Computers & Graphics 35 (2011) 1063–1069

Contents lists available at SciVerse ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Realistic modeling of spectator behavior for soccer videogames with CUDA ☆

Erdal Yılmaz [a], Eray Molla [b], Cansın Yıldız [c], Veysi İşler [d,*]

[a] Informatics Institute, Middle East Technical University, 06531 Ankara, Turkey
[b] École Polytechnique Fédérale de Lausanne, Department of Computer Science, Lausanne, Switzerland
[c] Department of Computer Engineering, Bilkent University, 06535 Ankara, Turkey
[d] Department of Computer Engineering, Middle East Technical University, 06531 Ankara, Turkey

ARTICLE INFO

Article history:
Received 3 April 2011
Received in revised form
5 October 2011
Accepted 6 October 2011
Available online 3 November 2011

Keywords:
Soccer game
Spectator behavior
Crowd simulation
CUDA

ABSTRACT

Soccer has always been one of the most popular videogame genres. When designing a soccer game, designers tend to focus on the game field and game play due to the limited computational resources, and thus the modelling of virtual spectators is paid less attention. In this study we present a novel approach to the modeling of spectator behavior, which treats each spectator as a unique individual. We also propose an independent software layer for sport-based games that simply obtains the game status from the game engine via a simple messaging protocol and computes the spectator behavior accordingly. The result is returned to the game engine, to be used in the animation and rendering of the spectators. Additionally, we offer a customizable spectator knowledge base with well structured XML to minimize coding efforts, while generating individualized behavior. The employed AI is based on fuzzy inference. In order to overcome additional demand for computing realistic spectator behavior, we use GPU parallel computing with CUDA.

1. Introduction

Soccer, due to its very nature, has always attracted millions of fans, not only in the real world, but also in the virtual universe. Spectators play a highly significant role during real soccer matches, and it has been shown that the support of an excited crowd of fans can offer significant advantages to a home team [1]; indeed the term "12th man" is often used to highlight the importance of fan support. Soccer game developers, and indeed developers of other sport-based games, have always taken the presence of spectators into account, visualizing them either as static bitmap images or as dynamic geometric models since early 1980s. In Fig. 1, two screenshots from the online soccer game "I Can Football" are provided to show the representation of virtual spectators in a typical soccer game; the spectators visualized in this game are similar to those found in most popular soccer games. Significant progress has been achieved in the simulation of spectators in recent years; however, the visual and behavioral realism of the spectators still lags far behind the realism of the players on the pitch, as can be seen in Fig. 1. The reasons for this are twofold: first, the prioritized use of limited computational resources for the game play; second, the prioritization of the game field, which takes precedence over action in the background. The rest of the paper discusses that the first issue is no more a challenge, if GPU resources are used efficiently. Regarding the necessity of realistic background, there are several counter-arguments such as simulating the camera shots to show spectators and generating life-like tribune motion.

In this study, we propose an architecture to process spectators in a modular way. The game engine can request spectator processing at any instance to reflect any change in the behavior of the spectators. This processing is performed by a separate layer, which we refer to as the Spectator Behavior Engine (SBE). SBE computes the behavior of the spectators according to the status of the game, which is provided by the game engine. Finally, SBE updates and sends the spectator behavior to the game engine, which in turn animates and renders the spectators. SBE performs all of these computations on the GPU, where each spectator is considered as an intelligent individual, based on fuzzy inference. SBE can also be used by non-sport game genres that contain a huge number of spectators. A typical example is gladiator-style events in arenas. Another example might be music and dance games that simulate stadium concerts. Crowd-based simulation applications and serious games can also utilize the proposed method.

This study is based on an extensive use of fuzzy inference on the GPU for the modeling of realistic soccer spectator behaviors via a user-defined XML script. We have observed that soccer games generally treat spectators as a whole rather than as individuals, which results in repeated and user-predictable spectator gestures

☆ This article was recommended for publication by O. Staadt.

* Corresponding author. Fax: +903122105564.
E-mail address: isler@ceng.metu.edu.tr (V. İşler).

0097-8493/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2011.10.001

Fig. 1. Comparison of the visuals of spectators and players (courtesy of Sobee).

that may annoy the player. Although fans within a stadium usually make similar moves, it is not very impressive to see exact clones, and for this reason our intention was to produce behavioral differences in the crowd, as would be found at real soccer matches. When each spectator is modeled as a unique individual, the reactions of the spectators can be produced based on the psychology of spectators resulting from various factors, such as the characteristics of the game being played, the reputation of the players, the current score, cultural issues, etc.

