Discrete Events Simulation as a Computer Game Kernel

Inmaculada García Ramón Mollá
Computer Graphics Section
Dep. of Computation and Computer Systems
Technical University of Valencia
Camino de Vera, s/n
46022 – Valencia – Spain
{ingarcia,rmolla}@dsic.upv.es

Abstract

Many computer games follow a scheme of continuous simulation. This scheme is not able to support all possible interactions produced among different objects within the system. To join the visualization part with the simulation process is inefficient. DESK is a simulator kernel that may be used as a computer game kernel. It allows decoupling simulation and visualization, increasing calculation efficiency. Nervous sampling improves simulation accuracy, avoiding incorrect behaviors in the system object interactions, typically performed by continuous simulators.

Keywords

Hybrid simulation, computer game engine

1. INTRODUCTION

A real-time graphic application, as virtual reality (VR), computer games or simulation, may be considered a *system*, following the definition of system of Banks and Carson [Banks95]. As a system, it can be represented using *modeling* [Wainer96] and *simulation* [Banks95] techniques. Attending to the systems classification, based on the way the system evolves in time [Wainer96], a real-time graphic application could be considered as a *hybrid system*. In that, the continuous system evolution in time may be altered by events not associated to the sampling period.

The earlier real-time graphics systems worked with a main loop that coupled graphics rendering phase to simulation phase. If animation and rendering are decoupled, scenes are render more quickly even when the higher-level animation computations become complex [Shaw92]. This decoupling increases system performance [Darken95]. Some real-time graphic systems that decouples simulation and visualization are among others: the *Cognitive Coprocessor Architecture* [Roberson89], *MR toolkit* [Shaw92] based on the *Decoupled Simulation Model*, *Virtual Builder II*, [Gobbetti95], *Alice & Diver* [Pausch94] or *Bridge* [Darken95].

The rendering and simulation decoupling allows not only to increase system accuracy and speed, also allows the independence of other process in the system [Giachetti02]. The natural evolution of decoupling was 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. Network games allows multi-users, but not distribute process in the net.

1.1 Computer games

Many different source code of computer games available correspond to non-commercial free games made by enthusiastic people. These computer games lack from internal organisation. They employ rudimentary simulation techniques. That is the reason why they have not been included in the present study [ZIRON].

Only a few commercial computer games have their source code published. Among them, we emphasise DOOM v1.1 [DMCd] or QUAKE v2.3 [QKDoc] [QKCd] games and the Fly3D kernel [Watt01] (a true 3D object oriented multipurpose game kernel developed in C++) because of their importance in computer games. Their working model may be seen in Figure 1.

An events management study of different games has been performed in order to verify the use of simulation techniques. Simulation techniques applied in many cases suppose considering computer game as a continuous system, although computer games are hybrid systems. So they have large limitations when working.

2. OBJECTIVES

Hybrid Simulation (HS) allows a more accurate simulation than Continuous Simulation (CS), avoiding lost events and a disorderly execution of events when the simulator has a limit in the computing power. The objective of this paper consists in the application to video games the following issues:

- Improve simulation accuracy using a HS model.
- Verify that HS techniques can improve the system events management, increasing game efficiency.
- Decouple rendering and simulation phase, as in high-end real-time graphics applications.

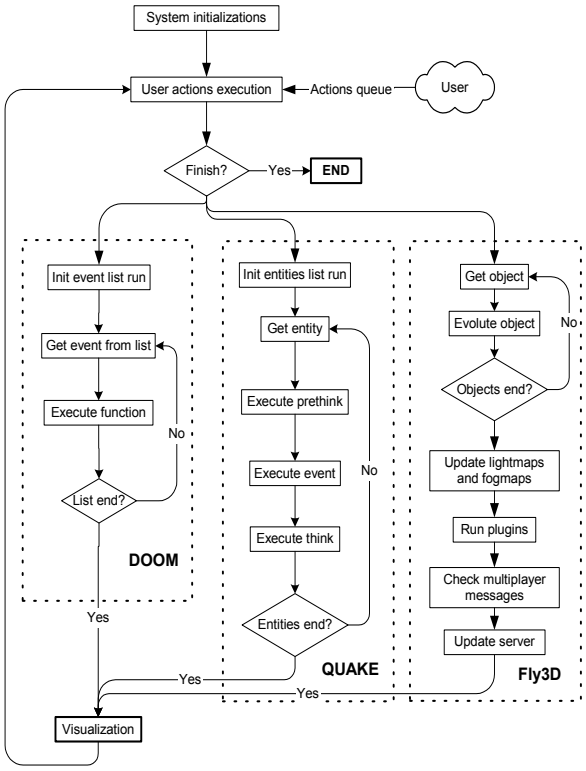

Figure 1: DOOM , QUAKE and Fly3D main loop

The saving in computer power may be used to improve the Human Computer Interface.

