary attributes are calculated (b), the selection is extended for each cell (c), and the results are consolidated (d). If requested, another extension is triggered.*

operations or reserve additional memory for the necessary data-tuple. As described by Corman et al. [CLRS09], counting-sort is divided into a count-phase, during which the occurrences are counted into an auxiliary data structure and a sorting-phase where this data structure is used to sort the data. The sorting step itself is destructive on the intermediately generated auxiliary array during sorting. This creates the necessity of a partitioning phase, as described by Green [Gre10]. To omit this partitioning phase, we extend the parallel prefix sum by Harris et al. [HSO07], creating a copy during the consolidation phase, and keep this copy as it stores the pointer offsets to every cell and the number of particles per cell. These resulting indexes of cell boundaries within the sorted particle indexes ensure parallel access of $O(1)$ on each particle by cell id. We proceed with counting sort as defined by Corman et al. [CLRS09] on the original count array. The additionally needed memory, which is one of the drawbacks of using counting-sort, is used afterward to store links to our dynamic data structure as described in Section 4.2.1.

The resulting indexed spatial decomposition is visualized as a data structure in Fig. 4(b).

4.2. Selection-Growing Based on Attributes

After selecting regions of interest in the 3D view by brushing, the selection can be extended using provided or dynamically calculated attributes with scalar values. The selection is extended to particles with similar values (within an adjustable range), all particles with smaller or larger values, or the one particle in the neighborhood with the maximum or minimum value.

Extending the selection to neighboring particles is executed by the steps ‘Extend selection’ and ‘Consolidation’, as shown in Fig. 5(c) and Fig. 5(d). The actual extension is executed for each cell containing selected particles, ignoring interdependencies between cells. During this parallel process, the particle selection might be extended to other particles within one cell or neighboring cells within separate parallel execution scopes that match the same particles in other parallel executions. Fig. 5(c) illustrates the issue of two particles within the same cell being selected from different parallel executions in detail, visualizing the necessary consolidation. Efficiently consolidating data from separate parallel execution scopes makes high demands on used data structures that are discussed in the following section.

4.2.1. Used Data Structures

According to our requirement *R5 - Good scalability*, as described in Section 3, we add as little as possible additional data while keeping the original information intact.

We use an additional dynamic data structure that scales with the number of cells that contain relevant particles during the extension and another dynamic data structure that scales with the number of calculated particle attributes. Particle attributes are stored for each particle within our *Dynamic Cell Information Offset* by cell, keeping particles’ order, and storing the cells attributes’ index as a field within our dynamic cell information. This dynamic data structure creates an organizational overhead of 64 bits per cell used during extension. The essential link between our static cell information available for all cells and our dynamic cell information is stored in the *Dynamic Cell Information Offset* as shown in Fig. 6. Using already allocated intermediary memory by counting sort, as described in Section 4.1, we compensate the overhead of 32 bits for each cell within the subspace for analyzing and, therefore, the general drawback of memory consumption of counting sort within our implementation.

Synchronization between different scopes of parallel execution is done using flags embedded into every data structure mentioned above. We introduce two synchronization flags within our static cell information. One flag uses the least significant bit and marks a cell-content as dirty to sync between the two execution phases, extend selection and consolidation. This flag is labeled with the letter ‘S’ in Fig. 6. Another flag, labeled with the letter ‘A’ in Fig. 6, dynamically uses one to three bits, indicating whether attributes for this cell are already available. This flag’s dynamic size reflects the requirement *R2 - Lazy evaluation of attributes on data*, defined in Section 3, limiting the maximum capacity to eight attributes per

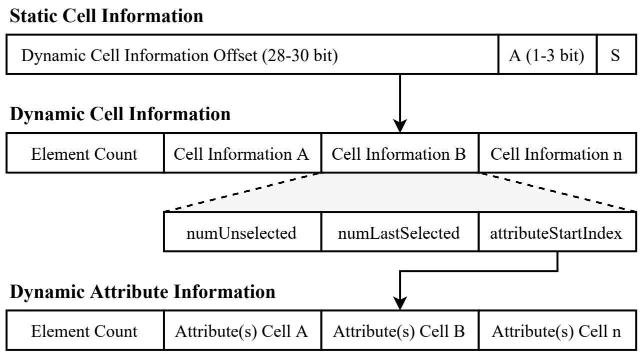

Figure 6: *Structure of dynamic particle data in global memory. The static cell information (top), with two flags keeping information on whether a cell needs attributes or resorting. The dynamic cell information (middle), with its included fields and the reference to the dynamic list of lazily evaluated attributes (bottom).*

particle, and giving at least 28 remaining bits to our dynamic cell information. Our solution can address more than 250 million cells containing dynamically calculated attributes, not making a relevant sacrifice to our scalability requirement without using additional memory.

