Improving Quality of Service in Videogames*

Inmaculada García
Technical University of Valencia
Camino de Vera, s/n
46022 - Valencia - Spain
ingarcia@dsic.upv.es

Ramón Mollá
Technical University of Valencia
Camino de Vera, s/n
46022 - Valencia - Spain
rmolla@dsic.upv.es

ABSTRACT
Every object in a videogame has different aspects to simulate as behavior, physics, graphics or sounds. Every aspect requires its own Quality of Service (QoS) levels (complexity and sampling). Traditional videogames follow a scheme of continuous coupled simulation. This rigid scheme does not allow to define a specific QoS sampling frequency. It is defined implicitly for the whole system and it is hardly dependent on the videogame, system load and the machine characteristics. A discrete simulation paradigm allows to define a private QoS criteria for each aspect of each object in the videogame. This discrete system allows a Smart System Degradation (SSD) and may redefine the objects QoS while the videogame is running. Objects may adapt its QoS based on the system status or on local performance degradation. This paper shows the results of the transformation of a continuous simulation videogame kernel (Fly3D) into a discrete one by integrating the discrete event simulator DESK.

Categories and Subject Descriptors
I.6.8 [**Simulation and modeling**]: Types of Simulation

General Terms
Simulation Applications on Games

Keywords
Simulation, Computer Games, Quality of Service

1. INTRODUCTION
The videogames QoS uses to be restricted to graphic parameters, but the QoS includes other videogame characteristics, such as behavior, physics or sampling frequency. The three main aspects in the videogame QoS are:

*Supported by grant GV04B-497 of the Valencia State Government, by grant TIC2002-04166-C03-01 of the Spanish MCYT, and by a STREP project IST-004363 of the 6th Framework Program of the European Union.

Family	UF	Simulation	Physics	Sample
Strategy	Low	High	Low	Chess, Warcraft
Arcade	High	Low	High	Quake, Doom
Flyght simulators	High	High	High	Comanche, ATP, Lock On
Other simulators	Low-Medium	High	Low	Simcity, The Sims
Graphic adventures	Medium	Medium	Low	Broken Sword, The Westerner
Rol	Low-Medium	Medium	Low	Knights of the Old Republic

Table 1: Videogames families.

- User Feedback (UF). Traditional Human Computer Interface (HCI) reference to videogame graphics (object geometry, textures, Screen Refresh Rate (SRR), anti-aliasing, motion blur, screen resolution, lighting or lightmaps) and sound (surround, quality, midi). But, other HCI aspects start to be included in videogames like stereo vision, tactile feedback or visual user recognition [6], or other haptic interfaces [7].
- Physics. Inverse kinematics, collision detection, inertia, dynamics or forces.
- Behaviors strategies or Artificial Intelligence (AI).

Those aspects have not the same relevance for all videogames families. Table 1 show some examples of videogames families and the level (low, medium or high) of significance of each QoS aspect. The programmer defines some videogame objects QoS parameters such as object geometry, polygons number, object size, texture color depth or amount of textures. Programmers traditionally allow the user to define some videogame rendering characteristics like anti-aliasing or screen resolution. This is a way to adjust the render object QoS to the computer power where the videogame is executed.

The traditional videogames simulation paradigm does not allow to define the videogames QoS properly.

1.1 Videogames Simulation Paradigm
A computer game may be considered a system [2] and it can be represented using modeling and simulation techniques. Attending to the systems classification [11] a videogame could be considered a hybrid system. This means that the continuous system evolution in time may be altered by events

not associated to the world sampling period. Attending to algorithm 1 [8], traditional videogames follow a continuous simulation scheme since the whole scene graph objects are sampled once in every world evolution. Continuous simulator sampling period is defined by the time elapsed in a run of the program main loop although they are considered hybrid systems. Every world evolution always requires a previous user input and a full UF. This coupled scheme was used by the earlier videogames and it has survived until present (free source code videogames).

Algorithm 1 Traditional videogames main loop.

```
while true do
    Get information from input devices
    Compute a tick of the simulation. Evolute one step the
    whole universe
    Update UF (video, sounds, haptics,...)
end while
```

1.2 Discrete over Continuous Simulation

Implementing computer games as continuous systems have many disadvantages:

- All objects in the scene graph (or in the active objects list) are simulated, although many objects will never generate events. So, the simulation may be erroneous because of disorderly events execution (depending on the objects position in the scene graph) or even the execution of canceled events.
- The videogame objects sampling frequency is the same for all objects. If objects behaviors do not match Nyquist-Shannon theorem they will be undersampled or oversampled.
- The videogame is hardly dependent on topics that can change during the game, such as available computer power, world complexity, other active tasks in system or current simulation and rendering load. The sampling frequency depends on the system load; so, it is variable, not predefined and not configurable for each particular object or object aspect.

