

Neuronal Editor Agent for Scene Cutting in Game Cinematography

Erick B. Passos
Media Lab - UFF

Anselmo Montenegro
Media Lab - UFF

Vinicius Azevedo
UFSM
Cezar Pozzer
UFSM

Vitoria Apolinario
Media Lab - UFF

Esteban W. G. Clua
Media Lab - UFF



Figure 1: Shot library of an editor agent

Abstract

The use of cinematography techniques in games aims to provide high level abstractions for operating the virtual camera based on concepts borrowed from the movie industry such as scene, shot and line of action, among others. One usual approach is the development of agents to execute tasks that are similar to their counterpart in a real movie set: director, editor and cinematographer. However, a game is an interactive application and resembles a live TV-show, where the actions taken by all the actors is not known previously. In such scenario, the role of the editor is of great importance since he is the one who ultimately decides what point of view should appear on the screen. In the context of game cinematography, most previous works have proposed ways of mapping scene concepts and automatically controlling the virtual camera but without paying much attention on the role of an editor agent. Our paper shows an intelligent editor agent that uses neuronal network classifiers to decide shot transition and has an intuitive user interface to the learning mechanism.

Keywords:: Game Cinematography, Virtual Camera, Shot Transition, Cut, Neuronal Networks

Author's Contact:

{epassos,anselmo,esteban}@ic.uff.br
socorosca@gmail.com
vitoriamachay@hotmail.com
pozzer@inf.ufsm.br

1 Introduction

Visual simulations such as games have always relied on cutting edge real-time graphics. However, the search for simulation complexity has also given rise to new challenges in other areas such as artificial intelligence, physics and also storytelling. All the efforts in such areas account for the level of immersion in the virtual world and, in this trend, intelligent real-time camera handling plays an important role. Good use of the virtual camera has gaining even more importance because of trends such as games with spectators [Drucker et al. 2002], the use of real-time game engines to create films [Elson and Riedl 2007; Morris et al. 2005] and also storytelling research [Courty et al. 2003; Amerson et al. 2005; Pozzer 2005]. Previous research has been applying movie industry standards and concepts, such as scenes and shots, to video games in order to create higher levels of abstraction to manipulate the virtual

camera.

The most common concept in game cinematography is that of a film idiom, which represents the most usual way to present an specific type of scene event such as an *over the shoulder shot* for a dialogue sequence between two characters, or a *helicopter-mounted camera* for a fast paced car chase in a highway. Film idioms are good to represent specific camera behaviors that are then isolated and treated as a problem of its own with different possible solutions. Complete solutions for game cinematography, however, have to deal with other problems as well and are sometimes organized as agents that represent the various roles people play in a movie set [Hawkins 2004]. The most common approach is to consider three types of agents: director, editor and cinematographer.

Usually, the director agent is responsible for analyzing the scene and proposing film idioms to present it to the player. The editor agent is responsible for choosing which one to use, while the cinematographer agent directly controls the virtual camera [ref-hawkins]. It is important to remember that games are different from films in the sense that, being interactive applications, there is no prior knowledge of future events. A better analogy is to think of them as live TV-shows, where the scene and the actors are known, but not all the dynamic dialogues, events and actions taken are. In this kind of environment, one of the most important roles is that of the editor, who ultimately decides what to present to the audience. Based on these concepts, a distributed multi-agent system was proposed in [ref-short-paper-ours] to deal with the issues related to massive online games.

In such system, the role of the editor agent is of great importance, and these are the main concerns related to its implementation:

- What information should be available from the scene in order to enable good decisions;
- How to decide which of the available film idioms to use at each time;
- When to cut from one shot to the other without breaking frame-coherence.

In general, the ultimate goal is to create an agent that mimics the behavior of a human editor. In this paper we present the details of an editor agent that uses neuronal networks to enable consistent shot transitions in dynamic environments and also:

- Provides for a very fast learning mechanism based on real-time teaching by example;

- Requires minimal, not film-structured, information from the scene;
- Uses an intuitive, edit-like, real-time interface for the learning mechanism that requires no programming experience from the user.

The rest of the paper is organized as follows: section 2 compares our approach with previous research. Section 3 presents the system architecture and explains some details about the different agent layers. Section 4 brings a detailed look at the editor agent implementation, while section 5 details the game prototype developed to the experiments while section 6 analyzes these experiments and results. Finally, section 7 concludes the paper and presents our future work on the subject.

