

# A GPU-based Interactive Bio-inspired Visual Clustering

Ugo Erra

Dipartimento di Matematica e  
Informatica  
Università della Basilicata  
Potenza, Italy  
Email: ugo.erra@unibas.it

Bernardino Frola

Dipartimento di Informatica ed  
Applicazioni  
Università di Salerno  
Fisciano, Italy  
Email: frola@dia.unisa.it

Vittorio Scarano

Dipartimento di Informatica ed  
Applicazioni  
Università di Salerno  
Fisciano, Italy  
Email: vitsca@dia.unisa.it

**Abstract**—In this work, we present an interactive visual clustering approach for the exploration and analysis of vast volumes of data. The proposed approach is based on a bio-inspired collective behavioral model used in a 3D graphics environment. The paper illustrates an extension of the behavioral model for clustering and a parallel implementation using Compute Unified Device Architecture to exploit the computational power of Graphics Processor Unit. The advantage of this approach is that, as the data enters in the environment, the user is directly involved in the data mining process. Our performance experiments illustrate the effectiveness and efficiency provided by our approach over a number of real and synthetic data sets.

## I. INTRODUCTION

Today, data-intensive science consists of the analysis of vast volumes of scientific data captured by instruments or generated by simulations, resulting in data with a high dimensionality. For instance, CERNs Large Hadron Collider and astronomys Pan-STARRS5 array of celestial telescopes are capable of generating several petabytes of data per day; nevertheless gene sequencing machines are capable of producing thousands or millions of sequences of the genome. The amount of data is so vast that in order to verify and validate the underlying model, the data must be visualized and analyzed. Therefore, both visualization and data analysis require new approaches that enable us high performance processing and intuitive representation when dealing with large amounts of data.

Visual data mining enables to gain insight into data-intensive science through new approaches to visual data analysis and knowledge discovery. The detection and validation of expected results is facilitated by interactive interfaces that improves interpretation of data. Scientists can use visual data analysis systems to explore several scenarios and examine data using multiple perspectives and assumptions. There are a number of visualization techniques that have been developed to support data mining tasks through visualization as for instance clustering.

Clustering is essentially a data mining approach which help to address the problems of the growing data and the scarcity of human attention to discover groups of similar objects.

Video of this work is available only for this review at <http://isis.dia.unisa.it/projects/behavert/cidm2011.wmv>

Each group, called cluster, consists of objects that are similar between themselves and dissimilar to objects of other groups. The meaning and measure of similarity is referred to as the distance metric used to define the closeness between a pair of objects. According to a given similarity, the organization of data into cluster is performed by an unsupervised learning approach starting from an unlabeled dataset from which we want discover how the objects are organized [1]. In addition to discover groups in data mining applications, clustering is a common operation that has numerous applications as for instance, in the study of social networks to recognize communities, in the image processing for object recognition, in the grouping of genes performing similar functions in bio-informatics, and so on.

Many diverse clustering techniques have been developed in the past. The most widely used techniques are: Hierarchical clustering [2], K-means clustering [3], and Self-organized maps [4]. In any case, it is impossible to say that one is better than another because each algorithm has its advantages and disadvantages and performs differently for different problem. For instance, K-means is simple and has good time complexity but suffers the initial positions of centroids which produces a non-deterministic output and also one must decide a priori the number of clusters. On the other hand, hierarchical clustering does not require the number of clusters a priori but has a complexity that is at least quadratic in the number of inputs compared to linear complexity of K-means. Moreover, these algorithms are useful for organizing static data but do not fit well in the analysis of extremely large datasets that can not be hold entirely in the system memory or in the situation where the time delay before new data is assigned into clusters is essential to refresh the organization of data. However, with two- or three-dimensional data the results of different algorithms can be easily explored using simple visualizations but data with a high dimensionality are much more difficult to visualize and understand. Several techniques try to determine a projections of the data in two- or three-dimensional space in order to show the properties of the high-dimensional clusters [5].

Thanks to their good price/performance ratio and hardware programmability, Graphics Processor Unit (GPU) is used today

not only for 3D graphics rendering but also in general-purpose computing. Several works have demonstrated how GPU can lead to a significant performance increase for agent-based simulation compared to CPU [6] [7]. In this paper, we present a clustering algorithm inspired by a flocking behavioral model that exploits the parallel architecture of the GPU. The high-dimensional data are mapped as agents' features. Each agent is assigned a local behavioral model and moves by coordinating with the motion of other agents in a 3D environment space. The effectiveness of our approach relies on the natural organization that arise through a group of agents that interact using local behavioral model and enable the organization of the agents with similar features in clusters. We illustrate the model and an efficient implementation that exploits the parallel architecture of the GPU using the NVIDIA's Compute Unified Device Architecture (CUDA) as programming environment. The proposed clustering does not require the number of clusters as input and data can be introduced interactively on the fly. As general results, the approach enables high performance processing in data analysis and visualization based on an intuitive representation that avoids the projection of the high-dimensional data in two- or three-dimensional space. Experimental results show that the quality of the clustering of our algorithm is guaranteed and the GPU implementation offers a good scales performance.

The remainder of this paper is organized as follows: in section II, we review previous clustering approach based on the GPU. In section III, we describe the behavioral model which inspired our clustering approach. In section IV, we illustrate our clustering algorithm. In section V, we present the implementation on the GPU. Section VI, illustrates some experiments in terms of efficiency and performance scalability. Finally, section VII concludes and discusses future works.

