ADAPTIVE SAMPLING FOR VERY LARGE PARTICLE SYSTEMS USING AN INCREMENTAL SELF-ORGANIZING FEATURE MAP: AN APPLICATION IN MOLECULAR DYNAMICS

Laurent Balmelli

Laboratory for Audio-Visual Communications
Swiss Institute of Technology

ABSTRACT

This paper describes an improvement of the self-organizing feature map (SOFM) obtained with the Kohonen neural network. The ameliorations are dedicated to its usage in computer graphics and mainly in animation of particle-based systems. We show its application in the context of the visualization of molecular dynamics systems. Finally, we compare this solution with other works based on particle systems.
Animations: http://lcavsun6.eplf.ch/EGCAS96

1. INTRODUCTION

The visualization of molecular dynamics systems can be achieved with several graphic techniques. In this paper, we consider a polygon-rendering method, which consists of displaying each atom with a sphere. Property mapping is then applied on the spheres, by colouring them using the temperature of the corresponding atom. If an animated visualization is wanted, this method is suitable as long as the system size is small (500-2000 atoms). For large system ($10^6 - 10^7$) atoms), this technique becomes unusable, as the scene complexity is too high for today's machines. This paper proposes a method to animate the visualization of large molecular dynamics (MD) systems by reducing the amount of information (essentially the set of atom coordinates) and keeping the main interesting features: the system organization and the properties (like temperatures) distribution. This technique can be applied in several other applications based on particle systems. Suggestions are made in Section 4 to include it in other models. Comparisons are made with a crowd visualization model, a model for human hair structure and finally with a model for the visualization of water flow.

2. BACKGROUND

2.1. molecular dynamics

A molecular dynamics problem consists of the physical simulation of a set of atoms. Each atom is considered as a point mass and Newton's equations are integrated to calculate its motion. In the basic model, electronic interactions are not taken into account. This model is however sufficient to simulate substances like gas, fluids (in [4], a simple MD model to simulate water is given), solids...etc. In this paper, we consider the issue of the visualization of a large set of atoms, where one wants to observe macro-phenomenons like aggregations, micelles or simply global system evolution.

A solution for visualizing and animating efficiently a very large MD system would be to discretize it according to its topology and its distribution, each discrete value representing a local density of atoms - in [3], the inverse problem is related, where the authors propose to simulate large particle sets using density functions based on the evolution of smaller sets -. This operation can be performed using a SOFM like the Kohonen neural network. To obtain the property distribution of the system, property mapping can be applied on the discrete representation. This technique can also be used to enhance the discrete model, as the network convergence quality depends heavily on the shape of the system.

To produce an animation of the system, a more efficient discretization method has to be defined such as one which considers the slow convergence of the Kohonen network. For this reason, an incremental algorithm is proposed to update the discrete representation. The advantage of having the possibility to quickly recalculate the discrete model from a previous configuration is that an animation can be produced with it. Visualization could even be achieved while the system is effec-

tively computed using molecular dynamics algorithms, assuming that those are effective enough to not slow down the frame rate of the animation. This feature would allow us to study MD systems in "real-time". More simply, visualization can be achieved with sets of pre-computed data which in this case are atom positions at successive time steps.

2.2. Neural nets and the Kohonen network

This section presents the bases of neural networks, however more details can be found in [6]. The mathematical neuron model is an information-processing unit. Three basic elements can be distinguished:

- A set of synapses (inputs), each of which is characterized by a weight of its own. A weight is positive if the associated synapse is excitatory, else the synapse is inhibitory.
- An adder for summing the input signals, weighted by the respective synapses of the neuron.
- An activation function for limiting the amplitude of the neuron output. Typically, the normalized amplitude range of the output belongs to the closed interval [0:1] or [-1:1].
- A threshold that has the effect of lowering the net input of the activation function.

A neuron is defined by the following pair of equations

$$u_k = f_k(\vec{x})$$

$$y_k = \varphi(u_k - \theta_k)$$

where f_k is a fixed function, k is the neuron index (In the basic neuron model described in [6], f_k is a weighted summation), u_k is the adder output, θ_k is the threshold and y_k is the output signal of the neuron. The activation function in can be any "useful" function, depending on the usage context.

Figure 1 gives a representation of the basic neuron model. Each synapse j of neuron k is weighted by $w_{k,j}$ and x_j denotes the input value on synapse j.

The Kohonen network is composed of a set of neurons, usually organized as a 2D mesh (see Figure 2). The input is propagated to all neurons and each of them computes the output according to its characteristics. Neurons are only differentiated by their weight values and they all compute the same function.

