Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Supervised learning for boundary detection on particle systems

Marcos Sandim [1]
ICMC-USP
Afonso Paiva [2]
ICMC-USP

Abstract. In particle-based physics simulations, the information about which particles belong to the boundary of the system and which are considered internal is, in general, an information that is useful but hard to obtain in an efficient way. This information can be applied to generate the free surface of the fluid or to compute the surface tension, among other applications. Techniques found in the literature may present satisfactory results, but in general they are sensible to the problem scale, particle distribution and involve computationally expensive operations such as matrix inversion. The goal of this study is to present an alternative with lower computational cost at runtime by transferring some of the work to a preprocessing step using offline learning.

Keywords. computational geometry, supervised learning, particle systems

1 Introduction

The area of Computational Fluid Dynamics (CFD) has received attention for years due to its application on many fields, ranging from industry to entertainment. Meshless methods such as Smoothed Particle Hydrodynamics (SPH) [7], although very popular, still have a series of problems with suboptimal solutions. One of these problems is the detection of boundary particles [5, 10] This information is useful on other steps of a simulation such as the pressure computation on incompressible simulations, which needs a boundary condition to make its solution unique. Other uses are the application of surface tension, and visualization of the free-surface of the fluid.

An interesting approach to this problem that hasn't been explored on the Computer Graphics or Computational Physics literature, is the use of Machine Learning (ML) to solve a classification problem that separates the boundary particles from the internal ones. During a simulation step using the SPH method, many intermediary attributes are computed for each particle of the system, but in most cases they are discarded at the end of the step. The main idea of this article is to use these attributes along with a few others

[1]sandim@icmc.usp.br
[2]apneto@icmc.usp.br

that can be easily computed to build a feature vector that represents the particle and ML to classify the particles. A recent study about this problem, using a different approach, was made recently by Sandim *et al.* [10] and it can be used as a primer on the subject.

To successfully apply an supervised learning method a ground truth is needed, since supervised learning methods need labels in its training phase. This ground truth is computed trough a more complex and expensive method proposed by Dilts [5] which uses a notion that can be interpreted as an ε-boundary definition that will be discussed ahead. With the labels provided by Dilts' method and feature vectors for a representative subset of the data, we can train a classifier that can be applied to other particle systems with similar characteristics.

2 Supervised learning

In a classification problem, supervised learning consists in feeding data with known labels to an algorithm so it can learn a mapping between the data and the labels. This mapping can then be used to assign labels to new data samples. In our case, the data comes from a set of features computed for each particle and the labels are, at first, unknown. In the next sections we'll discuss in detail how we obtain both the data and the labels used in the training step.

2.1 Feature vectors

During the simulation process of a particle system, most methods – like SPH – produce a lot of data for each particle of the system. In many cases a lot of this data is discarded at the end of the simulation step. This data is composed by some simple variables, like the number of neighbors inside the influence radius of a particle, some are more complex: the particle's velocity, density or pressure, and derivatives (gradient, divergent, curl) of some of them. Since the behavior of each particle is a result of these variables, we can use them to create a feature vector that represent the particle.

Two points arise from the idea of using this data on a feature vector: which variables are useful to separate the boundary from the interior, and how are they related to each other. Variables that don't carry any information about the surroundings of a particle aren't really useful when trying to separate the boundary of the system. The particle's position in $\mathbb{R}^n$, for instance, describes its location in space but doesn't say much about its position relative to other particles or the system itself. On the other hand, information that is derived from the surroundings of a particle are more useful for us. Particles on the boundary, or near it, tend to have fewer neighbors, lower density, it's density have a larger gradient, and so on.

The problem with some of these variables is that they are highly correlated, which makes them redundant. On Fig. 1 we show a correlation matrix created from a small sample of the data that can be extracted from particles. This is very useful to help us better understand this problem as it helps to identify and eliminate variables that add little to no information.

