REAL-TIME SIMULATORS DEVELOPMENTS: CURRENT AND NEW TRENDS

Rafael J. Martínez Durá

Instituto de Robótica, Universidad de Valencia
C/ Catedrático José Beltrán, 2
46980, Paterna, SPAIN

Rafael.Martinez@uv.es

ABSTRACT
This article briefly describes some technologies applied to current real-time simulation training systems. It focuses on nowadays and future trends both in hardware and software architectures. The most important open source and commercially available physics engines are briefly described along with some scene graph management libraries. Additionally, cluster-based simulator issues are also discussed, including the main problems that arise from using distributed GPU programming. An analysis of current state of the art regarding web browser-based simulators is also done. Finally new trends and cutting-edge research that is done in the real-time simulation field are discussed.

Keywords: real-time simulation, browser-based simulation, GPU clusters, shaders

1. INTRODUCTION

Simulation issues always deal with a set of techniques that allow reproducing the behaviour of both physical processes, such nuclear reactions, store stocks fluctuations or vehicles, and human behaviours, such as evacuation systems or collaborative works. Nowadays, due mainly to the availability of high performance computer systems, it is possible to simulate in real time the behaviour of devices, machinery, vehicles, control rooms, allowing the user to be trained to obtain the necessary skills to handle the real system with high confidence.

A training simulator is built based upon a set of functional blocks, being the most important the image generation subsystem, the dynamical models that compute the behaviour of the simulated objects, the input/output subsystem that connect the user interfaces, the instructor subsystem, the user management subsystem and the evaluation subsystem. All of them are closely coupled, in a way that the actions over any simulator controller are computed by the dynamical models and produce a reaction that is observed immediately by the simulator operator, indicating some kind of interactivity.

This capability presented in any real-time simulator is achieved whenever the computation time that the systems dedicates to calculate the evolution of the simulated environment is smaller than the time that is being simulated. As a consequence, every simulation model, or the image rendering phases, should not be as complex as desired, having to assume some kind of simplifications (reduce the integration step, simplify the dynamical model, reduce the number of polygons in the scene, use levels of detail in the objects, etc.).

Training by means of simulation has several advantages against the use of the real machine, although in any case the simulator can substitute the real one, due mainly to the fact that the simulator will always simplify the working environment. The main characteristics that make the simulators an indispensable element in the training process are:

- Risk avoidance
- Training costs savings
- Ability to reproduce extreme situations
- Ability to train with faulty machinery
- Objective evaluation of user skills

When designing a training simulator, special care must be taken with the details of the simulated environment in order to keep it close to reality, since this will greatly influence in the decrease of the time involved in the training process. The term *presence* (Sheridan 1992 and Slater, Khanna, Mortensen and Insu Yu 2009) refers to the user impression of "being there", in the virtual world that the computer simulates. In order to achieve a high presence, every user sense must be stimulated as much as possible and also as real as possible. Therefore, when integrating a simulator, it is highly recommended to use immersive visualization technologies, moving platforms, real dimension cabins and controllers, 3D sound surround systems and mainly graphical scenes and dynamical models highly accurate.

Along this paper a review of the technology related with real-time training simulator design will be done. Section 2 deals with new trends in computers and I/O hardware that will be used in future simulators. Following, in Section 3, the most important simulation engines are described and also the new trends that exist in multi GPU programming. Next section describes the state in the art in browser-based simulators and finally some open problems and current research lines regarding simulation are explained.

2. HARDWARE ISSUES

When building a simulator, it can be done using many hardware alternatives. The decision will depend on the presence and accuracy that the simulator must provide. Following, a description of the future trends in the computational systems commonly used in the implementation of training simulators is done.

2.1. Computer Architectures

The computer used to execute a simulator always depends on the number of different views that the simulator employs. There are simulators that have one or two screens, like those used in crane simulators, three screens, used in vehicle simulators and even five in general purpose simulators implemented with CAVES (C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon and J. C.Hart 1992). Every view has to be computed by a graphic card, so the simulator computer system at least must provide the capability to include all of them inside, and additionally, it must have enough performance to be able for rendering the images at high refresh rates.