Assuming that the network input is a tridimensional vector $\vec{x}$, the first feature of the Kohonen network is

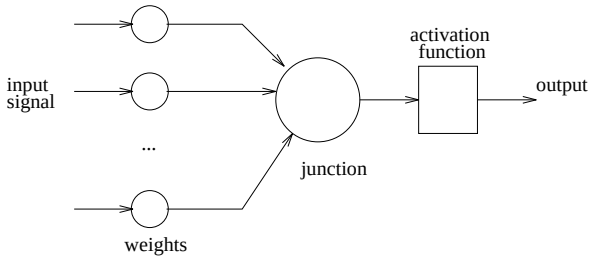

Figure 1: Basic neuron model

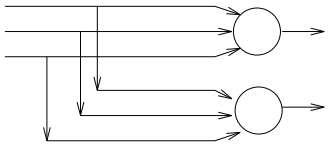

Figure 2: Example of a two-neurons network

that each neuron k simply computes the Euclidean distance between the input and its weights. In this model, θ_k is null and φ is the identity function.

$$y_k = \sqrt{(x_1 - w_{k,1})^2 + (x_2 - w_{k,2})^2 + (x_3 - w_{k,3})^2}$$

The *winner neuron* is defined as the closest neuron from the input (see Figure 4). The second feature is that a static neighborhood is initially defined between the neurons of the network. Every neighbouring neurons are linked together by a line for representation. Figure 3 shows a tridimensional cubic neighborhood between the neurons of the network.

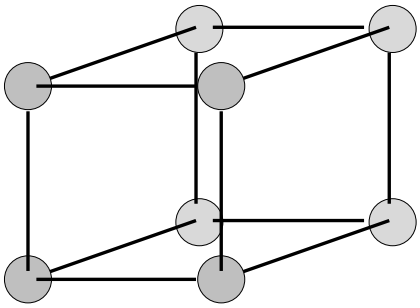

Figure 3: Example of a 3D neighborhood in the Kohonen network

We now present an algorithm for a Kohonen network having a tridimensionnal input (each neuron has, in this case, three synapses). The inputs are chosen randomly in the input space to obtain a feature of generalization. One could say that the network "learns" the distribution of the input space by modifying the weights of its neurons. This concept is discussed in more details in [6].

initialization

- input: $(\vec{x_1}, \ldots, \vec{x_n})$ a set of tridimensional vectors
- set $\vec{w_k}$ to random values for each neuron.
- define a static neighborhood between the neurons of the network.
- α is a learning factor and $0 < \alpha \leq 1$.
- V_k is a set of neighbours for the neuron k.
- let $t = 0$

algorithm

1. Pick up randomly an input vector
2. Search the winner neuron
   The winner neuron (i.e the closest neuron) is defined as the one with $y_k = \min_{1,\ldots,p}(y_l)$, where p is the number of neurons and $y_l = \|\vec{w_l} - \vec{x_i}\|$.
3. For this neuron k and $V_k(t)$, do the update $\vec{w_j}(t) = \vec{w_j}(t-1) + \alpha(t) * (\vec{x_i} - \vec{w_j})$, where $j \in \{k, V_k(t)\}$.
4. $t = t + 1$, update α and V_k for the next iteration
5. return to step 1 until the network has converged.

The complexity of the weight update depends directly on the chosen network topology. The level of convergence of the network can hardly be evaluated and the convergence speed depends of the chosen parameters for the algorithm. One can say that the network converges when it presents an organization (this term will be clear with the following examples). It is very important to note that the neuron weights are initialized to random values. These values should be uniformly distributed in a bounding box enclosing the MD system. After having randomly chosen an input, the winner neuron and its neighbours are approached from it using a displacement vector proportional to a α factor and to the distance they are separated by. Figure 4 illustrates this operation, with $\alpha = 0.5$.

This operation is performed by the Kohonen algorithm in each iteration. The learning factor α is continuously decreased during the iterations. The chosen decreasing function is

$$\alpha(t+1) = \alpha(t)(\frac{1}{1 + \Phi \cdot \alpha(t)}$$

where $\alpha(t)$ is the value at iteration t and Φ is an increment set by the user. For instance, a common value for $\alpha(0)$ is 0.9. The value $\alpha(t)$ is updated after every T_α