In addition to data structures that are needed overhead for our solution, the results, i.e., whether a particle is selected, need to be stored. Our solution distinguishes two states of selection, internal and external selection, as described in Section 4.2.2, which needs two bits of additional memory. Contrary to our requirement to work non-destructively on the provided data, we utilize the two least significant bits of each particle's alpha-channel to store this information. This solution is a trade-off to our requirement of adding the least possible amount of additional data. However, it fits all standard particle-rendering systems, as alpha-rendering is commonly not supported because of performance reasons. In addition to this static data structure, we use dynamically extended memory to store information on attributes and enable on-device communication between our solution's different execution steps.

4.2.2. Particle Selection States

To exclude already selected particles from being checked during further extension steps and enable synchronization between parallel execution of cells to extend particles from, we introduce three categories of particles within one cell. Previously selected particles, the latest selected particles (during the last iteration), and unselected particles can be distinguished. Isolating particles that we selected during the previous extension step enables us to focus on particles that are the source of extending the selection, while we exclude already selected particles from any further considerations. With all particles-indexes within one cell matching the same sorting criteria, we can split the particles into three categories by sorting each cell and implementing two counters. The counters are stored within the dynamic cell information, as shown in Fig. 6. Subsorting the particle indices by selection state within a cell is possible without losing the original intent of accessing all cell particles as described in Section 4.1. Additionally, it adds the possibility to sequentially access particles by selection state and thus help perform efficient calculations during an extension step. The structure of sorted data within a cell is shown in Fig. 4. The subsorting is done upon two additional particle selection states, distinguishing whether a particle has been selected within a cell, is internally selected, or whether the reason for this selection was located outside the same cell. Particles are then marked as externally selected. We mark particles within two bits each particle, as mentioned previously, and cells using a flag created within our static cell information, as shown in Fig. 6.

Algorithm 1: Extend particle selection

```
for all particles in base cell to extend from do
    copy particle information of base cell to shared memory
    while num latest selected particles $\neq 0$ do
        if particle is latest selected then
            for all unselected particles do
                if EvalExtend(myAttr,myAttr[i]) then
                    MarkInternallySelected(i)
                    SetCellSortFlag()
        MoveLatestSelectedToSelected()
        SortCellInSharedMem()
    MoveNewlySelInBaseCellToLatestSel()
    BlockSync()
    copy particle information of base cell to global memory
    for all neighbor cells of base cell do
        copy particle information of cell to shared memory
        if particle in base cell is latest selected then
            for all unselected neighbor particles do
                if EvalExtend(myAttr,neighbAttr[i])
                 then
                    MarkExternallySelSafe(i)
                    SetCellSortFlag()
        copy neighbors' particle information to global
          memory
```

4.2.3. Selection Growing

The above-mentioned categorization into particles that have been selected within a cell or from a neighboring cell is fundamental for reducing workload during extension. As already mentioned, particle extension is done isolated on regions in parallel. All cells containing particles selected by either newly created selections or neighboring cells are checked for further extension during each selection extension. At first, all particles within the same cell are considered and extended until no other particles are selected. With each execution, the number of possibly selectable particles decreases, which reduces necessary comparisons between particles. This operation is illustrated in the second picture of Fig. 5(c). We use subsorting on the cell's isolated execution content, as described in Section 4.2.2 and shown for global execution in Fig. 4(c). For the upstream extension within a cell's boundaries, we require an additional temporary selection state for subdividing the group of

Algorithm 2: Consolidate particle selection

```
input  : Cells that were involved in latest selection
output: Cells that need attributes, cells that are source of
        extension
for all cells involved in selection do
    ExternallySelectedToLatestSelected()
    SortCellInSharedMem()
    WriteSortedIndexesToGlobalMemory()
    ResetCellSortFlag()
    if cell contains latest selected particles then
        AddToCellsAsSourceOfExtension()
        for all neighbor cells do
            if neighbor cell has attributes not available then
                AddToCellsNeedingAttributes()
```

Algorithm 3: Dynamic attribute computation

```
for all particles in cell to calculate attributes for do
    copy particle information of base cell to shared memory
    for all particles in cell do
        myResult[$i$] ←
          EvalNeighborFunc(myInfo,neighbInfo[$i$])
        myResult ← ReductionFunc(myResult[$i$])
    for all neighbor cells of base cell do
        copy particle information of cell to shared memory
        for all neighbor particles do
            myResult[$i$] ←
              EvalNeighborFunc(myInfo,neighbInfo[$i$])
            myResult ← ReductionFunc(myResult[$i$])
    copy myResult to global memory
```

unselected to a new temporary group of newly selected within the cell. We realize this by temporarily introducing another counter as a border between the subsorted particles of different selection types. When the extension within the cell to extend from is completed, these are moved to the latest selected particles by adding the temporary counter to the number of latest selected particles, merging the two groups. These particles are then marked as selected internally and create the basis of extending to neighboring cells. All particles in neighboring cells are then marked as externally selected. In addition to the particles being marked, the containing cell is marked with a sort flag when parts of its content are selected. The whole process is outlined in Algorithm 1.