For these two main reasons, PC-based simulators and virtual reality systems were traditionally built with computer clusters, since a graphic card could be plugged in every node and there are also several CPUs that allow the implementation of parallel algorithms and complex dynamical and visual models. In such systems, designers had to cope with the problems derived from the synchronization of simulation data, scene graphs and user inputs.

Nowadays, the evolution of PCs performance and the ability to plug several graphic cards in the same computer, make clusters being superseded by low-cost single computers, avoiding the synchronization issues present in clusters.

2.1.1. PC based configurations examples

Following some advice is given for choosing the hardware to build a simulator, based, as viewed before, in the number of views it has to provide.

When building a simple simulator, a single graphic card with several heads can be used (the Matrox M9140 LP has four inside) for connecting several displays. If high quality images are needed, as motherboards with up to four PCI-e 16X slots are already available, it is possible to have several graphic cards plugged in the same PC. They can be configured to act as a single card with high resolution and antialiasing capabilities (SLI from NVidia and CrossFire from ATI) or in an independent way providing several outputs. Another similar solution is provided by NVidia. Its Quadro Plex Model II is capable of connecting eight displays (it uses four NVidia Quadro FX 4500 X2) and the NVIDIA Quadro Plex Model IV, is capable of connecting four synchronized displays (it uses two NVidia Quadro FX 5600 with GSync boards).

2.1.2. Game Consoles based Systems

Another important line to take into account is all the hardware related with video games. Video games are similar to simulators in many aspects, so the computer architectures specially developed for them also have enough power to drive a simulator, but with very low costs. The only restriction is that video game consoles are thought to have just a single screen, so they are not good for multi display systems, however clusters of them can be easily built in the way that traditionally has been done with the PC-based simulators. The most important game consoles that are currently available are:

Sony PlayStation 3 (http://www.playstation.com): Its parallel Cell processor and the NVidia RSX architecture (based on a GeForce 7800) provide high performance to this videoconsole, allowing the development of highly parallel applications. Although an official software development kit exists, it is also possible to install Linux Yellow Dog and other distributions for the PS3. They include a set of compilers and libraries for programming the Cell Processor. The main drawback they show is that their kernels restrict the access both the GPU and some peripherals, avoiding the development of high performance applications (See Barttlet 2007).

Microsoft XBOX 360: It is based on a 3 core Xenon CPU with an ATI Xenos graphic card that supports unified shaders. Microsoft has developed a free API (Application Program Interface) that allows the development of applications for this architecture. In order to develop professional games a subscription to the Xbox Registered Developer Program must be done (http://www.xbox.com/dev).

Computer architectures for real-time simulation are thus evolving in two main directions. On the one hand, the use of Personal Supercomputers offers an alternative to clusters, reducing cost and system complexity. On the other hand, the use of clusters is also being reviewed by the introduction of Video Game Consoles which provide a step ahead in parallelism and reduce costs.

2.2. Input Devices

Professional simulators trend to use the same devices employed in game simulators and other video game applications. Main reasons are that its interfaces are very well defined and supported by the operating system and that these devices have very low costs.

One example is the WiiRemote. It is one of the most common computer input devices in the world. It also happens to be one of the most sophisticated. It contains a 1024x768 infrared camera with built-in hardware blob tracking of up to 4 points at 100Hz, a 3-axis accelerometer also operating at 100Hz and an expansion port for even more capability. Its Bluetooth interface allows connecting the device to any personal computer. Many simulation related applications are including the wireless WiiRemote, since it can provide positional information that can be used as a tracker or a wand. See http://www.wiimoteproject.com

Future simulators will also include personal trackers. The high reality you get with this devices increment the presence of the simulator at very low costs. They are based on infrared cameras, like the ones included in the wiimote.

One example to take into account is the Microsoft Natal Project (http://www.xbox.com/live/projectnatal), that uses a video camera that tracks where your body is and what you're doing with it. It also uses a monochrome camera (it works with infrared) that reads depth — how far away your body and its component parts are — and a highly specialized microphone that can pick up voice commands. Along with all this hardware, it's got a ton of software that tells the Xbox how to find your body's various joints (it tracks 48 of them) and how to keep track of multiple players at the same time.