3. ANALYSIS

Simulation cycle is defined as the time elapsed in a run of the program main loop (sampling period of a continuous simulator). In the games analysed, each world evolution always requires a full visualisation of the entire world. All events are obligatory evolved at the higher speed that the computer system can supply, following a CS scheme. This means:

- The system is not sensitive to times lower than the sampling period.
- Events are executed in the order in which events, entities or objects management structure are accessed. They are not time ordering.
- Events are artificially synchronized matching the sampling period. They are not executed in the very moment when they happen.
- The sampling frequency depends on topics that can change during the game, as available computer power, world complexity, other active tasks in system, network overload or current simulation and visualisation load. So, the sampling frequency is variable and not predefined.
- The sampling frequency is the same for all objects, independently of their requirements. If objects behaviours do not match Nyquist-Shannon theorem, they will not be simulated properly, producing events loss, not detected collisions,... On the other hand, an object with a very slow activity may be oversampled.

3.1 Criticism

Current graphic cards support Screen Refresh Rates (SRR) over 75Hz. Higher frequencies are redundant since no flicker effect can be appreciated in practice from 72Hz on. The tests carried out upon QUAKE 3 v1.17 in current equipments employing last generation cards obtain refresh rates between 130 and 250 fps [TOM02].

This behaviour is inefficient due to the loses of time generating renderings that will never be appreciated on screen. The 70% rendering power is lost when generating 250 fps. An improvement of this operation scheme would be to separate simulation from rendering. The goal is to match the system sampling frequency to the SRR, typically 75Hz. Then, the system calculates only as many frames as the number of times the screen is refreshed. Released computer power provides a bigger amount of simulation cycles used to calculate simulations more precisely [Reynolds00]. Decoupling the system is not the solution if the simulation scheme remains continuous, because:

- Simulating at the fastest speed is inefficient since it spends calculations simulating intermediate states not rendered on screen (unadjusted sampling period).
- Scene graph must be accessed twice.

A common practice in the earlier computer games, with low scene complexity, was to simulate and render each object, accessing the scene graph once [Bishop98]. Current situation is different because rendered frame rates are higher than SRR [TOM02].

On the other hand, access through the scene graph when many objects will never generate events, is quite inefficient. It would be better to use an events queue. Only those objects that generate events will be checked, avoiding to access the remainder objects.

3.2 Improvements

After analysing how these programs operate, the following improvements are proposed:

- All the objects will have the same priority, including player avatar.
- Nervous sampling will be supported. This means:
  - No event may be lost.
  - Events should be attended and simulated at the moment in which they are produced, not in the following cycle.
  - Each object may have its own sampling period. Therefore, those objects generate events at a higher rate have to be sampled at a higher frequency. Also, the objects that do not generate events, do not overload the simulation engine.
  - Events should be simulated ordering in time.

4. DISCRETE EVENTS SIMULATION KERNEL

DESK [Garcia00] is a C++ object oriented Discrete Events Simulation Kernel API. This simulator allows changing dynamically system topology. Also, it allows include any continuous or discrete behavior in the system. It uses to

be faster than Smpl [SMPL] for almost all simulated models. DESK can manage models with high complexity and size. The programmer only focuses on the model description, not in events or programming bugs. The effort to describe a behavior can be reused (in the same model or others). The programmer can implement the definition of the behavior functions. This gives flexibility enough to use it as a game kernel.

5. USING DESK AS A COMPUTER GAME KERNEL

Using DESK as a simulation engine for computer games does not imply to change topics as the structure of the scenes description files or characters. It does not modify the file parser, the scene graph, the rendering techniques applied or video game style. It only modifies the system events management and introduces a HS scheme.

