Improving lagrangian methods. Toward an agent-particle based method

Jean Marie Dembele
Laboratoire Modélisation et Applications Thématiques
University Cheikh Anta Diop - IRD/UR GEODES
BP 1386. Dakar/Sénégal
(221) 77 645 45 75
dembele@ird.sn

Christophe Cambier
Université Paris 6, Laboratoire du Lip6.
BP 1386. Dakar/Sénégal
cambier@ird.sn

ABSTRACT
An agent-based modeling procedure is proposed in this paper in order to improve particles methods (like Smooth Particle Hydrodynamics, Vortex methods...) in the context of modeling and simulating physical or social systems described by partial differential equations. The procedure suggests, for avoiding some of the lagrangian methods limitations, to replace the classical particle with an autonomous process – an agent-particle: AP – able to implement skills like vicinity perception, capacity of evaluating mutual contribution, testing complex behaviors… A complete description of concepts and tools that the AP-based method might use for limiting the combinatory complexity is first given; sending messages for symmetric contributions, quad-Lattices for neighbors searching, ray tracing and kd-tree-domains for handling obstacles… In a second part, a continuous convection-diffusion problem and a discrete animals aggregation are simulated to show what the AP-based method can bring to classical resolution schemes.

Categories and Subject Descriptors
[**I. Computing Methodologies**]: I.6 SIMULATION AND MODELING: I.6.5 Model Development – *Modeling methodologies.*

General Terms
Algorithms, Performance, Experimentation, Theory.

Keywords
Particle methods, dynamical systems, partial differential equations (PDE), agent-based modeling.

1. INTRODUCTION
Simulations of physical systems described by partial differential equations are more and more using particle methods [10]. Indeed, mesh-based methods like finite element or finite difference are not well suited in the simulation of complex phenomena occurring on irregular geometries (impact/penetration, explosion, fluid-structure interactions, sediments transport, erosion...).

Lagrangian methods are then more promising in the study of such phenomena. However, they still have some conceptual limitations especially with complex boundary conditions or functions approximations.

Since parallel CPU computing or GPGPU[1] programming [14] – performing a way to handle very large numbers of threads – are widely applied now in agent-based modeling, we naturally suggested in [7] to build from the lagrangian particle an agent-particle (AP) with vicinity perception and capacity of evaluation of their mutual contribution (for functions approximations). In the same physical context, other previous works dealing with such a paradigm can be cited; simulating shallow water with "water-ball-agents" [16] or building "vortex-agents" in a turbulent flow with emerging coherent structures [5]. Lagrangian methods should therefore, in some cases, define their elementary particle as an autonomous process in order to avoid the aforementioned conceptual limitations and to be able to implement more skills with particles; aggregating [16], changing level of description [5], exploring more or less complex scenarios…

The purpose of this present paper is to provide a more complete description of what an agent-particle-based method should be and should use (section 2) and to illustrate the new method with well-known systems usually described by partial differential equations (section 3). After giving the conceptual basis (the SPH formalism) of our AP, a focus is made on the reduction of the combinatory complexity; particle methods requiring an important number of elements for good approximations, a great effort must be completed on reducing complexity. A quad-Lattice structure is consequently defined, providing to the AP a faster way to find its neighbors. In the same spirit and for handling complex boundary conditions or obstacles collisions an adaptation of ray tracing with the use of kd-tree is proposed.

Two cases are afterward debated. A first one implying continuous diffusion-based processes and the second one dealing with discrete systems. In both of them, the AP is autonomous in its dynamics computing; neither using random walk, nor Laplancian approximation by quadrature, but diffusion velocity that shows good results. This allows exploring more easily some dynamics like animals aggregation.

[1] General Purpose Graphics Processor Units

SIMUTools 2009, Rome, Italy.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5658
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5658

2. THE AP-BASED MODEL GUIDELINES

A particle means in this paper a virtual mobile entity with its own attributes and identity in a whole dynamical system. Its type and size are related to the field of study and scale of description.

In this way, it can represent an electron, an atom, a molecule… a granule, a drop, or more specifically a relevant volume of matter that can be taken elementary in a discretization of a physical continuous dynamical system. In discrete systems, a particle might also represent a star, a planet, a galaxy… for higher scales, or an individual for social phenomena like populations dynamics or moving flocks of fishes or birds. A particle system is then a collection of interacting particles (by attraction, repulsion, attenuation, random forces…), which are submitted to the resulting dynamic.

The agent-particle that we want it to simulate the behavior of the previously defined particles requires the use of a formal concept; we will use to that end the particle method SPH (Smoothed Particle Hdrodynamics).

2.1 Build the AP from SPH particle

Originally defined for astrophysics problems like evolution of proto-stars and galaxies [9], SPH (Smoothed Particle Hydrodynamics) is a lagrangian numerical method widely used now, specially in fluids dynamics. It had been well specified in [12] and improved many times [10].

SPH uses particles positions x as quadrature points in order to approximate any field function f (density, pressure, viscosity…).

$$< f(x) >= \int_{\Omega} f(y)W(x-y,h)dy \qquad 2.1$$

$$< f_i >\approx \sum_{j=1}^{N} f_j . \frac{m_j}{\varphi_j} W(x_i - x_j, h) \qquad 2.2$$

$$(\nabla f)_i = \frac{1}{\phi_i} \sum_{j=1}^{N} \frac{m_j}{\varphi_j} \phi_j (f_j - f_i) \nabla_i W(x_i - x_j, h) \qquad 2.3$$

Ω, W, h, f_i, m_i, φ_i are respectively the domain, the smoothing kernel, the core radius, the field function value at the particle i, the mass of the particle i, and its density.ϕ_i a scalar field used for making the gradient vanished when the scalar field is constant.

