EG UK Computer Graphics & Visual Computing (2018)
G. Tam and F. Vidal (Editors)

Screen Space Particle Selection

M. Köster[1] and A. Krüger[1]

[1]DFKI, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

Analyses of large 3D particle datasets typically involve many different exploration and visualization steps. Interactive exploration techniques are essential to reveal and select interesting subsets like clusters or other sophisticated structures. State-of-the-art techniques allow for context-aware selections that can be refined dynamically. However, these techniques require large amounts of memory and have high computational complexity which heavily limits their applicability to large datasets. We propose a novel, massively parallel particle selection method that is easy to implement and has a processing complexity of $O(n*k)$ (where n is the number of particles and k the maximum number of neighbors per particle) and requires only $O(n)$ memory. Furthermore, our algorithm is designed for GPUs and performs a selection step in several milliseconds while still being able to achieve high-quality results.

CCS Concepts

•***Human-centered computing*** → *Visualization systems and tools; Interaction techniques; Scientific visualization;*

1 Introduction

Data visualization and exploration are essential concepts to analyze 3D particle datasets. A crucial technique for different analyses is the selection of interesting subsets or patterns [Tuk77]. In contrast to simple 2D selection methods that work well with 2D data, 3D selection tasks are more challenging due to the high number of degrees of freedom; especially in the case of particle datasets that typically contain several hundred thousand of data points. In the case of 3D particle datasets, the selection is a 3D selection volume that can be difficult to specify and is time consuming and error prone to refine [ONI05]. In order to determine a sub-volume based on user input, dataset attributes like scalar properties per particle (data point) or the particle density are typically used.

Over the years, a variety of different selection techniques have been developed to provide feasible solutions for these kind of datasets. The state-of-the-art methods are context-aware selection techniques (*CAST*) [YEII12, YEII16] that work on 2D selection lassos which are drawn by the user. A user's intended selection is determined based on the lasso, the current camera view and the attributes of the dataset used. Specialized methods from the CAST-algorithm family can be used in different scenarios; for example, the selection of volumes by clicking or the selection of sub-volumes by analyzing the drawn contour of the lasso. These methods use a particle density estimator to compute density information that is required for the selection process. Although they deliver high-quality selections, they require multiple processing steps with high computational complexity. The high memory consumption (due to global uniform grids) and computation times dramatically limit their applicability in general.

In order to circumvent these limitations, we propose a novel selection method called *Screen Space Particle Selection (SSPS)*. It also uses the basic idea of a 2D lasso-based selection via mouse or touch input. Like the CAST algorithms, our method works on scalar properties per particle and density information. In contrast to existing selection methods, we do not use a global density information grid but apply a method based on the *smoothed particle hydrodynamics (SPH)* model. Using this approach, we can realize fast and efficient computations of the density per particle. Furthermore, we do not perform any operations that require an explicit 3D grid in memory (like the *Marching Cubes* algorithm [LC87]). Instead, we render the visualized dataset once into several screen space buffers and perform all analyses in screen space. This decouples the computational complexity of the selection analyses from the actual dataset. We determine the intended selection based on the drawn lasso by first computing a selection mask (a weighted 2D selection shape) and mapping the selection information back into 3D space. Finally, we apply an SPH-based flood-filling algorithm to actually select the desired particles in the dataset. Our method is designed to run on massively-parallel processors like graphics-processing units (GPUs). The allows us to compute a particle selection in a few milliseconds even on large datasets.

2 Related Work

Single or multiple object selection are well-known tasks and have a long tradition in the field of HCI and visualization. Especially ray-casting and image plane methods are popular approaches for the selection of individual 3D objects [Min95, PFC*97]. The selection concepts of pointing or touching in stereoscopic settings [DFK12, DSG*14] or in virtual reality [VSB*10] are also widely spread. They focus on the selection of a relatively small number of objects, either in 2D or 3D space. However, analyses of large amounts of

DOI: 10.2312/cgvc.20181208

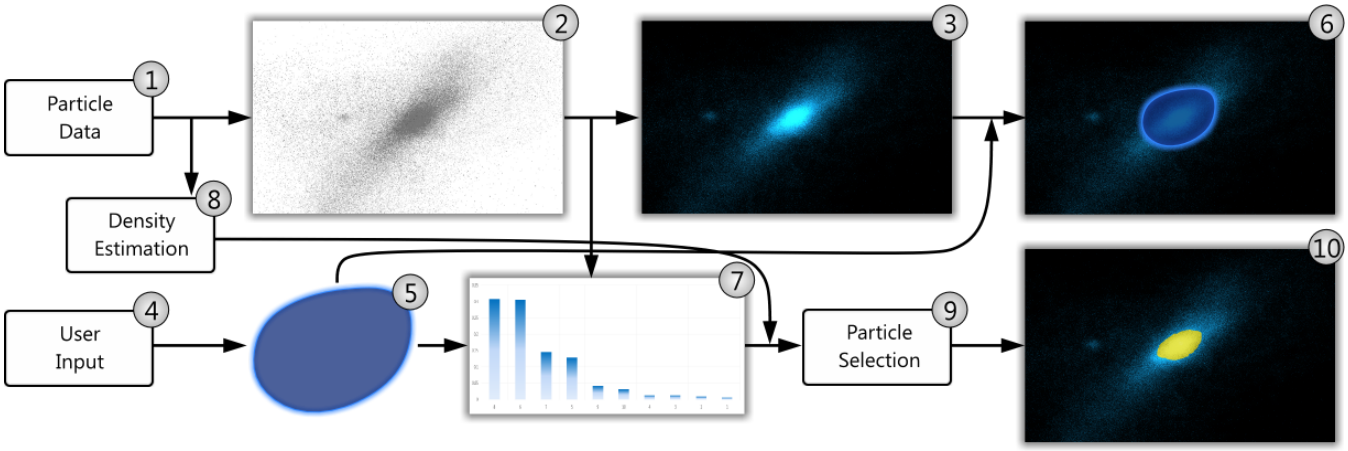

Figure 1: *A visualization of the general processing pipeline. Particle data (1) from the dataset is rendered using the default rendering pipeline. An intermediate result is a depth image (2) that is used for further processing. The composed image (3) is displayed to the user and they can perform lasso selection operations (4). A selection lasso (5) is derived from the user input and is also displayed to the user (6). The lasso and depth information are used to resolve the intended region of interest, called the focus area (7). The actual 3D particle selection operation (9) uses a specially designed density estimation (8) and information from the focus area (7). The final selection output is then displayed to the user and further analyses can be applied to the selected subset (10).*

data affect hundreds of thousands of data objects (particles, in our case) such that a single object is typically not the focus of a selection task. Instead, we are interested in larger groups of up to thousands of data elements from particularly interesting substructures or sub-volumes.