Others examples are the NaturalPoint low cost trackers (http://www.naturalpoint.com/trackir). One of its products, TrakcIR5, makes 6 DOF head tracking by means of detecting some marks on the head of the user. They provide a free software development kit that can be used to include tracking in your own simulators.

Also, new low-cost data acquisition boards are currently emerging, allowing the connection of any analogue o digital device to the computer through the usb interface in a straightforward way. They are HID compatible, so another advantage is that they are programmed very easily, as this protocol is supported by any operating system since it is the one employed to connect the standard game devices. In the same way, you can use your home-built simulator device to play with every commercial game simulator.

2.3. Motion Systems

In order to avoid sickness, motion systems must be included in simulators with immersive displays. This will be another future trend, since their installation is quite easy, programming the motion cueing algorithms it is already not necessary and their cost is not very high.

Electro-hydraulic technology has been used for over 30 years in probably 20,000 6DOF simulators around the world, since its response curves denote they can work at high frequencies (over 15Hz), however the shift to all electric systems is the ground breaking event that will change the world of simulation.

There will be small increments in software improvements, including special effects, buffets and the use of white noise generators to complement the increased fidelity of the visual displays. User interface continues to be improved. Automatic testing is being integrated into the core software so that acceptance testing can be run daily and compared to yesterday/last week/last year's data to show a trend in degraded performance indicating a need for maintenance, or no degradation which would indicate the system is being properly maintained. Moog Inc. is one of the main companies that supply full 6 DoF motion platforms. See http://www.moog.com/products/motion-systems.

Apart from this, in the near future, independent actuators will be used to build your own motions seats. They are mainly used in games, although for medium-size simulators it is very easy to build your own 2-3 DoF motion cabin. SimCraft has recently announced a motion simulation development kit, that allow to built your chassis from one of SimCraft's various free plan options, order a bolt-together kit, or design your own within the SimCraft architecture specification. See http://www.simcraft.com/star.html.

2.4. Display Systems

When dealing with high immersive simulators, today emerging technologies trend to substitute the projection systems that use rear-projection screens and mirrors with embedded in-cabin display systems. Stereoscopic images will be a must, since simulation for training demands a high degree of depth perception. Passive stereo technologies, although more comfortable, need to duplicate the number of visuals generated, so its uses will still remain be very restricted.

New solutions will use 3D LCD monitors with large 46-inch screens and Full HD 1920x1080 resolutions. Nowadays cutting-edge display technologies are based in new single chip solid state illuminated DLP projectors, with WUXGA resolutions (1920x1200). The main drawback of these systems is that brightness is only up to about 700 ANSI lumens, however they are capable of processing and displaying infrared content for simultaneous display of both visible light and the infrared spectrum, and the solid state illumination allow a nearly virtually maintenance-free system.

In the very near future, customizable high resolution panels will be built, by means of replicating low profile tileable single displays. Also some research is being done in auto-stereo systems, avoiding the necessity of wearing glasses, but this technology is still to come.

3. SOFTWARE TRENDS

The main problem that arises when designing a training simulator is that its software is compound by many different modules: the user data bases management system, the graphical user interfaces for the exercises and the instructor, the report generators, and the most important, the scene graph management system and the dynamical models for calculating the behaviour of the active objects. Everyone needs to be programmed using different tools.

The design and implementation of such software elements strongly depends on the computer systems they are thought to be executed. The introduction of new processor architectures, such as GPUs and PPUs, with many cores capable of execute instructions in parallel, has made that new programming paradigms have to be used for them. Furthermore, new algorithms and techniques to increase the simulation realism and performance should be applied to the existing methods. In order to understand what new problems arise when designing the software for current simulation systems, a brief description of the advances in computer architectures follows.