2 Related Work

A good amount of previous works were already dedicated to applying cinematographic techniques or other intelligent mechanisms of virtual camera control in video games. The majority of this research proposes the creation of higher level mechanisms for controlling the virtual camera with the adoption of cinema concepts, constructs and language such as scenes, shots, cuts, directors, editors and cinematographers [Drucker and Zeltzer 1995; wei He et al. 1996; Christianson et al. 1996; Halper and Masuch 2003; Hawkins 2004; Amerson et al. 2005; Tomlinson 2000; Hornung et al. 2003]. Some of these works use the concept of a *film idiom*, which encapsulates the combined knowledge of several personal roles in a traditional filming set. In the context of a complete solution, however, film idioms are good to solve only two parts of the problem: direct manipulation of the virtual camera; and creating a simple API for communicating with other layers.

In a work by Hawkins [Hawkins 2004], a three-layer architecture is proposed to split the issues through different agents categories: directors, editors and cinematographers. The director agent recognizes actions in the scene and suggests shot options to an editor agent, that chooses the best way to present them. This later issue is the responsibility of the third layer: a cinematographer agent, which directly operates the virtual camera. Bringing this organization to virtual cameras in computer graphics makes it easier to decouple the different modules needed by such system. Our work proposes a novel approach for the design and implementation of an editor agent that doesn't require the scene information to be structured in the form of scene events.

Other previous works are foundations on how to solve the issues related to the implementation of the virtual camera manipulation itself. Drucker [Drucker and Zeltzer 1995] studies the basic elements that compose and motivate the movements of a virtual camera, ending up proposing a agent-based framework to solve this task. Camera modules are designed as independent tasks that can be used independently or combined. Hermann e Celes [Hermann and Celes 2005] proposed a system that dynamically translates and orientates the camera with the use of physical constraint satisfaction. This later work is a good approach for implementing cinematographer agents, being complementary to our proposed editor.

Kneafsey and McCabe [Kneafsey and McCabe 2005] create an intelligent cinematography system for first person shooters based on a commercial engine. Their system dynamically switches camera shots based on specific information extracted from the 3D scene and use a Finite State Machine (FSM), augmented with fine grained activation rules, to represent the knowledge used to decide which shot to use at each time. Although appropriate to represent this kind of decision making knowledge, FSMs lack the ability to learn, and the user has manually specify the conditions that trigger each shot type.

He [wei He et al. 1996] also uses a FSM to represent film idioms but organizes them as a hierarchical structure. Each film idiom includes activation information as well as camera-modules, which are the equivalent to our cinematographer agent. Other works [Christianson et al. 1996; Amerson et al. 2005] also use hierarchical data structures to encode film idioms and provide a decision mechanism for shot transitions. Our system uses neuronal networks to store this knowledge and also present a very intuitive learning interface.

Hornung [Hornung et al. 2003] proposes an agent based system that drives the virtual camera based on relevant scene information of the cut-scenes from the game Half-Life. The editor agent of this system also use neuronal network classifiers to choose shot types, but uses a different approach (from ours) in that it receives information about the scene in the form of narrative events, containing coded information such as actors and targets. More important, however, is that it needs data to be translated from the scene domain to that of a narrative: emotional level, action level, among others. Because of this dependency on domain translation, this approach relies on a very complex director agent and is only appropriate to narrative applications that can provide such semantic information. Our editor agent does not need any domain translation and only requires input normalization.

Our work provides for a shot transition system that is more suitable to games, but its also important to mention research on storytelling since intelligent cinematography is one of the building blocks of this area. Pinhanez [Pinhanez 1999] studied the foundations for the representation and recognition of action events in interactive systems and developed a discrete mechanism and algorithm for the temporal structure of such actions called PNF (past, now, future) networks. This propagation algorithm is used to recognize the actions in an interactive context in replacement of other temporal representations such as Finite State Machines. Our system does not rely on action recognition to learn the rules for shot transition.

Pozzer [Pozzer 2005] developed an architecture for the generation and representation of dynamic interactive stories, including the presentation on a 3D engine and cinematographic techniques. The goal of the work is to provide an integrated tool to manage story generation for the brazilian digital TV system. In a more recent work, Guerra [Guerra 2008] proposes the term *story engineering* for this process, that is divided into three sub-problems: ontologies for story-driven data; story generation and storytelling, in which intelligent cinematography is included. Even being game-driven, our proposed editor agent is a suitable option for the shot transition issue, given its learning mechanism and intuitive interface. In the next two sections, our game cinematography system will be explained together with the editor agent's architecture.