These scenarios require a 3D-volume based selection. A straightforward idea is to use a cone-based selection that selects either a single object closest to the center of the cone or all objects within the selection cone [SP04]. More involved concepts are structure-aware selection methods [WVH11, HWVF12] that use analyses to infer knowledge about the underlying dataset. Yu et al. [YEII12] introduced two concepts (namely *TeddySelection* and *CloudLasso*), which use a 2D selection lasso. Both methods use a particle density estimation and require a 3D discretization (a grid) to construct a selection mesh in 3D. Yu et al. have shown that CloudLasso is more flexible, but also more computationally intensive than TeddySelection. Yu et al. improved their selection mechanisms by introducing the CAST-algorithm family [YEII16]. Similar to CloudLasso and TeddySelection, CAST use computationally expensive analyses based on 3D volume information for every step. These methods are based on the notion and semantic concept of a *cluster* to realize the actual selection process. From a higher-level perspective, a cluster can be considered a region in 3D space of a certain particle density that can be selected by users. Furthermore, a cluster can be visually perceived by humans using rendered particle sets since high density in the dataset typically leads to a very dense area in the final image.

Both papers by Yu et al. are mostly related to our approach since they link the 2D inputs to the intended particle selection in 3D space. 2D inputs are particularly interesting because they are typically used by data analysts in the scope of their common work flows. All these methods leverage the Marching Cubes algorithm [LC87] and a uniform 3D grid to compute selection meshes and to explicitly track clusters in memory. When focusing on the density estimator, the CAST algorithms rely on a modified Breiman kernel density estimation method (MBE) [BHS*11]. It leverages a uniform density grid of $O(m^3)$ memory, where m refers to the number of grid cells in one dimension. The grid is required to reduce its quadratic processing complexity of $O(n^2)$ and to adaptively calculate the influenced grid cells per particle by tracking an ellipsoid per particle. Real-world sets, however, require a reasonable high resolution in terms of grid cells in order to achieve a precise selection. The decision for using the MBE was motivated by Fredosi et al. [BHS*11] who determined that this method is perfectly suitable for astrophysical data sets. The SPH method was not considered by Fredosi et al. but was originally developed for astrophysical problems [Luc77, Mon92] and recent papers from the field of particle-based simulations by Macklin et al. [MM13, MMCK14] and Köster et al. [KK16] successfully used the SPH method for their real-time density estimations. They rely on a radix-sort based approach by Green [Gre10a, Gre10b] thats avoids an explicit construction of a 3D density grid in memory.

3 Screen Space Particle Selection

The general processing pipeline is visualized in Figure 1. As shown in the diagram, only one analysis is applied to the particles directly: the density estimation. All other information can be resolved from a screen space depth buffer.

The basic idea behind our approach is the fact that a selection of potentially interesting structures in space has a specific shape in screen space. Since the user performs a selection based on the visualized dataset, we can leverage the same information to perform the intended selection operation. Similar to the CAST-algorithm family, our selection method also requires an initial 2D selection lasso on the screen to be drawn by the user. This selection lasso spans the desired region of interest. The important and difficult part is to determine the actually intended selection volume based on the 2D selection information.

However, since we want to avoid excessive memory consumption and high computational overhead, we first map the required parts of the 3D volume information to a screen-resolution-dependent 2D texture and an intention-detection buffer. The texture contains linearized screen space depth information in $[0.0, 1.0]$ that is typically already available through and/or required for default rendering or visualization purposes. The intention buffer stores floating-point values and is basically a weighted distribution buffer. It is computed by combining the linearized screen space depth using per-pixel weighting information that is resolved from the selection lasso. The resulting intention weights are sorted according to the highest rating and allow an inference of the intended selection, called *focus area*, in 3D space. Based on this area, we can resolve the desired density information in the computed region and select all related particles with a similar density. The actual selection process involves projecting candidate particles onto the screen. A particle is selected, if it falls into the region of the 2D lasso in screen space. However, similar to TraceCast [YEII16] this test can also be disabled for context-aware selection of larger volumes near the focus area.

3.1 Mask Construction

The first step during selection is the construction of the selection mask. The input for this phase is the drawn lasso in form of a set of points in screen pixel coordinates. We follow the approaches from related work and close the lasso if it is not closed by automatically connecting the start and the end point of the user's lasso contour. These points will be converted into a 2D polygon that is used for further processing. Additional de-noising steps can be applied to smooth the shape of the selection polygon (like Laplacian smoothing [VMM99]). Afterwards, the resolved polygon is rasterized into a 2D floating-point mask image, which was previously initialized with 0. The size of the mask image is given by the rendering resolution.