3.1. GPU and PPU Programming

Current trends in computer architectures use replicated parallel cores to increment performance. This is done both at CPU level (for example Intel is currently developing a processor with 80 cores inside, see S. Vangal et all 2007) and also at the graphic processing units (GPU) level (for example see NVidia Testla architecture in E. Lindholm, J. Nickolls, S. Oberman, J. Montrym 2008). The GPU evolved from a fixed function graphics pipeline to a programmable parallel processor with computing power exceeding that of multicore CPUs. The main impact of these current architectures was that now it is possible to execute programs at GPU levels through what are called *shaders*. They are simple programs that describe the traits of either a vertex or a pixel. Following the Direct3D version 9 it is possible to write vertex and pixel shaders in a more abstract, readable and reusable fashion, using a high level C-Style like language called High Level Shader Language (see St-Laurent 2005). Later OpenGL 2.0 introduced a common high level shading language for vertex and pixel programs called OpenGL Shading Language (see Rost 2006). New GPUs incorporate a new architecture called "unified shading core" (Blythe 2006) allowing that both vertex and pixel processing can be handled by one single programmable unit instead of having separate programmable units for both the vertex and pixel pipeline and also added to the pipeline another programmable stage called the geometry shader.

In parallel, NVIDIA developed CUDA (Compute Unified Device Architecture), where the GPU is seen as a massively parallel set of multiprocessors, capable of executing a high number of threads in parallel. In the same way, AMD's response to GPU programming for high-performance data parallel tasks was the AMD Stream Computing Model. Both programming models are based on an extension of the C programming language. Furthermore, NVIDIA and ATI recently have added to their consumer level graphics cards a new concept: the Physics Processing Units (PPUs). These units implement the most common physics simulation techniques, accelerating their calculation. They are also programmed by means of CUDA or other high level libraries. Additional details can be seen in NVidia 2008.

This hardware evolution has stated a revolution in the programming paradigms of the new simulation algorithms, since from then, GPUs can be used for general purpose computations, and therefore have a specific weigh much more important than before. Now, they are responsible of taking care of light, shadows, particle systems, and complex dynamical models more and more. In Borgo and Brodlie (2009) a beginner's introduction to GPUs, from both hardware and software point of view, is done.

3.2. New programming paradigms

Nowadays PC clusters are still being used for implementing multi-view training simulators. In the past, software developers must cope mainly with the problems derived from the multichannel synchronization (gen-lock & frame-lock issues) and the data distribution between the I/O devices (sensors, tracking), the dynamical model and the different nodes that were replicated in the visual scene graphs. Within this scheme, only a centralized node that executes a single dynamical model must cope with the dynamics properties of the scene, sending at visual loop rates the position of the different objects that move around the virtual word and the position of the observer. The rest of the nodes execute an instance of the scene graph, modifying it according to the data received from the dynamical models. Then every graphic card renders locally the images that were displayed on the screen.

Current architectures trend to build many cores for increasing program concurrency. This is not a problem since compilers can manage processor affinities very well and also operating systems provide the necessary services to cope with multi-thread applications. Additionally, programmable GPUs allow, also without any problem, the implementation of a dynamical scene graph whose nodes where directly controlled by shaders entirely executed in them. If everything runs in a single personal supercomputer with several graphics boards inside, still we don't have any problem with the renderization of our multiple views.

But however, if we get a cluster of GPUs connected, with everyone executing its own shaders, current software technologies are not ready to cope with the distributed pipelines configuration necessary to execute the shader in parallel, so new paradigms are still pending to come.

Some developments are already being done in what respect to general purpose computing in the GPU. For example OpenCL (Open Computing Language), described in http://www.khronos.org/opencl, aims are the design of an open standard for parallel programming of heterogeneous computational resources at processor level. More than just a programming language it includes an API, libraries and runtime system for software development.

The framework aspires at enabling portable and efficient access to general purpose parallel programming across CPUs, GPUs, Cell and ManyCores architectures for both HPC and commodity applications. The main key is to allow applications to use a host and one or more OpenCL devices as a single heterogeneous parallel computer system. Experienced programmers are supported throughout the process of developing general purpose algorithm without the necessity of mapping the algorithm onto architecture/platform specific features like 3D graphics API such as OpenGL or DirectX.