Since the extension is done in parallel on possibly overlapping cells and thus identical particles, downstream synchronization on global memory is necessary. A visual example of the necessity is shown in Fig. 5(d), where the initially selected particle of both displayed blocks selects other particles within a second iteration. In this example, the second iteration results in the selection of a particle in overlapping cells.

During consolidation, all cells that were processed during extension are evaluated for further extension. Based on each particle's external or internal selection state, particles are grouped into unselected, latest selected, or selected. Particles that are selected internally have already been considered for extensions to neighboring cells and are therefore directly moved to the selected group, excluding them from further considerations. Externally selected particles are moved to the category of the latest selected particles. Cells containing particles of this type are added as a source of extension for the next extension cycle. Additionally, their neighboring cells are checked for attribute availability and scheduled for dynamically calculating attributes, if necessary. The above process is outlined in Algorithm 2.

4.3. Dynamically Computing Attributes

As soon as lazy evaluation on attributes is triggered, the calculation process is conceptually straightforward. Several provided and configurable neighbor functions f are executed for each particle and neighbor-combination, accepting each combination's positions and additional information, e.g., previously calculated attributes as input. Subsequently, the neighbor function's results are passed to another configurable reduction function r and stores the result back to the particle's attribute. By choosing f and r, as well as the number of execution cycles, we are able to calculate several attributes on first-, second-, or third-ring neighbors. With each additional ring, our solution dynamically extends the range of needed cells, omitting multiple calculations using the respective flags, as mentioned earlier. The calculation is integrated within our 'preparation step' as shown in Fig. 5(b) and outlined in Algorithm 3.

4.4. Implementation Details

Our solution is implemented in CUDA, using predefined structs as separated parameters to store basic configuration and pointers to device memory in addition to the mentioned data structures as described in Section 4.2.1. The parameters are available separately for grid-, particle-, attribute- and extension parameters, as well as optional uploaded attributes for attributes saved within the externally provided dataset. The selection-growing process is fully multi-threaded on the GPU and is divided into four main steps, which are synchronized by CUDA kernel launches. Each launch is adapted to the amount of work to be done within the step and can be repeated until all necessary criteria to process the next steps are met.

We built our examples using our plugin integration for the open-source particle rendering framework MegaMol [GKM*15]. This approach enables us to use different data sources compatible with MegaMol and use several rendering engines provided by the framework. To be able to visually distinguish particles of different selection states as described in Section 4.2.2, we created a rendering module based on Nvidia OptiX [PBD*10]. Using this renderer and other provided MegaMol plug-ins, additional features like coloring on attributes or hiding of (un)selected particles are additionally available.

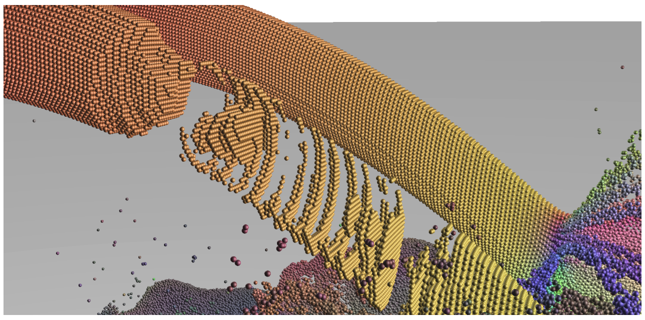

Figure 7: *Analysis of the inflow scenario simulated with SPH. Inverting the view, i.e., hiding all selected particles, reveals some shocks present in the jet that arise from the inflow zone.*

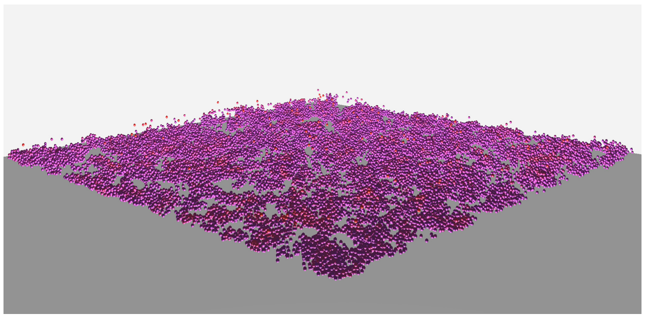

Figure 8: *Analysis of hot spots on a liquid's evaporation along a stationary planar interface. Particles on the surface with certain attributes are displayed isolated.*

5. Evaluation

We perform two kinds of experiments to evaluate our technique. First, two case studies are performed showing the versatility of the presented method and, second, performance and scaling studies are conducted discussing the techniques' efficiency.