## II. RELATED WORK

An example of a system that uses visualization techniques for high-dimensional clustering is OPTICS [8]. In this work, authors create a one-dimensional ordering of the database representing the density of clustering structure. Points within a cluster are close in the generated one-dimensional ordering and their reachability is provided using a distance plot. This visualization system is valuable for understanding the clustering and guiding the clustering process. Another approach is the HD-Eye system [9]. This system considers the clustering problem as a partitioning problem and allows the user to directly interact in the crucial steps of clustering process. That is, the choice of dimensions to be considered, the selection of the clustering the paradigms, and the partitioning of the data sets.

In the context of clustering, GPUs have demonstrated interesting results. K-means clustering is probably the most studied clustering algorithm on the GPUs. The first implementations that showed how GPU can be used to significantly accelerate K-means are [10] [11]. In these works, authors used an obsolete approach based on shader languages to exploit the computational capabilities of GPUs. Today, general purpose

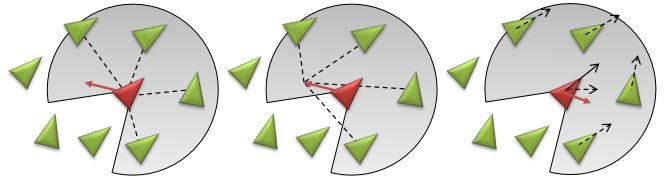


Fig. 1. From left to the right the steering behaviors: separation, cohesion and alignment. In all cases the agent has a local perception of the other agents limited by its field of view.

languages, such as CUDA, offers better support to GPU architectures. In [12] [13], authors have tried to improve the efficiency of the K-means using CUDA and in particular through optimizations directly targeted at parallel architectures. In these works, authors obtain an overall speed up of respectively 14 times and 13 times over the sequential computation to a CPU.

Authors in [14] used a shader language to implement hierarchical clustering. They achieved 2-4 times of speedup compared to a CPU implementation. While [15] explores parallel computation of hierarchical clustering with CUDA, and obtains an overall speed-up of up to 48 times.

In [16], the authors present an implementation on the GPU of a flocking clustering for analyzing and grouping documents. In this implementation, each document is a bird and flies toward other document that are similar to it. As the authors recognize, the limitation of the approach is its complexity  $O(n^2)$ . This inefficiency is caused because in each step of the simulation they create a thread for each agent pair ( $n^2$  thread in total) in order to compares their location in 2D virtual space and then compute the distance between them.

Our approach tackles the problem of  $O(n^2)$  complexity by using a static grid that subdivides the 3D space in cubic cells of the same size. This approach enable us to identify in parallel all agents inside the same cell or within a given region considering agents inside the cells that overlap a region of interest. Experiments showed that this approach can speed-up the clustering algorithm allowing a faster data exploration and dealing with data sets containing thousands of data items.

## III. THE BEHAVIORAL MODEL

Our clustering approach is inspired by the original behavioral models proposed by Reynolds [17]. In this model, each agent has a strictly local perception of the space it occupies. None of the creatures being part of group has a full knowledge of the entire group. Hence, the decisions must be taken by every agent only from neighbors that are perceived from its fields of view (Figure 1). Based on the visibility of each agent the synchronized aggregated motion of the group is achieved performing a weighted sum of *steering behaviors*. In particular, Reynolds defined three basic steering behaviors as illustrated in Figure 1. The first, *separation* tends to keep distance from other neighbors. This behavior is necessary to prevent agents collision. A repulsive force  $\vec{f}_s$  is calculated as the difference vector between current agent position and every

neighbors while the steering force is calculated as the average vectors between all the repulsive forces. The *cohesion* moves the agents toward the center of his local neighborhood. This behavior is useful in order to give to the flock an aggregated aspect. The cohesion force  $\vec{f}_c$  is obtained computing the average position of neighbors. The *alignment* tends to align the agent with other neighbors group computing. The alignment force  $\vec{f}_a$  is calculated as difference between the average of the forward vectors of the neighbors and the forward vector of the agent itself.

The overall steering force  $\vec{f}_R$  of the Reynolds model, for the agent  $i$ , is achieved by means of the sum of the steering forces produced by behaviors

$$\vec{f}_R = w_s \vec{f}_s + w_c \vec{f}_c + w_a \vec{f}_a$$

where,  $w_s$ ,  $w_c$ , and  $w_a$  are weights that manages the behavior impact on the whole steering force.

According to the steering model proposed by Reynolds, the 3D movement is defined for each agent  $i$  using the following equations:

$$\begin{aligned} \vec{a}_i &= \vec{a}_i + \text{smoothRate}(\vec{f}_R - \vec{a}_i) \\ \vec{v}_i &= \vec{v}_i + \vec{a}_i \\ \vec{d}_i &= (\vec{d}_i + \vec{a}_i) / \|\vec{d}_i + \vec{a}_i\| \\ \vec{p}_i &= \vec{p}_i + \vec{v}_i \end{aligned}$$

Where  $\vec{a}_i$  is the agent's acceleration vector,  $\vec{v}_i$  is the velocity vector,  $\vec{p}_i$  is the position vector, and  $\vec{d}_i$  is the direction unit vector. The parameter *smoothRate* indicates how much the steering force influence the agent's acceleration. Both the lengths of  $\vec{v}_i$  and  $\vec{f}_R$  are truncated to a maximum speed.

The model illustrated requires to identify neighbors out of whole environment and in particular determines all agents that fall inside the field of view of a agent. This operation is fundamental because each agent must take decisions only according to its neighbors, and so it must be able to pick out efficiently these agents. A brute force approach requires  $O(n^2)$  steps for a proximity screening, i.e., compares each agent to all others and gathers all agents within a given range. This approach is sufficient for a hundred agents but it is clear that is computationally inefficient for interactive results of thousands agents.