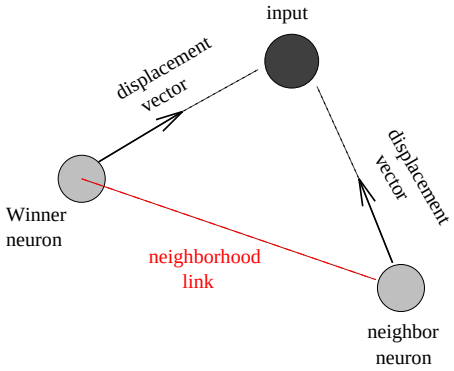

Figure 4: Illustration of the Kohonen algorithm

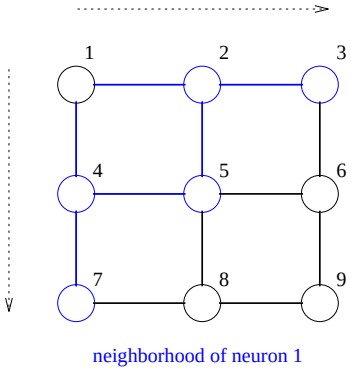

Figure 5: Illustration of an 2-extended neighborhood for the neuron 1

time interval. Initially, it is necessary to artificially extend the neighborhood for each neuron to the complete network, and decrease it during the iterations.

The extended neighborhood is not used for the graphic representation and only the links with direct neighbours are displayed. In the example of figure 5, the initial neighborhood of neuron 1 is extended to neurons within a distance of two. In this case $V_1(0) = \{2, 3, 4, 5, 7\}$. The neighborhood decreasing function is then

$$V_k(t+1) = \begin{cases} V_k(t) - 1 & \text{if } t \bmod T_v = 0 \text{ and } V_k \notin \emptyset \\ V_k(t) & \text{otherwise} \end{cases}$$

where T_v is the decay interval. The operator $\ominus$ denotes a reduction of the neighborhood set according to the network topology. Figure 5 shows an example of a bidimensionnal neighborhood topology (planar grid). For instance, $V_k(0) - 1$ would be $\{2, 4\}$ according to the indices in figure 5).

3. APPLICATION IN VISUALIZATION OF MOLECULAR DYNAMICS

3.1. Discrete representation

The following examples show the results obtained using different atom distributions. The chosen neighborhood topology is a tridimensionnal grid, whose size is given by three integers (n_x, n_y, n_z) representing the number of neurons in each dimension of the topology.

In the first example, the original system size is 250000 atoms and their distribution is approximately uniform. The network convergence is obtained using the parameters $\alpha(0) = 0.9$, $T_\alpha = 1000$, $\Phi = 1$, $T_v = 2500$, $\delta = 6$, $n_x = n_y = n_z = 6$ after 20000 iterations of the Kohonen algorithm.

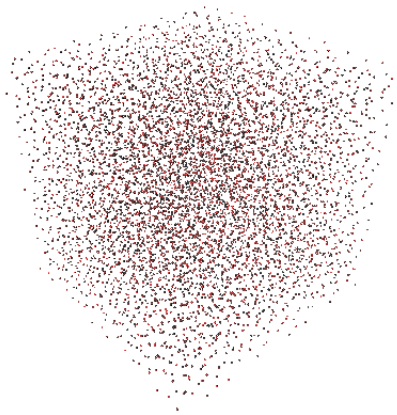

Figure 6: original system

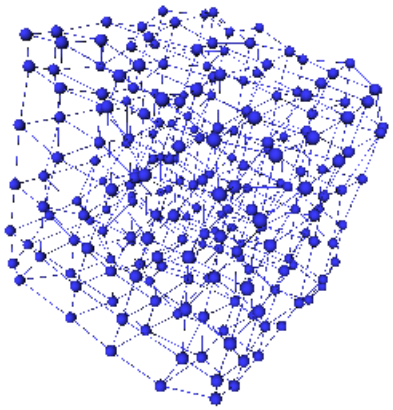

Figure 7: discret system

One can see that the neuron distribution is an approximation of the system distribution. One obtains high quality convergence (i.e approximation) on continuous distributions of atoms for both uniform or non-uniform distributions. On detached distributions -when the system have more than one separated region where atoms are located- the convergence is harder, because of the regular structure of the network. This case is illustrated below.

In the next example, the system size is 170000 and the atoms are quite uniformly distributed in each of the boxes 1 and 2 (see configuration scheme on Figure 8), but the density of atoms in box 2 is half of the density in box 1. A network with a neighborhood configuration of $n_x = 5$, $n_y = 5$, $n_z = 5$ was used. A neighborhood extension was made along dimension n_z where the system is extended by box 2. The parameters used were $\alpha(0) = 0.9$, $T_\alpha = 1000$, $\Phi = 1$, $T_v = 2500$, $\delta = 6$, $n_x = n_y = 5$ and $n_z = 6$ and 20000 iterations of the Kohonen algorithm were performed.