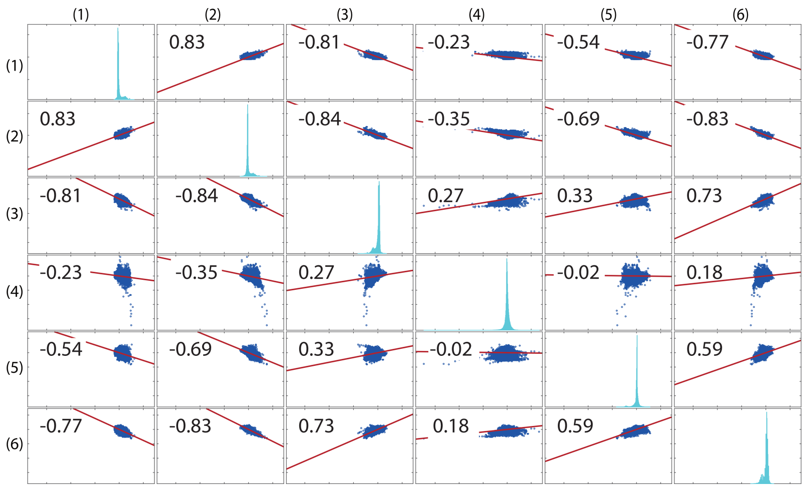

Figure 1: Plot of the correlation matrix of the chosen features: (1) modulus of the gradient of the density function, (2) divergence of the density function, (3) smallest eigenvalue of the covariance matrix, (4) largest eigenvalue of the covariance matrix, (5) density function, (6) number of neighbors inside influence radius.

For instance, the density of a particle in the SPH method uses a density estimator based on the Parzen window [8] with an additional scale by the particle's mass: $\rho_i = \sum_{j \in N_i} m_j W_{ij}$. Here ρ_i represents the density of the particle i, $\mathcal{N}_i$ the set of the neighbors (particles inside the influence radius) of i, m_j is the mass of j and W_{ij} is the value at the position of j of a kernel function, usually the gaussian kernel, centered on i. If all the particles have the same mass, it can be moved outside the summation and becomes just a scale: $\rho_i = m \sum_{j \in N_i} W_{ij}$. Since the chosen features will be normalized in a further step, this scale by the mass is useless and can be removed. This leaves us with a simple density estimator based on the Parzen window, which is simpler and carries similar information.

The density function in the context of the SPH method is a bit noisy thanks to the discretization used and the sum of kernel functions. This variation can be useful to identify regions near the boundary of the system. The gradient of the density function is greater near the boundaries of the system, so the modulus of the gradient of density function is greater at the position of particles near or at the boundary of the system.

By using Principal Component Analysis (PCA) on the neighborhood of a particle, we can differentiate between particles near the boundary as they have very different eigenvalues. Particles inside the system have roughly the same eigenvalues as their neighborhood is evenly distributed around them.

The final set of features used to train a classifier capable of extracting the boundary of a system are: the density function, the modulus of its gradient and divergence, the number of neighboring particles inside the influence radius of the particle, and the smallest and largest eigenvalues of the covariance matrix of the positions of the particles on the neighborhood of each particle.

2.2 Ground truth and labels

A solid ground truth that can be used as labels for training is essential in training and testing a classifier. Here we discuss how we use the method proposed by Dilts [5] to generate the labels that we use.

Consider a point set $\mathcal{P}$ that represent the positions of particles in a system, this set can be seen as a sampling of a closed region Ω that contains the system and has a surface $S = \partial\Omega$. As discussed by Sandim *et al.* [10], a boundary point of $\mathcal{P}$ is a point that lies on S. Since it is quite hard to know if a point lies exactly on S, a more useful definition takes into account the sampling density of the set, as proposed by Sandim *et al.*. By taking into account the sampling density, Sandim *et al.* give the definition of ε-boundary points. These definitions match the method proposed by Dilts, so we can use this method to compute our labels. Albeit faster, the method proposed by Sandim *et al.* can't be applied here since it gives an approximation of the set of boundary particles of $\mathcal{P}$.

By applying directly the *arcs* method from Dilts [5], we obtain a reliable set of labels that follow the definitions given by Sandim *et al.* [10]. These labels can then be used by us to train our classifier in order for it to identify ε-boundary particles, which will be called only boundary particles for simplicity.

2.3 Training and testing

We chose to use a Support Vector Machine (SVM) classifier was used. This choice was based on the fact that its training phase may be costly but the classifying phase has linear complexity on the number of support vectors. This means that the majority of the computational effort is moved to the training phase of the method, and allowing the already trained classifier to be used inside the pipeline of a particle-based simulation.