#### IV. THE CLUSTERING MODEL APPROACH

In addition to the model proposed by Reynolds, we defined two new behaviors, called *Cluster-Cohesion* and *Cluster-Alignment*. These two behaviors implement the agent-based clustering algorithm.

The cluster-cohesion force  $\vec{f}_{cc}$ , for a specific agent  $i$ , is computed as

$$\vec{f}_{cc} = \sum_{j \in Neighs(i)} sim_{ij} \vec{s}_{ij} + (1 - sim_{ij}) \vec{f}_{ij}$$

where,  $Neighs(i)$  are the nearest neighbors of the agent  $i$ . The vector  $\vec{s}_{ij}$  is the seeking force between agents  $i$  and  $j$  obtained as  $(\vec{p}_j - \vec{p}_i) - \vec{v}_i$ , while  $\vec{f}_{ij}$  is the fleeing force obtained as  $(\vec{p}_i - \vec{p}_j) - \vec{v}_i$ . The function  $sim_{ij}$  computes a similarity factor between the features vectors associated to agents  $i$  and  $j$  and

must be between 0 and 1. The cluster-alignment force  $\vec{f}_{ca}$ , for a specific agent  $i$ , is computed as

$$\vec{f}_{ca} = \sum_{j \in Neighs(i)} sim_{ij} \vec{d}_j$$

Then, the steering force  $\vec{f}$  used in the flocking clustering algorithm is achieved by means of the sum of the Reynolds's steering forces and these new forces

$$\vec{f} = \vec{f}_R + w_{cc} \vec{f}_{cc} + w_{ca} \vec{f}_{ca}$$

Also in this case, we use two weights  $w_{cc}$ , and  $w_{ca}$  to manage impact of these behaviors of the clustering algorithm.

#### A. Similarity

In our model, each agent represents an object of the data set and the features associated to each object define the character of the agent. The agents move in the 3D environment that use as the search space to find the best similar agents. The overall effect is such that when agent find a similar agent, it begins to stay nearby this agent, but, on the other hand, it continue to explore the 3D environment in order to join similar groups of agents. The idea to use the 3D environment as search space is twofold. Firstly, the 3D environment enables clustering of high-dimensionality data set without loss of features. Secondly, the clustering process is visualized in an intuitive and natural fashion independently of the data set dimensionality.

The similarity between two agents is computed by using values of the associated features vectors. The implementation described in this paper uses the angular separation between features vectors

$$sim_{ij} = \frac{\vec{c}_i \cdot \vec{c}_j}{\sqrt{\|\vec{c}_i\| \|\vec{c}_j\|}}$$

where,  $\vec{c}$  is the agent's features vector. The resulting factor is between -1 and 1 and must be ranged between 0 and 1. To achieve this value, the similarity is recalculated as

$$sim_{ij} = (sim_{ij} + 1)/2$$

This similarity factor yields to poor results when features vectors are not normalized taking into account the mean value and variance of all the features vectors. This kind of normalization is unfeasible when data represent continuous streams. For this reason we adopted a dynamic adjustment of the similarity value based on neighbors agents similarity statistics.

Each agent at each simulation step collect information about minimum value ( $s_{min}$ ), maximum value ( $s_{max}$ ) and average value ( $s_{avg}$ ) computed among similarities of neighbors agents. The adaptive similarity *asim* is then computed as follow:

$$asim_{ij} = \begin{cases} \text{lerp}(sim_{ij}, s_{min}, 0.0, s_{avg}, 0.5) & \text{if } sim_{ij} \leq s_{avg} \\ \text{lerp}(sim_{ij}, s_{avg}, 0.5, s_{max}, 1.0) & \text{else} \end{cases}$$

where,  $\text{lerp}(val, x_a, y_a, x_b, y_b)$  gets the linear interpolation of  $val$  on the line whose vertexes are  $(x_a, y_a)$  and  $(x_b, y_b)$ . Figure 2b shows the relationship between *sim* and *asim*.

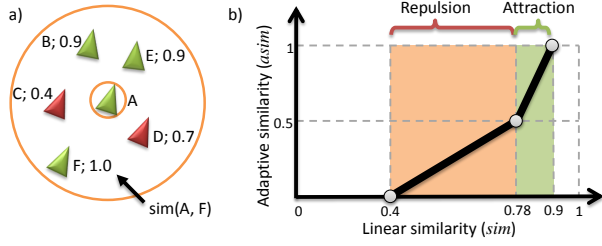


Fig. 2. a) A group of 6 agents (little triangles). First step: the agent A computes the linear similarity ( $sim$ ) for each neighbor. It also computes the values of minimum (0.4), maximum (0.9) and average (0.78)  $sim$  among them. Second step: the agent A computes the value of  $asim$  for each neighbor taking into account the minimum, maximum and average  $sim$  computed at the previous step. b) Adaptive similarity vs. linear similarity. Agent A is repulsed by agents with  $asim < 0.5$  (agents C and D) and attracted by those with  $asim > 0.5$  (agents B, E and F).

### B. Cluster Identification

We implemented a simple local label propagation algorithm for cluster identification. The algorithm is composed by two steps:

- I. Assign a unique label to each agent.
- II. Each agent looks to each of its neighbors in turn. If its neighbor's label is smaller than its own label, then it replaces its label with that of its neighbor. Repeat this step  $LPIterations$  times.

The value of  $LPIterations$  can be set at run-time, by using one of the slides of the user interface. Figure 3 shows an example of local label propagation.