Let it be: T sampling period of the whole system, SSF sampling frequency of the whole system, OS collection of objects in the videogame, O_i collection of the object i aspects, $OSF_{i,j}$ object i sampling frequency that models the aspect j and $OSF_{i,j_{min}}$ minimum $OSF_{i,j}$ to simulate the aspect j of the object i properly. Videogames continuous simulation evolute the whole system at a time. Every evolution (world sampling) is a main loop step (algorithm 1). The whole system is sampled following the equation $SSF = \frac{1}{T} = OSF_{i,j}$. This simulation scheme has disadvantages. Different object aspects may need different $OSF_{i,j}$.

Continuous simulation paradigm supports discrete simulation in a very inefficient way while discrete simulation scheme supports simultaneously discrete but also continuous simulation in a very smart way [2]. Discrete simulation paradigm has advantages over continuous simulation since only those objects that change their state produce events and consume computer power. The object priority in simulation depends on the time its events are set. The next event to simulate the aspect j of the object i will be sent to itself $1/OSF_{i,j_{min}}$ time units later.

There is no restriction in the sampling frequencies. They are constant or may change dynamically if the programmer wants. Even different object aspects may have different sampling periods if required and they may change during the whole videogame execution. So, the QoS can be defined for each object aspect by the programmer. The system distributes the computer power according to each objects needs.

1.3 Decoupling over Coupling

If animation and rendering are decoupled, scenes are rendered more quickly even when the higher-level animation computations become complex [9]. This decoupling increases system performance [3]. The rendering and simulation decoupling allows the independence of other processes in the system [1]. Decoupling was created to distribute the system processes in a computer network or to use parallelism. However such distribution is not possible in games created to run in a single PC or console[1]. Tests [10] show that more than 70% rendering power may be wasted if the SSF overpasses the SRR. The SRR can not be defined by the programmer or by the user in a continuous videogame. Discrete videogames allows the independence of the SRR and the system load. The SRR can be defined and/or adjusted during the videogame execution automatically or explicitly by the programmer.

2. OBJECTIVES

The discrete event simulator DESK [5] has been integrated into the Fly3D kernel [12][13] to test if the new simulation paradigm improves both the performance and the simulation quality. DESK is a library developed by us to solve discrete simulation. GDESK [4] is DESK adapted to videogames. DFly3D (Discrete Fly3D) is the modified Fly3D v2.0 kernel result of using GDESK to manage the Fly3D events. A videogame created using DFly3D is a collection of objects interchanging messages [4] managed by GDESK. The object defines its behavior as the response to an incoming message. The number of messages generated in an interaction depends on how the programmer models the objects interaction or the object behavior. Different system objects or the same object may have a discrete or continuous behavior, instantaneously or during the videogame execution. The objects priority depends on the events generation process, completely defined by the videogame programmer for each object. The Render Process (RP) is managed by a specific object. The RP generates a event each time a rendering must be performed. So, the RP is decoupled from the Simulation Process (SP). Each peripheral is managed by a different object in order to adapt its sampling frequency to the device characteristics (as keyboard process or sound process).

3. RESULTS

[1]New game platforms like PS3, based on grid computing, will put again on the table the necessity of this paper paradigm shift in videogame technology

We made a videogame consisting on some balls jumping and colliding. Two videogame versions have been implemented for both Fly3D and DFly3D kernels. The QoS aspect considered in tests has been the number of balls movements and the accurate collision detection. The balls number for tests has been changed increasing both simulation and rendering load. Both systems have been tested in a PC Pentium 4 (2 GHz) with 512Mb Ram with the Nvidia GeForce 4 MX440 graphic card.

3.1 Simulation Times Comparison

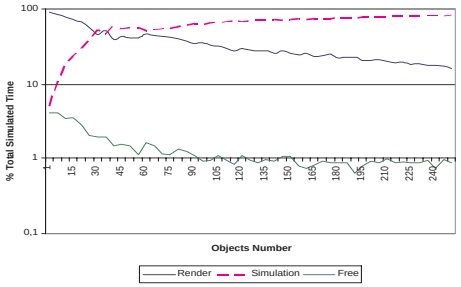

Figure 1: Simulated time in Fly3D (logarithmic).

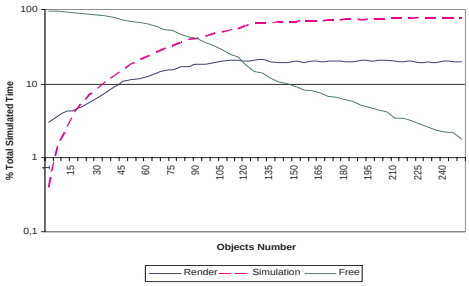

Figure 2: Simulated time in DFly3D (logarithmic).