5.1. Case Study

In the following, we consider two use cases with different characteristics. The first one is a large-scale smoothed particle hydrodynamics (SPH) dataset from Reinhardt et al. [RKEW19], where we analyze one simulation state and perform spatial reconstruction of attributes. The second one is a molecular dynamics dataset, simulating a liquid's evaporation on a stationary planar interface. It was supplied by Heinen and Vrabec [HV19]. In this case, we use temporal and spatial relations to reconstruct attributes.

5.1.1. SPH Inflow Case

In this case, we analyze a fluid simulation conducted with SPH. While initially invented by Gingold and Monaghan [GM77] to simulate astrophysical phenomena, SPH is nowadays often used in the field of computer graphics to animate liquids. The fluid quantities A_i at particle i's location $\mathbf{x}_i$ are smoothed over a compact neighborhood N_i using a weighting function (called smoothing kernel) W via

$$A_i = A(\mathbf{x}_i) = \sum_{j \in N_i} \frac{m_j}{\rho_j} A_j W(\|\mathbf{x}_i - \mathbf{x}_j\|, h), \tag{1}$$

where h is the so-called smoothing length, m_j the mass represented by particle j, and ρ_j the density at $\mathbf{x}_j$. The density ρ is one of the most important quantities with SPH as it is characteristic for the pressure and the resulting pressure forces, which on the other hand, are the dominating forces of the simulation system. For a particle i, it is computed using equation 1 and reads as

$$\rho_i = \sum_{j \in N_i} m_j W(\|\mathbf{x}_i - \mathbf{x}_j\|, h). \tag{2}$$

For a more detailed introduction to SPH, we refer the reader to the tutorial of Koschier et al. [KBST19].

A simulation state of the scenario we are investigating is depicted in Fig. 2. The scenario consists of up to 10 million particles. To safe disc space, only particle positions and velocities, as well as particle type (fluid or rigid particle), were stored within the dataset. It was computed using weakly compressible SPH [BT07] with consistent Shepard interpolation [RKEW19]. To simulate fluid and rigid coupling, the method of Akinci et al. [AIA*12] was used. As smoothing kernel W, the cubic spline kernel [Mon92] was used.

As mentioned, density is one of the characteristic fluid quantities. It is decisive for the volume conservation and the stability of the overall simulation process. Large density differences result in pressure shocks, which negatively influence the stability, and, therefore, smaller simulation time steps are needed. Identifying such shock waves is a challenging topic and typically achieved by visualizing densities as colors and looking at the simulation.

Initially, some particles on the surface of one of the liquid jets are selected manually. When growing the selection for larger density values, it is observable that the particles are selected that not all particles down the liquid jet are selected. In fact, when inverting the view, i.e., hiding the selected particles (Fig. 7), we can observe that these particles form some discs. These differences in density result in pressure differences and can, therefore, produce shocks, which have a negative influence on the stability of the system.

5.1.2. Molecular Dynamics Evaporation Case

The second case study is a molecular dynamics (MD) simulation from Heinen and Vrabec [HV19]. In this scenario (shown in Fig. 1), a liquid's evaporation along a stationary planar interface is investigated. They have obtained promising results for a vapor-liquid equilibrium investigations [VKFH06] and evaporation simulations [HVF16] using MD. For a detailed introduction to MD, we refer the reader to the lecture notes from Allen [All04].

The investigated scenario is part of a study where the influence of the hydrodynamic velocity on the evaporation flux is investigated using the transversal temperature $T_{i,xy}$ of a particle i. It is defined by

$$T_{i,xy} = \frac{m_i}{2N_i k_B} \sum_{j \in N_i} v_{j,x}^2 + v_{j,x}^2, \tag{3}$$