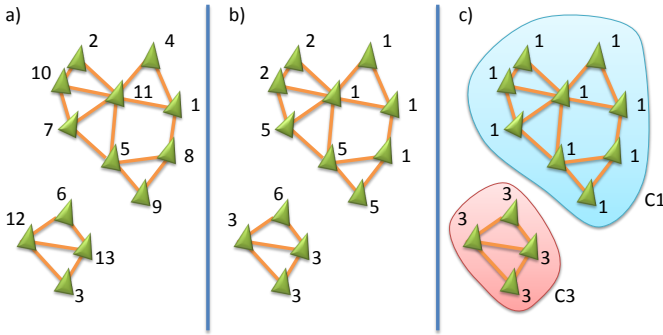


Fig. 3. Example of local label propagation for cluster identification (with  $LPIterations$  equal to 2). a) Step I: assign a unique label to each agent. b) Step II, iteration 1: propagate minimum values through neighborhood connections. c) Step II, iteration 2: again, propagate minimum values. Agents with the same label represent clusters.

## V. IMPLEMENTATION

Recently, GPUs have evolved to address programming of general-purpose computations through programming models such as CUDA [18]. CUDA is a minimal extension to C language which permits the writing of a serial program called *kernel*. A kernel executes in parallel across a set of parallel threads. Each thread has a private local memory. The programmer organizes these threads into a hierarchy of thread blocks and grids. A thread block is a set of concurrent threads that can

cooperate among themselves through barrier synchronization and have access to the shared memory with latency comparable to registers. The grid is a set of thread blocks that may each be executed independently. All threads have access to the same global, constant or texture memory. These three memory spaces are optimized for different memory usages and thus have different time access. For example, the read-only constant cache and texture cache are shared by all scalar processor cores and this speeds up reads from the texture memory space and constant memory space.

The grid and block sizes must be defined for every kernel invocation. Each block is mapped to one multiprocessor and then multiple thread blocks can be mapped on the same multiprocessor and are executed concurrently. Multiprocessor resources (registers and shared memory) are split among the mapped thread block. As a consequence, this limits the number of thread blocks that can be mapped onto the same multiprocessor. In order to maximize the number of threads supported by a multiprocessor it is important to take into account the resources required by each kernel.

In order to avoid the  $O(n^2)$  complexity of the neighbors search, we adopt a strategy based on the assumption that interaction of steering behavior drops off with distance [19]. Then, we are interested only to compute efficiently a limited amount of neighbors agents. This biologically-based assumption alleviates the computational effort required by the neighbors search as well as the difficult to manage dynamic data structures which are not trivial to implement on the GPU.

Then, to accomplish this task, a static grid subdivides the 3D environment into cubic cells of the same size. For each agent we assign a hash value based on its cell and a radix sort based on these hash values is performed on the GPU. At the end of this step, groups of agents belonging to the same cell will be located in continuous regions of the GPU memory. The output of this process enables to obtain a neighbors lists. In fact, each agent looks for neighbors in the cell where it currently is and in adjacent cells. A simple linear search starting from a proper index based on the cell hash function is then sufficient. Further information, including software, performance and scalability evaluations, of this approach is discussed in more details in our previous work [20] [21].

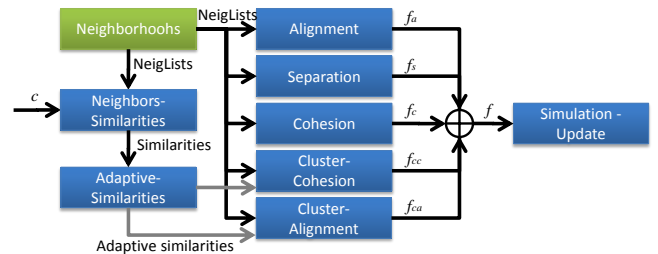


Fig. 4. Implementation schema. Rectangles are kernels and arrows are data streams. Neighborhoods represents a set of kernels.

Figure 4 shows main kernels and data streams implementing the proposed clustering method. Neighborhoods is composed of several kernels and generates neighbors lists as

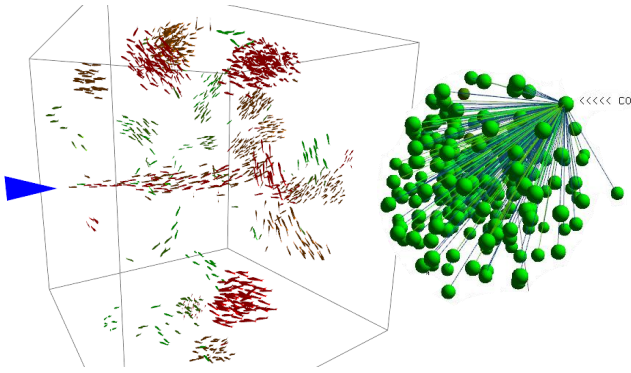


Fig. 5. (left) Agents introduced on the fly in any place in 3D environment. Agents as rendered with a 3D model of fish. (right) 3D representation of a cluster. Agents are rendered with billboards. Each agent is connected to the agent with the lowest label value among agents belonging to the same cluster.

described above. Given the neighbors' list of each agent, the kernel `NeighborsSimilarities` generates a similarity value for each neighbor of each agent. Therefore, this kernel launches one thread for each neighbor of each agent. Let be  $maxNeighbors$  the maximum number of neighbors for each agent. One agent is associated to  $maxNeighbors$  thread, each of them, sharing its features vector and neighbors list of the agent. These data are read in parallel and put in the shared memory. Other kernels, `Separation`, `Cohesion` and `Alignment`, computes the three steering forces composing the Reynolds' flocking behavior while kernels `ClusterCohesion` and `ClusterAlignment` turn similarities and neighbors lists into steering forces, respectively,  $f_{cc}$  and  $f_{ca}$ . These kernels launch a thread for each agent. Steering forces yielded by each behavior are blended in a common accumulator ( $f$ ), taking into account force weights associated to each behavior. The `SimulationUpdate` kernel applies the resulting steering force to the 3D motion of agents, as described by equations in Sect. IV. Each thread computes and stores similarities and similarities statistics (useful for the adaptive similarity method, described in Sect. IV-A) of only one neighbor agent.