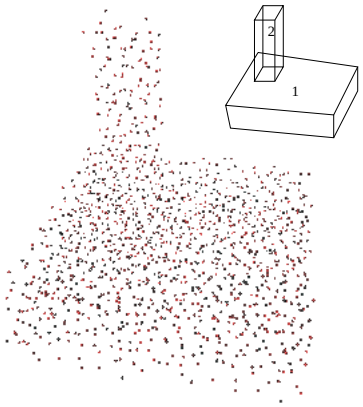

Figure 8: original system

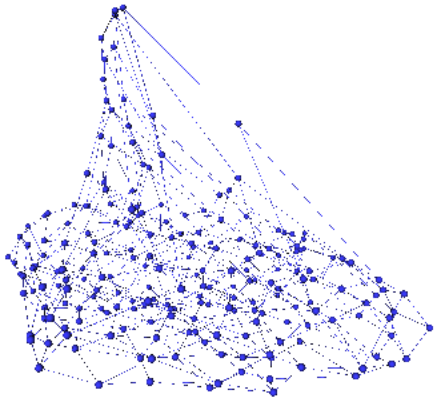

Figure 9: discret system

In this case, convergence quality is lower and one can see that some neurons appear in locations where no atom is present in the original system. We call these neurons *"lost" neurons*. This is due to the chosen neighborhood topology for the neural net. Some neurons have neighbours in both regions represented with boxes 1 and 2 and are "lost" between these. In the next sections, two techniques to improve visualization

of such systems are described.

3.2. Property mapping

Each neuron (after convergence) can be coloured using information such as the average temperature of the surrounding atoms. The property function applied to the neuron i is

$$p_i = \frac{1}{n_i} \sum_{j=1}^{n_i} T_j$$

where n_i is the number of atoms surrounding this neuron and T_j is the temperature of atom j. Other functions for p_i could be used to filter atom properties, and thus enhancing visualization in some applications. The property mapping on the neural net must be applied after convergence because of the random initialization of the neuron weights. The neurons will then "travel" in the discrete space before reaching their final location. After convergence and for each atom of the original system, the winner neuron is searched. Then, the atom property is added to the neuron property. Finally, the final neuron property is the average of the properties which were added to it. The next illustrations show the results obtained using this technique. The MD systems are the same as those used in the previous examples.

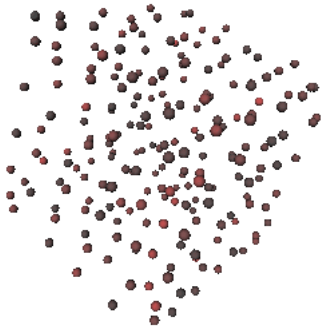

Figure 10: Exemple of property mapping

In figure 10, one can observe now a "line" of warm neurons (high average temperature p_i). With this result, one can see that in the discrete model the user can track more efficiently the properties of the system. Information that are not visible in the real system, due to the too great amount of data, can now be observed. The region emphasized on the discrete model shows a warm region of atoms which is not discernable in the real system. If after the property mapping operation some neurons were never "hit" by atoms (this will be the case for the lost neurons), they will be deleted from the representation. In the second example, property mapping on the detached distribution used in a previous example is applied.

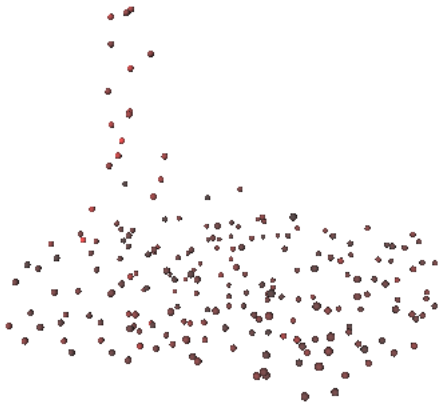

Figure 11: Higing of neurons using property mapping

As shown in Figure 11, The "lost" neurons are successfully hidden and the visualization of the discrete system has been enhanced.

3.3. Introducing a varying radius in the representation

An efficient method to handle the lost neurons is to represent each neuron with a radius proportionnal to the number of atoms it represents (by counting for how many atoms each neuron is the winner neuron). We will see that this solution is very usefull when we will introduce the algorithm for incrementally updating the network. The radius size is increased logarithmically in function of the size of the input. We give here some example of the convergence obtained with a detached distribution. The atoms are simply distributed on two parallel squares, having both the same density of neurons. Figure 12 show the convergence without using a varying radius, whereas figure 13 show the result while using it.