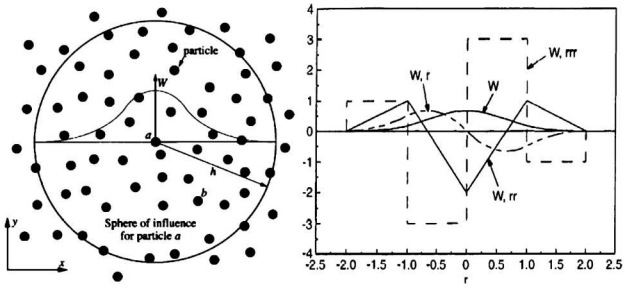


Figure 1. Influence zone of a particle and the kernel cubic B-Spline function and derivatives

For each particle, the contribution of the others is proportional to the distance between them and the smoothing kernel is chosen in a way to ignore particles over a core radius distance and to have continuous derivatives.

This allows rewriting (with 2.3 for example) governing PDE of motions in a set of ODE for numerical time integration by predictor-corrector, leap-frog or Verlet schemes.

2.2 The general AP procedure

Our AP is more than a "virtual mobile entity with its own attributes" (positions, mass, velocity…). He becomes an autonomous process able to implement other behaviors (independently to the physical dynamics) during the system evolution. Here is a general way an AP should behave:

- find its neighbors (section 2.4)
- compute their contributions and send messages (section 2.3)
- calculate its dynamic (with diffusion velocity for example in section 3.1)
- look for obstacles (section 2.5) and move accordingly
- implement other skills, (varying neighbors perception; section 3.2, aggregation, changing scale of description…)

This gives an idea of the AP procedure. Of course according to the problem, the new skills will change the resulting interactions (for example an aggregated structure will have a different influence on a single particle) and the AP procedure.

2.3 Send messages for symmetric contribution

Looking at equation 2.2, one can notice the symmetric contribution between particles assuming that their motions are synchronous[2]. An AP should then, once having calculated the contribution from another AP, send him the information and keeping him from computing again the same routine. This can improve the computation time if the sending message mechanism is well scheduled and the structure chosen for storing information received from other AP is efficient (i/o time access for lists may be different form one platform to another).

Sending messages will be very helpful when computing many contributions at a time (velocity, energy, density, pressure…) or when searching neighbors.

2.4 Quad-Lattices for neighbors searching

2.4.1 The Quad-Lattice (QL) structure

Particles methods are N-Body problems and require quadratic complexity with some operations that need to be computed at each time step. For example, in a neighborhood query, the particle has to compare its position to the position of all others to determine the particles in its core radius.

A way to reduce such a high combinatory procedure is to build a spatial indexation of elements. If particles were immobile, a simple Delaunay triangulation or an spatial adaptable splitting procedure like Quadtree or Octree [2] or Kd-tree [3] could have fit to the problem. In our case, particles are very mobile and require not a tree but a lattice indexation.

An appropriate method is the Bin-Lattice one [15]. A rapid overview of the principle is to split the domain into regular cells

[2] In a synchronous mode, all particles calculate their future states with the same distribution in particles positions.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5658
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5658

(or cubes in 3D) containing particles. A particle then belongs at one cell and a cell can contain several particles and should maintain a list of them. In the neighborhood query, all cells beyond the neighborhood distance are ignored and only those in the query range will be considered. In some particle methods [8], a particular case of Bin-Lattice is taken by given to the cell a length equal to the core-radius.

The particle will look for its neighbors in its cell and in the 8 surrounding ones, North: N, South: S…North East: N-E, South West: S-W…(Figure 2-left). In order to reduce again the number of interacting elements in the neighbors searching, we decide to divide cells into four (4) sub-domains (00, 01, 10, 11) and call the resulting structure a Quad-Lattice (Figure 2-right).

According the position of an aP in a sub-domain, It may consider a contiguous cell entirely or not (Table 1).

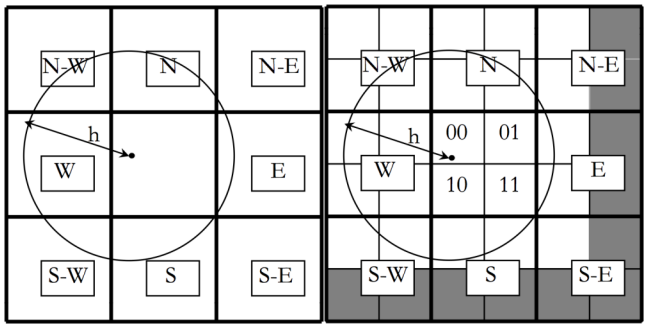


Figure 2. left: domain segmentation into cells of length h. right: reduction of the research area with QL

Table 1. The AP sub-domain position and its corresponding looking neighbors' areas in contiguous QL

	N-QL	**S-QL**	**E-QL**	**W-QL**
00	ALL	(00, 01)	(00, 10)	ALL
01	ALL	(00, 01)	ALL	(01, 11)
10	(10, 11)	ALL	(00, 10)	ALL
11	(10, 11)	ALL	ALL	(01, 11)
	NW-QL	**NE-QL**	**SW-QL**	**SE-QL**
00	ALL	(00, 10)	(00, 01)	00
01	(01, 11)	ALL	01	(00, 01)
10	(10, 11)	10	ALL	(00, 10)
11	11	(10, 11)	(01, 11)	ALL