## VI. EXPERIMENTAL RESULTS

The proposed model and the efficient implementation on the GPU enables to introduce agents on the fly in any place in 3D environment by using a sort of agents fountain that avoids to restart the algorithm when new data are available (see right Fig. 5 (left)). The objective of this feature is to maintain a consistently good clustering of sequence observed so far, using a small amount of time. The data stream of agents once entered in the environment immediately seek for clusters more similar in a natural fashion. This feature is implemented by pre-allocating buffering space in the GPU memory and by using these buffers whenever new data is available.

During the simulation several acts can occur related to the groups formation. Agents belonging to the same cluster move together and form flocks. These flocks explore the 3D environment, looking for similar groups to join up with. In a case of an impact between flocks representing well defined

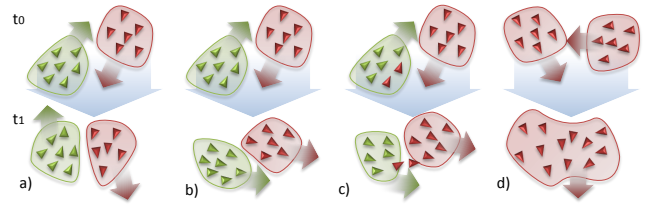


Fig. 6. Some common situation occurred during our experimentation. Impact between flocks representing well defined cluster (a) and similar clusters (b). Individual exchange between flocks (c). Flocks mixing (d).

clusters, they bounce and follow different paths (Fig. 6a). Flocks representing similar clusters move close, not mixed. Some agents act as cluster bridges, moving in the middle between the two flocks (Fig. 6b). These agents change flock membership whether the cluster of other flock match the features vector better than the cluster of the current flock (Fig. 6c). Flock representing the same cluster (according to the metric used as similarity function) merge in a big flock (Fig. 6d). In our experiments, we observe that 2000 iterations suffice to reach a stable state even for many thousands agents. Therefore, we use 2000 iterations throughout the rest of our experiments.

Several parameters decide the formation of clusters. Aside from the weights of the model illustrated in Sect. IV, we have  $worldRadius$  which is the size of the world,  $searchRadius$  is the size of the range query,  $separationRadius$  is the distance of separation between agents, and  $maxNeighbors$  is the maximum number of neighborings for agent.

The visual interface (Fig 7) supports the user in the classification and verification process of the output clusters using the Visual Information Seeking Mantra "Overview first, zoom and filter, then details-on-demand" [22] which are described below:

- *Overview first.* The application supports the visualization of the flocking approach during the creation of clusters. The overview provides the user with a visual summary of the clustering result and allows a first evaluation of the number of clusters and relations between clusters. As described in Sect. IV-B, each agent is connected to the agent with the lowest unique index in the cluster. The name of the cluster is the label the lowest unique index in the cluster. During the clustering, the user can modify the simulation parameters at running time using several sliders and move the point of view in the 3D environment
- *Zoom and filter.* Because we can also handle vast volumes of data, the visual interface provides the user with the capability to zoom in from the initial overview and filter information in order to refine the current view. If the user identifies clusters of interest in the overview, they can be selected individually to isolate them or remove from the clustering process.
- *Details-on-demand.* The user can select one or more agents and show their properties (position, class membership, etc.). Each input data is labeled with actual

class membership, the application shows the confusion matrix. Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class.

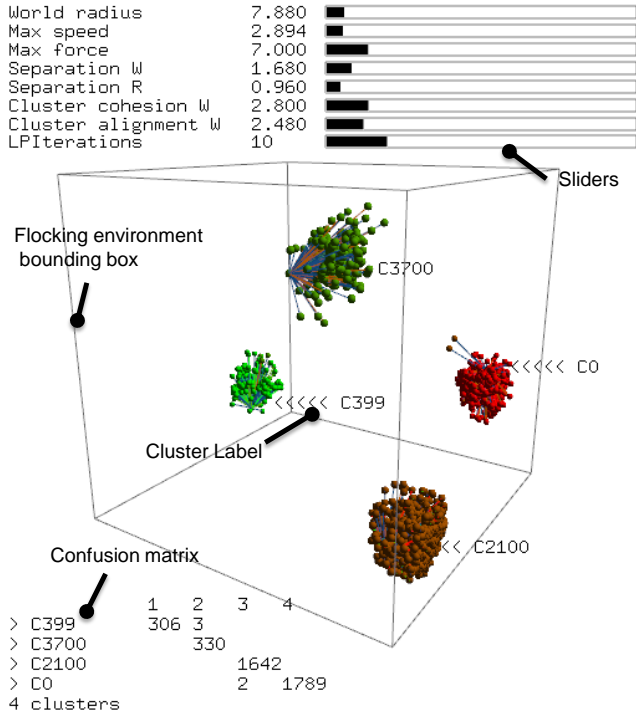


Fig. 7. The software Graphical User Interface (GUI).

### A. Quality

For the quality tests, we selected six of the most popular dataset from UC Irvine Machine Learning Repository [23]. The selected dataset are Iris, Wine, Yeast, Breast Cancer Wisconsin, Abalone and SPECT Heart. Furthermore, we used synthetic datasets generated by using a Gaussian cluster generator proposed in [24]. For each test we split the given data set in a 50/50 ratio into training and testing data.