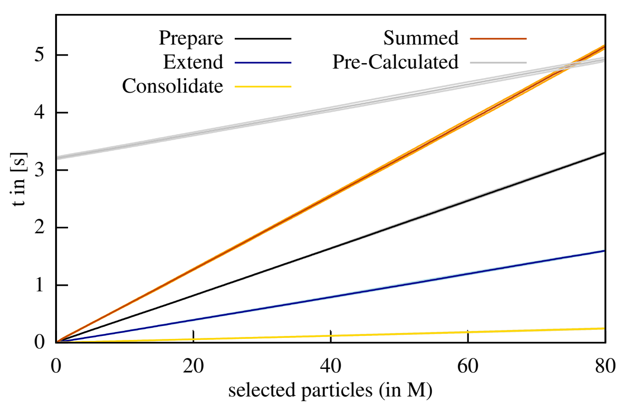

Figure 9: *Time measurements for the overall time to select particles in the 80M particles fluid pillar scenario. The extension on pre-calculated attributes is compared to lazily evaluated attributes. All sub-steps of our solution are shown and scale linearly with respect to the particle count. Median and interquartile boundaries (p25, p75) are plotted for all measurements.*

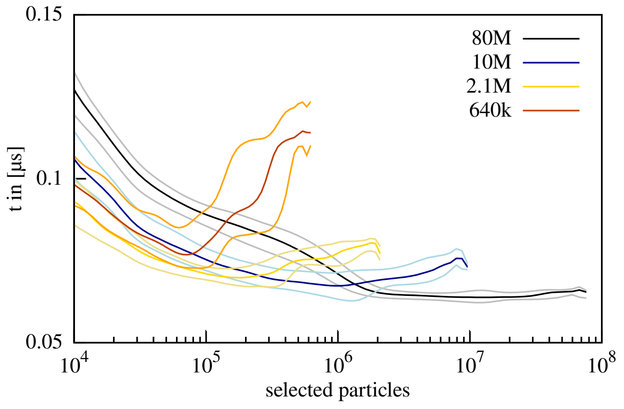

Figure 10: *The overall time to process one particle while increasing the selection. Median and interquartile boundary values (p25, p75) out of 100 tests. It can be observed that the process costs per particle slightly decrease until scenario boundaries are reached.*

where k_B is the Boltzmann constant, m_i the particles' mass, $v_{j,x}$ and $v_{j,y}$ the thermodynamic velocity in x- and y-direction. A detailed discussion can be found in Heinen and Vrabec [HV19].

This dataset consists of about 4.44M particles. The goal is to identify hot spot regions, i.e., particles with high transversal temperature, as in these regions, evaporation is analyzed. To compute $T_{i,xy}$, we need the particles' thermodynamic velocity. Therefore, the attribute calculation for the selection-growing is performed in two steps. First, the velocities are reconstructed from two subsequent time stamps, and, secondly, the transversal temperature $T_{i,xy}$ is computed as described in Equation 3. Additionally, the evaluation range has been extended from the usual four to eight times the radius of one particle.

The result in Fig. 8 can then be achieved using our growing method for similarity, combined with hiding unselected particles. The evaporation of particles on the surface can now be investigated in subsequent frames while keeping the selection.

5.2. Performance

We conducted two different studies to measure the performance of our solution and one study on memory usage. The measurements have been executed on a notebook with NVidia GTX 1070 graphics card with 8 GB memory using CUDA 9.0, measuring our solution's execution time separated for the three execution steps. As input data, we used an SPH fluid pillar in seven different resolutions from 80k to 80M particles with an average amount of 8.5 particles per cell, resulting in a total number of 11.2k to 9.4M cells. We executed all measurements 100 times when measuring time values and are displaying the median value and the interquartile boundary values (p25, p75) in all figures.

In the first study, we selected one random particle for each measurement within the 80M particle scenario and extended the selection until all particles were selected. After each particle extension, the number of additional particles and the required time has been measured. The result is displayed in Fig. 9. The total time to perform all selection steps, and the time consumed by the monitored sub-sets, scales linearly with the number of selected particles is shown. It can be observed that the lazy evaluation of attributes consumes about two-thirds of the total computation time. This ratio is valid for our density evaluation and will differ for different workloads when calculating other types of attributes. In addition to our lazily evaluated attributes, we measured the time consumed using pre-calculated attributes, shown in Fig. 9. As expected, the time to calculate all attributes upfront does not differ from the time that is consumed by the preparation-step for all 80M particles. As the extension itself does not change, the resulting difference is found in the phase of consolidation. In fact, the total time used for consolidation is reduced when calculating all attributes before starting the selection. However, as this step takes a neglectable portion of the total processing time, our solution, using lazily evaluated attributes, outperforms this approach until almost 90% of all particles are selected.