3.4. Adding an incremental mechanism

We call an incremental update a method which allows, from an initial network configuration (defined by a set of neuron weights), to find the resulting network configuration after having modified the input space used for the previous network convergence. The construction of such an algorithm is possible for two reasons:

- Because of the very small time step used, the atoms in the MD system move slowly. Then, the

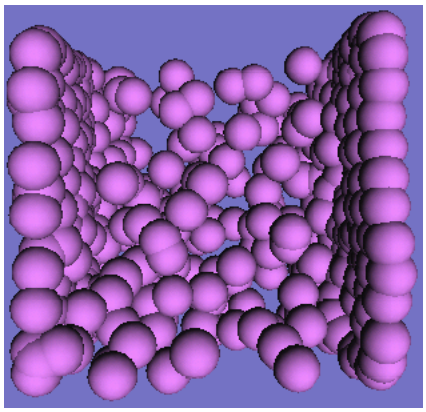

Figure 12: Convergence without using a varying radius

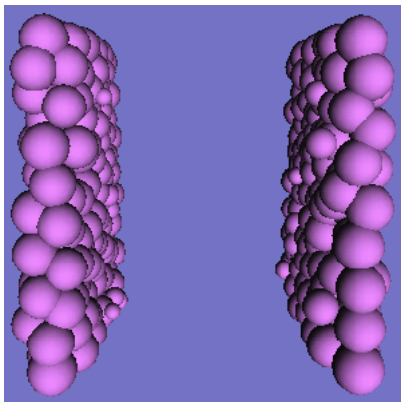

Figure 13: Convergence using a varying radius

values of the set of atom coordinates obtained at one iteration is not very different from the ones obtained at the next iteration.

- The values of a set of successive locations of an atom are correlated during the simulation.

The issue is that some atoms in the system move faster than others, and those ones ought to influence more the network (i.e the discrete representation), as the difference between their previous locations and the new ones is greater than that of the other atoms. For this reason, we propose a mechanism for choosing randomly these "interesting atoms" more often than the ones whose position do not vary greatly. The core of the incremental algorithm relies on this idea. The velocity of an atom determines its activity. This value can be used to assign a probability to each atom for the random selection. A probability mass function can then be constructed for the whole system. Even if the network have previously converged before using the incremental algorithm, the atoms still have to be chosen randomly for the following reason: the Kohonen network gives an approximation of the distribution of its input and has a feature of generalization, and by choosing with a higher probability the fast atoms, the discrete model will reproduce more faithfully the system motion. This insight allows the network to be modified on only a small random subset of the database (the atom coordinates) and not the complete set, and thus lower the number of necessary iterations to obtain the network update. A simple and efficient algorithm for the random selection of the atoms is given below. The first task is to normalize the atom velocities and to sort them in a decreasing order. The normalization is done using the value , where

$$\Gamma = \sum_{n} \sqrt{v_{x,i}^2 + v_{y,i}^2 + v_{z,i}^2}$$

where $v_{x,i}$, $v_{y,i}$ and $v_{z,i}$ are the velocity of atom i on each axis and n is the system size. Each velocity

$$0 \leq \frac{v_i}{\Gamma} = \frac{\sqrt{v_{x,i}^2 + v_{y,i}^2 + v_{z,i}^2}}{\Gamma} = \bar{v_i} \leq 1$$

Finally, by simply generating uniformly a random number in the range $[0 \ldots 1]$ and using the following algorithm, an atom can by chosen according to its assigned probability:

Initialization

- $[v_1^{\bar{a}_1}, \ldots, v_n^{\bar{a}_n}]$ is the sorted velocity vector, where a_i correspond to the atom index.
- $i = 0$

Algorithm

1. generate a random number $0 \leq \xi \leq 1$
2. $i = i + 1$, $c = \sum_{j=1}^{i} v_j^{\bar{a}_j}$ and $index = a_i$
3. if $\xi \leq c$ then take the atom which index is $index$ else return to 2

The great advantage of sorting the normalized atom velocities in a decreasing order is that one can minimize the number of steps of this algorithm for finding the chosen atom. Initially, the non-incremental algorithm (see Section 2.2) is used to obtain an initial state for the incremental algorithm. During the update only a few iterations are sufficient to update the network (usually 5-10% of the number of iterations necessary for convergence when using the non-incremental algorithm), but this depends on the system shape and the topology of the network. This makes this method suitable

for animation. To keep a fine discrete representation of the system, it is very important to have a very small learning value (computed by the $\alpha(t)$ function) and to have a 0-extended neighborhood, meaning that an input modifies only the winner neuron.

