CHAMELEON: A LEARNING VIRTUAL BOT FOR BELIEVABLE BEHAVIORS IN VIDEO GAME

Fabien Tencé[1,2], Laurent Gaubert[1], Pierre De Loor[1] and Cédric Buche[1]
[1] UEB – ENIB – LAB-STICC
[2] Virtualys
Brest – France
{tence,gaubert,deloor,buche}@enib.fr

KEYWORDS
Bayesian inference, believability, behavioral simulation, experiments.

ABSTRACT

The believability of a virtual world can be increased by improving the behavior of the characters in it. Considering literature, we choose a model developed by Le Hy to generate the behaviors by imitation. The model uses probability distributions to find which decision to choose depending on the sensors. Then actions are chosen depending on the sensors and the decision. The core idea of the model is promising but we propose to enhance the expressiveness of the model and the associated learning algorithm. We hope the model will be able to generate more believable behaviors and learn them with minimal *a priori* knowledge. We first revamp the organization of the sensors and motors by semantic refinement and add a focus mechanism in order to improve the believability. To achieve believability, we integrate an algorithm to learn the topology of the environment. Then, we revamp the learning algorithm to be able to learn much more parameters and with greater precision at the cost of its time of convergence.

INTRODUCTION

Presence is one of the main goals of virtual worlds. It consists in making the users of those environments feel like they are in the simulation. To do so, there must be many rich interactions between the user and the virtual world. One option is to populate the simulation with entities which exhibit a *believable behavior* (Bates 1994). The main difficulty is that in virtual worlds, some entities may have unpredictable behaviors, like for instance users' avatars. It may lead to unexpected situations which artificial entities may not be able to handle correctly, exhibiting unadapted behaviors. Regarding this kind of problems, it is necessary for the entities to be able to dynamically evolve. We propose to provide our entities with *imitation learning* abilities in order to adopt a real player's behavior.

Video game companies want the players to be immersed in the simulation. To achieve this, they create rich and complex virtual worlds. Thanks to these kind of work, researchers can avoid some technical difficulties (rendering, physics, networking, etc.) by using such games and then focus on the entities to be studied (Mac Namee 2004). Furthermore, and as a feed-back effect, since video games are made for human users and often popular, they offer a real challenge for the entities to be believable.

In this article we first give an overview of the kind of models which drive entities' behaviors in video games. Then we focus on a model from Le Hy, which seems to fit our need for both believability and imitation learning. Then we propose some modifications in order to make the virtual character more believable and capable of learning almost all the parameters of the model.To conclude, we give explanations about how we would like to evaluate the believability of our model.

LITERATURE

In the video game industry, most of the games are scripted to have a storyline, models must be flexible and readable enough to be adjusted by game designers. Animation is also important so models have to handle low level details. As a consequence, very simple models are still used such as finite state machines (FSMs). They are easy to read and give a good control on how the character will behave. The main drawbacks are that it is hard to code complex behaviors, difficult to maintain and they need a lot of time to be parametrized correctly.

An alternative to FSM are behavior trees. They give the same control over the character's behavior but it is a lot easier to code and maintain. However, they are still not widely used is the video game industry and, as FSM, they lack of expressiveness for complex behaviors.

Bayesian-based approaches

Recently, some Bayesian-based approaches have been used to control characters in video games. The model described in (Gorman et al. 2006*a*) tries to apply to a video game a Bayesian model of imitation previously developed in (Rao et al. 2004). The believability of this model has been studied in (Gorman et al. 2006*b*). The model, however, does not seem to offer easy generalization. But it shows that a Bayesian approach fits to our need for believability and compatibility with imitation learning.

A specific Bayesian-based model (Le Hy et al. 2004) has been developed for characters in video games. The advantage of this model is that it is quite easy to modify so that the character acts as wanted. It is also possible to learn the parameters of the model by imitation. The believability of the behaviors has not been carefully evaluated but some preliminary tests show that it may be comparable to models from industry. This last model will be our base for future work because it is quite close to FSM which give good results. It has also more expressiveness, partly because it uses probabilities, and has an imitation learning algorithm. We believe that choosing a model, more complex than what is used in industry but less complex than cognitive or multi-agents model, is a good mean to generate believable behaviors. Indeed, using that approach, it is possible to generate complex behaviors, still being able to understand and modify the internal parameters to achieve the best believability.