3 Cinematography System Architecture

The proposed system architecture comprises three separate layers modeled as agents: scene mapper, editor and cinematographer. The role of the scene mapper agent is to gather relevant information about the 3D scene and to map it to normalized values (between -1 and 1). This approach provides a knowledge mapping of a scene data converted to a float-valued domain, which is independent of the specific 3D environment implementation, enabling a simple data structure to be used by the editor agent, which is called input-hub. This data structure does not require information to be coded with movie semantics such as other works [Hornung et al. 2003].

The movie semantics are not required because our shot-transition system relies much more on learning by examples than in predefined cuts specified in declarative languages [Christianson et al. 1996; Amerson et al. 2005] The information kept at the input hub will be the sole input of the editor agent, which is thus independent of the specific 3D environment. Figure 2 illustrates the role of the scene mapper: feed an input hub (float array container) with normalized information gathered from the 3D scene.

The editor agent contains a set of neuronal network classifiers, one for each available shot (cinematographer). This agent uses the output of each neuronal network to classify the candidate cinematographers at runtime. The key feature of the system is the use of an interactive training mechanism that enables the user to teach his shot preferences at runtime while watching gameplay. The basic architecture of the editor agent is illustrated in Figure 3, while a detailed explanation is given in the next section.

In our architecture, cinematographer agents are components implemented almost independently of the other ones. Their responsibility is to represent film idioms, such as a chasing camera or an over the shoulder shot, among others. The available cinematographer imple-

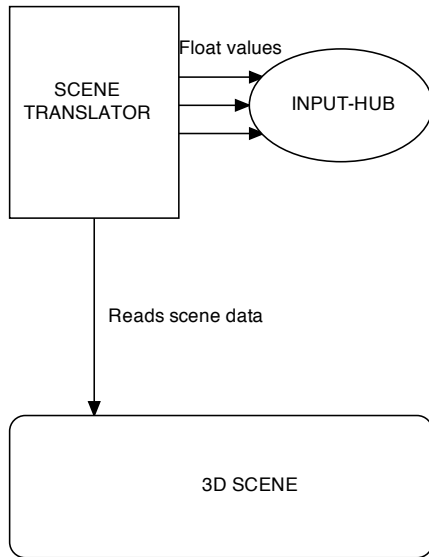


Figure 2: Scene Mapper

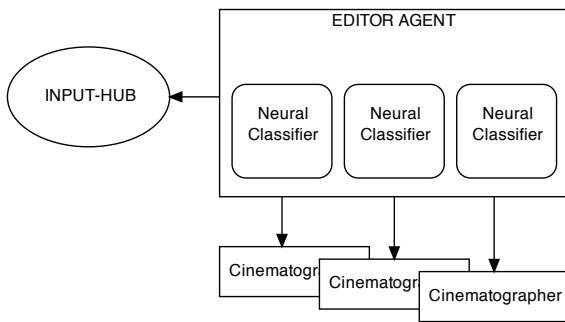


Figure 3: Editor Agent

mentations are enabled or disabled by the editor agent at runtime, based on the values given by the respective classifiers. In this current work, we are more interested in the cut/transition editing issue, so one is encouraged to read relevant previous research on the topic of film idiom implementations such as those pointed out in the related work section.

The communication between the three layers, shown in Figure 4, is made as follows:

1. The scene mapper, with the knowledge of the 3D environment, gathers and normalizes information about the scene to an array of float values within the range of $[-1,1]$. It constantly feeds these values into an input-hub, that is the only data-structure needed by the editor agent's neuronal network classifiers;
2. The editor keeps references to cinematographer agents and decides which one to use at each time, based on the output of the respective classifiers;
3. In our prototype, the cinematographers operate directly on the 3D environment, each one controlling its own virtual camera (viewport);
4. The position and orientation of the camera controlled by the chosen cinematographer are replicated to the editor viewport of the system.