In the following example, a MD system containing 2000 atoms has been discretized using a 4x4x4 Kohonen network. The system has been simulated using molecular dynamics algorithm during 100 time steps. After computing each time step, the network has been incrementally updated using the new atom coordinates. Figure 14 shows the MD system evolution after 100 timesteps. We obtain a similar initial discret state as the one shown in figure 7. The result of this computation is taken as the initial network state by the incremental algorithm. Figure 15 shows the resulting incrementally modified network.

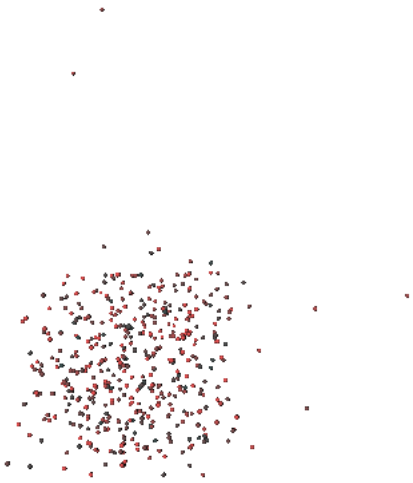

Figure 14: system evolution after 100 timesteps

One can observe in this example that the system begins to burst after some timesteps. The network tries to follow its motion and is heavily modified by the atoms "escaping" the system. In this kind of simulation, no boundaries for the MD system has been simulated. This is definitely the worse case for the incremental modification as the network structure cannot burst. On system having quasi-periodic motion, for example on a melting structure, we obtain a very good approximation of the atoms motion. In this case, the varying radius concept introduced in section 3.3 is very useful, as during the incremental modification, each neuron does not represent anymore an equal density of atoms, as it was the case in the original model. We can now approximate the motion of the system and have a representation of the amount of input that are effectively

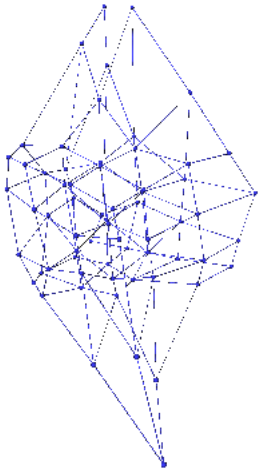

Figure 15: discrete system obtained with the incremental update

moving. We give here an example of a 2000 atoms systems. This system simulates a set of atoms where only a few of them are sufficiently excited to bounce on the structure and then to return in the group. We give approximations of the system at several resolutions to show that, even with a low complexity model (i.e small value of the parameters n_x, n_y, n_z) one obtain a good approximation of the original model. We can then easily adapt the resolution to the capabilities of the machine. We give here only a representation of the original model: figure 16 shows the initial state of the periodic motion, whereas figure 17 shows the highest location reached by the atoms. The resulting approximations are given in appendix.

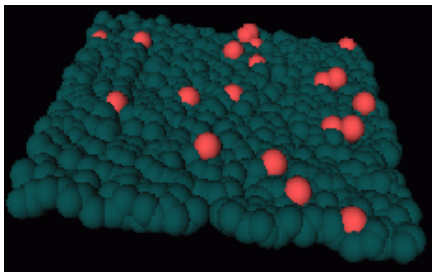

Figure 16: Initial state. a few excited atoms are emerging of the structure

The incremental mechanism added to the Kohonen algorithm is the key to obtain animated visualization of large systems because of the low number of iterations necessary to obtain the network update. This feature allows visualization of an MD system while this one is

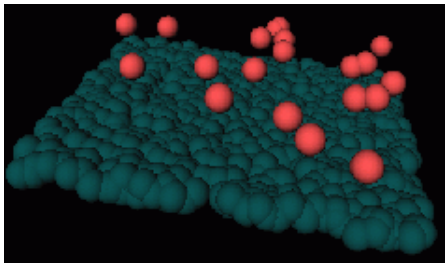

Figure 17: The excited atoms reached their highest location and will now fall down into the structure

computed. In the case where the user works on precomputed data, the incremental algorithm can be used to obtain a sufficient frame rate for the animation of very large molecular dynamics systems.

4. DISCUSSION

The methods described in this paper can be used to improve rendering performances and/or to animate very large particle-based systems having drastic density variation over space and time. It can also be used to visualize them at different levels of resolution, depending on the chosen resolution level (i.e the values (n_x, n_y, n_z)). These new possibilities can be exploited in many applications and some examples are presented below.