2.4.2 The combinatory complexity comparison

Let us call by A1 the algorithm with cells and A2 the one with QL, *LpN* the list of potential neighbors, L_i the list of AP of a cell *i* and L_{ik} the list of AP of a sub-domain *k* of a cell *i.*

A1:
If none of AP of same cell have already found *LpN* then
Begin
Ask for L_j from contiguous cells;
Append to the list L_i of my own cell: $LpN \leftarrow L_i \cup (\cup L_j)$;
Send *LpN* to AP of same cell; (*symmetry*)
End
Find the list of neighbors LN_i from *LpN*;

A2:
If none of AP of same sub-domain have already found *LpN* then
Begin
Ask for L_{jk} from contiguous cells; (*use Table 1*)
Append to the list L_i of my own cell: $LpN \leftarrow L_i \cup (\cup L_{jk})$;
Send *LpN* to AP of same sub-domain; (*symmetry*)
End
Find the list of neighbors LN_i from *LpN*;

Let us now call by anpc the average number of AP/cell. The algorithmic complexity of A1 for all the *N* agent-particles is approximately 9.*anpc*.*N* and the A2 complexity is approximately 25.*anpql*.*N* (*anpql* = *anpc*/4 denotes here the average number of AP/sub-domain). The ratio of A2 complexity to A1's one is 0.694 (or 1.44 of A1 to A2). Indeed, QL cuts the area of research by 30.6% (Figure 2-left).

Table 2. Combinatory complexity according to the number of sub-domains/cell (N_{sub})

N_{sub}	**Research area**	**Combinatory complexity**
1(Cell,p=0)	1 or 100%	9.*anpc*.*N*
4(QL, p=1)	0.694	25.(*anpc*/4).*N*
16(4^2, p=2)	0.5625	81.(*anpc*/16).*N*
4^p (p≥0)	$\left(\frac{2*4^{p/2}+1}{3*4^{p/2}}\right)^2$	$(2*4^{p/2}+1)^2.(anpc/4^p).N$

The combinatory complexity actually decreases with the subdivision (Table 2). The complexity ratio between any level of subdivisions and the first classical cell corresponds to the percentage of research area of the subdivision area. Therefore, we could think about keeping the recursive process of cells subdivision to minimize the research area.

Nevertheless, even by reducing the combinatory complexity, it is not evident that the ensuing structure will be better than the classical one (the simple cell). In fact, splitting a domain requires four times the number of AP lists of the same domain. The question is: does reducing the combinatory complexity save enough time computing comparing to the time spent by lists[3] management?

2.4.3 The time computing comparison

The experiment that we drive here in Netlogo™ proposes to show the time computing differences between A1 and A2. Considering a domain]-a,a[^{2} of R^2 (a=1/2) and a random initial distribution of agents-particle. We want to execute the two algorithms on the same data and record their time computing. Particles will determine their neighbor lists using both A1 and A2, move randomly and repeat the same operation many times.

The displacements of particles will make the results independent from their spatial distribution and several time steps should minimize the noise from the operating system. We can also, in addition to these two precautions, take an increasing number of particles (10^3, 2.10^3 then 8.10^3 and 16.10^3).

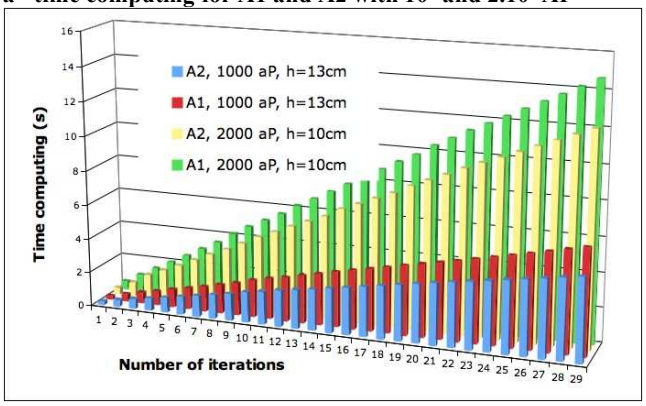
[3] Notice that the number of lists increases in an exponential way while the combinatory complexity rapidly decreases and tends to a finite limit.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5658
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5658

The results (Figure 3-a & 3-b) show a common linear factor[4] between A1 and A2 time computing that we found to be 1.251. Less than what was predicted by combinatory complexity (1.44). As expected, this is normal and is due to AP lists management. Comparing now the QL and a deeper level of subdivision, for example p = 2, will give a ratio of 1.23. This is too close to 1 (keep in mind that AP lists management reduces this ratio in time computing).

Since we are at the first steps of the AP-based model construction, we will retain the QL that seems to be at a right intersection between the time computing improvement and the use of supplement structures.

a - time computing for A1 and A2 with 10^3 and 2.10^3 AP



b - time computing for A1 and A2 with 8.10^3 and 16.10^3 AP

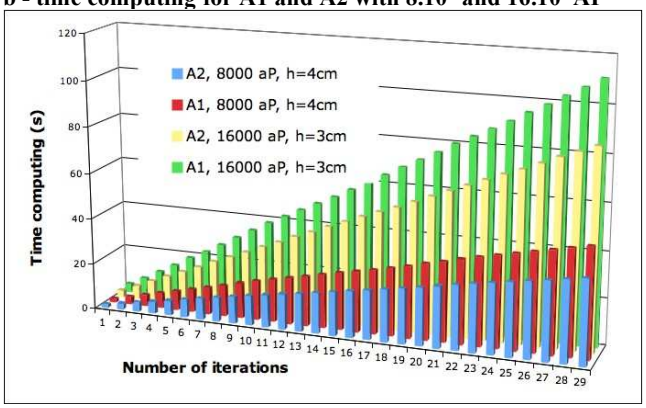


Figure 3. Time computing comparison between A1 & A2

2.4.4 Updating core-radius

SPH needs a minimum number of neighbors in order to obtain good approximation (in a 2D it requires 25 to 30 particles in a uniform distribution). So, when building the QL (or even the classic Bin-Lattice), the size must be chosen to be the maximum core-radius possible during the entire dynamic to make A1 and A2 still adequate.