The literature of Support Vector Machines dates back to 1964, with Vapnik and Chervonenkis [11] first introducing the idea of statistical learning, which is the basis for the SVM method. The SVM method itself was introduced by Boser *et al.* [2] and ,from then on, many others extended the idea to adapt and use the SVM in different applications.

The training data is extracted from preprocessed SPH simulations, which contains the necessary information to compute the feature vector for each particle. All steps of the simulations used in training and testing have their labels computed according to Section 2.2. To build a training set that contains different cases from different configurations, we do the sampling in two phases: sampling of the time steps of the simulation and sampling of particles inside the sampled steps.

To sample the time steps of the simulation we use a random sampling of 30% of the available steps to be used as training, leaving the other 70% to testing. From the training steps we do a hold-out validation in which we chose again 30% of the particles from the step to be used as training and leaving the rest to testing. Note that the 30% of the particles chosen from a step are composed by roughly the same amount of internal and boundary particles to ensure that both cases are properly represented in the training data.

With this partitioning scheme we use only 9% of the available data as training, but if we consider that the total size of the datasets available from our SPH simulations are

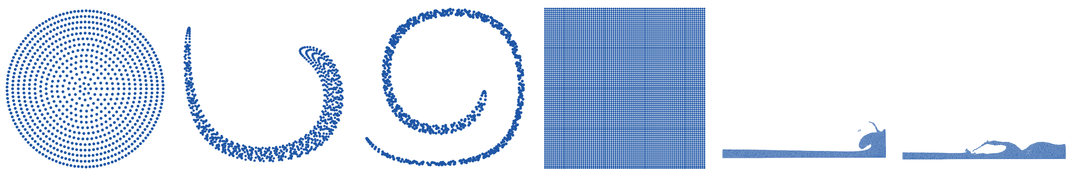

Figure 2: Sample steps from the training data.

often around 100 steps with 500 thousand particles each, adding up to a total of 50 million different instances in a single simulation, the training set with 9% of this still has a total of 4.5 million samples. This training set is quite large and can be problematic to process in a conventional workstation with 8GB of RAM. The use of Sequential Minimal Optimization (SMO) to solve the optimization step of the SVM, proposed by Platt *et al.* [9], reduces the memory needed during the training phase, since its memory requirements are linear to the training set size. An extended version of the SMO was made by Fan *et al.* [6].

The training is made using the SVM implementation from LIBSVM [3], with an extended version of the SMO algorithm made by Fan *et al.* [6] and a Radial Basis Function (RBF) kernel. The choice of the RBF kernel over a linear was based on the fact that the groups are not linearly separable using the chosen features, preventing the convergence of the SMO algorithm in an acceptable number of iterations. After the training phase, the remaining data is used as testing and the confusion matrix is computed. Along with the confusion matrix we plot the particle system with the labels from the ground truth and the classifier's results, this plot shows not just the number of correct or wrong cases but also where they occur. The information of where the particles are classified on the wrong group is useful to understand the effect of the chosen features on the classifier.

3 Results

The method was implemented using MATLAB, which uses LIBSVM as the back-end for its SVM tools. The tests were run in a workstation equipped with an Intel® Core$^{\text{TM}}$ i5-3570 processor and 8GB of RAM. The data used was obtained from SPH simulations made with DualSPHysics, which is an open source SPH simulation tool built and maintained by Crespo *et al.* [4] and others. Another dataset is the single vortex spin by Bell *et al.* [1], which is a common benchmark in the computational physics literature.

We trained a classifier using the information from the single vortex spin and a dam break simulation. Both start with a well behaved distribution but evolve to configurations with thin layers, sharp features, drops and holes. These features are fundamental to the training since the feature vectors of the particles in these regions carry the information needed to identify these structures. Fig. 2 shows a few steps of these datasets with the mentioned features. It is important to include extreme cases in the training data, with bad particle distribution, such as the tail of the system in the single vortex dataset. These cases are the ones that push the classifier to its limit.

The results obtained with the training data are consistent, even between runs with different partitioning of the data. This means that the partitioning scheme is successfully

Table 1: Confusion matrix from the classification of the testing data.

		Predicted class	
		Internal	Boundary
Actual class	Internal	562,246	14,813
	Boundary	1,600	86,251

Table 2: Confusion matrix from the classification of the double dam break data.

		Predicted class	
		Internal	Boundary
Actual class	Internal	3,917,101	285,354
	Boundary	11	204,090

Table 3: Time taken in seconds by the classification of the double dam break data.