In the second study, we evaluate the mean cost to process one particle depending on the total amount of selected particles on four different scenarios with varying particle count ranging from 640k to 80M. The result of this performance study is shown in Fig. 10. There are two main observations: first, the time needed to process one particle in the selection is independent of the scenario size. Secondly, the cost to process one particle stays almost constant regardless of the overall number of selected particles. It is even slightly reduced the more particles are selected. Increasing costs per particle close to the end of the measured selection processes are caused by the reduced number of unselected particles remaining during extension close to the scenario's boundaries. This leads to fewer particles being left for selection during each extension and, therefore, an increasing execution time per particle. The randomly chosen first particle leads to differences in unselected areas between the conducted experiments during the last extension steps, explaining an

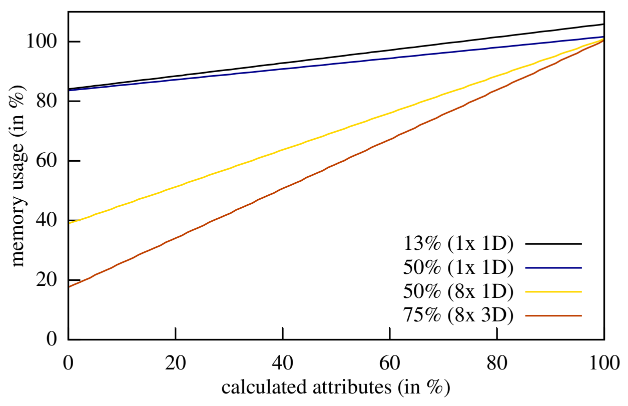

Figure 11: *The memory used while lazily evaluating attributes and extending the selection in percent of using precalculated attributes. Depending on the average occupancy of a cell (13%, 50% and 75%), as well as the amount and type of evaluated attributes, up to 80% less memory is required.*

increasing variance in this extension phase. In essence, the overall time needed to process one particle is about 0.1 microseconds.

Within the same scenario, we measured the amount of necessary memory for lazily evaluating attributes while increasing the selection. We compared the results to the amount of memory needed when using previously uploaded attributes for the full dataset. The numbers are shown in Fig. 11. The result mainly depends on the average occupancy of particles within a cell and the number of attributes calculated. For good coverage, we consider the following four configurations: One low cell-occupancy of 13% with one 1D particle evaluated, which matches the presented 80M scenario, requires up to 18% less memory. The 75% occupancy with an evaluation of eight 3D attributes, which represents the maximum capacity of the presented solution, can use up to 80% less memory for small amounts of attributes calculated. The memory increments depend on the used resize strategy and can be optimized for better memory usage or performance. In general, our method is recommended when smaller percentages of attributes are required and especially useful with increasing numbers and dimensions of attributes.

The dynamic memory usage D depends on the number of particles within the scenario n, the maximum number of particles to extend within one extension step $e_{\max}$, the average cell occupancy p_{avg}, the number of attributes x, as well as the dimensions per attribute d_a. Its minimum percentage of the corresponding fixed memory usage can be calculated via

$$D(x) = \frac{5n + \frac{2n}{p_{\text{avg}}} + d_a n x + 2 + \frac{3nx}{p_{\text{avg}}} + e_{\max}}{5n + \frac{2n}{p_{\text{avg}}} + d_a n + 1}, \tag{4}$$

where a static approach uses three dimensions of position-data, color information and indexes ($5n$), intermediate and count array, as well as all attributes for each dimension. Additionally, one element to calculate the number of particles in each cell with increased performance is required. Our dynamic approach requires attributes for each dimension only for calculated attributes but adds one additional counter, the dynamic data structure of three elements per cell, and a sync list containing all particles to extend within one step.

The average cell occupancy and the maximum number of particles to extend within one step depend on the scenario. The presented scenario of 80M particles contains an average of 8.5 particles per cell with an $e_{\max}$ of 250,000.

6. Conclusion

We presented an efficient technique for selecting particles based on attributes in an interactive visualization of the simulation. The region of interest is first brushed manually and afterward grown intelligently, based on attribute comparisons. These attributes may even not necessarily be present in the dataset. If they are not present (e.g., for disk space reasons), they are lazily reconstructed, outperforming precalculated attributes with respect to performance and memory usage for amounts of up to 90% attributes calculated within most particle-based use cases. In this way, even new attributes or measures may be constructed, which may arise during the analysis. These could also be used to compare different simulations conducted with a concurring model. A more in-depth analysis of such a use case would be a new topic in future work. For example, measuring the density noise to compare as presented by Reinhardt et al. [RKEW19] might be an interesting area for such cases. As our presented solution uses a grid-based spatial decomposition, it might be interesting to investigate the technique for other methods like octrees or kd-trees.

Acknowledgements

This work was supported by “Kooperatives Promotionskolleg Digital Media” at Hochschule der Medien and the University of Stuttgart.