Le Hy's model

In Le Hy's model, the agent has sensors named $S = (S_0, .., S_n)$. They give information on internal and environment's state like for instance the character's inventory and the position of another character. Agent's movements are driven by motors named $M = (M_0, ..., M_p)$ which can be rotation, jump commands and so on. Both sensors and motors take discrete values. In order to simulate the character's behavior, the notion of decision has been introduced, the associated variable is named D and may have different values like searching for an object or fleeing.

The value of D^t, where t is the time, is chosen according to the value of the sensors, following the probability distribution $P(D^t|S)$, and to the previous decision, following the probability distribution $P(D^t|D^{t-1})$. Thus the value of D is chosen following the probability distribution $P(D^t|S^tD^{t-1})$, the decision model, computed using the two previous distributions. As S is the conjunction of n variables, Le Hy introduces the notion of *inverse programming* to reduce the complexity: $P(D^t|S^t)$ is computed using $P(S_i^t|D^t)$ (and not $P(D^t|S_i^t)$!) as they are assumed to be independent, which is a strong assumption.

Once the value of D^t is chosen randomly following the distribution $P(D^t|S^tD^{t-1})$, the model must decide which motor command should be activated. The value of each motor command is chosen following the distribution $P(M_i^t|S^tD^t)$, the motor model. Again, to reduce the complexity, Le Hy introduce the notion of *fusion by enhanced coherence*. Each command is computed separately then they are combined using the formula $P(M_i^t|S^tD^t) = \frac{1}{Z}\prod_j P(M_i^t|S_j^tD^t)$ where $1/Z$ is a normalization factor.

Thus the model, which can be categorized as an input-output hidden Markov model, is composed of three types of parameters whose relations are summarized in figure 1:

- $P(D^t|D^{t-1})$
- $P(S_i^t|D^t)$
- $P(M_i^t|S_j^tD^t)$

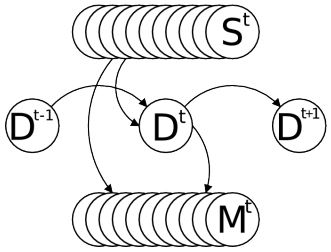

Figure 1: Summary of the influences between model's variables (Le Hy et al. 2004).

Those parameters can be specified by hand or learned by imitation. Results seems to be better in term of believability and performance with learned parameters. The imitation is done by observation of the virtual representation of the player, his avatar also named the demonstrator. By monitoring at each time step the values for S and M for this demonstrator, it is possible to update the value of the parameters. The learning algorithm developed by Le Hy is based on (Florez-Larrahondo 2005) but only updates the decision model, the parameters $P(D^t|D^{t-1})$ and $P(S_i^t|D^t)$.

This algorithm, a modified version of the incremental Baum-Welch, updates at each time step the parameters with the following formula:

$$P_n(D^t|D^{t-1}) = \frac{1}{Z}\Big(P_{n-1}(D^t|D^{t-1}) + \Delta P_{n-1}(D^t|D^{t-1})\Big)$$

$$P_n(S^t|D^t) = \frac{1}{Z}\Big(P_{n-1}(S^t|D^t) + \Delta P_{n-1}(S^t|D^t)\Big)$$

$1/Z$ is a normalization factor. The Δ are computed using the motor model, $P(M_i^t|S_j^t)$, and the actual values of the decision model, $P_{n-1}(D^t|D^{t-1})$ and $P_{n-1}(S^t|D^t)$.

Limits

As believability is based on the feeling of an observer, we have to examine the behaviors produced by the model to see its real advantages and drawbacks. Based on those observations, we can propose some modifications to compensate potential flaws.

The main flaw of agents using Le Hy's model is the way they move in the environment. We noticed that its movements were not very smooth and gave an overall feeling of non-humanness. Moreover, it often chose motor commands which seem not to satisfy any goal. How the agent act is very important because it gives the first impression to observers.

On top of the movements, the paths the agent uses to go from one point of the environment to another do not look like the ones a player would take. This problem does not comes from the model itself but from the representation it uses for the environment. Indeed, the agent uses navigation points placed by the designers of the environment which may not represent well how players prefer to use the environment.

The implementation of Le Hy's model has not enough sensors to exhibit the whole range of behavior a layer would exhibit. Le Hy's model is flexible enough for new sensors to be added. However increasing the number of perception make the learning more difficult because it increases the parameters. A compromise should be found to give enough information to the agent without adding unnecessary complexity to the model.