In this study, GPU parallel processing with Nvidia CUDA (Compute Unified Device Architecture) is used to overcome the additional computational power required to create realistic and very large numbers of spectators. Using CUDA, it is possible to treat each spectator as a unique individual on a GPU parallel architecture. Thus, CUDA helps us to deal with this problem with minimal coding effort and using negligible CPU resource.

The rest of the paper is organized as follows. The next section summarizes CUDA. This is followed by Spectator Behavior Modeling with Fuzzy Logic Control (FLC), and includes three subsections: the first subsection explains the Fuzzy inference mechanism; the XML structure that is used for the fuzzy knowledge base is covered in the next subsection; while the final subsection gives several fuzzy variables and fuzzy sets as examples in order to better explain the AI process of SBE. Section 3 discusses implementation and summarizes performance results and gives the simulation results of a Mexican wave implementation using SBE. The performance results section includes comparisons of both the performance and the visual output between the GPU parallel implementation and CPU serial implementation. Finally, Section 4 presents conclusions of this work.

2. Related work

2.1. GPU parallel processing with Nvidia CUDA

Nvidia introduced CUDA as a parallel computing architecture that allows programmers to use C language while programming GPUs, starting from the GeForce 8 series. State-of-the-art Nvidia GPUs provide many-cores (e.g. the gtx580 has 512 processor cores), known as stream processors. Each stream processor has special hardware thread manager that is capable of running multi-threads. Considering the huge number of stream processors and the multi-thread capability, the total number of concurrent threads can easily reach several thousands. Nvidia calls this architecture SIMT (single-instruction, multiple-thread), the details of which can be found in the CUDA Programming Guide [2]. This many threads can help deal with computing intensive parallelizable problems. Since the release of CUDA technology, many studies have reported speed-ups of order of magnitude, and even two orders of magnitude, in various applications [3–8], and an increase in these reported computational performances may be possible with the use of multi-GPUs in a cluster.

Although state-of-the-art GPUs offer several teraflops of processing power, special care must be given to certain issues [2]. First and the most important is to convert the problem into data-parallel structure. This issue is quite applicable to massive crowd simulation domain since it is possible to assign each character to a single thread. The second issue is to use the bandwidth efficiently. This issue both covers the data transfer between the CPU/GPU and the memory operations on the GPU. The final issue is the optimization via instruction usage. The techniques that we employed to address the mentioned issues are given in the implementation section.

Using the GPU instead of the CPU, some of the CPU resources that are critical for the game engine may be released. As discussed in the results section, GPU parallel processing causes no practical delays in the game loop and rendering, since the fuzzy inference of tens of thousands of spectators is completed in only a few milliseconds. This is clearly beyond the computational performance offered by the CPU as shown in Section 3.1.

Using GPU parallel processing in crowd simulation is not limited to real-time applications. A well-known Massive software, which is mainly used by motion picture industry uses this technology to speedup offline rendering process [9].

Some other researchers have utilized non-GPU parallel architectures in crowd simulation as in [10–12].

2.2. Behavior modeling and fuzzy inference

Behavior modeling is one of the most extensively studied areas in crowd simulation. There have been different approaches and studies into modeling how virtual characters “make decisions” and behave in virtual worlds. In such worlds, characters simply perceive the surrounding world and react accordingly. In a soccer game, since all the spectators in the stadium perceive the same event(s), other parameters must be employed to generate behavioral differences, as in the real life. Chittaro and Serra have demonstrated autonomous virtual characters with behavioral differences [13], bringing personality to virtual characters and employing probabilistic influence on behavior selection to produce distinct behaviors. Badler et al. also used personality in EMOTE (Expressive Motion Engine), which influences the character perception and action [14]. The basis of the behavioral modeling used in this study was introduced by Bécheiraz and Thalmann [15], who used perception to generate emotion, and then used both perception and emotion to invoke a certain behavior to a corresponding action. Ayesh et al. have tried fuzzy individual model (FIM) for the behavioral modeling of virtual individuals [16], while also using perceptions to update emotions that trigger different behaviors. Rudomin and Millan used XML scripting to specify behaviors, and employed a Finite State Machine as a processing method [17], further developing this study by

implementing probabilistic FSMs, hierarchical FSMs and layered FSMs to produce non-deterministic results [18].

In this study, FLC is used to model spectator reactions. Fuzzy sets were first introduced by Lotfi A. Zadeh in 1965 [19], and over the last 40 years, fuzzy logic has been used extensively in many application areas, including crowd simulation [8,20] and agent behavior modeling [16,21]. Our reasons for choosing fuzzy logic for behavioral modeling are:

- Ability to produce more realistic and less predictable reactions.
- Ability to capture a real human knowledge base and use it extensively with minimal coding.
- Use of an AI technique that is more suitable to model complex spectator behavior.