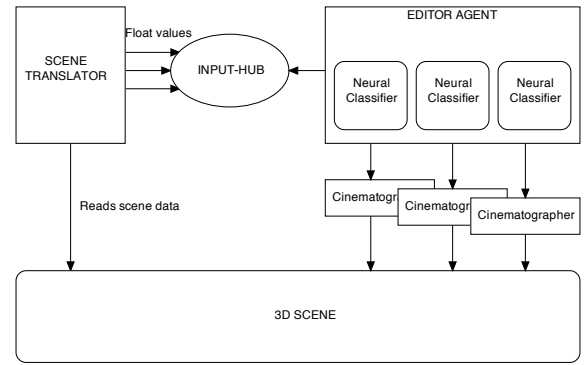


Figure 4: Communication Architecture

4 The Editor Agent

The role of the editor agent is to choose which shot, provided by the cinematographer agent, to use at each time for a given dynamic scene. It feeds all the values kept at the input-hub into several neuronal network classifiers simultaneously, one for each cinematographer. The output of each classifier is used as an activation value for the respective cinematographer. The goal of the training mechanism is to teach each classifier to recognize the situations that are favorable to its cinematographer. This training is performed by a user who only has to indicate his favorite camera at each relevant situation while interactively watching a player testing the game. In this section we explain in more detail the features of our proposed editor agent: the neuronal network classifiers; the learning mechanism; and the user interface.

4.1 Neuronal Network Classifiers

Neuronal network building blocks are called neurons, which are computation units that take as input a collection of values in a normalized range - usually from -1 to 1 - and compute an output. Each input value has an associated weight that, after training, indicates its importance to that neuron. The output is given by two functions: input and activation. The input function often calculates the weighted sum of the input values, while the activation function is commonly the step, sigmoidal functions, among others. Figure 5 presents a schematic representation of a neuron. In our system, we chose to use the sigmoidal function because the output of the network has to be a float activation value instead of the binary output of the step function. The formulae for computing the output of each neuron is given below:

$$output = \frac{1}{1 + e^{-\sum W_j * I_j}}$$

Where,

- j is an index for the neuron input and weight vectors, ranging between $[0, size - 1]$;
- W is the weight vector;
- I is the input vector.

Neuronal networks can be composed by sets with only one to several neurons, which are thus arranged in layers. Single layer networks are also called perceptrons. In our system, it's possible to use either perceptrons or multilayered networks as classifiers for the cinematographers. The choice will depend on the complexity of the scene input and the number of available cinematographers. Figures 6 and 7 illustrate how the perceptron and multi-layered neuronal networks are built in our system, showing the connection between input neurons and the input-hub. One can notice that all classifiers share the same input-hub, which is just a simple data-structure used for distributing the input vector, not a weighted neuron.

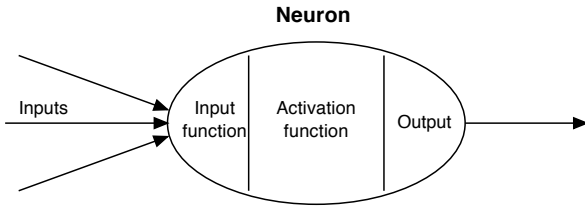


Figure 5: Schematics of a Neuron

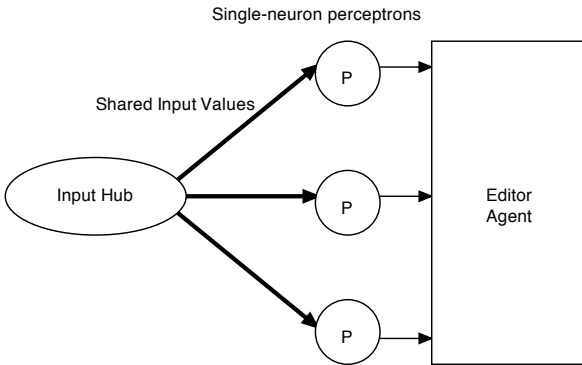


Figure 6: Editor with Perceptron Classifiers

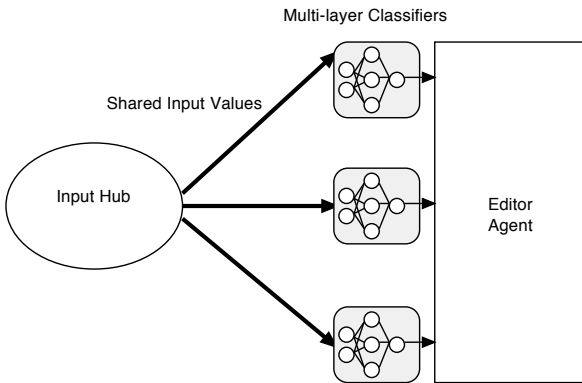


Figure 7: Editor with Multi-layer Classifiers

4.2 The Learning Mechanism

By modifying the weights associated to the inputs, one can teach each neuron the importance of each input value for its respective output. A set of know tuples of inputs together with the associated desired output is called a training set. To train a neuron network, one should feed its input with each example from the training set, comparing the calculated output with the desired one and feeding back the computed error to a training function, which adjusts the weights properly. Each time the training set is fed through the network is called a *training age*. For single neuron networks, the simple training rule given in the following formulae is applied to each classifier while for multi-layered networks the back propagation algorithm is used.

$$W_j = W_j + \alpha \times (\text{desired} - \text{current}) \times I_j$$

Where,

- α is the learning rate for the training mechanism;
- *desired* is the correct output for the training sample;

- *current* is the calculated output using current weights;

One can notice that the training function updates the weights of the neuron proportionally to the input (I_j), the output error ($\text{desired} - \text{current}$) and the training rate (α). It is important to balance between large training rates and the number of training ages. In section 6 we present experiments and analysis about these variables with a game prototype implemented exclusively for this work.