Note that the shape of a selection polygon has a lot of information about the intended selection (see Figure 2). For example, sharp corners help to separate desired from non-desired regions, whereas large enclosing regions indicate a large intended area. We propose a straight-forward and fast selection-mask approach that adaptively weights all included pixels of the selection polygon to encode a user's intended selection. For this reason, we apply a position-dependent weighting kernel $W_M : \mathbb{N} \times \mathbb{N} \mapsto [0.0, 1.0]$ to the rendered mask image. The kernel is applied to every pixel inside the rendered selection polygon and yields an importance (influence) factor for a given mask pixel. The higher this factor is, the more important the pixel coordinate is in the scope of the user's desired selection. We use a radial-weighting kernel in all cases that pays special attention to the shape of the user's selection lasso (see Figure 3). The overall shape of the selection lasso is taken into account using the diameter of the polygon. All points inside the selection mask are weighted according to their distance to the centroid.

3.2 Focus Area Computation

Once the selection mask has been constructed, it has to be linked to the structure of the dataset in order to compute the actual focus area. The high-level idea behind this step is to infer a 3D volume slice that corresponds best to the intended selection lasso. We use depth-distribution histograms by accumulating depth values that fall into certain depth intervals or bins. The number of bins S_I is a custom selection parameter and is typically set to 16 in our experiments. Increasing S_I yields a more fine-grained selection that pays more attention to the mask on the one hand; a smaller number of bins allows to conveniently select larger (but more coarse-grained) regions on the other hand.

From a high-level point of view, we are looking for two cut-off planes (the minimum and maximum planes) in the depth-distribution histogram that reflect the desired volume slice; how-

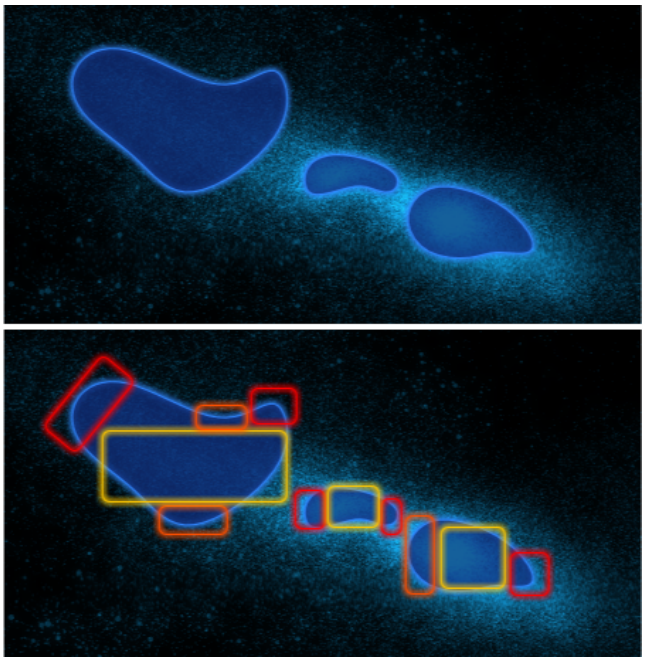

Figure 2: *A dataset with three user-defined selection lassos (top). An analysis of the masks' shapes allows to distinguish more from less important areas. The bottom image highlights several regions of the selection lassos and their related selection intentions. For example, corners separate desired from non-desired areas (red). Less important areas are color-coded with orange and the least important areas are yellow.*

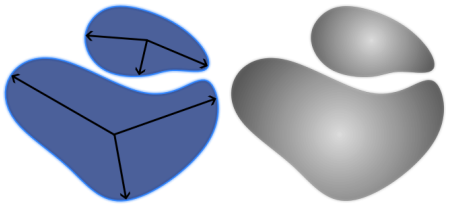

Figure 3: *Two different user lassos (left) and their corresponding mask images (right). The black arrows indicate the distance of a point from the centroid of its selection polygon. Note that the larger this distance is, the larger the weighting factor will be. The color gradient from white to black visualizes the weighting kernel. Points near the centroid will receive a low weight (white), whereas others in the more interesting regions will receive a high weight (black).*

ever, since we want to reflect the user's intention, a simple depth-distribution histogram is not sufficient. Instead, we accumulate the corresponding mask weights in form of a distribution histogram called the *intention buffer* I_B. This allows us to infer the intended selection based on the values of the intention buffer: a high rating implies a high selection intention.

As previously mentioned, we use linearized depth information from the depth buffer of the rendering pipeline for this purpose (see Figure 1), where 0 refers to a near and 1 to a far particle[†]. These depth values $D(x,y)$ are mapped to bin-indices in the intention buffer. Hence, the available mask information ($M_I(x,y) > 0$) is used to compute the involved minimum $D_{\min}$ and maximum depth $D_{\max}$ values from the depth image (see Algorithm 1). Next, the depth-sampling interval is computed in order to determine the value range of a single bin in the scope of the intention buffer. For every pixel, the corresponding mask values $M_I(x,y)$ will be accumulated in the corresponding bins, if there was a particle at pixel location (x,y) ($D(x,y) < 1$). If not, the mask information will not be stored in the intention buffer since it does not contain valid depth information (see Figure 4).

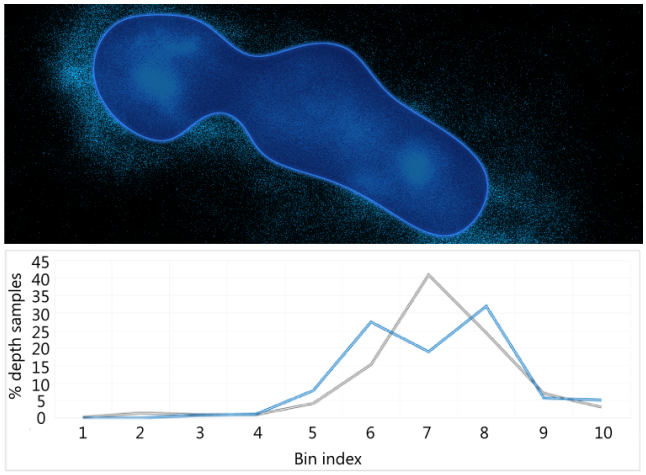

Figure 4: *A sample selection mask (top) and the corresponding (normalized) depth distributions from the mask area (Y-axis). The X-axis indicates the bin indices. Color coding: Grey, the default distribution of the depth values. Blue: weighted depth values according to the mask weights.*

Finally, the actual focus area in 3D space can be inferred from the intention buffer. To this end, the buffer is sorted in descending order according to the accumulated weights (see Figure 5). Then, we compute the average difference between a certain bin and its successor. Starting with the first bin, all successor bins are included in the inference step as long as the difference between its value and its successor's value is smaller than the average difference. This feature significantly relaxes the greediness of the area computation since it also includes selection ranges with similar ratings.