For the quality tests, we used  $w_a = 0$ ,  $w_c = 0$ ,  $searchRadius = 4$ ,  $separationRadius = 1.5$ ,  $maxNeighbors = 32$ . We used training data to empirically find the values of  $w_s$ ,  $w_{cc}$ ,  $w_{ca}$  (showed in Table I). The value of  $worldRadius$  is computed such that the agent density in the 3D environment is always 0.05 world units per agent (in order to ensure a good level of interaction among agents).

The Iris dataset contains information about Iris flowers. There are three classes of Iris flowers, namely Iris Setosa, Iris Versicolor and Iris Virginica. The dataset consists of 150 examples with 4 attributes. One class is well separable the other two. The others have a large overlap.

The Wine dataset is the result of a chemical analysis of wines grown in a region in Italy but derived from three different cultivars. There are three classes. The dataset consists of 178 examples each with 13 continuous attributes. The dataset

contains distribution 59 examples of class 1, 71 examples for class 2 and 48 examples for class 3.

Yeast data set contains 1484 records. The cellular localization sites of proteins are to be determined. There are ten classes.

The Breast Cancer Wisconsin (B. C. W.) dataset has 699 records covering benign and malignant breast tumor cases. Goal is to explain the difference between the two diagnoses.

The Abalone data has a total of 4177 records. Each record represent an abalone instance. Goal is to decide the number of rings of the instance using various measurements. The number of rings ranges from 1 to 29. A 3-class classification problem is defined for the Abalone dataset analogously to the Iris dataset, except that here class 1 has the records with 1-8 rings, class 2 has those with 9 or 10 rings, and class 3 contains those with 11-29 rings.

The SPECT Heart dataset has 267 records. In contrast to above datasets, all attributes are binary. The goal is prediction of a 0, 1 DIAGNOSIS variable.

We used three synthetic datasets where each one contains 4000 records. The first has 10 classes (Synth. 10C), the second has 20 classes (Synth. 20C) and the third 40 classes (Synth. 40C) as illustrated in Table II.

For each dataset, we evaluated the correctness of the classification result by using *precision* ( $P$ ) and *recall* ( $R$ ). A further measure we used is  $F$ -measure ( $F$ ) which is the harmonic mean of precision and recall. These measures are defined as:

$$P = \frac{tp}{tp + fp} \quad R = \frac{tp}{tp + fn} \quad F = 2 \cdot \frac{PR}{P + R}$$

where  $tp$  is the number of true positive patterns,  $fp$  the number of false positive patterns and  $fn$  the number of false negative patterns.

Table II shows the averages and standard deviations of the results of the proposed clustering algorithm of 500 iterations after the simulation reached a stable state (2000 iterations). We compared our results with K-means clustering [3] and hierarchical clustering (single-linkage) [2].  $P_{KM}$ ,  $R_{KM}$  and  $F_{KM}$  are, respectively, precision, recall and F-measure of the K-means clustering results.  $P_{SL}$ ,  $R_{SL}$  and  $F_{SL}$  are, respectively, precision, recall and F-measure of the hierarchical clustering results. We executed the K-means clustering algorithm 500 times with each dataset. Compared to K-means, we achieved better results with Iris, Wine and SPECT Heart. We achieved similar results to K-means with Yeast and Abalone and slightly worse results with Breast Cancer Wisconsin. Tests with synthetic data show that datasets with high number of class are properly classified. These experimental results show that the proposed clustering approach is quite good but it needs some minor adjustments for maturity.

### B. Performances

All the tests have been performed on an AMD Athlon 2800+ CPU, 2GB RAM and a NVIDIA GTX 470 1280Mb

TABLE I  
VALUES OF PARAMETERS

	Iris	Wine	Yeast	B. C. W.	Abalone	SPECT H.	Synth. 10C	Synth. 20C	Synth. 40C
$w_s$	2.0	3.0	2.0	1.0	2.0	0.5	0.5	0.5	0.5
$w_{cc}$	3.0	4.0	4.8	2.0	4.0	1.0	3.0	3.0	3.0
$w_{ca}$	2.0	4.0	3.0	4.0	3.8	6.0	2.5	2.5	2.5

TABLE II  
RESULTS OF QUALITY TESTS

	Iris	Wine	Yeast	B. C. W.	Abalone	SPECT H.	Synth. 10C	Synth. 20C	Synth. 40C
$P$	$0.98 \pm 0.00$	$0.77 \pm 0.00$	$0.33 \pm 0.01$	$0.86 \pm 0.00$	$0.54 \pm 0.00$	$0.67 \pm 0.00$	$0.91 \pm 0.01$	$0.92 \pm 0.01$	$0.79 \pm 0.03$
$P_{KM}$	$0.83 \pm 0.15$	$0.75 \pm 0.01$	$0.33 \pm 0.04$	$0.96 \pm 0.00$	$0.51 \pm 0.00$	$0.56 \pm 0.02$	$0.85 \pm 0.06$	$0.79 \pm 0.06$	$0.77 \pm 0.05$
$P_{SL}$	0.84	0.47	0.27	0.83	0.28	0.46	0.74	0.74	0.73
$R$	$0.97 \pm 0.00$	$0.72 \pm 0.00$	$0.33 \pm 0.01$	$0.87 \pm 0.00$	$0.50 \pm 0.00$	$0.70 \pm 0.00$	$0.90 \pm 0.00$	$0.91 \pm 0.01$	$0.77 \pm 0.05$
$R_{KM}$	$0.85 \pm 0.12$	$0.72 \pm 0.00$	$0.34 \pm 0.04$	$0.95 \pm 0.00$	$0.52 \pm 0.00$	$0.69 \pm 0.03$	$0.87 \pm 0.07$	$0.78 \pm 0.06$	$0.75 \pm 0.05$
$R_{SL}$	0.68	0.36	0.22	0.50	0.41	0.48	0.71	0.70	0.72
$F$	$0.97 \pm 0.00$	$0.71 \pm 0.00$	$0.29 \pm 0.01$	$0.86 \pm 0.00$	$0.51 \pm 0.00$	$0.68 \pm 0.00$	$0.90 \pm 0.00$	$0.92 \pm 0.01$	$0.76 \pm 0.03$
$F_{KM}$	$0.84 \pm 0.13$	$0.70 \pm 0.01$	$0.30 \pm 0.03$	$0.95 \pm 0.00$	$0.51 \pm 0.01$	$0.43 \pm 0.02$	$0.85 \pm 0.06$	$0.77 \pm 0.07$	$0.75 \pm 0.05$
$F_{SL}$	0.58	0.25	0.20	0.40	0.33	0.48	0.68	0.66	0.73