2.5 Agents Kd-tree-domain (AK) for obstacles

In a natural environment often heterogeneous, AP should be able to deal with obstacles (or different objects type) that may appear, be in motion or even disappear, and disturb the normal behavior of the system. To manage the effect of an obstacle, the AP has to calculate the intersection between its trajectory and the frontal plans of the obstacles and determine its behavior according to that. It is then necessary to endow the AP of a certain perception of the environment in order to evaluate that intersection and its resulting behavior only if it is really interacting with the obstacles.

This can be done either by maintaining a global topologic map in each AP (individual perception) or by using external sensors indexing structures. The second solution, cheaper than the first one redundant, is used here. In view of the fact that the intersection problem is well studied in algorithmic geometry, robotic field, images rendering, video games, virtual reality… see [11] for an overview, we propose here to use some of their concepts and tools…

2.5.1 The ray tracing for collision detection

The ray tracing [17] allows the reconstruction of a scene by taking the opposite direction of the light (i.e. from the observer to the scene), in order to generate pixels representing the received picture by the observer.

Given a direction, the ray is thrown toward the scene containing many primitives to figure out the first intersection point with the primitives or objects. Others rays (reflected, refracted…) involved in the color of that intersection point are also taken into account in the synthesis of the corresponding pixel (Figure 4).

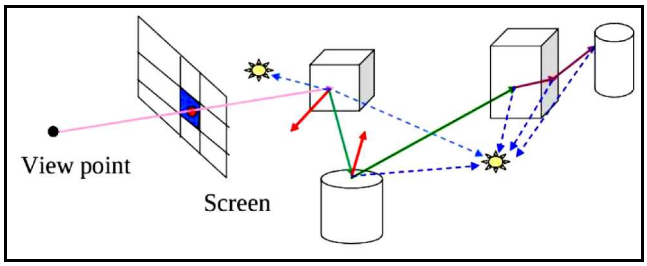


Figure 4. Ray tracing, the extern ray (pink) and intern (reflected, refracted, normal,…) rays of a scene

The image rendering can therefore be divided into 5 steps: the ray setup, the spatial subdivision for indexing the scene primitives, the ray-primitive intersection [1], the determination of light or shadow sources on the intersecting point and finally the picture generation. If we make an analogy between our AP and the observer – assuming that the thrown ray corresponds to the AP displacement vector in a time step – we will only use the three first steps of the image rendering. In fact, the AP doesn't need a complete image of the obstacle but just the first intersection point.

2.5.2 Kd-tree and heuristics for indexing obstacles

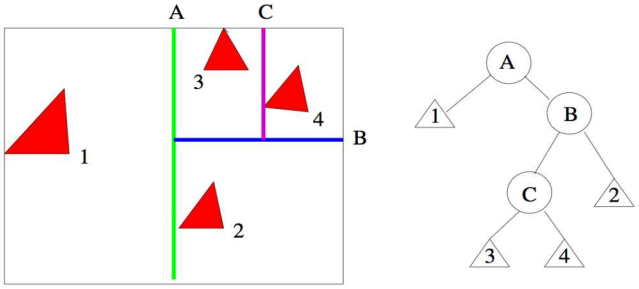


Figure 5. Adaptable subdivision and its kd-tree.

[4] This factor doesn't depend on spatial repartition or number of time steps or number of AP.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5658
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5658

For the spatial subdivision, kd-trees [3] are used to localize the objects of the scene, limiting the number of comparisons between the ray and primitives. They split the domain into two sub-domains with horizontal or vertical plan depending on the tree deepness (see Figure 5). Partitioning the domain into two equal parts may not be efficient.

To optimize the tree construction, a cost heuristic is used in ray tracing: the Surface Area Heuristic (SAH) [17]. This method minimizes the cost for a domain (Equation 2.4):

$$C(D) = K_t + P_{[D_g|D]}C(D_g) + P_{[D_d|D]}C(D_g) \qquad 2.4$$

where $P_{[S|D]} = SA(S)/SA(D)$ is the probability for the ray to cut the sub-domain S given that it had already cut the domain D. SA gives the surface area, K_t the known cost for cutting a domain.

Instead of finding all generated trees and selecting the one minimizing the cost, SAH proposes to suppose at each time step that the sub-domains are leafs. To minimize the cost of our kd-tree is equivalent to maximize the size of empty sub-domains reducing, by the way, the number of AP interacting with obstacles. We also need a way to stop the subdivision; otherwise the deepness of the tree will be related to the size of the obstacle. A threshold surface percentage of sub-domain comparing to the obstacle's one can be set.

2.5.3 Kd-tree-domains, what for?

Moving objects need to update (or even rebuild) the kd-tree. To avoid as possible the reconstruction of the tree, we propose to build, in addition, some objects kd-tree-domains indexing subdomains (bounded with plans limits), and keeping the information on its family: brother, father and grandfather… the contiguous kd-tree-domains.