2.2.1. Fuzzy inference

In this study we have used the fuzzy controller approach proposed by Mamdani [22]. This approach simply performs a processing using scalar input values and fuzzy control elements (fuzzy variables, fuzzy sets and fuzzy rules), and produces an output represented by a scalar value. These fuzzy control elements are organized in a knowledge base that is mostly based on the experience of domain experts. One of the most important tasks in the development of an FLC system is the design of a knowledge base. For the purpose of this study, we have used an XML-based solution, which is discussed in the next subsection.

A Mamdani-style fuzzy inference [23] is a four-step process: in the first step, known as fuzzification, scalar values are used to calculate degrees of membership of corresponding fuzzy set/sets within a fuzzy variable. Input values can either be static or dynamic throughout the game; the refresh rate of dynamic variables can also be different. In a soccer game "Game Field Perception," "Neighborhood Perception," "Personality" and "Mood" can be used as input variables. It is possible to expand this list through the addition of more parameters.

Rule evaluation is the second step in fuzzy inference. In this step, the output of the first step is used to produce the value of output rule using two input rules and a fuzzy operator. The corresponding fuzzy set in the output rule is clipped to produce a new shape using an output scalar value. This step is followed by rule aggregation, which combines clipped fuzzy sets to generate a new geometric shape. Finally, the fuzzy inference is completed with the defuzzification step, in which the aggregated geometric shape is converted into a scalar value, which is used as the output of the fuzzy inference.

In this work, we exploited the fuzzy inference CUDA kernels that we have developed in our previous work where the details of the implementation can be found [8]. We improved the previous approach to make it more flexible and GPU friendly as discussed in the following sections.

2.2.2. XML structure

As mentioned previously, one of the main goals of this study is to reduce the workload of soccer game developers in incorporating intelligent virtual spectators into their games. When fuzzy logic is used for modeling spectator behavior, a highly detailed human knowledge base and fuzzy rules are required to generate life-like outputs. For this purpose an XML script is provided to the game developers, either to construct the knowledge base from scratch or to customize an existing knowledge base. This XML scripting is partly inspired by the legacy FCL defined by the International Electrotechnical Commission (IEC) [24], and by the Fuzzy Markup Language (FML) introduced by Acampora and Loia [25]. We built our own XML scripting (hereafter referred to as Spectator Behavior Modeling Markup Language – SBML) using FML as a template in order to maintain a similar terminology to previous studies. Since SBML is intended to be used solely by sports game developers, our main goals have been simplicity and performance.

Fig. 2 gives the general structure of SBML as a tree view. In this structure, "FuzzyControl" is the outermost tag, with sub-tags entitled "KnowledgeBase" and "RuleBase." The order of "FuzzyVariable," "FuzzySet," "RuleBlock" and "Rule" can also be seen in this hierarchy.

The "KnowledgeBase" contains a single, or a set of, "FuzzyVariable/s", which corresponds to a term used in fuzzy inference operation. This fuzzy element, which may be composed of a single or several "FuzzySet/s", has "Name" and "Description" attributes. In Listing 1, we give an example of a fuzzy variable, "Mood", which contains fuzzy sets, "Calm," "Stressed" and "Angry." In SBML, the shapes of the fuzzy sets are given within the "Coordinates" tag. In this listing "Calm" and "Angry" are defined as trapezoids, and "Stressed" is defined as a triangle.

The "RuleBase" section covers "RuleBlock(s)," with each "RuleBlock" containing only two input terms and one output term. This is preferred for the sake of easy implementation. The rules within the "RuleBlock" are evaluated using only fuzzy "AND" and fuzzy "OR" operators. We have also developed a tool with