CHAMELEON: LE HY'S MODEL ENHANCEMENTS

Despite the problems raised, the implementation of Le Hy's model showed that the decision sequencing using Bayesian programming is quite efficient and simple to settle. The concept of decision makes the behavior easy to adjust by modifying the probabilities value. This is why we decided to follows Le Hy's general idea. First, a decision is chosen knowing the previous one and some information about the environment. Then actions are done depending on the decision and the environment. However some limitations were raised during our different experiments so we modified the way the decision and the actions are chosen and how the sensors and actions are linked to the decision in several ways.

Improvements On Le Hy's Decision Model

The first change we made is to apply a semantic refinement on sensors, splitting them in two types: high level ones (random variables H_i) and low level ones (random variables L_j). As decisions represent general behaviors (attacking, fleeing, etc.) the agent does not need a very accurate information about the environment to make its choice. However, in order to accomplish any action according to the chosen decision, the agent needs much more accurate values (to aim or to run avoiding walls for instance). This should increase the amount of available information for the agent without increasing too much the parameters.

The second change is to regroup actions into three types: motion, interaction and reflexive actions. A motion action (random variable M) gathers all the motor commands needed for the agent to be able to move in the environment. In our case it is a combination of five motors: pitch, yaw, run, lateral movement and jump. The difference with Le Hy's model is that we do not assume that all the motor commands are independent. As a consequence, the agent has a better control of how it moves at the cost of increasing the number of parameters in the model. An interaction action (random variable I) regroups all the motor commands needed for the agent to interact with other players or objects in the environment. In our case there is only one motor: shooting or not. The last kind of action is the reflexive action (random variable R): it models any action that the agent applies to itself. In our case there is only one motor: changing the current weapon. The main difference between reflexive actions and the other actions is that they depend on the current decision and the high level sensors whereas the other actions depend on the current decision and the low level sensors. This models the fact that the agent does not need very accurate information about its surroundings in order to achieve reflexive actions.

The third and last change to the model is to replace the mechanisms to reduce the complexity resulting from the number of possible values for sensors. The first mechanism used by Le Hy is the inverse programming where $P(D^t|D^{t-1}S^t)$ is computed using $P(S_i^t|D^t)$, assuming that all sensors are independent knowing the decision. This assumption may be wrong depending on the chosen sensors, and moreover, the more sensors used the higher the chances the assumption may be wrong. The second mechanism is the fusion by enhanced coherence. This technique suffers some simple problems: it makes use of probability distributions, but handle them in total opposition with their natural properties. The main aim of this technique is, in the end, to consider a weighted sum of probability distributions, which is easily and rig-

orously achieved with the sum of random variables over a random index. Therefore, we propose to use instead a mechanism where the agent focus on one high level sensor and one low level sensor. This focus mechanism use two distributions $P(G^t|H^t)$ and $P(J^t|D^tL^t)$ where the random variables G and J gives respectively the index of the high level sensor the agent focus on and the index of the low level sensor the agent focus on. H^t and L^t are respectively the conjunction of all the high and low level sensors. As a result, we must simplify the expression of the two distribution because they may take far too many values to be tractable. We propose to express the distributions as follows:

$$\mathbb{P}\left(G^t = i \,\middle|\, H^t\right) = \frac{\theta_i\left(H_i^t\right)}{\sum_n \theta_n\left(H_n^t\right)} \quad (1)$$

$$\mathbb{P}\left(J^t = j \,\middle|\, D^t, L^t, K^t\right) = \frac{\lambda\left(D^t, L_j^t\right)}{\sum_m \lambda\left(D^t, L_m^t\right)} \quad (2)$$

The higher the values of θ and λ, the more likely the agent will focus on the associated sensor. This greatly reduces the number of parameters still giving the agent a mechanism to focus only on one sensor.