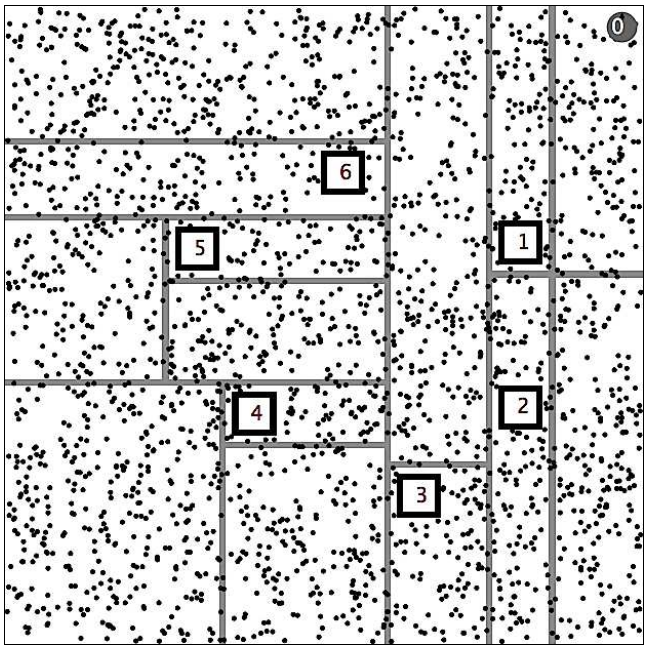


Figure 6. Kd-tree-domains built iteratively and maximizing the empty sub-domains.

When the object indexed is moving, the plans limits of the contiguous kd-tree-domain are only updated, not the kd-tree. For example, when obstacle 3 is moving (Figure 5), the tree is not updated unless one kd-tree-domain disappears. The kd-tree-domains will have two principal rules: indexing the obstacles and giving to the AP the way to behave with the primitives (after ray-primitive intersection: repulsing, sticking, bouncing, disappearing…).

3. CASE STUDY: DIFFUSION

Diffusion described by equation 3.1 is one of the basic transport processes in physical and even natural systems. It can be solved easily with mesh-based methods (finite element, finite difference). The derivatives are in that case discretized on time and space and a stability condition, involving the time step, the mesh elements size and the diffusion coefficient D, is needed to ensure the convergence of the used scheme.

Nevertheless, diffusion in natural or physical systems is often nonlinear and may also introduce other phenomena making the numerical resolution nontrivial. It is the case with convection-diffusion, reaction-diffusion…

So, if we want the AP to implement others skills in a diffusion-based process, we need to make him autonomous in the Laplacian approximation by using diffusion velocity.

3.1 The diffusion velocity of the AP

Given a scalar field u in a domain $\Omega \in \mathrm{R}^d$ its (*linear) diffusion is governed by the following PDE:

$$\begin{cases} \dfrac{\partial u}{\partial t} + \nabla.(-D\nabla u) =^{*} \dfrac{\partial u}{\partial t} - D\Delta u = 0 \\ u = f \text{ on } \partial\Omega,\ u(x, t=0) = u_0(x) \end{cases} \qquad 3.1$$

The problem with diffusion in Lagrangian methods is the discretization of the Laplacian. in fact, its approximation with quadrature points is very sensitive to the positions of particles:

"The particle adaptation comes at the expense of the regularity of the particle distribution as particles move in order to adapt to the gradients of the flow field" [4]. We prefer therefore to calculate the velocity diffusion [6] of each particle.

For that we can rewrite Equation 3.1 in a purely transport equation with advective velocity A (Equation 3.2). Each particle can now find its velocity buy using the neighborhood information (the gradient is determined from equation 2.3 that involves the smoothed kernel derivative).

$$\frac{\partial u}{\partial t} + \nabla.(Au) = 0 \text{ where } A = -D\frac{\nabla u}{u} \qquad 3.2$$

In the convection-diffusion problem for example (Equation 3.3):

$$\begin{cases} \dfrac{\partial u}{\partial t} + \nabla.(Vu) + \nabla.(-D\nabla u) = 0 \\ u = f \text{ on } \partial\Omega,\ u(x, t=0) = u_0(x) \end{cases} \qquad 3.3$$

the dynamic of the AP is now due to two velocity vectors. The AP procedure is as follow: adapt the core radius, find the neighbors, compute the contributions and send message, calculate both the diffusion and fluid velocities and move, proceed to aggregation or structures detection [7]

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5658
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5658