We showed in this paper that, using a topological and adaptive sampling method, one can represent the organization of a molecular dynamics system with less information. In the example of Figure 10, one can see that the system is represented with 1000 times less information (if we consider the factor between the amount of atom coordinates and the amount of neuron weights). Animation of very large system can be thus performed. A multi-resolution concept can be introduced with the following example. Assuming that an user wants to initially visualize the organization and behaviour of a atoms system by representing each atom with a sphere, then zooming into the "interesting" parts of the system to observe phenomenons like aggregation or micelles. Obtaining an animation with a sufficient frame rate with such a great amount of information is still a challenge in computer graphics, even when using level of details. This can be achieved by animating a discrete representation of the system. Zooming into the system will reduce the size of the visible scene. So discretization can be targeted on the observed sub-system, improving resolution of the discrete scene. One can obviously represent the real system when the scene complexity is sufficiently low, depending on the hardware used.

Assigning properties to the components of the discrete system is done with the property mapping process. Each discrete value characterizes its surrounding environment. We have also shown that the property mapping technique on the discrete system can also be useful to observe phenomenons that cannot be seen in the real system, because of the great quantity of information. This remark is also illustrated by figure 10.

In [2], a crowd in a virtual building is simulated using a model in which each individual is an independent entity. To optimize the position of signs, the corridors dimensions or the location of emergency exits, the pedestrian flow is visualized by representing each person with a geometric model. To prevent graphic bottlenecks, several levels of details are defined for those models, allowing faster animations when large crowd are visualized. An alternative to this graphic representation would be to discretize crowd before representation, as only global movement tendencies want to be observed. This approach would allow the simulation of a much bigger crowd. Geometric models should then be designed to represent groups of people instead of individuals. According to the chosen discretization level, our method would allow us to observe the evolution of crowd flows at different levels of resolution.

There is very little literature which directly concerns the simulation of human hair structure using particle systems. Particle systems have been previously used by Reeves [7] to model fire and by Reeves and Blau [8] to represent tree and grass. The application of our method is exposed for human hair representation, but can also be applied to those former works. Usual hair structure of a human is composed of 100000 to 200000 strands of hair, whose diameter ranges from 40 to 120. In [5], Rosenblum, Carlson and Tripp proposed a method in which each strand is represented using a series of connected straight strand segments. Each strand segment is rendered as a fixed width line and its position is determined by a simple dynamic simulation. Each strand structure is static, preventing the animation of large hair structures in case where many virtual humans are represented. Using our method, a entire hair structure could be discretized with a level of resolution depending on the distance to the viewpoint. This could be used to model a crowd of virtual humans. The discretization level could be adapted according to the distance from which the observation is made, or according to the graphic capabilities of the system used. To have a realistic motion of the hair structures, physical equations would need to be solved for the total number of strands in the scene and visualization would be achieved using the discrete systems, with an appropriate geometric model for each discrete value. Obviously, in the case where many humans, i.e

many hair structures, are modelled, an equivalent number of neural net would have to be used. Discretization can still be done even when the observation of the hair structure is made at a close distance as the neuron-hiding capability of our method can be used to "erase" neurons not converging on a hair strand.

The visualization of water currents can also be efficiently achieved with particle-based techniques. In [4], Chiba and al. propose to model water using a set of imaginary particles called "water-particles". To generate images, they determine the water surface with a scalar field $F(x, y, z)$, using a sum of Gaussian density distribution (one for each water particle). The equation $F(x, y, z) - c = 0$ is then solved using an algorithm proposed by Doi and Koide [1]. This equation becomes complex when large expanse of water are modelled. Rendering can thus be improved by initially discretizing the set of water particles, which will obviously reduce the scalar field complexity. Our discretization method is convenient in this case, as the density of water particle is not necessary uniform, especially when simulating waves.

5. ACKNOWLEDGMENTS

This work has been realized during a project at Silicon Graphics, Cortaillod in Switzerland with the collaboration of the Computer Graphics Lab at EPFL Lausanne, Switzerland. The project topic was to visualize large atoms systems simulated using parallel algorithm on SGI SMP machines. We greatly appreciate Silicon Graphics for providing an exciting environment in which this research was conducted. We also thank Prof. Daniel Thalmann of the Computer Graphics Lab at EPFL Lausanne for the coordination between EPFL and SGI. Last but not least, we would like to thank Ronan Boulic and Tolga Capin from the Computer Graphics Lab at EPFL Lausanne for their great advices, support and discussions and Grace Chang for her final review.

6. REFERENCES

[1] A.Doi and A.Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Transactions E74*, 1:214–224, 1991.

[2] E.Bouvier and P.Guilloteau. Crowd simulation in immersive space management. In *3rd EUROGRAPHICS Workshop on Virtual Environments*, 1996.