The model works in the following way (see figure 2):

- Pick an index i of a high level sensor, using (1)
- Pick a decision using $P(D^t|D^{t-1}H_i)$
- Pick an index j of a low level sensor, using (2)
- Pick a motion action using $P(M^t|D^tL_j)$
- Pick an interaction action using $P(I^t|D^tL_j)$
- Pick a reflexive action using $P(R^t|D^tH_i)$

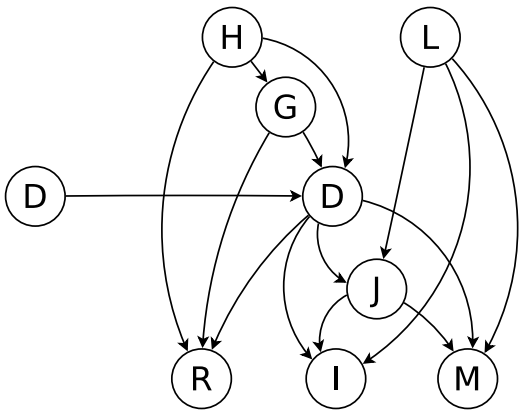

Figure 2: Summary of the relation between the random variable of the model.

Learning the Environment

To be able to learn the parameters of the model, the learning algorithm first needs sensors and motors which represent best how players interact with the game. While most of them are defined by hand, creating a representation of the environment is a quite tedious task. As a consequence, we looked for techniques to learn by imitation such a representation. Few works had been done on this subject, but an interesting technique (Thurau et al. 2004) tries to learn the movement in a virtual world by imitation. It uses a algorithm named neural gas to learn navigation points from the positions of a player's avatar. This algorithm has the advantage of being on-line: it learns at each observation of the player. Therefore, the neural gas can be used and learned at the same time, the character evolving during the game.

As this technique seems promising, we tried to apply a Growing Neural Gas (GNG) on the observed positions. The GNG (Fritzke 1995) is a graph model which is able to achieve incremental learning. Each node has position (x,y,z) in the environment and has a cumulated error which measures how well the node represents its surroundings. Each edge links two nodes and has an age which gives the time it was last activated. This algorithm needs to be omniscient, because the position of the imitated player (the demonstrator) has to be known at any time. The principle of the GNG is to modify its graph, adding or removing nodes and edges and changing the nodes' position for each input of the demonstrator's position. For each input the closest and the second closest nodes are picked. An edge is created between those nodes and the closest node's error is increased. Then the closest nodes and its neighbours are attracted toward the input. All the closest node's edges' age is increased by 1 and too old edges are deleted.
We trained 2 GNG on 2 different maps. The first one is a simple map, called Training Day, it is small and flat which is interesting to visualize the data in 2 dimensions. The second one, called Mixer, is much bigger and complex with stairs, elevators and slopes which is interesting to see if the GNG behaves well in a real three dimensional environment. The results are shown in figure 3 (a) for the simple map and in figure 3 (b) for the complex map.

In order to study the quality of the learned topology, we first chose to compare the GNG's nodes with the navigation point placed manually by the map creators. Of course, we do not want the GNG to fit exactly those points but it gives a first evaluation of the learned representation. In our case we know those navigation points but our goal is that they become not longer necessary for a character which evolves in a new environment. Figure 4 shows both the navigation points and the GNG's

nodes. As we can see, the two representations look alike which indicates that the model is very efficient in learning the shape of the map. However, there are zones where the GNG's nodes are more concentrated than the navigation points and other where they are less concentrated. We cannot tell now if it is a good behavior or not as we should evaluate an agent using this representation to see if it navigates well. Even in the less concentrated zones, the nodes are always close enough to be seen from one to another, so it should not be a problem.