† Note the assumption that the depth texture is initialized with 1 (infinite depth) before the rendering step and the depth test is set to less or less equal.

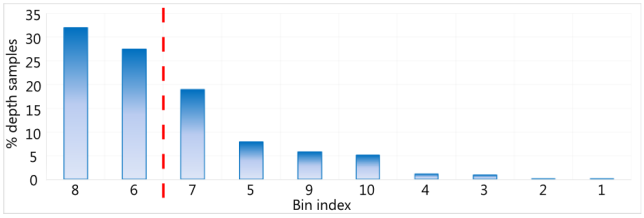

Figure 5: *Sorted focus area bins from Figure 4. The numbers on the X-axis refer to the original bin index. In this case, the average difference between all entries is approximately 0.04. Consequently, the third bin (bin number 7) will not be taken into account (red line). Using a greedy approach would result in bin 8 only.*

Algorithm 1 Focus Area Construction Algorithm

Require: Mask image M_I **and** linearized depth image D
Require: Intention buffer I_B
  $D_{\min} \leftarrow 1, D_{\max} \leftarrow 0$
  for all pixels (x,y) **where** $M_I(x,y) > 0$ **and** $D(x,y) < 1$ **do**
    $D_{\min} \leftarrow \min(D_{\min}, D(x,y))$
    $D_{\max} \leftarrow \max(D_{\max}, D(x,y))$
  end for
  if $D_{\max} < D_{\min}$ **then**
    return No focus area found
  end if
  Compute sampling interval $D_S \leftarrow \frac{D_{\max} - D_{\min}}{\text{number of samples}}$
  Initialize I_B with 0
  for all pixels (x,y) **where** $M_I(x,y) > 0$ **and** $D(x,y) < 1$ **do**
    $i_b \leftarrow \frac{D(x,y)}{D_S} \cdot \text{number of bins}$
    $I_B(i_b) \leftarrow I_B(i_b) + M_I(x,y)$
  end for
  Sort I_B in descending order of ratings
  $d_I \leftarrow$ average difference of $I_B(i)$ and $I_B(i+1)$ in I_B
  $i \leftarrow 0$
  while $i <$ number of bins $- 1$ **and** $I_B(i) - I_B(i+1) < d_I$ **do**
    $i \leftarrow i + 1$
  end while
  return $(\text{depth}(I_B(0)), \text{depth}(I_B(i)))$

3.3 Density Estimation

We realize density estimation via the SPH method, which was original developed by Lucy [Luc77], Gingold and Monagan [GM77] for astrophysical problems. Discrete data points p and an associated scalar property A are the input arguments for SPH, which approximates a continuous value distribution based on the property values of the given data points. An interpolated quantity A_i (smoothed quantity A) for the i-th particle at particle position p_i is given by the weighted sum over all particles [Mon92]:

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W(||p_i - p_j||, h), \tag{1}$$

where m_j, ρ_j, A_j are the mass, the density and the scalar property value of the j-th particle. h is the applied smoothing length and W is the smoothing kernel (weighting function), that is positive, even, normalized [Mon00, Kel06], and typically has a finite support in practice [MM13, KK16]. In order to approximate the local density

of a particle, we can also use the general SPH method [Mon02]

$$\rho_i = \sum_j \frac{m_j}{\rho_j} \rho_i W(||p_i - p_j||, h) = \sum_j m_j W(||p_i - p_j||, h). \quad (2)$$

The mass m_j can be replaced by an arbitrary generic scalar property which we want to base our density computation on. However, SPH requires a certain amount of neighboring particles that fall into the radius h to ensure a stable computation [KSW99]. Hence, h is often chosen adaptively in such a way that every particle has a reasonable number of neighbors [NP94]. In our case, we cannot determine the smoothing length h off-line since the dataset is unknown in advance.

As mentioned in section 2, related approaches use online ways to compute the density with adaptive influence radii for every particle. There are many applications of dynamically-adjusted smoothing lengths in SPH-based simulations. We also use an online computation concept and use an adaptive smoothing length h_i per particle for the density computation. In order to determine a high-quality estimate for h_i per particle, we use a formula based on the ideal number of neighbors N_h that depends on the shape of the used smoothing kernel W. For instance, a cubic-spline kernel is a common choice for astrophysical simulations and requires $N_h \approx 42$ [DA12]. We use the relation by Winchenbach et al. [WHK16] that associates h_i with the ideal number of neighbors for a given kernel. They use the adaptive particle volume by Solenthaler [SP08] to compute a scalar value per particle that gives information about the spatial distribution of particles

$$V_i = \frac{1}{\sum_j W(||p_i - p_j||, h_i)}. \quad (3)$$

Note that V_i is equal to the inverse of the previously presented density computation with a constant value $A_j = 1$. The basic idea behind the approach by Winchenbach et al. is that a number of particles with radius V_j intersect with the sphere of radius h_i around a particle. The adaptive smoothing length h_i is then given by [DA12]:

$$h_i = s_c V_i^{\frac{1}{3}} \left(\frac{N_h}{\frac{4}{3}\pi} \right)^{\frac{1}{3}}, \quad (4)$$

where s_c is a global scaling factor.