Also, other studies have been recently made, trying to uniform both the GPU and CPU calculations and the communications between them. The most relevant are the Zippy project (Fan, Qiu, and Kaufman 2008) and the CUDASA project (Strengert, Mller, Dachsbacher and Ertl 2008) that present a non-uniform memory access scheme between GPUS, the work done by Moweschell and Owens (2008), that implements a multi-GPU distributed shared memory architecture and DCGN (Stuart and Owens 2009), a message passing interface usable on systems with data-parallel processors. All of them ease the development and integration of parallel visualization, graphics, and computation modules on a heterogeneous cluster, but still lack of generalities to share a dynamic scene graph between the nodes of a GPU cluster.

3.3. Software for simulator developments

The core of a simulator is, at the end, a software application which computes the evolution of the simulated environment. There are a lot of libraries that allow developing such kind of applications; however it is difficult to choose which one better fits to our needs. Following a brief description is done of those that are the most active in the field of real-time simulation, divided in two: physic engines and scene graph management engines.

3.3.1. Physics Engines

Traditional physics engines are mainly based on solid-rigid simulations coupled with links. Their main drawbacks are that when developing the models, many simplifications have to be done in order to guarantee real-time and stability. Simulation aims, in this case, are limited to have a realistic behaviour of a few scene objects, the most important.

Nowadays, as viewed before, GPUs are in charge of executing the dynamical models, allowing the simulation of physically based visual effects, complex systems and the calculation of real-time collision detections. Nowadays everything is simulated, including plants movements, trees, atmospheric effects, particle systems, fire, fluids and deformable objects.

Programmers must choose one of the many physical engines available to develop the dynamical models of the scene objects. It is worth mentioning that if different engines are used for physics and graphics, special care must be taken when matching the dimensions and positions of the objects in the scene graph with those equivalents that the physical engine handles for calculating collisions. The whole information should be extracted from the 3D models and from the scene graph in order to avoid discrepancies between the calculated behaviour of the objects and what you see.

The most important physics engines that today are used in real-time simulations are the following:

ODE (http://www.ode.org): ODE is an open source library for simulating rigid body dynamics. It has advanced joint types and integrated collision detection with friction capabilities.

Bullet physics (http://www.bulletphysics.com): It is a professional open source collision detection, rigid body and soft body dynamics library. It is also integrated in MAYA and Blender3D.

Newton Dynamics (http://newtondynamics.com): It is an integrated solution for real time simulation of physics environments. The API provides scene management, collision detection and dynamic behaviour of objects.

Vortex (http://www.vxsim.com): It simulates the behaviour of vehicles, robotics, and heavy equipment in real-time synthetic environments for operator training and testing. It is integrated in OSG and VEGA.

PhysX (http://nvidia.com/object/physx_new.html): It delivers real-time, hyper-realistic physical and environmental gaming effects: explosions, reactive debris, realistic water, and lifelike character motion. Everything is computed in the NVidia GPU.

Havok FX (http://www.havok.com): It is a physic engine that runs entirely on the GPU and provides failure-free physic simulation using proprietary techniques for ensuring robustness, collision detection, dynamics and constraint solving. It provides integrated vehicle solutions and other tools available for simulating clothes, skeletons physics and rigid body destruction.

3.3.2. Scene graph management

Regarding scene graph management, there are many sdks dedicated to that. This topic is highly influenced by game technologies, where commercially available graphics engines use shaders executed in the GPU to increase performance. Mostly game engines also incorporate other features like sound, networking, artificial intelligence, collision, physics, etc.

Following are described the most important scene graph managers, game engines and cluster related applications that make possible the integration of a high performance real-time simulator.

OpenSceneGraph (http://openscenegraph.org): It is an open source high performance 3D graphics toolkit, used by application developers in fields such as visual simulation, games, virtual reality, scientific visualization and modelling. Written entirely in Standard C++ and OpenGL it runs on all Windows platforms, OSX, GNU/Linux, IRIX, Solaris, HP-Ux, AIX and FreeBSD operating systems.