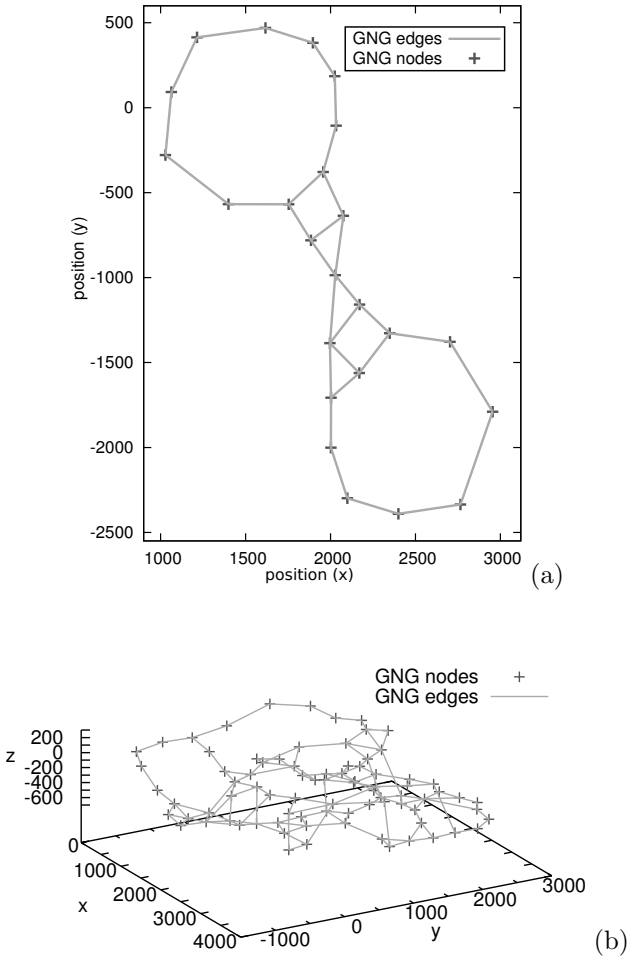

Figure 3: Result of a growing neural gas learned from a player for a simple map, top view (a) and for a complex map (b).

As the attraction applied to the nodes for each input is constant, the GNG is not converging to a stable state. This is a desired behavior, allowing the GNG to adapt to a variation in the use of the map: if the teacher suddenly uses a part of the map which he/she has not yet explored, the GNG will be able to learn this new part even if the GNG has been learning for a long time.

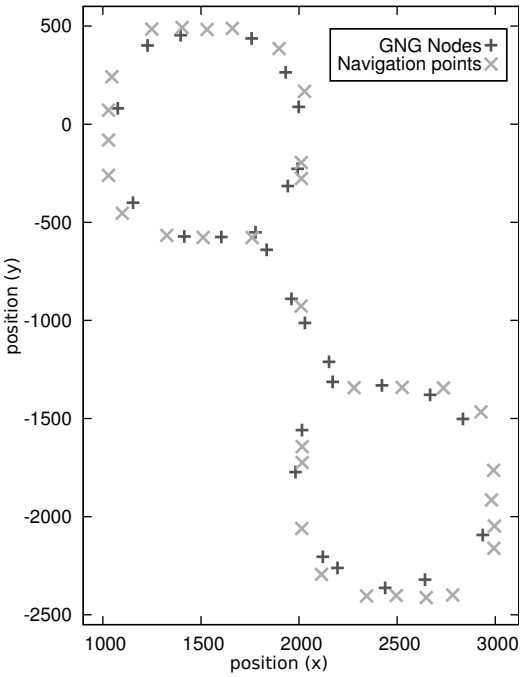

Figure 4: Comparison of nodes learned by the growing neural gas with the navigation points placed manually by the game developers.

Learning the Parameters Of The Decision Model

In order to learn the parameters of the model, Le Hy uses a modified version of Florez-Larrahondo's incremental Baum-Welch algorithm. We prefer not to use this algorithm because it has to approximate very roughly the probabilities computed by the backward procedure. In our case such probabilities give the chances of taking a certain decision at t knowing what the demonstrator did at $t+1...T$. This information should not be lost (for instance, if the demonstrator is looking for something specific, the learning algorithm cannot know what it is picked up). As a consequence, it seems wiser to learn on a whole sequence of observations (from time 0 to time T) instead of using an incremental version. We choose to begin the sequence when the demonstrator's avatar appears in the environment and end the sequence when the avatar dies.

We also want to extend the learning to all the distributions and not only the ones used to pick a decision. In order to do that, we apply a EM algorithm to our model to optimize the parameters. For each sequence of observations of a demonstrator, our algorithm gives a local maximum of the likelihood function. The quality of this local maximum depends on the initialization of the distributions and some modification we can do to the parameters during each maximization step. At the time we write this article, we use a random initialization and a technique to make the distributions less uniform

using the formula $a = \frac{1}{Z}a^n$, a being a probability, $\frac{1}{Z}$ a normalization factor and $n > 1$.

θ and λ are the only parameters that are not learned. Our first attempts to find solutions that optimize the likelihood function using a gradient method did not give good results. On the contrary, specifying them by hand and making the model learn on demonstrators gives quite good results, the agent being able to move in the world and shoot at enemies (although an accurate shooting procedure is hard to learn and it may be enhanced by specific heuristics).