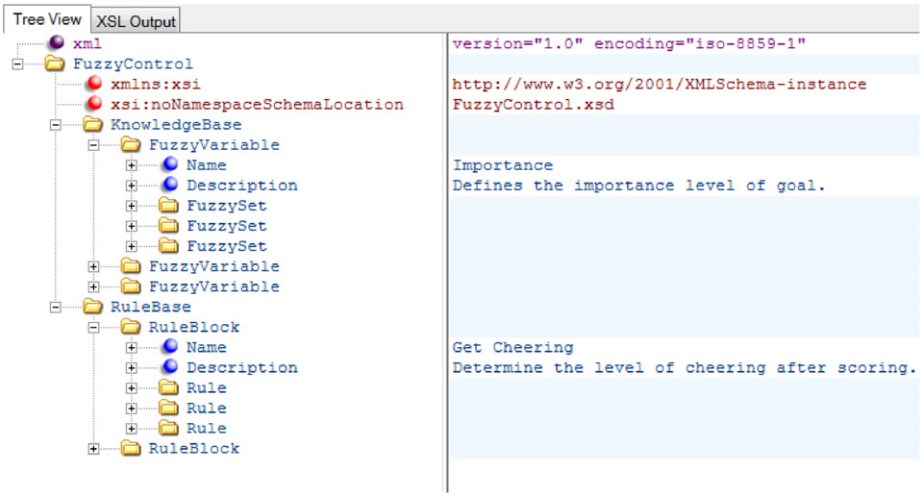

Fig. 2. Tree view of SBML elements.

```
- <FuzzyVariable>
    <Name>Mood</Name>
    <Description>Defines the mood of the spectator.</Description>
  - <FuzzySet>
      <Name>Calm</Name>
      <Description>Spectator is calm</Description>
      <Coordinates>0,1,30,1,50,0</Coordinates>
    </FuzzySet>
  - <FuzzySet>
      <Name>Stressed</Name>
      <Description>Spectator is stressed</Description>
      <Coordinates>35,0,50,1,65,0</Coordinates>
    </FuzzySet>
  - <FuzzySet>
      <Name>Angry</Name>
      <Description>Spectator is angry</Description>
      <Coordinates>60,0,80,1,100,1</Coordinates>
    </FuzzySet>
  </FuzzyVariable>
```

Listing 1. Fuzzy variable and fuzzy sets. .

a user friendly GUI to edit the knowledge base, using Eclipse environment.

2.2.3. *Examples of fuzzy variables*

In this subsection we highlight some of the fuzzy variables that can be used in fuzzy inference. As mentioned before, it is quite easy to define new fuzzy elements, since SBML offers a highly flexible environment and tools. The following examples are only a small part of possible fuzzy variables that can be considered within the fuzzy inference process of SBE.

- **Game_Importance:** This fuzzy variable defines the importance of the game, which is quite significant in determining spectator behavior. Needless to say, a friendly match during the off-season is not as important as a championship final.
- **Attitude_Against_Opponent:** This variable can be used to evaluate the general attitude towards the opposing team. The level of hostility or friendliness towards an opposing team is sometimes hard to predict; however, there are certain teams against which a hostile attitude is guaranteed.
- **Current_Attitude:** The performance of the team during the season may affect the behavior of its spectators. If the team suffers a series of bad results, the tolerance and attitude of the supporters may change. Spectators certainly tend to protest their team during the game if the game-play or score do not meet their expectations.
- **Spectacular_Move:** A spectacular action, such as an incredible goal, may draw applause from whole stadium, including the opposing fans.
- **Attitude_Against_Referee:** This fuzzy variable is used to determine the reaction against a referee's decision. Spectators usually protest the decision in order to influence the referee, which can be important, especially in critical decisions.

3. Implementation

In this study, a software module was designed to compute spectator reactions using an agent-based approach. Each spectator was considered as an independent individual. This module is called as Spectator Behavior Engine (SBE) and designed to run on the GPU using the NVidia CUDA technology. Fig. 3 shows the interaction of SBE with an imaginary game engine.

First, the game engine initializes SBE and transfers individual spectator attributes to the device via the "cudaMemcpy" function. The fuzzy knowledge-base is also transferred to the device's cached constant memory to improve computational performance.

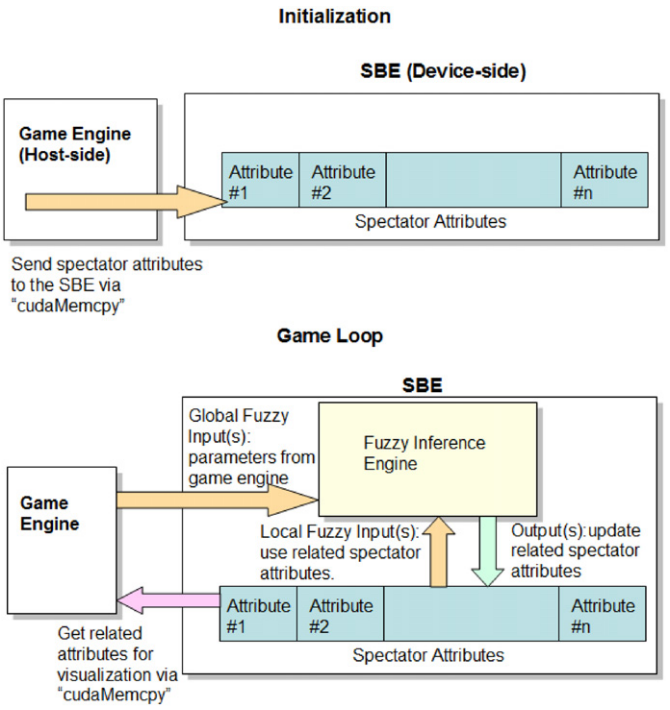

Fig. 3. SBE architecture.