Method	Total time (sec)	Avg time/step (sec)
SVM	317.80	1.26
Reference	4,979.28	19.83

including representative samples into the training set. The confusion matrix on Table 1 shows a very low False Negative Rate (FNR) meaning that a low amount of boundary particles were wrongly classified as internal. A low FNR also indicates that the boundary has few or no holes. On the other hand, the False Positive Rate (FPR) is somewhat high, indicating that there is a considerable amount of internal particles being classified as boundary particles. This higher FPR may be a problem to some applications, but this can be solved by using a more expensive method such as Dilts' method [5] to filter this cases and reduce the FPR.

After training and testing, we used a different simulation made with DualSPHysics to analyze the use of a classifier built with one simulation to classify another one and test the generalization of the model. In this case we used a double dam break scene, with a higher number of particles and finer discretization. The results were similar to the testing data but with a higher FPR as seen on Table 2, this is a grave issue that needs to be solved. On the other hand, the time to classify all the particle of each time step of the simulation using the SVM model is lower than the time needed to compute the reference method used as ground truth. After all 250 time steps were processed, the total time spent by the ML approach is considerably lower than the reference method. Table 3 shows the time in seconds spent by each technique to process the entire dataset. The geometric approach consumes more than 15 times more time to label the same data, this shows that while the approach using SVM may yield a high FPR it can be much faster than the geometric approach. Fig. 3 shows a few steps comparing the reference method with our proposal.

4 Conclusion

With this article we can conclude that it is possible to employ ML techniques to solve the boundary detection problem with low time overhead on the moment of classification. The running time of the proposed approach is more than 15 times lower than the reference method, this was one of the objectives and was successfully met. However the high FPR may turn the method unusable in some contexts where lower error rates are a requirement.

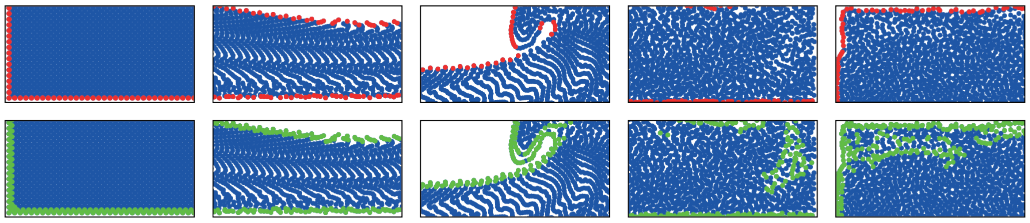

Figure 3: Results of the classification of the double dam break simulation. The top row contains the results from the reference method, and the bottom row the results obtained by the proposed method. It is clear that the proposed method gives a closed boundary but has a high FPR.

A third possibility, apart from a pure ML or pure geometric solution, is to combine both approaches in a two phase method: coarse phase using SVM and a fine phase using Dilts' method [5]. The second phase can be run only on the particles labeled as "boundary candidates" by the classifier used in the first phase. With this we may be able to achieve smaller running time than the reference method, but with FPR in acceptable levels.

References

[1] J. B. Bell, P. Colella, and H. M. Glaz. A second-order projection method for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 85:257–283, 1989.

[2] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92*, pages 144–152, 1992.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2:27:1–27:27, 2011. Code at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] A. Crespo, J. Domínguez, B. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, and O. García-Feal. DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH). *Comput. Phys. Commun.*, 187(0):204 – 216, 2015.

[5] G. Dilts. Moving least-squares particle hydrodynamics II: conservation and boundaries. *Int. J. Numer. Meth. Eng.*, 48(10):1503–1524, 2000.

[6] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using second order information for training support vector machines. *The Journal of Machine Learning Research*, 6:1889–1918, 2005.

[7] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Mon. Not. R. Astron. Soc.*, 181:375–389, 1977.

[8] E. Parzen. On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[9] J. Platt et al. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods—support vector learning*, 3, 1999.

[10] M. Sandim, D. Cedrim, L. G. Nonato, P. Pagliosa, and A. Paiva. Boundary detection in particle-based fluids. *Comput. Graph. Forum*, 35(2):215–224, 2016.

[11] V. Vapnik and A. Y. Chervonenkis. A class of algorithms for pattern recognition learning. *Avtomat. i Telemekh*, 25(6):937–945, 1964.