In our system, the training set is initially empty, and is populated as the user chooses favorite shots while watching gameplay. Each time he informs the system of a preferred shot, a snapshot of the current input is taken and the desired value for each cinematographer is set: 1 for the chosen one and 0 for all the others. Each sample consisting of the input values and the desired output is then added to the respective cinematographer agent's training set. At each time a new sample is included, the weights are reset and a new training is performed for all the classifiers. To keep the training cost low throughout the simulation, we use the minimum necessary number of training ages, which will be better explained in Section 6.

4.3 The Learning Interface

The proposed system interface used for training the editor agent in our work, shown in Figure 8, is composed of several viewports to show at the same time:

- All the available shots (cinematographers) - seen in the small viewports on the right;
- The current choice of the editor agent - lower-right viewport;
- The main screen for the player during training - bigger viewport.

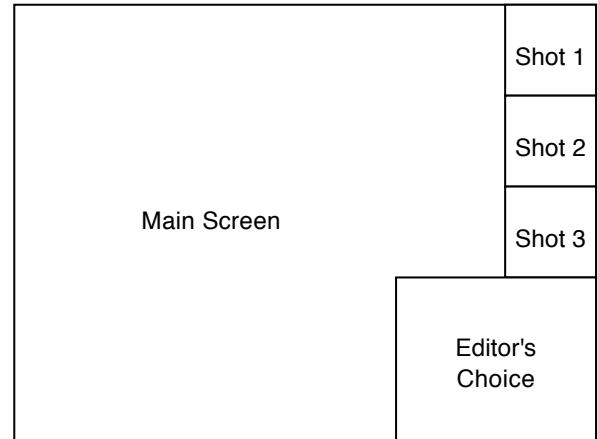


Figure 8: Learning Interface with 3 Cinematographers

When the system starts, the training set is empty and will be filled by the user (or a spectator beside him) as he plays. At each time the game developer finds the simulation state in a representative situation for a specific shot, he should press the numeric key of that cinematographer. This will fire the mechanism that feeds each cinematographer's training set by taking the input snapshot, adding the respective desired output. After the new samples are included in each set, a training age is performed for all cinematographers. In the next sections we show the prototype, exclusively developed to test the system, and the results of experiments performed to verify the accuracy of the learning mechanism.

5 Prototype: Race Game

We developed a game prototype to illustrate the use of our editor agent. It consists of a simple race game where the goal is to make the editor agent learn to use different camera shots to increase the dramatic appeal of action situations such as jumps and rocks crossing the road. The game takes place on a small modeled island with

a race track around it. There are several bumps through the track that make the car jump. In addition, there is a volcano that throws rocks over the track in the back part of the circuit. The player controls the car as usual, using the directional keys, while the editor agent training controls are the number keys corresponding to the available shots. Figure 9 shows a high view of the island, showing the race track and the volcano.

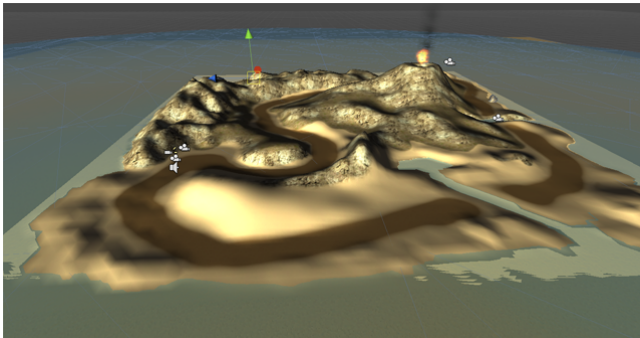


Figure 9: Screenshot of the Island and the Race Track

5.0.1 Editor Agent

The editor agent for the prototype includes three shot options and the respective classifiers, which are single neuron perceptrons. This example does not need multi-layer networks because the functions we are trying to teach for each classifier are linear. The goal is to use a chasing camera for normal racing situations, and different ones for jumps and for when the car is near the falling rocks. One can notice that it would be easy to implement a rule based system to achieve the same goals, but our intelligent editor is flexible to learn any other set of rules with the same simple teaching interface, and also, we target it for non technical users, who would find it difficult to specify logical rules.