To be processed by the SBE, a spectator must have several distinct attributes, which also provide the basis of behavioral variety. These attributes are related to behavioral modeling and transferred to the GPU during the initialization of the application. Thus the spectator related non-graphical attributes are stored on the device memory during run-time. This is computationally more efficient, as it does not cause an increase in bandwidth when transferring these attributes. The fuzzy inference engine uses these attributes as spectator dependent local fuzzy inputs as shown in Fig. 3.

As mentioned in the introduction section, each spectator is treated as a unique individual to make variety among spectators. As stated in [26], variety is an important aspect of crowd simulation. We initialize the game with individual-based unique attributes and update them separately as shown in Fig. 3. The fuzzy inference produces distinct scalar outputs when we use unique attributes as inputs. Thus, during the game, updates of SBE still preserve the initial attribute differences of the spectators.

The game engine can request spectator processing anytime to reflect any change in the individual behavior of the spectators. The fuzzy inference engine uses the inputs provided by the game engine as global fuzzy inputs. The fuzzy inference engine generates scalar outputs to reflect any spectator change of behavior, using global fuzzy inputs (from game engine) and spectator-dependent inputs (using related attributes of the spectator). The generated output(s) updates the respective spectator attribute(s). The fuzzy inference engine is capable of running nested inferences, which means the output of a fuzzy inference can be used as an input for the next fuzzy inference. Thus, it is possible to run several dependent/independent inferences in a single run.

Finally, whenever required, the game engine puts the updated spectator outputs to the host-side to animate and render the spectators. Therefore, the game engine has all the control in this architecture and SBE has no direct connection to the game field, it only produces outputs. This approach provides flexibility, since any application that processes spectators can use SBE with little effort. Additionally, there is no need to update spectators in each frame and return the results immediately.

3.1. Performance tests

As previously mentioned, computational and rendering resources are very valuable for achieving life-like visualization of the game field. Minimal resources must be used to process spectators at the background. In this performance setup, it was assumed that several fuzzy inferences were enough to update spectator behaviors. Therefore a two minutes test (120,000 ms) was run using different spectator populations. In this test the game engine was requested to update spectator behaviors in different intervals. Depending on the call parameters, the related spectator attributes were updated by SBE using fuzzy inference engine (1–10 inferences per request). Finally, the related attributes were copied to the host, whenever needed. The size of the related attribute(s) was assumed 32 bits. Fig. 4 shows the results of this performance test for 65,536 spectators using NVidia GTX 295 GPU. Fig. 4(a) indicates that memory copy operations occupied only 7% of the total GPU processing time. This performance was achieved by getting updated results whenever required, not in every frame. Fig. 4(b) shows that during two-minute simulation period, SBE was called 120 times (at different periods) and the attributes were copied to the host 47 times (at irregular intervals). Additionally, the quantity of the fuzzy inferences changed at each SBE call (Fig. 4(b); "fuzzyInferenceTest" bar size is different in each run). This figure indicates that SBE processing completed in 103.77 ms (96.11 ms for fuzzy inferences, 7.66 ms for memory copy from device to host). Since the total simulation time was 120,000 ms, this processing time corresponded to only 0.09% of the GPU time, which is negligible in whole process chain. The same test was also run on a single CPU core to show the benefit of running SBE on the GPU. Table 1 gives the CPU and GPU performances. For the population of 65,536 spectators, it took 16% of the single core resource to run the same simulation on the CPU, an unacceptable value for sports-based video games. One of the most important issues regarding the use of SBE is the update frequency. The more update signal we produce the more GPU occupancy occurs. That is why the update frequency should be kept within reasonable rate. Table 1 shows that the SBE GPU occupancy rate is almost negligible if the average update frequency is 1 call/second or less even for the most crowded stadiums. It is certain that during a soccer videogame we usually do not need to update SBE that often.

3.2. Visualization of spectators

To demonstrate the functionality of SBE, we have populated a virtual stadium with nearly 50,000 spectators using the techniques described by Ciechomski et al. [27]. To visualize the fuzzy inference outputs several actions such as "cheer," "applause," "protest," "stand up," "sit down," "jump", etc. have been modeled using the MoCap data. Although our behavioral modeling may produce a rather rich set of outputs, we have to map the outputs to a limited number of animations in our crowd simulation tool.