Figures 1 and 2 show the simulated time percentages used for both systems to simulate, render and remain idle. Time in the continuous system (figure 1) is shared by the SP and the RP. For each main loop step each object is simulated once and the scene is rendered, at the maximum speed. An increase in the SP load supposes to decrease the RP and viceversa.

In the discrete system (figure 2) the RP and the SP are not dependent (decoupling). So, the videogame time is not shared by rendering and simulation. The system uses only the 100% of CPU time if the system is collapsed (the system load is bigger than the computer power). The discrete system always freed more time than the continuous system. If the computer power is enough to simulate the system objects properly, the released computer power may be used by other system applications. The discrete system uses only the necessary computer power to simulate properly the system objects.

3.2 Sampling Frequency

An object in a videogame may have different aspects (graphic, AI or physics aspects) that may need to be sampled separately. For example, a ball is distorted once every second, but its speed forces to change its position every 0.05 seconds and to detect its collisions. The sampling frequency is the same for all objects in a continuous system and for all the objects aspects. This sampling frequency depends on the computer power and the videogame load. All objects are sampled to the higher frequency the system can achieve. It is quite difficult for the global system sampling frequency to match the ideal objects sampling frequency. So, if the object QoS requires a higher sampling frequency, it is undersampled. If the object needs a lower sampling frequency, the object is oversampled. The discrete system allows to define different sampling frequencies for each object aspect. DFly3D objects sampling frequency may change to adjust the sampling to the current object behavior or to the object QoS.

We say that the computer system is collapsed when a system is not able to show the scene properly (incorrect objects behavior). Let it be: CF_S global system collapse factor and $CF_{i,j}$ object i aspect j collapse factor. A system could be collapsed due to the videogame scene or simulation complexity or because of the low computing power. If the system is collapsed both kernels do not allow running the videogame properly. The continuous kernel typically undersamples the objects making their movements more chaotic and the collision detection fails many times ($\exists i \in OS, \exists j \in O_i : OSF_{i,j_{min}} > SSF$). The bigger the load, the more aspects and objects that do not match the Nyquist-Shannon theorem. The discrete kernel produces a correct output since discrete simulators always match $\forall i \in OS, \forall j \in O_i : OSF_{i,j} > OSF_{i,j_{min}}$, but the system evolution is slower. That is, simulated time cannot follow real time. The given moment of system collapse depends on the videogame complexity and the kernel used. The bigger the simulation load, the slow the simulation runs. The collapse can be measured by the CF_S. Each object i aspect j have its own collapse factor $CF_{i,j}$ ($\forall i \in OS, \forall j \in O_i : OSF_{i,j_{min}} > SSF \Longrightarrow CF_{i,j} = OSF_{i,j_{min}} - SSF$). If the system is not collapsed, this factor is 0 ($\forall i \in OS, \forall j \in O_i : OSF_{i,j_{min}} \leq SSF \Longrightarrow CF_i = 0$). The global system factor is fixed by the collapse factors of each aspect of each object ($CF_S = \sum_i^{i \in OS} \sum_j^{j \in O_i} \mid CF_{i,j} \mid$).

A system is collapsed when there is at least an aspect j of the object i that does not maintain its $OSF_{i,j}$ ($\forall i \in OS, \forall j \in O_i : OSF_{i,j_{min}} > OSF_{i,j} \Longrightarrow CF_{i,j} = OSF_{i,j_{min}} - OSF_{i,j}$, $\forall i \in OS, \forall j \in O_i : OSF_{i,j_{min}} \leq OSF_{i,j} \Longrightarrow CF_{i,j} = 0$). If an object aspect $OSF_{i,j}$ is degraded, all the objects aspects sampling frequencies, are degraded too. The system degradation is uniform in a discrete system. The discrete system produces a SSD. If the system is collapsed and the sampling frequency of each object aspect is correctly selected, the system slows down, but the simulation remains correct. That is, the restriction is that the simulated time must follow always real time.

Each object aspect has a different percentage of degradation in a continuous system. The SSF decreases, so, each object sampling frequency decreases too. The new global sampling frequency can degrade each object in a different percentage. If the system is collapsed, the continuous system may have erroneous behaviors as losing events or not detected collisions in some objects. The computing overload

produced by the discrete events management in a discrete system is minimum, so both paradigms overheads are quite similar. The discrete system when there is not enough computer power to simulate the system following the real time. The amount of simulations remains constant although the system load grows. The QoS of each object aspect is maintained. So, this kernel can be used for off-line or not real time simulation also.