OGRE (http://www.ogre3d.org): The Object-Oriented Graphics Rendering Engine is a scene-oriented, flexible 3D engine written in C++ designed to make it easier and more intuitive for developers to produce applications utilising hardware-accelerated 3D graphics. The class library abstracts all the details of using the underlying system libraries like Direct3D and OpenGL and provides an interface based on world objects and other intuitive classes.

Irrlicht Engine (http://irrlicht.sourceforge.net): It is a cross-platform high performance real-time 3D engine written in C++. It is a powerful high level API for creating complete 3D and 2D applications like games or scientific visualizations. It integrates all the state-of-the-art features for visual representation like dynamic shadows, particle systems, character animation, indoor and outdoor technology, and collision detection.

Delta3D (http://www.delta3d.org): It is a widely used and well-supported open source game and simulation engine. Delta3D is a fully-featured game engine appropriate for a wide variety of uses including training, education, visualization, and entertainment. Delta3D is unique because it offers features specifically suited to the Modelling and Simulation community such as High Level Architecture (HLA), After Action Review (AAR), large scale terrain support, and SCORM Learning Management System (LMS) integration.

Unreal Engine 3 (http://unrealtechnology.com): It is under the hood of the most visually intensive computer and video games on the market. It is available under license for PC, PlayStation3, and Wii. Its main features are: multi-threaded rendering, 64-bit high dynamic range rendering pipeline with gamma correction, dynamic composition and compilation of shaders, post-processing effects (ambient occlusion, motion blur, bloom, depth of field, tone mapping), artist-defined materials, dynamic fluid surfaces, soft body physics, deformable geometries, texture streaming system for maintaining constant memory usage, particle physics and skeletal animation.

Cry Engine 3 (http://www.crytek.com): It gives developers full control over their multi-platform creations in real-time. It features many improved efficiency tools to enable the fastest development of game environments and game-play available on PC, PlayStation® 3 and Xbox 360™. Its main characteristics are: road and river tools, vehicle creation, multi-core support, and multithreaded physics, deferred lighting, facial animation editor, dynamic pathfinding, rope physics, parametric skeletal animation and soft particle systems.

OpenSG (http://opensg.vrsource.org): It is a scene graph system to create real-time graphics programs, e.g. for virtual reality applications. It is developed following Open Source principles, LGPL licensed, and it can be used freely. It runs on Microsoft Windows, Linux, Solaris and Mac OS X and is based on OpenGL. Its main features are advanced multithreading and clustering support (with sort-first and sort-last rendering, amongst other techniques), although it is perfectly usable in a single-threaded single-system application as well.

VR Juggler (http://www.vrjuggler.org): It is a platform for virtual reality application development. This component allows a user to run an application on almost any VR system. VR Juggler acts as "glue" between all the other Juggler components. VR Juggler is scalable from simple desktop systems like PCs to complex multi-screen systems running on high-end work stations and super computers. Its development environment supports many VR configurations including desktop VR, HMD, CAVE™-like devices, and Powerwall™-like devices.

Equalizer (http://www.equalizergraphics.com): It is the standard middleware to create parallel OpenGL-based applications. It enables applications to benefit from multiple graphics cards, processors and computers to scale rendering performance, visual quality and display size. An Equalizer-based application runs unmodified on any visualization system, from a simple workstation to large scale graphics clusters, multi-GPU workstations and Virtual Reality installations.

4. BROWSER-BASED SIMULATION

Regarding games and simulation, one of the trends that emerged recently was the implementation of applications that are performed through a browser-based interface or even on hand-held mobile devices. The motivations were the necessity for extending the use of the simulators in order that they arrive to the maximum people. By means of the installation of an activex component, every simulator component (graphical models, dynamical models, textures and user interfaces) is downloaded automatically and the most important, is executed locally, without the necessity of installing anything, just the mentioned ocx. Additional advantages are that the software maintenance is done in a way completely transparent to the user (whenever you execute a new instance of the program) and the software protection can be performed using encrypted keys or restricting the clients by its internet address.