[3] A. Luciani, A. Habibi, A Vapillon, and Y. Duroc. A physical model of turbulent fluids. In *Computer Animation and Simulation 95, Springer Computer Science. ISBN 3-211-82738-2*, 1995.

[4] N.Chiba, S.Sanakanishi, K.Yokoyama, I.Ootawara, K.Muraoka, and N.Saito. Visual simulation of water currents using a particle-based behavioural model. *The Journal of Visualization and Computer Animation*, 6(3), July-September 1995.

[5] R.E.Rosenblum, W.E.Carlson, and E.Tripp III. Simulating the structure and dynamics of human hair: modelling, rendering and animation. *The Journal of Visualization and Computer Animation*, 2(4), October-December 1991.

[6] S.Haykins. *Neural nets, a comprehensive foundation.* IEEE Computer Society Press.

[7] W.T.Reeves. Particle systems, a technique for modelling a class of fuzzy objects. *IEEE Computer Graphics*, 17(3):359–376, 1983.

[8] W.T.Reeves. Approximate and probabilistic algorithms for shading and rendering structured particle systems. *IEEE Computer Graphics*, 19(3):313–322, 1985.

A. APPRENDIX

We give here the result of an approximation at several resolutions of the model described in section 3.4 (figure 16 and figure 17). Obviously the state reached in figure 17 is the most critical to approximate, as some atoms are separating themselves from the structure. We give then illustrations only for it, as the approximation of the initial one (figure 16) is already considered in the previous section of this paper. We can observe that, thanks to the varying radius introduced in section 3.3, we have a notion of the amount of atoms which follow the approximated motion.

In the first illustration (figure 18), one can see that the motion is roughly approximated. Because of the small amount of neurons, each one is approximating a large number of atoms. This implied that the neurons are not separating themselves very much from the structure and their property (depicted by their color) poorly locate the excited atoms. The second approximation (figure 19) is done with 64 neurons. Here we begin to see the location of the excited atoms and the varying radius is showing explicitly that only a small density of them are following the approximated motion. The next approxmation (figure 20) gives a better result as we raised the resolution level to 100 neurons. Finally, in figure 21, with a resolution of 144 neurons, we obtain an approximation which is very close to the

original model. The results obtains are very interesting as we made absolutely no assumptions on the model to approximate (except that we adapted the parameters of the topology (n_x, n_y, n_z) to its shape) and on its motion.

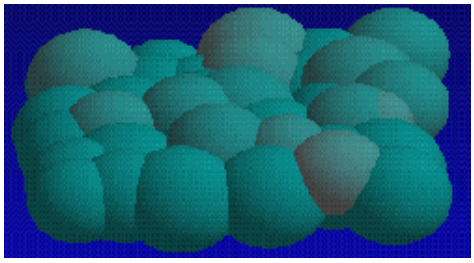

Figure 18: approximation $(n_x = 6, n_y = 6, n_z = 1)$

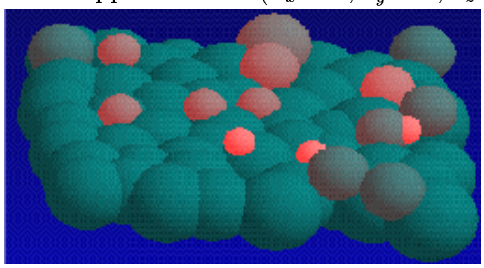

Figure 19: approximation $(n_x = 8, n_y = 8, n_z = 1)$

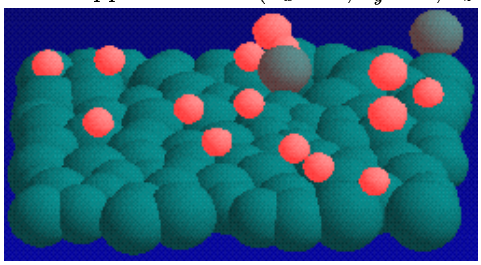

Figure 20: approximation $(n_x = 10, n_y = 10, n_z = 1)$

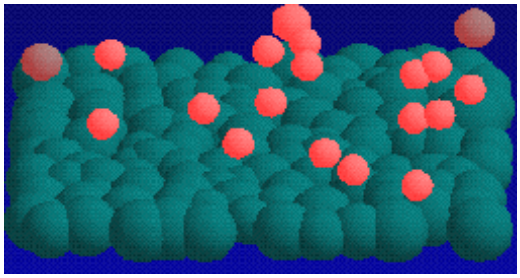

Figure 21: approximation $(n_x = 12, n_y = 12, n_z = 1)$