CONCLUSION

Virtual worlds, like for example video games, need believable characters for users to feel in the environment. For virtual characters, actual solutions in research focus on intelligence with cognitive or multi-agents approach. In the industry, the goal is mainly believability but the models often generate too simple behaviors.

Le Hy's model seems to fill the gap between the two approaches, focusing on believability but trying to produce quite complex behaviors. To have a first look on the results of the model we implemented it. The result is very flexible, behaviors can be easily specified. However agents are not as believable as what is done in the industry. This difference can come from the assumption done in the model and because characters designers in the industry spend a lot more time to specify the behavior rules.

We first propose some modifications to Le Hy's model by rearranging the sensors and the motors and by adding a focusing mechanism. We hope these changes will enable the model to produce more complex behaviors. We then introduce the growing neural gas to learn the topography of the environment. Finally we apply an EM algorithm to learn how find a decision and to carry it out.

The learning algorithm still need some enhancements to be complete. θ and λ are not learned yet and techniques to find maxima close to the global maximum are still to be found. Then, the next step is to evaluate our work by gathering a pool of player to let the agent learn from their behavior. When the learning is done, we will try to assess the believability of our agent. Believability is very difficult to evaluate because it is subjective. We will do a test based on (Turing 1950; Gorman et al. 2006*b*), using humans as judges.

REFERENCES

Bates J 1994 The Role of Emotion in Believable Agents *Communications of the ACM* **37**(7), 122–125.

Florez-Larrahondo G 2005 Incremental learning of discrete hidden Markov models PhD thesis Mississippi State University.

Fritzke B 1995 A growing neural gas network learns topologies *in* 'Advances in Neural Information Processing Systems 7' MIT Press pp. 625–632.

Gorman B, Thurau C, Bauckhage C and Humphrys M 2006*a* Bayesian imitation of human behavior in interactive computer games *in* 'Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06)-Volume 01' IEEE Computer Society Washington, DC, USA pp. 1244–1247.

Gorman B, Thurau C, Bauckhage C and Humphrys M 2006*b* Believability testing and bayesian imitation in interactive computer games *in* 'From Animals to Animats 9' Vol. 4095 Springer pp. 655–666.

Le Hy R, Arrigoni A, Bessiere P and Lebeltel O 2004 Teaching bayesian behaviours to video game characters *Robotics and Autonomous Systems* **47**, 177–185.

Mac Namee B 2004 Proactive Persistent Agents: Using Situational Intelligence to Create Support characters in Character-Centric Computer Games PhD thesis.

Rao R, Shon A and Meltzoff A 2004 A bayesian model of imitation in infants and robots *in* C. L Nehaniv and K Dautenhahn, eds, 'Imitation and Social Learning in Robots, Humans, and Animals' Cambridge University Press pp. 217–248.

Thurau C, Bauckhage C and Sagerer G 2004 Learning human-like movement behavior for computer games *in* 'Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB'04)'.

Turing A 1950 Computing machinery and intelligence *Mind* **59**(236), 433–460.

BIOGRAPHY

FABIEN TENCÉ received his Ph.D. in computer science in 2011 concerning "Probabilistic Behaviour Model and Imitation Learning Algorithm for Believable Characters in Video Games". Now, he is working at Virtualys.

LAURENT GAUBERT is an associate professor of mathematics at the Brest National Engineering School (ENIB), which belongs to the European University of Britain (UEB). He is a member of the LAB-STICC IHSEV team, member of the European Center for Virtual Reality (CERV). His research concerns the simulation of adaptive behaviors, the synchronization of periodic coupled systems, the study of links between individuals and populations in abstract models and applied models, such as data produced by DNA chips.

PIERRE DE LOOR is a professor at the Lab-STICC laboratory (UMR-CNRS). He is the head of the IHSEV team since 2012 and the head of the CERV. He wrote his doctoral thesis in Automatic, Signal and Software engineering (1996) and his Accreditation to Direct Research (HDR) in Computer Science (2006). He's interrested on the link beetween artificial intelligence, cognitive science and virtual reality. He's also interrested on phenomenology, epistemology and links between art and science.

CÉDRIC BUCHE is an associate professor at the ENIB/UEB Member of the IHSEV team of the LAB-STICC, member of the CERV. He wrote an Accreditation to Direct Research (HDR) concerning the simulation of adaptive behaviors (2011). He is the leader of the CHAMELEON project.