Objects in DESK, interact by *messages passing*. A message is modeled by means of a client that contains the necessary information to develop an adequate interaction. For example, when a projectile collides with a wall, it sends a message to the wall indicating that it has collided with it. The message must includes the necessary parameters (like mass, point of impact, velocity or projectile kind). The behavior of the receiving object depends on the object transmitter and the kind of message sent. The receiving object determines what kind of behavior must be presented. For example, if a small stone collides with a wall, perhaps the wall attributes will not be affected with that message (depending on the weight of the stone). If the projectile is a missile, the wall can disappear. The same event with different transmitting objects causes different behaviors in the same receiving object; but also in the transmitter (the bullet rebounds if the wall is made of concrete and it is incrusted if is a clay wall). Different kind of objects will have different behaviors.

On the other hand, objects spent time generating an answer to the message. Therefore, when a ball strikes a wall, it does not rebound immediately. It spent time changing its trajectory. The ball can be deformed as consequence of the impact, to subsequently recuperate its original form.

5.1 Creating a Game

When a game starts, all objects in game must be created, assigning them a behavior and grouping them into a hierarchy. The simulator will invoke the behavior assigned to an object at the precise moment. It is possible to define autonomous system behaviors.

The system generates a render event for each SRR. User events are registered in the events queue at the moment that are produced. So, they are mixed with the remainder of system events, being ordered in time. In this way, the player receives the same homogeneous processing that the remainder entities. Computer game implementation using DESK supposes:

- To define the behavior methods of each object:
  - A behavior differentiated depending on the kind of message received and the transmitter object.
  - Generation of new events produced as consequence of the defined behavior. It can include answers to transmitter object.
  - Statistical methods, if proceeds.
- To define the system behavior. This behavior should include, al least, the SRR.
- To invoke the simulator initialization function, in order to initialize simulator internal data structures.
- To start the simulation.
- To obtain results as statistics or accounting.

5.2 Working Model

When an animated object is created, a new event is sent to the system. Unanimated objects will not generate any event at the moment of creation. When the entire scene is created, the computer game starts.

Scene simulation process is carried out independently of the visualization process. During computer game execution, the system generates an event for each screen refresh. When events manager detects the event, two things may happen:

- Exist already a screen previously calculated which is waiting to be shown on screen. Therefore, there is no reason to process a new render.
- The last calculated image is being shown on screen, there is no screen available waiting to be shown. So, a new rendering must be calculated.

The visualization freezes the simulation while it generates the new image going through the scene graph. Therefore, there is only necessary to go through the scene graph once for each visualization as in typical games.

Each time a screen refresh ends, the graphic package activates the screen calculation. So, next time a refresh event is generated, will be carried out the visualization.

When the calculated frame rate is lower than the SRR, DESK behavior will be equivalent to the studied games. If the calculation power is high, computer will not lose time calculating an image that will never be rendered. This remainder computer power may be dedicated to improve the simulation accuracy or other needs, as opposed to these games (see point 3.1).

For example, let it be a hand grenade thrown to the air. At the moment in which it is thrown, an event of blowing is generated in 1.76 seconds, wherever it is. While it does not blow, the hand grenade travels through a continuous trajectory by air. Trajectory is sampled in screen SRR times by second. Therefore, the hand grenade position must be recalculated for each screen. That is done generating an event each 1/SRR seconds. It is not necessary to spend computer power simulating intermediate states, since only the last one is shown at screen. DESK simulation model is hybrid because it carries out a CS, sampling system each 1/SRR seconds, while it still bears discrete events.

DESK always performs SRR fps. So, sampling frequency in DESK is constant assuming that there is computer

power enough. It does not depend on the game load or computer power available.

The HS model bears implicitly nervous sampling because those processes that present a great variability will generate a great quantity of events consuming more computer power. Those objects that do not generate events do not overload the game. So, computer power is distributed among all objects depending on their behaviors.

5.3 Results

Coupled and decoupled models have been simulated in the laboratory. Let it be: T_S the time used by the videogame to simulate a game step, T_R the time spent to render a frame, and, finally, ν_V the fps actually generated by the videogame. The conclusion for CS decoupled systems are:

1. $SRR \geq \nu_V$
2. If $T_S+T_R \leq 1/SRR$ then $SRR=\nu_V$
3. If $T_S+T_R > 1/SRR$ then $T_S+ T_R \leq 1/\nu_V$
4. The amount of simulations is always equal or higher than the amount of renderings

Point 2 gives the maximum performance because SRR is achieved. Videogames behaves better in the point 2 situation than point 3. Using DESK HS reduces T_S, increasing the probability to operate under the conditions of point 2.