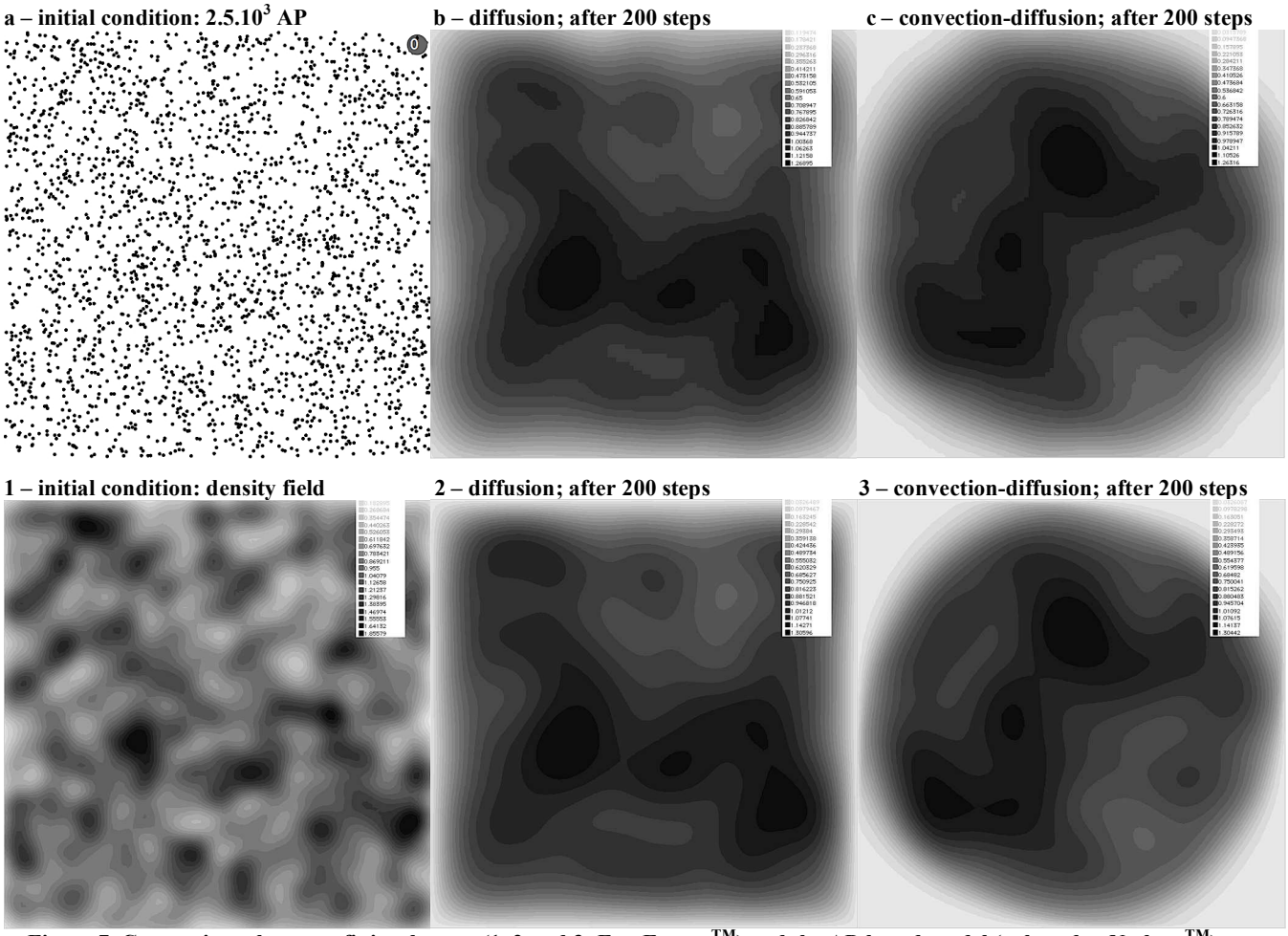


Figure 7. Comparisons between finite element (1, 2 and 3: FreeFem++^{TM}) and the AP-based model (a, b and c: Netlogo^{TM}) on a density simple diffusion (2 & b) and a low Peclet number convection diffusion: "rotating hill" (3 & c). $D = 10^{-3}$, $V = (y,-x)^T$. The results from Netlogo are imported in FreeFem++ which prints isovalues.

From the same initial condition (Figure 7-a, 7-1) we made a simulation with an agent-based platform (Netlogo) and with finite element method (using FreeFem++). Figure 7 depicted the results from both platforms and shows good results for the AP-based model.

Now that the diffusion is managed as a vector velocity, the AP can implement the obstacles perception described in section 2.4. The following simulation (Figure 8) treats this case with the obstacles indexed on figure 6.

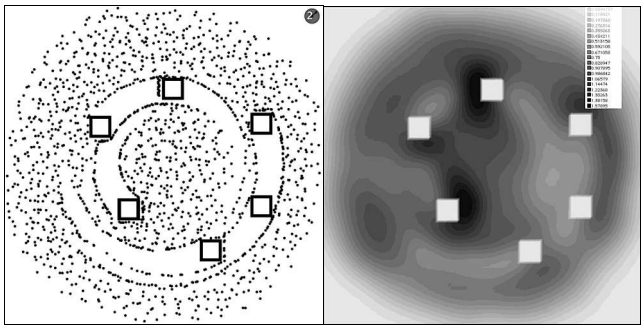


Figure 8. The resulting dynamics in presence of obstacles.

We just show here squares as obstacles but the most important is not the shape (we can just use the existing algorithms for polygons) but the way to let AP figure out if they are interacting with obstacles; when an AP wants to know if it will be in collision with an obstacle, it has to communicate its displacement vector to the top node of the tree, this one will tell him in which Kd-tree-domain to search, and so on until finding a empty Kd-tree-domain or one with obstacle.

Our Kd-tree-domains are well suited in the case where obstacles (or any foreign disturbing elements) are not moving a lot.

3.2 Application to a discrete system: exploring animals aggregation with AP

Let us now apply our model in a discrete system context. Assuming now that the AP are able to diffuse precisely, we can build on that, other behaviors in order to simulate for example animals aggregation.

Here is a description of the problem: given a population of animals (density: u) diffusing (diffusivity D) and producing at a rate α a signal (density: s). They are also sensitive at a rate λ to the total signal gradient and are advected (at a velocity V) following it. The produced signal is also diffusing (diffusivity D_S)

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5658
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5658

and evaporating at a rate β. This problem can be written in a continuous form with partial differential equations (Example equations 3.4).