As can be seen in Figs. 5 and 6, there are none of the repeated motion patterns or robotic actions that can usually be observed in soccer videogames. The most important achievement of this paper is the generation of a very large number of individually processed spectators for more realistic tribune scenes in soccer videogames. A comparison of Figs. 6 and 7 clearly highlights this, given the much greater variety of actions and reactions in Fig. 6 than in Fig. 7.

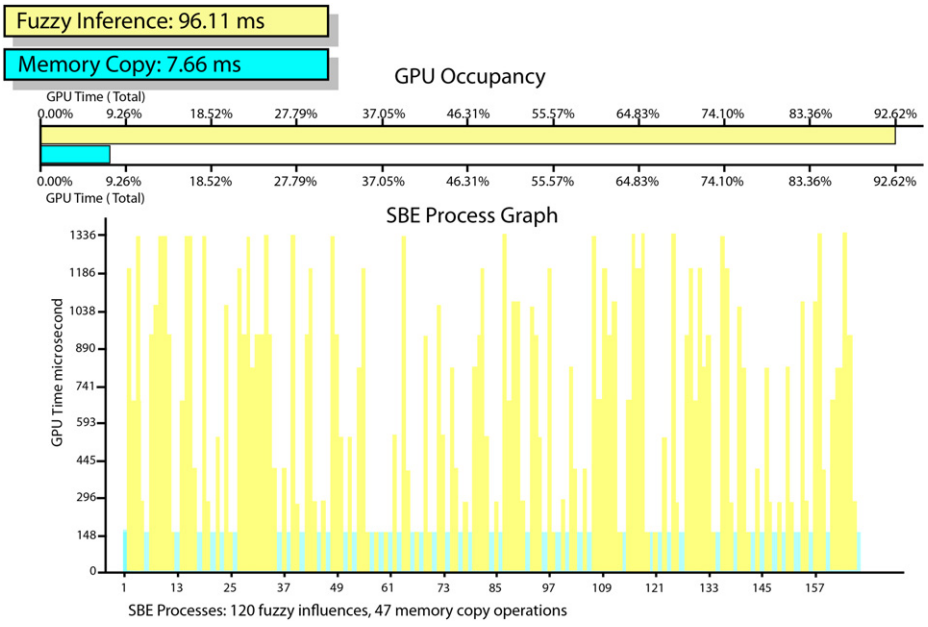

Fig. 4. Simulation of 65,536 spectators on the GTX 295 GPU. (a) GPU Occupancy (b) SBE Process Graph.

Table 1
CPU and GPU performances of the SBE.

Population	CPU (single CPU Core, Intel T 9550 @ 2.67 GHz)		GPU (GT 120 M)		GPU (GTX 295)	
	Process time (ms)	Processor occupancy (%)	Process time (ms)	Processor occupancy (%)	Process time (ms)	Processor occupancy (%)
16,384	4762	3.97	188.97	0.16	39.20	0.03
32,768	9470	7.89	335.74	0.28	61.81	0.05
65,536	18,992	15.82	651.32	0.54	103.77	0.09

Fig. 5. Individually processed fans celebrate a goal.

Fig. 6. Close-up view of individually processed fans celebrating a goal.

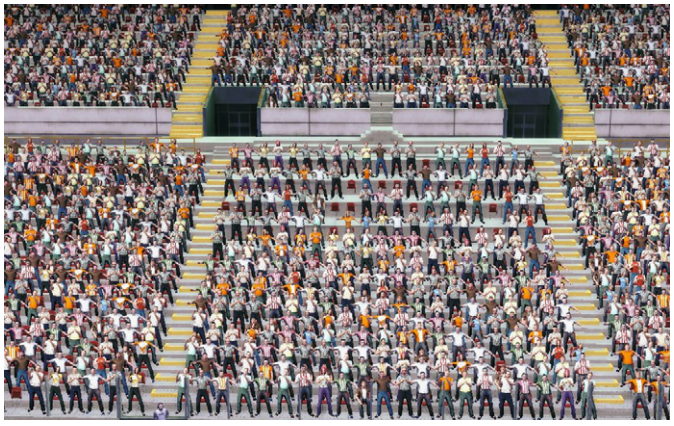

Fig. 7. Close-up view of fans supporting their team (courtesy of Sobee).

The proposed method requires high-end GPUs and parallel processing with CUDA, but offers a flexible mechanism for the handling of spectator behavior modeling. In this way, soccer videogames can benefit from the parallel computing power of the GPUs, however extra effort is needed to handle the additional rendering cost. To achieve this, various rendering techniques can be used to populate tribunes with large numbers of spectators [28–33].