By the other side, the disadvantages that this methodology shows are that it is still very new, so there are still many software development kits quite immature, showing many instabilities and also poor performance when rendering the images. Furthermore, it is necessary to receive through the network connection the whole data that will be displayed, including textures and 3D models that can weigh hundreds of megabytes, often requiring a high bandwidth connection. Another aspect to take into account is that the server is a single point of failure, and can be also stressed if it receives many connections simultaneously. Security is another issue, since a failure when programming an interface can lead to intrusions in the computer system.

This kind of applications is used to implement cheap simulators that use a single computer with a single display, so the presence they provide is very low. They are similar to computer based training applications (CBT), but provide a real-time 3D interactive environment.

Anyhow, the browser-based game industry is currently very active and their technologies would be translated in a short period of time to the simulation market.

4.1. Browser-based simulation technologies

Following a brief description of the most important libraries and engines used in browser-based distributed applications is shown.

4.1.1. Java

Mostly browser-based applications use Java, from Sun Microsystems (http://java.com), as it is possible to integrate any Java application in the navigator through applets. However, as the Java virtual machine consumes a lot of cpu and many applications depend closely from the navigator they are being executed, they have relative small use in simulation applications. Anyway if it is only required a 2D interface, it is worth having a look to JavaFX (http://javafx.com), an open source application that allow making very impressive interfaces.

In order to eliminate the explorer dependencies, Sun introduced in 2001 a new technology called Java Web Start (JWS), allowing the execution of any Java application outside the browser. This is done specifying the compiled Java binary archives (Java ARchives) to be executed in the virtual machine inside a XML file called JNLP (Java Network Launch Protocol). Using this technology and whatever library that links Java applications with OpenGL it is possible to build a 3D simulator accessible by web. The client only downloads the JNLP file that is automatically executed, allowing the downloading of the simulator and if necessary also the Java engine required in a transparent way to the user.

In order to build the Java application that links with OpenGL, there are many libraries that can be used. However, the most important are:

JOGL (https://jogl.dev.java.net): The Java OGL project hosts the development version of the Java™ Binding for the OpenGL® API. It is designed to provide hardware-supported 3D graphics to applications written in Java, and integrates with the AWT and Swing widget sets allowing the implementation of GUIs.

LWJGL (http://www.lwjgl.org): The Lightweight Java Game Library is a solution aimed to enable commercial quality games to be written in Java. LWJGL provides developers access to high performance cross platform libraries such as OpenGL (Open Graphics Library) and OpenAL (Open Audio Library) allowing for state of the art 3D games and 3D sound. Additionally LWJGL provides access to controllers such as Gamepads, Steering wheel and Joysticks.

java3d (https://java3d.dev.java.net): The Java 3D API enables the creation of three-dimensional graphics applications and Internet-based 3D applets. It provides high-level constructs for creating and manipulation of scene graphs, 3D geometry and building the structures used in rendering that geometry. It is similar to the both described before, but it uses high level structures.

jME (http://www.jmonkeyengine.com): jMonkey Engine is a high performance scene graph based graphics API. Using an abstraction layer, it allows any rendering system to be plugged in. Currently, both LWJGL and JOGL are supported. jME also supports many high level effects, such as imposters (render to texture), environmental mapping, lens flare, tinting, particle systems, etc.

4.1.2. Adobe Flash

Flash (http://www.adobe.com) is commonly used to create animation, advertisements and various web page components, to integrate video into web pages, and more recently, to develop rich Internet applications. Flash can manipulate vector and raster graphics, and supports bidirectional streaming of audio and video. It contains a scripting language called ActionScript allowing the design of graphic user interfaces. In order to facilitate their design, the Adobe Flex SDK can be used. It comes with a set of user interface components including buttons, list boxes, trees, data grids, several text controls, charts, graphs and various layout containers.

Following, the flash based libraries and engines most commonly used for 3D graphics rendering through the navigator are briefly described:

PV3D (http://www.papervision3d.org): It is a technology that uses Flash and ActionScript allowing the renderization of 3D graphics through the web browser. It is very efficient, but still lacks the ability of using hardware accelerated rendering techniques (only hardware polygon rendering it is allowed). PapervisionX will be the next version of Papervision3D, built from the ground up based on Flash10's new 3D api, that will take full advantage of the 3D features of Flash Player 10.