In order to compute the density per particle, we use an iterative approximation of h_i. In contrast to Winchenbach et al. we do not adapt h_i over the runtime and multiple steps of a simulation, and apply different initialization values and break conditions (see Algorithm 2). We perform the density estimation step once per particle selection operation and execute several iterations until the average Δh_i (referred to as Δh) is small enough between two iterations or the maximum number of approximation iterations is reached. The initial h_i is computed in two steps. First we compute an initial guess h_a, which assumes that the particles are evenly distributed in the domain. h_a is based on the maximum dimension of the dataset in 3D and the number of particles $d_i = \frac{dim_i(\text{dataset})}{\text{number of particles}}$ and $h_a = max(d_x, d_y, d_z)$. Secondly, we relate h_a to the kernel W and the neighborhood of every particle. The idea behind the formula is the assumption that a linear increase of the radius h_i causes a cubic increase in the number of neighboring particles:

$$\text{init}(h_i) = \left(\frac{\sum_j W_N(||p_i - p_j||, h_a)}{N_h} \right)^{\frac{1}{3}} h_a, \quad (5)$$

where W_N is a simple kernel that weights every particle that is included in the sphere h_a around the i-th particle with 1

$$W_N(d, h) = \begin{cases} 1 & \text{if } d < h \\ 0 & \text{else.} \end{cases} \quad (6)$$

We interpolate the estimations from two iterations linearly by weighting both values equally. Note that the initial assumption for h_i can be computed upon loading of the dataset and be reused for every selection operation if the dataset does not change over time. In case of a running simulation, the initial value for h_i has to be computed every time and should not be reused for faster conversion rates of the estimation algorithm.

Algorithm 2 Density-Estimation Algorithm

```
for all particles i do
    h_i ← init(h_i) (Equation 5)
end for
Δh ← 0, i ← 0
loop
    for all particles i do
        Compute V_i according to Equation 3
    end for
    Δĥ ← 0
    for all particles i do
        Compute ĥ_i according to Equation 4
        Δĥ ← Δĥ + ||h_i − ĥ_i||
        h_i ← h_i + (ĥ_i − h_i)/2
    end for
    Δĥ ← Δĥ / number of particles
    if ||Δh − Δĥ|| < ε or i ≥ i_max then
        break
    end if
    Δh ← Δĥ
    i ← i + 1
end loop
```

From a practical point of view, we do not compute and store particle neighbor lists. We separate the whole simulation domain into an uniform virtual grid. The virtual grid does not exist explicitly in memory in form of a 3D memory buffer. Every particle will be assigned the virtual grid cell the particle lies in. Afterwards, we apply the radix-sort based approach by Green [Gre10a, Gre10b] to sort all particles according to their grid cell. This dramatically improves neighbor-lookup performance later on and does not require additional memory larger than $O(n)$.

3.4 Particle Selection

Related papers perform the actual selection based on their triangulated meshes that are used to form clusters of a certain density. Access to such information is not available in our case, but we can work on the implicitly defined particle-neighbor relationship. This link inherently defines particle clusters via their density information, and thus avoids the explicit computation of clusters in memory. In general, we use a flood-filling approach that will mark all

neighboring particles, if a certain condition evaluates to true. We consider a particle p_j a neighbor to a current particle p_i, if their distance is smaller than the smoothing length h_i computed in the density estimation step.

In the first step, we mark all particles which have depth values that fall into the depth interval of the focus area, called *direct target particles*. While marking the start particles, we compute their minimum and maximum density values and the resulting density delta $\Delta\rho$. We mark all *indirect target particles* using an iterative marking process. We iteratively consider all neighboring particles of all marked particles and test whether $|\rho_i - \rho_j| < \Delta\rho$. We further transform the particle coordinates into screen space and test whether the projected position lies inside the selection mask. As previously mentioned, this test can also be disabled to allow the automatic selection or more involved structures. If these conditions evaluate to true, we will mark the corresponding neighbors. We stop the iterative marking process as soon as no particle is marked or when the maximum number of indirect marking iterations is reached. Note that the density delta $\Delta\rho$ is not adjusted in any iteration since it represents the initial density of the selection.

3.5 Complexity

The total complexity is given by the sum of the single complexities of every required operation since these are performed sequentially one after another

$$C_{\text{Total}} = O\left(C_{\text{Density}} + C_{\text{Selection}}\right). \tag{7}$$

The density estimation (Algorithm 2) is the most expensive operation in general since it works on particles from the dataset. It performs several passes over all particles and most loops require nested loops over the neighboring particles. This results in:

$$C_{\text{Density}} = O(n \cdot k + 2 \cdot n \cdot k \cdot i_{\max}) = O(n \cdot k \cdot i_{\max}) = O(n \cdot k), \tag{8}$$

where i is the number of density-adjustment iterations and k the maximum number of neighbors per particle. In practice, however, i cannot become greater than $i_{\max}$ which is a constant (typically $\in [1,3]$). Note further that k is approximately N_h, and thus $k << n$.

Similar to the density estimation, the particle selection step involves iterations over the neighboring particles. Every particle marks a neighbor if the density condition for marking evaluates to true. This results in several iterations over the particle set until a fixed point is reached; or in other words, until no additional particles will be marked any more. In the theoretical worst case, only a single particle is marked in the beginning (a single direct target particle). In the first iterative indirect marking iteration, only its k neighbors will be marked, resulting in $k+1$ marked particles in total. The next marking iteration will then result in $k \times k + 1$ marked particles. However, if we reconsider the worst case, only one additional particle is marked in every iteration. This results in a total number of n marking iterations. Hence, the complexity $C_{\text{Selection}}$ is $\in O(n \cdot k)$. In practice, however, approximately $k^i + 1$ particles are typically marked after the i-th iteration.