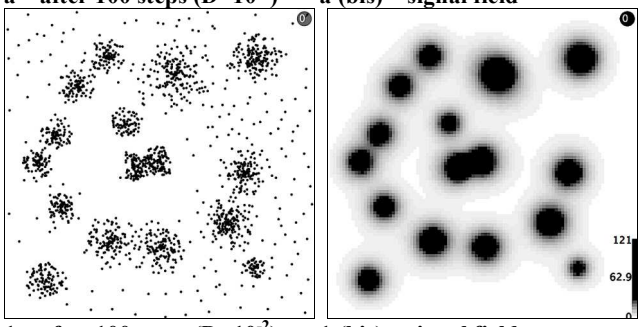
$$\begin{cases} \dfrac{\partial u}{\partial t} - D\Delta u + \nabla.(Vu) = 0 \\ V = \lambda \nabla s \\ \dfrac{\partial s}{\partial t} - D_s \Delta s + \alpha u - \beta s = 0 \end{cases} \quad 3.4$$

Nevertheless, we are not going to resolve numerically PDEs that are realistic for "large, dense aggregates with no sharp discontinuities" [13].

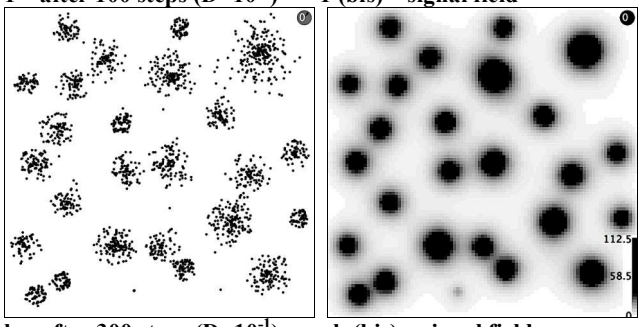
In our discrete model, the AP will have a vicinity perception that allows him to evaluate the signal concentration field and the AP density also. If the signal is created only on domain patches were there is an AP, it will create an attractive force (gathering the AP), and the diffusion is equivalent to a repulsive force (keeping a minimum distance between them).

While the AP are diffusing with velocity, the signal is diffusing using forward Euler since it is a simple diffusion process. In one time step, AP behaves as follow: adapt the core radius, produce the signal on the patch field, diffuse and follow the signal gradient (advection step).

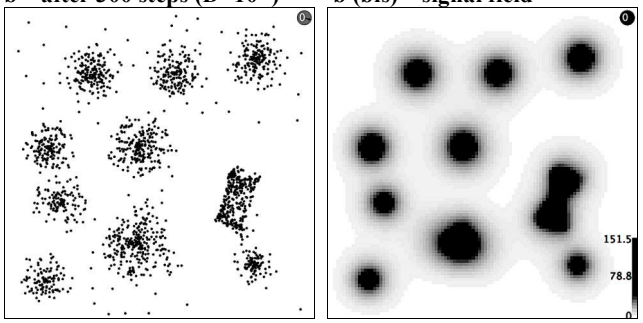
a – after 100 steps ($D=10^{-1}$) **a (bis) – signal field**



1 – after 100 steps ($D=10^{-2}$) **1 (bis) – signal field**



b – after 300 steps ($D=10^{-1}$) **b (bis) – signal field**



2 – after 300 steps ($D=10^{-2}$) **2 – (signal field)**

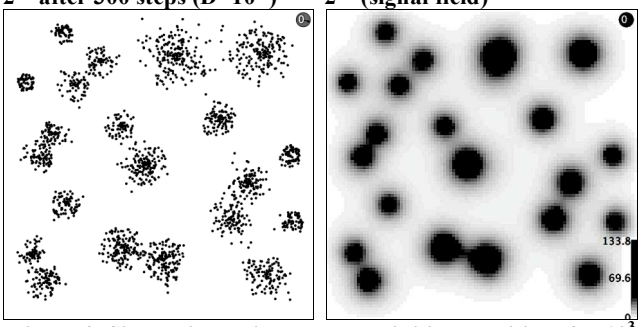


Figure 9. Simulations with a random initial condition: $2.5.10^3$ AP, α=0.3, β=0.1, λ=0.1, D_s=0.2

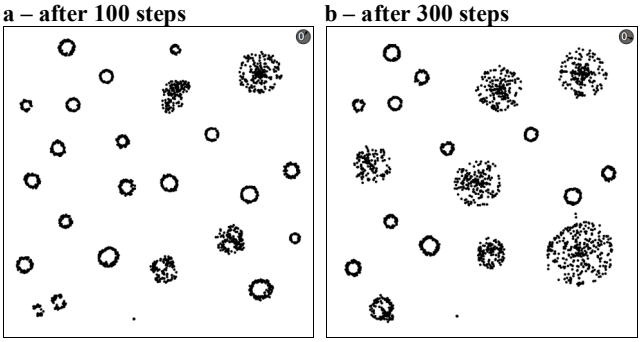
The simulation shows after few steps the emergence of more or less stable groups. For higher AP diffusivity (Figure 9-a&b), we can notice less emerging groups then lower diffusivity (Figure 9-1&2). In fact, when diffusing more rapidly they are getting far away emerging groups.

The AP based model can help in this context of discrete systems to explore some scenarios in the simulations more easily than using PDE based models. The AP trajectories can also be followed showing how groups occur (for migration flows).

Another thing to test is the behavior of the system if the perception of AP is more than a threshold in any case (even if the number of necessary neighbor is obtained). This should allow the AP to know more about a further group or another AP and create more rapidly stable groups (Figure 10).