The shot library comprises of three cinematographer agents, referenced by the editor and implemented by directly manipulating the virtual camera. Figure 10 shows a screenshot of the learning interface of the prototype game, while the description of the three available cinematographers is given below.

- *Chasing Camera* - follows the car from behind - default;
- *Front Camera* - keeps ahead of the car, pointing back to it - for jumping situations;
- *High-view Camera* - fixed at the volcano top, and its orientation follows the car - for the volcano area.



Figure 10: Prototype game interface, showing the available shots

The input for the editor, which is illustrated in Figure 11, comprises of only three values:

- Boolean value (0 or 1) saying if the car is touching the ground;
- Normalized values for the X and Z coordinates of the car, based on the limits and center of the island.

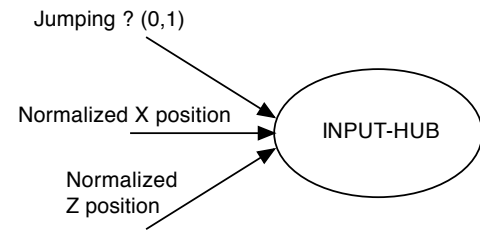


Figure 11: Input-hub for the prototype

6 Experiment Results

We were specially interested in experimenting with the learning mechanism to verify its performance with only a minimal number of training ages and a very small training set, so the user would not notice the computational penalty of this task. By classification performance we mean both classification quality, the correctness rate that the classifier generates against the expected output; and accuracy, characterized by small dispersion values over the test instances. We expected to confirm that the size of the training set could be small, so the user would not have to choose the adequate camera for each situation too many times. The less the user has to inform/teach the system, more simple it becomes to operate. But before actually performing interactive tests with real users and small training sets we did some experiments to find what was the ideal number of training ages and the influence of the learning rates of the neurons over the classification performance. This section describes both parts of the experiment: first, we analyze the influence of learning rates and number of ages over performance; second, we show the results of tests with real users.

6.1 Learning Rate and Ages

For both parts of the experiment we used an auxiliary sample set for validation purposes which consisted of 300 collected samples, with all three expected values for each cinematographer classifier. This training set was carefully built with actual application data to represent all possible situations during gameplay. The first experiment aimed to check the influence of the learning rate over accuracy. Using this validation sample set, we performed *Ten Fold Cross-validation* tests for each learning rate value, and measured the average output correctness and standard deviation for each instance of the test. *Ten Fold Cross-validation* works by shuffling the sample set and splitting it in ten parts. Then, each one of these 10% subsets are used to validate the system, which is trained with the remaining 90%.

We performed this test with the learning rates ranging from 0.05 to 0.5 and noticed that the learning rate does not influence much on the performance, given that the system converges to a trained state. The only influence is noticed in the training speed, meaning that small values take too long to converge, while higher values are too sensible to errors in the training set. Given this, we chose to run the remaining experiments with a learning rate of 10%.

With the second experiment, we wanted to find appropriate maximum value for the number of ages to be performed at each training of the networks, big enough to achieve good accuracy even if the system does not converge and still small enough to avoid high computational intensity. We ran the same kind of *Ten Fold Cross-validation* tests, this time fixing the learning rate in 10% and varying the maximum number of ages from 1 to 50. The results for this tests are presented in a chart in Figure 12. The horizontal axis represents the number of ages, while the vertical axis shows the measured classification performance (100% is full correctness).

One can notice from the chart that the number of ages has a strong influence over the trained network performance. Only with 30 or more ages has the accuracy converged to 100%, with a measured standard deviation of around 3%. Fortunately, even when executing 50 training ages with the small training sets expected to be used in

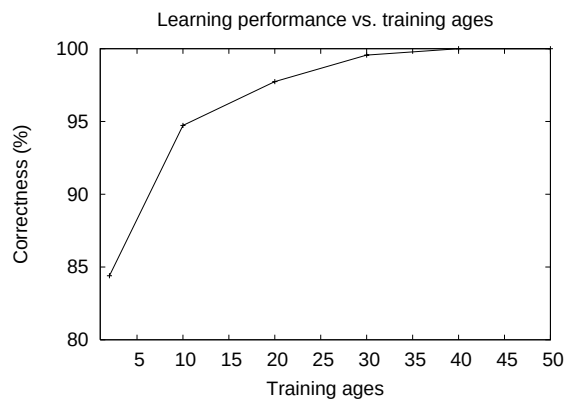


Figure 12: Classifier performance with varying ages number

actual interactive training, the computational penalty is almost imperceptible, so this was the number used in the final experiments. We also checked how robust the prototype classifiers were, by introducing random error on the validation sample set in the range of 5% to 25%. Figure 13 shows the influence of this errors over the different number of training ages. The axis represent the same variables as the previous test results, and the different lines represent the different error ranges.