Consequently, the total computational complexity C_{Total} of our method is

$$C_{\text{Total}} = O(n \cdot k). \tag{9}$$

Note that the complexity of the rendering steps are not included in this analysis. Rendering steps have to be performed for visualization purposes anyway. In terms of memory consumption, the algorithm requires a linear amount per particle only to store the selection state, the smoothing lengths and the actual density. In addition, we require several negligible screen space buffers to store mask information and bin values of the focus area. This results in a total memory complexity of

$$C_M = O(i \cdot n) = O(n). \tag{10}$$

4 Evaluation

The overall evaluation focuses on two aspects: runtime performance and selection quality. Our test scenarios are based on the ones used by related work [YEII16] in order to have a comparable evaluation in terms of selection quality. Furthermore, they reflect different common selection tasks that can arise in real-world datasets. In contrast to Yu et al. we fixed the virtual camera position and instead used a larger number of datasets during the evaluation. This yields further insights in terms of applicability to different general sets of particles.

We used 12 different scenarios that are visualized in Figure 6 with a screen resolution of 1920x1080 pixels while setting the number of focus area bins to 16. In practice, common datasets contain hundreds of thousands of particles. Therefore, our smallest set contains around 150.000 and our largest set around 460.000 particles. Additionally, we added noise particles to enhance the degree of realism (see Table 1). We follow the evaluation approach by Yu et al. and used scenarios that can be assigned to the following major categories: whole clusters, partial selection and occluded selection. Note that some scenarios can be assigned multiple categories.

Whole clusters The selection of a single or even multiple clusters at once is very common (Scenarios 1, 2, 4, 5, 6, 8, 9, 10 and 11). Consequently, most of our evaluation scenarios reflect these kind of tasks. From a user's point of view, this task is very easy and convenient: a single lasso is drawn on the screen that includes the target cluster/clusters. From an algorithmic point of view, the evaluation scenarios differ drastically as the shape of the selection lasso, the particle densities and the desired user intentions are different. *These scenarios focus on the general mapping of selection masks to visible regions of the dataset.*

Partial selection More complex selection tasks are partial selections, which are also more sophisticated to process in general since only parts of certain particle clusters should be selected (Scenarios 3, 7 and 12). Compared to other structure-aware methods, our algorithm does not require specific support to perform partial selections because it does not consider nor rely on the concept of explicit clusters (see subsection 3.4). *These scenarios focus on the precise mapping of selection masks to visible regions of the dataset.*

Occluded selection The most difficult selection type is an occluded selection: users want to select a subset that is either partially or fully occluded by other particles (Scenarios 2, 3, 4, 7 and 10). This is also the most difficult scenario type for our approach since we rely on screen space information in form of a depth buffer. *These scenarios focus on the mapping of selection masks to partially occluded regions of the dataset.*

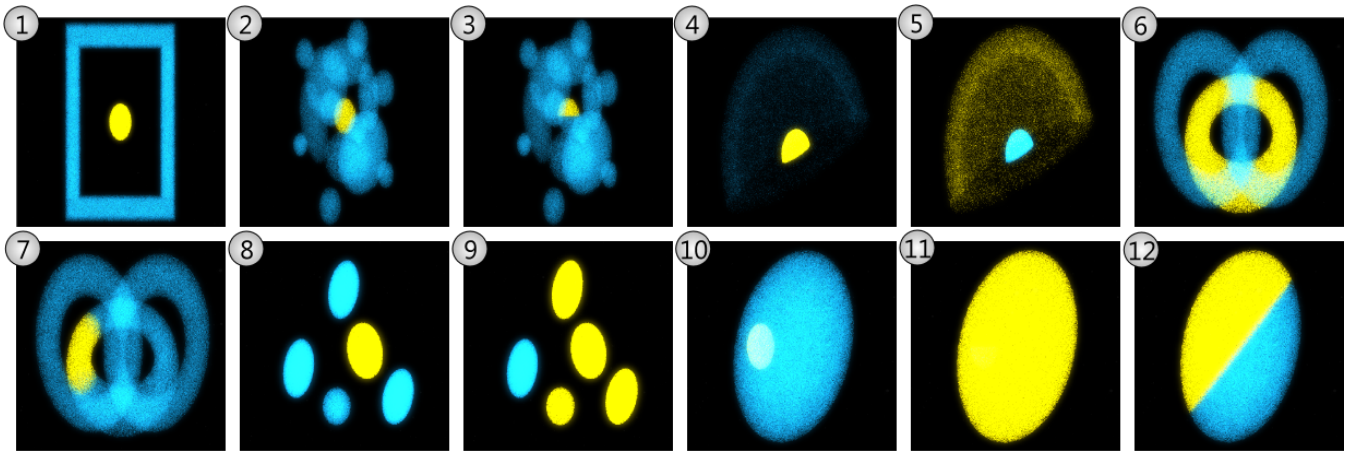

Figure 6: *A visualization of the different evaluation scenarios from the used camera perspectives. These images were displayed to the users during the online study. Note that the first scenario was used to familiarize themselves with the selection process.*

Table 1: *Detailed information about the evaluation scenarios (# Particles). Target particles are particles that have to be selected by the user with the help of a selection mask. Default particles represent valid data points in the scope of the dataset, but they should not be selected.*

Scenario	# Particles	# Target Particles	# Default Particles	# Noise Particles
1	457152	52158	399998	4996
2	286136	10476	270662	4998
3	286136	5236	275902	4998
4	153917	104752	44169	4996
5	153917	44464	104457	4996
6	251650	82194	164462	4994
7	251650	16986	229670	4994
8	449734	104865	339874	4995
9	449734	340279	104460	4995
10	371792	105053	261741	4998
11	371792	236099	130695	4998
12	371792	314331	52463	4998

4.1 User Study

We have conducted an online user study using an interactive web application to gather selection masks for evaluation purposes. The different scenario images from Figure 6 were displayed to users in a counterbalanced manner and the first scenario was used for training purposes only. Users were not able to rotate or move the virtual camera through the dataset as they worked on static images only. A set of several demonstration slides explained the basic selection concepts of drawing lassos. Afterwards, users could train the actual selection process on the first scenario to get initial insights. The selection masks were sent asynchronously to our server via Websocket requests and were processed in real time. The resulting image with the highlighted selection was rendered on the server and was sent to the users' browsers together with additional information (number of selected particles etc.). We considered selections as invalid and discarded the selection information if less than 1000 particles were selected. Users were not able to continue the user study as long as their selection on the currently displayed scenario was considered to be invalid. Besides raw selection masks, we asked users for their age, their gender, color blindness, their input device, their experience with their input device (from 0 to 5) and their experience in the field of particle selection (from 0 to 5). The recoded selection masks were used for the computation of the selection quality and runtime performance.