References

[AGC05] Ahrens J., Geveci B., Charles: Paraview: An end-user tool for large-data visualization. In *Visualization Handbook*, Hansen C. D., Johnson C. R., (Eds.). Butterworth-Heinemann, 2005, pp. 717–731. doi:10.1016/B978-012387582-2/50038-1. 3

[AIA*12] Akinci N., Ihmsen M., Akinci G., Solenthaler B., Teschner M.: Versatile rigid-fluid coupling for incompressible SPH. *ACM Transactions on Graphics 31*, 4 (2012), 62:1–62:8. doi:10.1145/2185520.2185558. 8

[All04] Allen M. P.: Introduction to molecular dynamics simulation. *Computational soft matter: From Synthetic Polymers to Proteins 23*, 1 (2004), 1–28. 8

[BT07] Becker M., Teschner M.: Weakly compressible SPH for free surface flows. In *Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation* (2007), pp. 9–18. doi:10.2312/SCA/SCA07/209-218. 8

[CBW*12] Childs H., Brugger E., Whitlock B., Meredith J., Ahern S., Pugmire D., Biagas K., Miller M., Weber G. H., Krishnan H., Fogal T., Sanderson A., Garth C., Bethel E. W., Camp D., Rübel O., Durant M., Favre J., Navratil P.: Visit: An end-user tool for visualizing and analyzing very large data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, Bethel E. W., Childs H., Hansen C., (Eds.). CRC Press/Francis–Taylor Group, 2012, pp. 357–372. 3

[CLRS09] Corman T. H., Leiserson C. E., Rivest R. L., Stein C.: *Introduction to Algorithms*, 3rd ed. MIT Press, 2009. 5

[DGH03] Doleisch H., Gasser M., Hauser H.: Interactive feature specification for focus+context visualization of complex simulation data. In *Eurographics / IEEE VGTC Symposium on Visualization* (2003), The Eurographics Association. doi:10.2312/VisSym/VisSym03/239-248. 3

[GKM*15] Grottel S., Krone M., Mueller C., Reina G., Ertl T.: Megamol - a prototyping framework for particle-based visualization. *IEEE Transactions on Visualization and Computer Graphics 21*, 2 (2015), 201–214. doi:10.1109/TVCG.2014.2350479. 2, 7

[GM77] Gingold R. A., Monaghan J. J.: Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society 181*, 3 (1977), 375–389. doi:10.1093/mnras/181.3.375. 8

[GRDE10] Grottel S., Reina G., Dachsbacher C., Ertl T.: Coherent culling and shading for large molecular dynamics visualization. *Computer Graphics Forum 29*, 3 (2010), 953–962. doi:10.1111/j.1467-8659.2009.01698.x. 2

[Gre10] Green S.: *Particle simulation using CUDA*. Whitepaper, NVIDIA, 2010. 4, 5

[HK98] Hojjatoleslami S. A., Kittler J.: Region growing: a new approach. *IEEE Transactions on Image Processing 7*, 7 (1998), 1079–1084. doi:10.1109/83.701170. 2

[HM03] Huang R., Ma K.-L.: RGVis: region growing based techniques for volume visualization. In *11th Pacific Conference on Computer Graphics and Applications.* (2003), pp. 355–363. doi:10.1109/PCCGA.2003.1238277. 2

[HNM*12] Harrison C., Navrátil P., Moussalem M., Jiang M., Childs H.: Efficient dynamic derived field generation on many-core architectures using Python. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis* (2012), pp. 583–592. doi:10.1109/SC.Companion.2012.82. 3

[HSO07] Harris M., Sengupta S., Owens J. D.: Parallel prefix sum (scan) with CUDA. *GPU Gems 3*, 39 (2007), 851–876. 5

[HV19] Heinen M., Vrabec J.: Evaporation sampled by stationary molecular dynamics simulation. *The Journal of Chemical Physics 151*, 4 (2019), 044704. doi:10.1063/1.5111759. 8, 9

[HVF16] Heinen M., Vrabec J., Fischer J.: Communication: Evaporation: Influence of heat transport in the liquid on the interface temperature and the particle flux. *The Journal of Chemical Physics 145*, 8 (2016), 081101. doi:10.1063/1.4961542. 8

[IHL20] Ibrahim S., Harrison C., Larsen M.: JIT's complicated: A comprehensive system for derived field generation. In *In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (2020), pp. 27—31. doi:10.1145/3426462.3426467. 3

[JMEL08] Jones C., Ma K., Ethier S., Lee W.: An integrated exploration approach to visualizing multivariate particle data. *Computing in Science Engineering 10*, 4 (2008), 20–29. doi:10.1109/MCSE.2008.88. 3