3.3 Dynamic Improvement and Degradation

The discrete system may be defined to adapt the computer power to the CF_S or to redefine the QoS of system objects, in order to adapt the objects behavior to the real system load. For example, let the RP to generate 50 FPS, and the ball is sampled 20 times per second. If the computer power is low and the ball needs to be sampled during a time interval 30 times per second, the RP may slow down to only 25 FPS, releasing computer power. The opposite situation can be done too. If the RP detects free computer power can generate more FPS or the ball may do it too. That supposes to use techniques to adapt the system degradation factor dynamically. The monitorization of the degradation factor can be done for:

- Each object. The object collects information about if its QoS has been accomplished and redefines its QoS as consequence. The object can get global system information too. The object decisions can involve other objects. For example, the programmer decides if the ball simulation has priority over the RP. If the ball detects that it is not simulated properly, the ball may send a message to the RP to decrease the RP QoS in order to improve the ball QoS.
- Each object aspect.
- The whole system. There are two possibilities:
  - A Monitorization Object (MO) is created. The MO sees system information. If any object aspect QoS must be changed, the MO sends messages to the objects involved. The message contains the necessary information to allow the object to change the object aspects as convenience.
  - Any system object monitorizes the whole system. It is no necessary to create a specific MO. Its functionality may be assumed by any other object.

4. CONCLUSIONS

The videogame QoS can be defined by three different aspects: UF, physics and simulation. Each object aspect in a videogame has its own QoS. Traditional videogame kernels define a global QoS based on the graphic videogame characteristics. The QoS is defined for the whole system because traditional videogames follow a scheme of coupled continuous simulation. The level of system degradation is not uniform; it depends on each object characteristics. Although continuous simulation games have been the main stream during the last years, this paradigm has many drawbacks, especially in current requirements: portable devices games (very low computer power) or last generation personal computer games (high load). The discrete simulation paradigm allows to define a QoS criteria for each aspect of each object in the videogame, as sampling frequency or simulation quality. This sampling frequency may change to adapt the object aspect QoS to the real computer power and distribute the computer power adequately among the objects. The objects events are executed ordered by time. The RP works as any other videogame object. So, the SRR can be adjusted to the system load or characteristics. The result obtained is a discrete system that allows a SSD and may redefine the objects aspects QoS. Objects can collect system information and use it to adapt their QoS.

5. REFERENCES

[1] M. Agus, A. Giachetti, E. Gobbetti, and G. Zanetti. A multiprocessor decoupled system for the simulation of temporal bone surgery. *Computing and Visualization in Science*, 5(1), 2002.

[2] J. Banks, J. Carson II, B. Nelson, and D. Nicol. *Discrete-Event System Simulation*. Prentice Hall International Series in Industrial and Systems Engineering, 2001.

[3] R. Darken, C. Tonnesen, and K. Passarella. The bridge between developers and virtual environments: a robust virtual environment system architecture. *SPIE*, 1995.

[4] I. García, R. Mollá, and A. Barella. GDESK: Game discrete event simulation kernel. *WSCG*, 2004.

[5] I. García, R. Mollá, E. Ramos, and M. Fernández. D.E.S.K.: Discrete events simulation kernel. *ECCOMAS*, 2000.

[6] T. Komura, A. Kuroda, and Y. Shinagawa. NiceMeetVR: Facing professional baseball pitchers in the virtual batting cage. *Cgforum*, 16(3):C347–C355, 1997. (Proc. Eurographics'97).

[7] S. Mokka, A. Väätänen, J. Heinilä, and P. Välkkynen. Fitness computer game with a bodily user interface. *Cgforum*, 16(3):C347–C355, 1997. (Proc. Eurographics'97).

[8] R. Pausch, T. Burnette, A. Capehart, M. Conway, D. Cosgrove, R. DeLine, J. Durbin, R. Gossweiler, S. Koga, and J. White. A brief architectural overview of Alice, a rapid prototyping system for virtual environments. *IEEE Computer Graphics and Applications*, 1995.

[9] C. Shaw, J. Liang, M. Green, and Y. Sun. The decoupled simulation model for virtual reality systems. *CHI*, 1992.

[10] Tom's Hardware Guide. www6.tomshardware.com.

[11] G. Warnier. Introducción a la simulación de sistemas de eventos discretos. Technical Report 96-005, Buenos Aires University, 1996.

[12] A. Watt and F. Policarpo. *3D Computer Games Technology: Real-Time Rendering and Software*, volume I. Addison-Welsey, 2001.

[13] A. Watt and F. Policarpo. *3D Computer Games*, volume II. Addison-Wesley, 2003.