We had 72 participants and were able to use 67 users for evaluation purposes. Data from five participants could not be used since they did not complete the online study for unknown reasons. All participants used a mouse as an input device and were able able to differentiate between blue (default), noise (gray) and target (yellow) particles. In sum, we gathered 737 selection masks, the ages of our participants ranged from 20 to 46 years ($M = 26.4, \sigma = 5.8$) while 9 were female. Most of them were not that familiar with the field of particle selection ($M = 2.1$) but very experienced with the mouse ($M = 4.5$).

4.2 Selection Quality

Following known approaches from related work [YEII12, YEII16], selection quality is measured in terms of two accuracy scores, the F1 score and the MCC (Matthews correlation coefficient) score. We applied the gathered selection lassos from the user study to the underlying data sets in order to determine the selection precision. Table 2 shows the average F1 and MCC scores for all user inputs and scenarios. All scores are very high (above 91%) and indicate a very good selection across all evaluation datasets. In some cases, however, the selection quality is slightly worse compared to other scenarios. For instance, scenario 3 has the worst scores with $F1 = 0.92$ and $MCC = 0.92$, which are still high but slightly lower than the others. First, the selection is a partial selection that is more sensitive to the actual mask information. Secondly, the density of the target particles is very similar to the density of its surrounding. These arguments also apply to scenario 7, where $F1 = 0.92$ and $MCC = 0.93$. However, even in these cases our approach is able

Table 2: *Selection quality measurements in terms of the average F1 and MCC scores.*

Scenario	2	3	4	5	6	
F1	0.98	0.92	0.98	0.96	0.95	
MCC	0.97	0.92	0.97	0.97	0.94	
Scenario	7	8	9	10	11	12
F1	0.92	0.99	0.99	0.96	0.96	0.98
MCC	0.93	0.99	0.99	0.96	0.96	0.97

to achieve high quality selection results and does not suffer from severe limitations.

4.3 Runtime Performance

We have implemented our system in C# and use the ILGPU[‡] compiler for all GPU kernels. We leverage Direct3D for visualization purposes that seamlessly integrate with the GPU kernel functionality. Performance tests were executed on two different GPUs: an Nvidia GeForce GTX 980 TI and an Nvidia GeForce GTX 1080 TI (see Table 3). A single performance measurement is the median execution time of 100 selection processes with the same captured selection mask from the user study. The actual density values were computed using a cubic spline kernel, where the ideal number of neighbors was set to 45. Although we limited the number of density iterations to be less or equal to three, our density estimation algorithm required only up to two iterations in order to reach a reasonable number of neighbors per particle (see Table 4).

The runtime of the density analysis heavily depends on the number of particles and the number of approximation steps to adapt the smoothing lengths. For example, scenario 2 requires a single iteration step with around 286.000 particles, whereas scenario 6 requires two density iterations with roughly the same number of particles. As expected, the time for performing two density steps is approximately twice as high as a single step on both GPUs with a comparable number of particles. Our initial guess of the smoothing length based on Equation 5 works well on the evaluation scenarios, since we do not even reach the predefined limit of three adjustment steps.

The runtime of the selection process on both GPUs does not exceed 25 milliseconds. If we compare the measurements of scenarios 8 and 9, we can see a significant difference between the performance. This is caused by the large difference in the number of selected particles: we have to perform more indirect marking iterations in general as the number of desired particles increases.

5 Conclusion

We present a novel particle selection method called SSPS that does not require high computational complexity and is easy to implement using the presented algorithms. In terms of runtime performance, SSPS is fast enough on current hardware (less than 50ms in the worst case on our evaluation scenarios) such that it is perfectly suitable for interactive applications. In the best case, however, we required only around 10ms to complete a selection process on a dataset with a significant number of particles. When focusing on selection quality, we have shown that our method performs as well as other state-of-the-art approaches that rely on different analyses.

Similar to other approaches, an inherent limitation of our method are the general parameters that control the actual selection process. For instance, our selected values for the number of focus area bins have proven themselves in our evaluation scenarios, for example, in order to achieve very high accuracy values. In particular, the number of bins have to be adjusted properly as they control the overall sensitivity of the selection process. This also affects our proposed mask weighting functionality to reflect a user's intention. Depending on the general domain (e.g. medical data) and the underlying structure of the particles, different values can lead to better results in terms of intended selections.

In the future, additional datasets from different domains have to be analyzed. Further insights will improve the parameter choices and can reveal novel application domains for our selection algorithm. A perfect extension to our proposed method would be an automatic parameter adjustment step that determines proper values based on several initial user selection steps. Another interesting next step can be the application to real-time simulations that can be analyzed on-the-fly as the simulations run.

Table 3: *Runtime performance measured on an Nvidia GeForce GTX 980 TI and on an Nvidia GeForce GTX 1080 TI in milliseconds. Density refers to the runtime of the density estimation step, whereas the Selection column includes the mask construction, the focus area computation and the actual selection.*

Scenario	Density	Selection	**Total**	σ	Density	Selection	**Total**	σ
2	10	11	**21**	0.8	6	4	**10**	0.6
3	10	10	**20**	0.8	6	4	**10**	0.6
4	8	18	**26**	1.1	5	11	**16**	0.7
5	8	11	**19**	1.1	5	5	**10**	0.7
6	16	14	**30**	0.9	11	9	**20**	0.8
7	16	10	**26**	0.9	11	5	**16**	0.8
8	24	16	**40**	0.9	15	12	**27**	1.0
9	24	25	**49**	0.9	15	16	**31**	1.0
10	20	18	**38**	1.2	14	13	**27**	1.1
11	20	22	**42**	1.2	14	14	**28**	1.1
12	20	24	**44**	1.2	14	16	**30**	1.1
GPU	**GeForce GTX 980 TI**				**GeForce GTX 1080 TI**			

Table 4: *Density information related to our density estimation algorithm. Note that all particle positions were normalized to fall into the range of* $[(-10,-10,-10)^T,(10,10,10)^T]$.