[JSZ18] Jiang M., Southern R., Zhang J. J.: Energy-based dissolution simulation using SPH sampling. *Computer Animation and Virtual Worlds 29*, 2 (2018), e1798. doi:10.1002/cav.1798. 3

[KBST19] Koschier D., Bender J., Solenthaler B., Teschner M.: Smoothed particle hydrodynamics techniques for the physics based simulation of fluids and solids. In *Eurographics 2019 - Tutorials* (2019), The Eurographics Association. doi:10.2312/egt.20191035. 8

[LMD*11] Linsen L., Molchanov V., Dobrev P., Rosswog S., Rosenthal P., Long T. V.: SmoothViz: Visualization of smoothed particle hydrodynamics data. In *Hydrodynamics - Optimizing Methods and Tools*, Schulz H. E., Simoes A. L. A., Lobosco R. J., (Eds.). InTech, 2011, pp. 3–28. 2

[MFR*13] Molchanov V., Fofonov A., Rosswog S., Rosenthal P., Linsen L.: Smoothviz: An interactive visual analysis system for SPH data. In *Proceedings of the 8th International SPHERIC Workshop* (2013), pp. 350–356. 2

[Mon92] Monaghan J. J.: Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics 30*, 1 (1992), 543–574. doi:10.1146/annurev.aa.30.090192.002551. 8

[PBD*10] Parker S. G., Bigler J., Dietrich A., Friedrich H., Hoberock J., Luebke D., McAllister D., McGuire M., Morley K., Robison A., Stich M.: Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics 29*, 4 (2010), 66:1–66:13. doi:10.1145/1778765.1778803. 2, 7

[Pri07] Price D. J.: Splash: An interactive visualisation tool for smoothed particle hydrodynamics simulations. *Publications of the Astronomical Society of Australia 24*, 3 (2007), 159–173. doi:10.1071/AS07022. 2

[RE05] Reina G., Ertl T.: Hardware-accelerated glyphs for mono- and dipoles in molecular dynamics visualization. In *EG/IEEE VGTC Symposium on Visualization* (2005), pp. 177–182. doi:10.2312/VisSym/EuroVis05/177-182. 2

[RHD*17] Reinhardt S., Huber M., Dumitrescu O., Krone M., Eberhardt B., Weiskopf D.: Visual debugging of SPH simulations. In *2017 21st International Conference Information Visualisation (IV)* (2017), pp. 117–126. doi:10.1109/iV.2017.20. 2

[RKEW19] Reinhardt S., Krake T., Eberhardt B., Weiskopf D.: Consistent Shepard interpolation for SPH-based fluid animation. *ACM Transactions on Graphics 38*, 6 (2019), 189:1–189:11. doi:10.1145/3355089.3356503. 8, 10

[SBSG06] Sereda P., Bartroli A. V., Serlie I. W. O., Gerritsen F. A.: Visualization of boundaries in volumetric data sets using LH histograms. *IEEE Transactions on Visualization and Computer Graphics 12*, 2 (2006), 208–218. doi:10.1109/TVCG.2006.39. 2

[SWB15] Sakou L., Wilches D., Banic A.: Region growing selection technique for dense volume visualization. In *International Symposium on Visual Computing* (2015), pp. 745–754. doi:10.1007/978-3-319-27863-6_70. 3

[SYM13] Sauer F., Yu H., Ma K.-L.: An analytical framework for particle and volume data of large-scale combustion simulations. In *Proceedings of the 8th International Workshop on Ultrascale Visualization* (2013), pp. 1–8. doi:10.1145/2535571.2535590. 2

[SYM14] Sauer F., Yu H., Ma K.: Trajectory-based flow feature tracking in joint particle/volume datasets. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 2565–2574. doi:10.1109/TVCG.2014.2346423. 3

[TLM05] Tzeng F.-Y., Lum E. B., Ma K.-L.: An intelligent system approach to higher-dimensional classification of volume data. *IEEE Transactions on visualization and computer graphics 11*, 3 (2005), 273–284. doi:10.1109/TVCG.2005.38. 2

[VKFH06] Vrabec J., Kedia G. K., Fuchs G., Hasse H.: Comprehensive study of the vapour–liquid coexistence of the truncated and shifted Lennard–Jones fluid including planar and spherical interface properties. *Molecular Physics 104*, 9 (2006), 1509–1527. doi:10.1080/00268970600556774. 8

[YEII12] Yu L., Efstathiou K., Isenberg P., Isenberg T.: Efficient structure-aware selection techniques for 3D point cloud visualizations with 2DOF input. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (2012), 2245–2254. doi:10.1109/TVCG.2012.217. 3