6. CONCLUSION

The computer games analyzed follow an inefficient simulation scheme because of they use CS and they do not decouple visualization from simulation. Although CS model is widely used in computer games and it works in practice, this model becomes insufficient to simulate accurately more complex behaviors like next generation computer games, VR applications or interactive real time graphics. HS allows a more accurate simulation than CS, avoiding lost events and a disorderly execution of events when the simulator has a limit in the computing power.

When DESK is used as a computer game kernel, it allows HS and forces to decouple the system. So, the calculation efficiency is increased and it improves the simulation accuracy performing a better final result. Visualization and simulation decoupling allows using parallel techniques in order to increase computer game speed, to bear more complexes computer games, professional applications or to improve final visual result.

Since DESK supports nervous sampling, simulation accuracy is improved, avoiding incorrect behavior to be produced. Computer power is concentrated in simulating those parts of the game with high rate of events generation, dedicating less attention to those parts with a lower events rate.

7. ACKNOWLEDGEMENTS

This work has been partially funded by the Spanish government CICYT TIC1999-0510-C02-01

8. REFERENCES

[Banks95] J. Banks, J.S. Carson, B.L. Nelson. Discrete-Event System Simulation. (Prentice Hall, 1995)

[Bishop98] L. Bishop, D. Eberly, T. Whitted. Designing a PC Game Engine. IEEE CG&A, vol. 18, no. 1, January/February 1998, pp. 46-53

[Darken95] R. Darken, C. Tonnesen, K. Passarella. The Bridge Between Developers and Virtual Environments: a Robust Virtual Environment System Architecture. Proceedings of SPIE 1995, No. 2409-30

[DMCd] www.idsoftware.com/archives/doomarc.html

[Fishman78] G.S. Fishman. Conceptos y Métodos en la Simulación Digital de Eventos Discretos. (Limusa, 1978)

[Garcia00] I. García, R. Mollá, E. Ramos, M. Fernández D.E.S.K.: Discrete Events Simulation Kernel, ECCOMAS 2000, September 2000

[Giachetti02] M. Agus, A. Giachetti, E. Gobbetti, G. Zanetti. A Multiprocessor Decoupled System for the Simulation of Temporal Bone Surgery. *Computing and Visualization in Science*, To appear

[Gobbetti95] E. Gobbetti, J.F. Balaguer. An Integrated Environment to Visually Construct 3D Animations. SIGGRAPH 95 Conference Proceedings, Annual Conference Series, pp 395-398, August 1995. ACM SIGGRAPH, Addison-Wesley.

[Pausch94] R. Pausch et al. Alice & DIVER: A Software Architecture for the Rapid Prototyping of Virtual Environments. Course notes for SIGGRAPH '94 course, "Programming Virtual Worlds".

[QKCd] www.quake2.com/kko/

[QKDoc] www.gamers.org/dEngine/quake/

[Reynolds00] C. Reynolds. Interaction with Groups of Autonomous Characters. Game Developers Conference Proceedings, March 2000.

[Roberson89] G.G. Robertson, S.K. Card, J.D. Mackinlay. The Cognitive Coprocessor Architecture for Interactive User Interface, UIST'89 Proceedings, 1989, pp. 10-18.

[Shaw92] C. Shaw, J. Liang, M. Green, Y. Sun The Decoupled Simulation Model for Virtual Reality Systems. CHI'92, May 1992, pp. 321-328.

[SMPL] http://www.autoctrl.rug.ac.be/ftp/smpl/

[TOM02] www6.tomshardware.com/graphic/02q1/020304/geforce4-09.html

[Wainer96] G.A. Wainer. Introducción a la Simulación de Sistemas de Eventos Discretos. Technical Report: 96-005. Buenos Aires University.

[Watt01] A. Watt, F. Policarpo, 3D Games. Real-time Rendering and Software Technology, (Addison-Wesley, 2001) http://www.fly3d.com.br

[ZIRON] http://www.ziron.com/links/