Some strategies can also be defined, as joining a closer (or further), an older (or younger) group (obtained by clustering or aggregating particles using a distance density criteria) than the highest attractive gradient one.

a – after 100 steps **b – after 300 steps**



c – after 475 steps **d – after 700 steps**

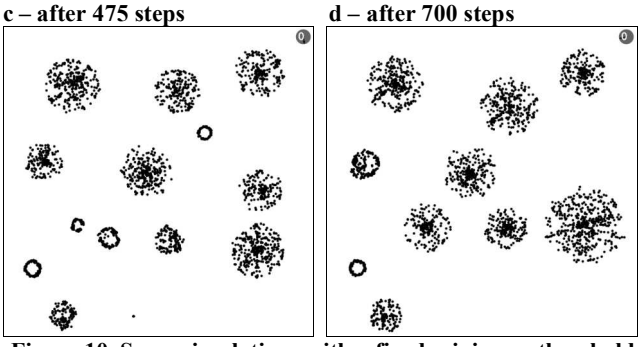


Figure 10. Same simulations with a fixed minimum threshold (h_{min} = 0.06)

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5658
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5658

4. CONCLUSION

We presented in this paper the first conceptual steps of an AP method (i.e. the use of messages for symmetric contribution, the new quad-lattice structure for neighbors searching and the kd-tree-domains for managing obstacles), which contribute to reduce the combinatory complexity that is an important issue in N-Body problem. We also furnished the ability to handle more complex behaviors with diffusion-based process, either on continuous systems or discrete ones.

Rather than resolving governing law-based equations with classical numerical schemes, the AP method tries if possible, to follow natural interactions rules in the studied system (the diffusion velocity, in our case, looks like the effect of a "natural pressure" from the closer neighbors trying to spread out uniformly in the environment).

The method seems promising in the simulation of more and more complex systems. In fact, its formulation gives the opportunity to increment the particle skills, from basic transport processes, according to the needs of simulation and/or the knowledge of the simulation domain expert. In the animals' aggregation for example, the given model helps to find the parametric values allowing gathering or dispersion of individuals. Others scenarios may after that help to easily discover others dynamics and learn about the social studied systems.

This study can therefore be expanded to turbulent fluids described by carrying vorticity particles (Navier-Stokes velocity-vorticity equations) since there are following a convection diffusion process. However, more validation tests have to be done and effort made for functions approximations that remain a limitation for lagrangian method (especially vortex method). The AP skills can also assist on coherent structures detection. In other fluids flows, like shallow water or breaking waves, the obstacles perception can be applied once the integration schemes provided to the particles their displacement vector.

Parallelizing the code, or building the corresponding architecture on GPU should help to run more significant simulation without using supercalculators.

5. REFERENCES

[1] Badouel D., 1990. An efficient ray-polygon intersection. Graphics gems, 390–393

[2] Bandi, S., Thalmann, D., 1995. An adaptative spacial subdivision of the object space for fast collision detection of animated rigid bodies. In EUROGRAPHICS'95 Conference, Computer Graphics Forum, Volume 14(3), pages 259-270, Maastricht.

[3] Bentley, J.L., Friedman, J.H., 1979. Data structures for range searching. ACM Computing Surveys, 11(4): 398-409.

[4] Bergdorf, M., Cottet, G.-H., Koumoutsakos, P., 2005 Multilevel Adaptive Particle Methods for Convection-Diffusion Equations, SIAM Multiscale Modeling and Simulation, 4, 328-357.

[5] Bertelle, C., Olivier, D., Jay, V., Tranouez, P., Cardon, A., 2000. A multi-agent system integrating vortex methods for fluid flow computation. In16th IMACS Congress Proceedings, Lausanne, Suisse.

[6] Degond, P., Mustieles, F. J.,1990. A deterministic approximation of diffusion equations using particles, SIAM J. Sci. Stat. Comput. 11(2), 293.

[7] Dembele, J.M., C. Cambier, 2008. Contribution to the modeling of complex systems described by partial differential equations: a 2D multi-agent model for convection-diffusion. InterJournal of Complex Systems, Article 2169.

[8] Gesteira, M. G., Rogers, B. D., Dalrymple, R.A., Crespo, A.J.C., 2008. Narayanaswamy, M. User Guide for SPHysics Code.

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

[10] Li, S. and Liu, W. K. 2002. Meshfree and Particle Methods and Their Applications. Applied Mechanics Review, vol. 55, pages 1-34, 2002

[11] Meseure, P., Kheddar, A., Faure, F., 2003. Détection des collisions et calcul de la réponse. Action spécifique CNRS N° 90.

[12] Monaghan, J. J. 1985. Particle methods for hydrodynamics. Comput. Phys. Rep. 3, 71–124.

[13] Parrish, J. K., Edelstein-Keshet, 1999. Complexity, Pattern, and Evolutionary Trade-Offs in Animal Aggregation. Science. Vol. 284. no. 5411, pp. 99 – 101

[14] Pharr, M., Fernando, R., 2004. GPU Gem 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation, Addison Wesley.

[15] Reynolds,W.,2000. Interaction with Groups of Autonomous Characters. Game Developers Conference 2000, proceedings, pp 449-460

[16] Servat, D., 2000. Modélisation de dynamiques de flux par agents. Application aux processus de ruissellement, infiltration et érosion. Thèse de Doctorat. Université Paris 6.

[17] Slusallek, P., Shirley, P., Mark, B., Stoll, G., Wald I., 2005. Siggraph 2005 course 41 : Introduction to real time ray tracing.

[18] Wald, I., Havran, V., 2006. On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N). In Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, pages 61–6

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5658
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5658