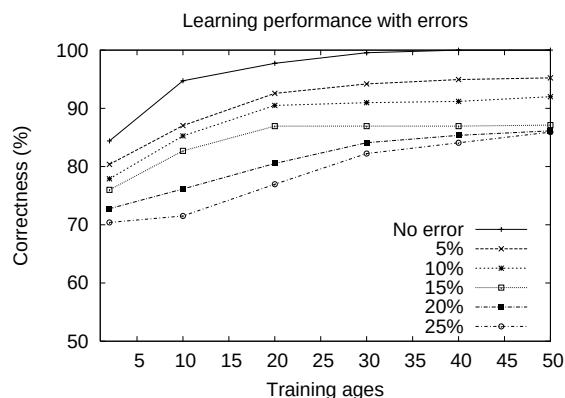


Figure 13: Classifier performance in the presence of errors in the training set

One can notice that the presence of error in the training set is particularly influent with the smaller number of ages. By training with 50 ages, even with the presence of 25% incorrect samples, the accuracy is still high, reaching more than 85%. This robustness is one of the reasons neuronal networks are a good choice for such system. Following, we analyze the results of interactive training experiments with real users creating very small training sets.

6.2 Interactive Experiments

For the interactive experiments, we wanted to check if the learning mechanism was robust enough to perform well even with a very small training set, which is mandatory for the system to be considered user friendly from an usability point of view. The tests consisted of a user watching live gameplay and choosing the correct shot (cinematographer) during the simulation. We only instructed the users to choose the first shot (chase camera) for normal situations, the second one (front camera) for jumping moments, and the last one (volcano shot) when the car was near that part of the island.

Since this prototype editor agent is composed of three different cinematographers, we used a minimal training set of at least three samples, one for each available shot. From this minimum value, the test users increased the number of samples by 3 for each new test category, always adding one more sample to favor each one of the cinematographers. To better validate the results through standard deviation, the users were asked to "teach" the system several times by starting from scratch for each desired training set size. Table 1 shows the computed results for this experiment with the training set size ranging from 3 to 12 samples, while Figure 14 shows the performance of the editor agent in a line chart.

Table 1: Performance and deviation with small training sets

Samples	3	6	9	12
Accuracy	83,134	95,945	98,894	99,447
Std. Dev.	17,020	3,938	0,232	0,463

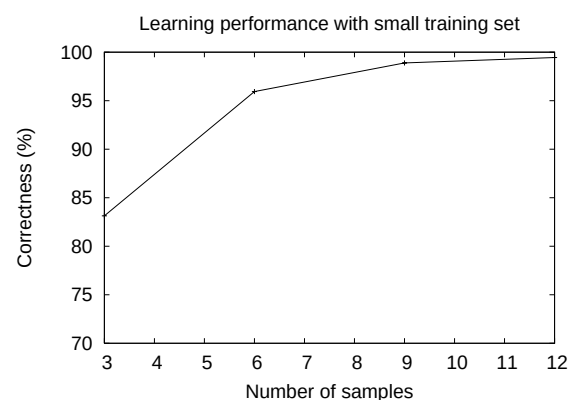


Figure 14: Classifier performance with small training sets

One can see that, apart from the trivial case of only one sample for each shot, the accuracy results are statistically very satisfactory. It is also important to say that again, apart from this trivial training set with only 3 samples, the system has always reached 100% correctness over the validation set in several test instances. The average accuracy was a little under this value probably because of some mistakes by the user, who failed to choose good samples, which is expected to happen even in production situations. The test results were very important because they showed us strong evidences that this mechanism is good enough for a real case production scenario, where such intelligent system becomes very handful, specially because non technical users are not expected to learn how to specify complex logical rules such as an equivalent decision tree.