3.3. A special case: Mexican wave visualization

In order to test the functionality of SBE we tried to simulate the Mexican wave, which is a collective human behavior where the spectators in the neighbor columns stand up while raising their arms up and then sit down again. This action triggers the neighbors to do the same. The stronger waves generally continue around the stadium several times. The Mexican wave phenomena was interpreted and quantified by Farkas et al. with a variant of models originally developed to describe cardiac tissue [34]. They examined several Mexican wave videos and the results they report are as follows:

- The wave usually rolls in a clockwise direction.
- The typical wave speed is 12 m/s (nearly 20 seats).
- The average width is 6–12 m (nearly 15 seats).
- The wave is initiated by no more than a few dozen people.

Table 2
Fuzzy knowledge-base for Mexican wave example.

Rule	
1	If ***Wave*** is ***Strong*** **OR** ***Mood*** is ***Normal*** then ***Involvement*** is ***Average***
2	If ***Wave*** is ***Strong*** **OR** ***Mood*** is ***Excited*** then ***Involvement*** is ***High***
3	If ***Wave*** is ***Weak*** **AND** ***Mood*** is ***Bored*** then ***Involvement*** is ***Low***

Fig. 8. A still image from the Mexican wave simulation.

Supplementary material related to this article can be found online at doi:10.1016/j.cag.2011.10.001.

Considering the facts provided in the literature we employed the fuzzy rules given in Table 2. Two inputs for fuzzification were used; the strength of the wave and the mood of the spectator.

The metrics provided by Farkas et al. were taken into account while designing fuzzy knowledge-base. The results illustrated in Fig. 8 reflect similarities to the above given metrics. The status of the neighbors was used to determine the first fuzzy input, which is the strength of the wave. We used "mood" as the second input parameter. As seen in this figure, some spectators do not join the wave, depending on their current mood. To simulate this specific soccer event, we assumed that all of the spectators support the same team. We also initialized spectator's mood in order to ensure that the excitement level is high. If there is nothing new to excite the spectators, SBE decreases the excitement level and the wave stops after a while.

4. Conclusions

We have presented an approach for the realistic modeling of spectator behavior in soccer videogames using the Nvidia CUDA parallel programming architecture. The results show that the high performance computing capability of the GPUs makes it possible

to process thousands of virtual spectators individually in real-time. The GPU processing and data transfer time to handle tens of thousands of spectators is almost negligible since it is no more than the 0.1% of the total GPU occupancy time. Thus, spectators in soccer videogames can be simulated more precisely, and with no regular motion patterns for almost no CPU cost. In this study, we have followed an XML-based approach to easily construct the knowledge base and the rule-base that is used by the fuzzy logic inference to be performed on the GPU. The usage of fuzzy logic provides behavioral variety and realism to the spectators, which is currently very limited in soccer videogames. A so-called Spectator Behavior Engine (SBE) computes the spectators' behavior in tribunes separately from the game engine to allow a flexible and pluggable software architecture; the SBE receives the status of the game at any instance and computes the behavior of the individual spectators accordingly, which is then sent back to the game engine. In this way, applications that contain huge number of spectators may incorporate this functionality in an easy and modular manner. Our initial efforts cover the details only for soccer videogames, but it is possible to extend this approach to other sports games and crowd simulation applications that contain large crowds.

References

[1] Boyko RH, Boyko AR, Boyko MG. Referee bias contributes to home advantage in English Premiership football. J. Sports Sci. 2007;25:1185–94.

[2] Nvidia. 2010; Nvidia CUDA C Programming guide version 3.2, 〈http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf〉.

[3] Weber O, Devir Y, Bronstein AM, Bronstein MM, Kimmel R. Parallel algorithms for approximation of distance maps on parametric surfaces. ACM Trans. Graphics 2008;27(4). (Presented at SIGGRAPH 2008).

[4] Silberstein M, Schuster A, Geiger D, Patney A, Owens JD. Efficient sum-product computations on GPUs using software-managed cache. In: Proceedings of the ACM ICS '08; 2008. p. 309–18.

[5] Nyland L, Harris M, Prins J. Fast N-Body Simulation with CUDA. GPU Gems 3. Addison-Wesley; 2008. p. 677–95.

[6] Le Grand S. Broad-Phase Collision Detection with CUDA. GPU Gems 3. Addison-Wesley; 2008. p. 697–721.

[7] Howes L, Thomas D. Efficient Random Number Generation and Application Using CUDA. *GPU Gems 3*. Addison-Wesley; 2008. p. 805–30.

[8] Yılmaz E, İşler V, Yardımcı YÇ. The virtual marathon: parallel computing supports crowd simulations. IEEE Comput. Graphics Appl. 2009;29(4):26–33.

[9] 〈http://www.massivesoftware.com/〉.