RAM (CUDA compute capability 2.0). Software configuration: CUDA v3.1, Windows 7. The rendering of the clusters is performed by using OpenGL [25].

For performance tests, we used  $w_s = 0.8$ ,  $w_a = 0$ ,  $w_c = 0$ ,  $w_{cc} = 3.0$ ,  $w_{ca} = 2.5$ ,  $searchRadius = 4$ ,  $separationRadius = 1.5$ ,  $maxNeighbors = 32$ . The value of  $worldRadius$  is computed such that the agent density in the 3D environment is always 0.05 world units per agent (in order to ensure a good level of interaction among agents).

We used synthetic datasets generated by using a Gaussian cluster generator proposed in [24]. We tested the performance of the proposed approach on datasets with different number of instances, features and classes.

We developed a serial version of the proposed approach (CPU implementation) by using the OpenSteer steering library [26]. The neighbors search is based on an efficient implementation of a grid subdivision of the environment. We compared performances of GPU and CPU implementations by comparing how long each iteration of the algorithm takes (in milliseconds). We also compared performances of our GPU implementation with those of the Matlab’s k-means serial implementation, in order to give an idea of the performances of a classical clustering approach. We executed the K-means clustering algorithm 500 times with each configuration and we took the average elapsed time of a single execution.

Figure 8 compares the performances with different number of instances of our GPU implementation, our CPU implementation and k-means. With 1000 instances the CPU implementation is more efficient than the GPU implementation (due to the data-reordering overhead, as described in [20]) but the second one scales better than the first one. We achieved a speed-up of 30 with a dataset of 65000 instances. Figure 8 also shows that the CPU implementation can run up to 2000 instances at interactive frame rates while the parallel implementation can run up to 32000 instances at interactive frame rates.

Figure 9 (left) shows the performances of the proposed GPU implementation with different number of features, compared to results of k-means. The performance of the proposed approach scales slightly worse than k-means. This problem is due

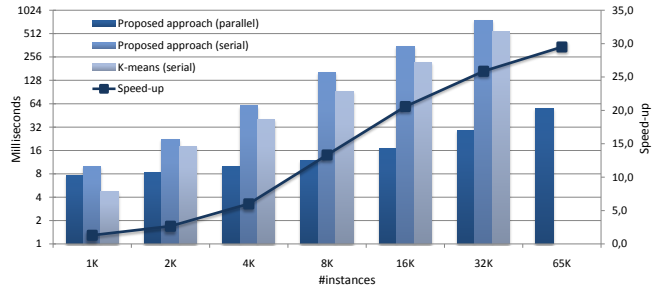


Fig. 8. The speed-up is between our CPU implementation (proposed approach - serial) and our GPU implementation (proposed approach - parallel). The GPU implementation scales better than CPU implementation. The GPU implementation is affected by an overhead which is dominant on the overall performance with a low number of instances (up to 1000).

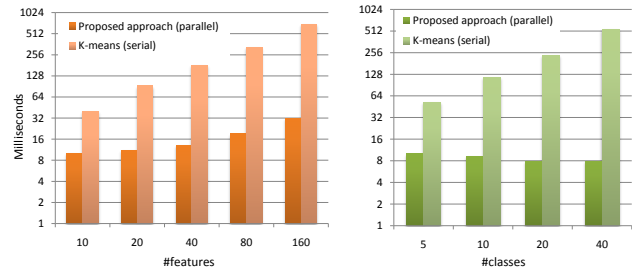


Fig. 9. (Left) The GPU implementation (proposed approach - parallel) scales worse than a classical clustering algorithm. This is due to the parallelization scheme we chose. Scalability on #instances is advantaged compared to scalability on #features. (Right) The performance of our GPU implementation does not decrease with a high number of classes.

to the `NeighborsSimilarities` kernel that computes similarities among agents. This kernel launches a thread for each neighbor of each agent, that ensures good performance with a high number of instances and a small number of features (up to 40). In future works a new version of this kernel will address this issue. A good solution to this problem is to launch a thread for each agent for each feature.

Figure 9 (right) shows an interesting point. The computation time of the classical k-means increases proportionally to

the number of classes. The computation time of our GPU implementation does not increase decrease (indeed, it slightly decreases). A high number of classes leads to a high fragmentation of agents in the 3D environment (a flock for each class). A high agents fragmentation leads to a smaller average size of agents' neighbors lists. Thus, with a high number of classes, the neighbors searching phase is slightly more efficient.