7 Conclusion

In this paper we presented an editor agent for intelligent game cinematography that features a learn by example mechanism with very good accuracy, does not impose big performance penalties and is suitable to be included in a production pipeline, given its very easy user interface. This editor agent is part of a cinematography architecture that has been designed and developed for games and other interactive applications, even those where there is no prior knowledge of the action events that will take place in the 3D environment. Although designed to work in games, we think our system is suitable for other types of interactive 3D visualization applications as well and we plan to experiment it with those scenarios. In our opinion, the major contributions of this work are the robust learn by example mechanism, the independency of complex domain translation of scene data and the intuitive user interface.

The experiments made with the neuronal networks classifiers provided very important results that have lead to some modifications on the implemented ideas. For instance, we originally thought it

would be possible to run smaller number of training ages, but the results showed us that this number is much more important than the learning rate and even the training set size. The final tests with real users showed us very important evidence that the system is viable to a production scenario, given the near-ideal performance that was obtained with very small training sets.

Future work on this cinematography architecture include:

- Experimenting with more complex environments, input sets and shot types, which will need multi-layer neural network classifiers;
- Design a learning mechanism to interactively specify cinematographer agent's constraints;
- Integrate the system with a distributed architecture targeting spectators of massive online games.

References

- AMERSON, D., KIME, S., AND YOUNG, R. M. 2005. Real-time cinematic camera control for interactive narratives. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACM, New York, NY, USA, 369–369.
- CHRISTIANSON, D. B., ANDERSON, S. E., WEI HE, L., SALESIN, D., WELD, D. S., AND COHEN, M. F. 1996. Declarative camera control for automatic cinematography. In *AAAI/IAAI, Vol. 1*, 148–155.
- COURTY, N., LAMARCHE, F., DONIKIAN, S., AND RIC MARCH. 2003. A cinematography system for virtual storytelling. In *Int. Conf. on Virtual Storytelling, ICVS'03*, Springer, 30–34.
- DRUCKER, S. M., AND ZELTZER, D. 1995. Camdroid: a system for implementing intelligent camera control. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 139–144.
- DRUCKER, S., HE, L., COHEN, M., WONG, C., AND GUPTA, A., 2002. Spectator games: A new entertainment modality for networked multiplayer games. <http://research.microsoft.com/~sdrucker/papers/spectator.pdf>.
- ELSON, D. K., AND RIEDL, M. O. 2007. A lightweight intelligent virtual cinematography system for machinima production. In *AIIDE*, The AAAI Press, J. Schaeffer and M. Mateas, Eds., 8–13.
- GUERRA, F. W., 2008. Engenharia de estórias: um estudo sobre a geracao e narrao automatica de estórias.
- HALPER, N., AND MASUCH, M. 2003. Action summary for computer games: Extracting action for spectator modes and summaries. In *Proc. of 2nd Int'l Conf. Application and Development of Computer Games*, 124–132.
- HAWKINS, B. 2004. *Real-Time Cinematography for Games (Game Development Series)*. Charles River Media, Inc., Rockland, MA, USA.
- HERMANN, R., AND CELES, W. 2005. Posicionamento automatico de camaras em ambientes virtuais dinamicos. In *Proceedings of IV workshop on games and digital entertainment of the Brazilian Symposium on Computer Games and Digital Entertainment*.
- HORNUNG, E., LAKEMEYER, G., AND TROGEMANN, G. 2003. Autonomous real-time camera agents in interactive narratives and games. In *Proceedings of the IVA 2003: 4th International Working Conference on Intelligent Virtual Agents*, 15.-17.9.2003, Isee, Germany, *Lecture Notes in Computer Science* 2792, Springer, 236–243.
- KNEAFSEY, J., AND MCCABE, H. 2005. Camerabots: Cinematography for games with non-player characters as camera operators. In *DIGRA Conf.*
- MORRIS, D., KELLAND, M., AND LLOYD, D. 2005. *Machinima: Making Animated Movies in 3D Virtual Environments*. Muska & Lipman/Premier-Trade.
- PINHANEZ, C. S. 1999. *Representation and recognition of action in interactive spaces*. PhD thesis, Massachusetts Institute of Technology. Adviser-Aaron F. Bobick.
- POZZER, C. T. 2005. *Um sistema para geracao, interacao e visualizacao 3D de historias para TV interativa*. PhD thesis, Pontificia Universidade Catolica - RJ.
- TOMLINSON, B. 2000. Expressive autonomous cinematography for interactive virtual environments. In *In Proceedings of the Fourth International Conference on Autonomous Agents*, ACM Press, 317–324.
- WEI HE, L., COHEN, M. F., AND SALESIN, D. H. 1996. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 217–224.