Scenario	2,3	4,5	6,7	8,9	10,11,12
Density # Iterations	1	1	2	2	2
Max. Smooth. Length	1.65	0.89	1.35	0.53	0.69
Avg. # Neighbors	42	43	44	47	44

[‡] www.ilgpu.net

References

[BHS*11] B. J. FERDOSI, H. BUDDELMEIJER, S. C. TRAGER, M. H. WILKINSON, J. B. ROERDINK: Comparison of density estimation methods for astronomical datasets. *A&A* (2011). 2

[DA12] DEHNEN W., ALY H.: Improving convergence in smoothed particle hydrodynamics simulations without pairing instability. 5

[DFK12] DAIBER F., FALK E., KRÜGER A.: Balloon Selection revisited - Multi-touch Selection Techniques for Stereoscopic Data. In *Proceedings of the International Conference on Advanced Visual Interfaces* (2012). 1

[DSG*14] DAIBER F., SPEICHER M., GEHRING S., LÖCHTEFELD M., KRÜGER A.: Interacting with 3D Content on Stereoscopic Displays. In *PerDis* (2014). 1

[GM77] GINGOLD R. A., MONAGHAN J. J.: Smoothed Particle Hydrodynamics-Theory and application to nonspherical stars. *Notices of the Royal Astronomical Society* (1977). 4

[Gre10a] GREEN S.: Particle Simulation using CUDA – Parallel Radix Sort, 2010. 2, 5

[Gre10b] GREEN S.: Screen Space Fluid Rendering for Games. Presentation at the Game Developers Conference, 2010. 2, 5

[HWVF12] HEGE H.-C., WIEBEL A., VOS F. M., FOERSTER D.: WYSIWYP: What You See Is What You Pick. *IEEE Transactions on Visualization and Computer Graphics* (2012). 2

[Kel06] KELAGER M.: Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics, 2006. 4

[KK16] KÖSTER M., KRÜGER A.: Adaptive Position-Based Fluids: Improving Performance of Fluid Simulations for Real-Time Applications. *CoRR* (2016). 2, 4

[KSW99] KODA J., SOFUE Y., WADA K.: How to Determine the Smoothing Length in Sph? In *Astrophysics and Space Science Library* (1999). 5

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics* (1987). 1, 2

[Luc77] LUCY L. B.: A numerical approach to the testing of the fission hypothesis. *Astronomy Journal* (1977). 2, 4

[Min95] MINE M. R.: *Virtual Environment Interaction Techniques*. Tech. rep., 1995. 1

[MM13] MACKLIN M., MÜLLER M.: Position Based Fluids. *ACM Transactions on Graphics* (2013). 2, 4

[MMCK14] MACKLIN M., MÜLLER M., CHENTANEZ N., KIM T.-Y.: Unified Particle Physics for Real-Time Applications. *ACM Transactions on Graphics* (2014). 2

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. In *Annual Review of Astronomy and Astrophysics* (1992). 2, 4

[Mon00] MONAGHAN J. J.: SPH without a Tensile Instability. *Journal of Computational Physics* (2000). 4

[Mon02] MONAGHAN J. J.: SPH compressible turbulence. *Monthly Notices of The Royal Astronomical Society* (2002). 5

[NP94] NELSON R. P., PAPALOIZOU J. C. B.: Variable Smoothing Lengths and Energy Conservation in Smoothed Particle Hydrodynamics. *Monthly Notices of the Royal Astronomical Society* (1994). 5

[ONI05] OWADA S., NIELSEN F., IGARASHI T.: Volume catcher. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (2005). 1

[PFC*97] PIERCE J. S., FORSBERG A. S., CONWAY M. J., HONG S., ZELEZNIK R. C., MINE M. R.: Image plane interaction techniques in 3d immersive environments. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics* (1997). 1

[SP04] STEED A., PARKER C.: 3D Selection Strategies for Head Tracked and Non-Head Tracked Operation of Spatially Immersive Displays. In *8th International Immersive Projection Technology Workshop* (2004). 2

[SP08] SOLENTHALER B., PAJAROLA R.: Density Contrast SPH Interfaces. In *Eurographics/SIGGRAPH Symposium on Computer Animation* (2008). 5

[Tuk77] TUKEY J. W.: *Exploratory Data Analysis*. 1977. 1

[VMM99] VOLLMER J., MENCL R., MÜLLER H.: Improved laplacian smoothing of noisy surface meshes. In *Computer Graphics Forum* (1999). 3

[VSB*10] VALKOV D., STEINICKE F., BRUDER G., HINRICHS K., SCHÖNING J., DAIBER F., KRÜGER A.: Touching Floating Objects in Projection-based Virtual Reality Environments. In *Joint Virtual Reality Conference of EGVE - EuroVR - VEC* (2010). 1

[WHK16] WINCHENBACH R., HOCHSTETTER H., KOLB A.: Constrained neighbor lists for sph-based fluid simulations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2016). 5

[WVH11] WIEBEL A., VOS F. M., HEGE H.-C.: *Perception-Oriented Picking of Structures in Direct Volumetric Renderings*. Tech. rep., ZIB, 2011. 2

[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* (2012). 1, 2, 7

[YEII16] YU L., EFSTATHIOU K., ISENBERG P., ISENBERG T.: CAST: Effective and efficient user interaction for context-aware selection in 3D particle clouds. *IEEE Transactions on Visualization and Computer Graphics* (2016). 1, 2, 3, 6, 7