## VII. CONCLUSIONS AND FUTURE WORKS

[ELENCCARE RISULTATI] In this work, we proposed a nature inspired clustering model and presented an efficient implementation for the GPU. In this model, each data is represented by an agent that follows simple local rules to move in the 3D environment. Agents following these simple rules emerge a complex global behavior and agents similar to each other gradually merge together to form a cluster. Also we propose an efficient implementation for the GPU based on a static grid that tackles the problem of identifying neighbors. This solution is a key factor in obtaining an interactive visualization-based result on the GPU that enables to cluster incoming data without take into account all the processed data. Then, the approach is able to diagnose changes in evolving input data and to distinguish data introduced in 3D environment that must join old clusters or represent a new cluster. An advantage of this approach is that it does not require to know a priori the number of clusters as well as not require to know a priori the number of data to cluster. As the input data stream evolves during the computation the number of natural cluster change. This feature enables to interactively introduce incoming data stream in an user defined 3D place in a way similar to the 'in vitro' procedure used in biology. As far as we know, this modus operandi is novel.

In the future, we would like to explore several improvements to this approach: (i) Perform further experiments to validate better the effectiveness of the clusters. (ii) find automatically the best parameters setting. (iii) evaluate other similarity metrics that perform better as the number of features increase. (iv) add a new feature called assisted clustering that enable the user to select at runtime two or more clusters and try to merge them. (v) improve the clusters identification.

## REFERENCES

- [1] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.
- [2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.
- [3] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, L. M. L. Cam and J. Neyman, Eds., vol. 1. University of California Press, 1967, pp. 281–297.
- [4] T. Kohonen, M. R. Schroeder, and T. S. Huang, Eds., *Self-Organizing Maps*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.
- [5] L. Yang, "Interactive exploration of very large relational datasets through 3d dynamic projections," in *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2000, pp. 236–243.
- [6] R. D. Chiara, U. Erra, V. Scarano, and M. Tatafiore, "Massive simulation using gpu of a distributed behavioral model of a flock with obstacle avoidance," in *VMV*, 2004, pp. 233–240.

- [7] K. R. Roshan DSouza, Mikola Lysenko, "Sugarscape on steroids: simulating over a million agents at interactive rates," in *Proceedings of Agent*, 2007, pp. 361–364.
- [8] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: ordering points to identify the clustering structure," *SIGMOD Rec.*, vol. 28, no. 2, pp. 49–60, 1999.
- [9] A. Hinneburg, D. A. Keim, and M. Wawryniuk, "Hd-eye - visual clustering of high dimensional data: A demonstration," *Data Engineering, International Conference on*, vol. 0, p. 753, 2003.
- [10] J. D. Hall and J. C. Hart, "GPU acceleration of iterative clustering," in *ACM Workshop on General Purpose Computing on Graphics Processors*, August 2004.
- [11] S. A. Shalom, M. Dash, and M. Tue, "Efficient k-means clustering using accelerated graphics processors," in *DaWaK '08: Proceedings of the 10th international conference on Data Warehousing and Knowledge Discovery*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 166–175.
- [12] M. Zechner and M. Granitzer, "Accelerating k-means on the graphics processor via CUDA," *Intensive Applications and Services, International Conference on*, vol. 0, pp. 7–15, 2009.
- [13] R. Farivar, D. Rebolledo, E. Chan, and R. H. Campbell, "A parallel implementation of k-means clustering on GPUs," in *PDPTA*, 2008, pp. 340–345.
- [14] Q. Zhang and Y. Zhang, "Hierarchical clustering of gene expression profiles with graphics hardware acceleration," *Pattern Recogn. Lett.*, vol. 27, no. 6, pp. 676–681, 2006.
- [15] D.-J. Chang, M. M. Kantardzic, and M. Ouyang, "Hierarchical clustering with CUDA/GPU," in *ISCA PDCCS*, J. H. Graham and A. Skjellum, Eds. ISCA, 2009, pp. 7–12. [Online]. Available: <http://dblp.uni-trier.de/db/conf/ISCApdcs/pdccc2009.html#ChangKO09>
- [16] J. S. Charles, T. E. Potok, R. M. Patton, and X. Cui, "Flocking-based document clustering on the graphics processing unit," in *NICSO*, 2007, pp. 27–37.
- [17] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1987, pp. 25–34.
- [18] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.
- [19] I. D. Couzin and J. Krause, "Self-organization and collective behavior in vertebrates," *Advances in the Study of Behavior*, vol. Volume 32, pp. 1–75, 2003.
- [20] U. Erra, B. Frola, V. Scarano, and I. Couzin, "An efficient gpu implementation for large scale individual-based simulation of collective behavior," *High Performance Computational Systems Biology, International Workshop on*, vol. 0, pp. 51–58, 2009.
- [21] U. Erra, B. Frola, and V. Scarano, "Behavert: A gpu-based library for autonomous characters," in *Motion in Games*, ser. Lecture Notes in Computer Science, R. Boulic, Y. Chrysanthou, and T. Komura, Eds. Springer Berlin Heidelberg, vol. 6459, pp. 194–205.
- [22] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proceedings of the 1996 IEEE Symposium on Visual Languages*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 336–. [Online]. Available: <http://portal.acm.org/citation.cfm?id=832277.834354>
- [23] <http://archive.ics.uci.edu/ml/datasets.html>.
- [24] <http://dbkgroup.org/handl/generators/>.
- [25] OpenGL ARB, D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, August 2005.
- [26] C. Reynolds, "Opensteer - steering behaviors for autonomous characters," 2004, <http://opensteer.sourceforge.net/>.