[10] Reynolds, C. Big fast crowds on PS3. In: Proceedings of the 2006 ACM SIGGRAPH symposium on videogames; 2006. p. 113–21.

[11] Steed A, Abou-Haidar R. Partitioning crowded virtual environments. In: VRST '03: Proceedings of the ACM symposium on virtual reality software and technology; 2003. p. 7–14.

[12] Berg J, Patil S, Seawall J, Manocha D, Lin M. Interactive navigation of individual agents in crowded environments. In: Proceedings of ACM symposium on interactive 3D graphics and games, 2008. p. 139–47.

[13] Chittaro L, Serra M. Behavioral programming of autonomous characters based on probabilistic automata and personality. Comput. Anim. Virtual Worlds 2004;15(3-4):319–26.

[14] Badler N, Allback J, Zhao L, Byun M. Representing and parameterizing agent behaviors. In: Proceedings of computer animation; 2002. p. 133–43.

[15] Bécheiraz P, Thalmann D. A behavioral animation system for autonomous actors personified by emotions. In: Proceedings of first workshop on embodied conversational characters '98; 1998. p. 57–65.

[16] Ayesh A, Stokes J, Edwards R. Fuzzy individual model (FIM) for realistic crowd simulation: preliminary results. In: Proceedings of FUZZ-IEEE conference '07; 2007. p. 1–5.

[17] Rudomín I, Millán E. XML scripting and images for specifying behavior of virtual characters and crowds. In: Proceedings of CASA '04; 2004. p. 121–8.

[18] Rudomín I, Millán E. Probabilistic, layered and hierarchical animated agents using XML. In: Proceedings of the 3rd international conference on computer graphics and interactive techniques in Australasia and South East Asia GRAPHITE '05; 2005. p. 113–6.

[19] Zadeh LA. Fuzzy sets. Inf. Control 1965;8(3):338–53.

[20] Chang J, Li T. Simulating virtual crowd with fuzzy logics and motion planning for shape template. In: Proceedings of IEEE conference on cybernetics and intelligent systems: 2008. p. 131–6.

[21] Zambetta F. Simulating sensory perception in 3D game characters. In: Proceedings of the 4th Australasian conference on interactive entertainment; 2007. Article no: 7.

[22] Mamdani EH. Application of fuzzy algorithms for control of simple dynamic plant. Proc. Inst. Electr. Eng. 1974:121–59.

[23] Byl P. Programming believable characters for computer games. Charles River Media; 2004.

[24] International Electrotechnical Commission (IEC) 1997. Fuzzy control programming IEC 1131-7; 1998. 〈http://www.fuzzytech.com/binaries/ieccd1.pdf〉.

[25] Acampora G, Loia V. Using FML and fuzzy technology in adaptive ambient intelligence environments. Int. J. Comput. Intell. Res. 2005;1(2):171–82.

[26] Maïm J, Yersin B, Pettré J, Thalmann D. YaQ: an architecture for real-time navigation and rendering of varied crowds. IEEE Comput. Graphics Appl. 2009;29(4):44–53.

[27] Ciechomski P, Schertenleib S, Maim J, Thalmann D. Reviving the Roman odeon of Aphrodisias: dynamic animation and variety control of crowds in virtual heritage. Virtual Syst. Multimedia 2005:601–10.

[28] Millán E, Rudomín I. Impostors and pseudo-instancing for GPU crowd rendering. In: Proceedings of the fourth international conference on computer graphics and interactive techniques in Australasia and Southeast Asia; 2006. p. 49–55.

[29] Millán E, Rudomín I. Impostors, pseudo-instancing and image maps for GPU crowd rendering. Int. J. Virtual Reality (IVJR) 2007;6(1):35–44.

[30] Millán E, Hernández B, Rudomin I. Large crowds of autonomous animated characters using fragment shaders and level of detail. In: Engel W, editor. Shader X5—advanced rendering techniques. Charles River Media; 2006. p. 501–10.

[31] Dobbyn S, Hamill J, O'Connor K, O'Sullivan C. Geopostors: a real-time geometry/impostor crowd rendering system. In: Proceedings of symposium on interactive 3D graphics and games '05; 2005. p. 95–102.

[32] Kavan L, Dobbyn S, Collins S, Zara J, O'Sullivan C. Polypostors: 2D polygonal impostors for 3D crowds. In: Proceedings of the symposium on interactive 3D graphics and games '08; 2008. p. 149–55.

[33] Dudash B. Animated crowd rendering. *GPU Gems 3*. Addison-Wesley; 2008. p. 39–52.

[34] Farkas I, Helbing D, Vicsek T. Social behavior: Mexican waves in an excitable medium. Nature 2002;419:131–2.