Sandy3D (http://www.flashsandy.org): Sandy is an object oriented ActionScript 3D library, for programming 3D scenes for the Flash Player. The engine's capabilities are related to the performance of the virtual machine, which does not provide any native 3D nor hardware acceleration (although the Flash 10 player already introduces some 3D features). Sandy3D currently provides transparent materials, video textures, several shadow techniques and Collada, 3DS and ASE external formats are supported. However it lacks support for new materials with bump mapping and the use of multiple cameras.

Away3D (http://away3d.com): It is another real-time 3D engine for flash. It uses the hardware acceleration techniques included in Flash10 and supports object culling (using frustum calculations), interchangeable camera lenses to allow for different types of projection, collada bones animation, shadows, fog and optimized corrective z-sorting. It is being actively used by now and very well documented.

4.1.3. Adobe Shockwave

It allows publishing Adobe Director applications on the Internet and viewed in a web browser by anyone who has the Shockwave plug-in installed. The Shockwave file format is .dcr, and it is created by the Director authoring application. Features not replicated by Flash include a much faster rendering engine, including hardware-accelerated 3D, and support for various network protocols, including Internet Relay Chat. Due to its hardware rendering acceleration capabilities (OpenGL and directX), it is often used in online applications which require a very rich graphical environment. Online Learning tools which simulate real-world physics or involve significant graphing, charting, or calculation sometimes use Shockwave. It is worth seeing the many applications developed with Adobe Shockwave in http://www.shockwave3d.com

4.1.4. Other alternatives

Apart from the graphic engines related with Flash or Java reviewed before, there are still some other alternatives for building 3D browser applications that should be taken into consideration.

O3D (http://code.google.com/apis/o3d): It is an open-source web API for creating rich, interactive 3D applications in the browser. It is open source and it is developed by Google. It uses OpenGL or DirectX and depends strongly from JavaScript.

OSG4Web: (http://www.virtualrome.itabc.cnr.it): It provides a framework for in-browser OpenGL-based application wrapping. The framework allows the development of OpenGL and OpenSceneGraph based applications that open windows within the browsers, allowing JavaScript bidirectional interaction with surrounding page elements.

Unity3D (http://unity3d.com) Unity is a multiplatform game development tool, designed from the start to ease creation. Unity has a highly optimized graphics pipeline for both DirectX and OpenGL. It supports animated meshes, particle systems, advanced lighting, shadows, shaders, physics, sound and networking. The Unity Web Player enables you to view blazing 3D content created with Unity directly in your browser, and updates automatically as necessary.

Torque-3d (http://www.garagegames.com): This commercial application 3D game engine features multi-player network code, state of the art skeletal animation, seamless indoor/outdoor rendering engines, drag-and-drop GUI creation, a C-like scripting language and a built in world editor. A great thing with The Torque Game Engine is that you will receive all C++ source code to the engine, this makes it a lot easier to add any extra additions you might need for your game. Torque 3D supports all major browsers and operating systems, including IE7, FF3, OS X and Chrome. Games perform at 100% native speed, with no performance cost, completely in your browser.

5. SIMULATION CUTTING-EDGE RESEARCH

From the late 60's, training simulators have been used in many military and civil applications. Simulation technologies, therefore, are not new to the research community, however, as has been viewed in previous sections, new hardware advances and new user requirements have made that new programming paradigms be used when developing a simulator, both in the hardware and the software side.

Following are described some of the main problems that emerge when integrating a real-time training simulator and that still need to be improved. Also some important cutting-edge research lines are included.

- Simulation for training requires a way to know if the simulator reproduces accurately the real environment, and also if it provides a good instructional design making it suitable for training purposes. This is done by means of certification and homologation tasks. These are very well established in military applications but not yet in the civil world.
- Improvements in 3D sound are still to come. Auralization and real spatial sound systems (appropriate for caves) must be improved in current simulation systems. (See V. Pulkki 2002 and R. Furse 2009 and T. Lentz, D. Schröder, M. Vorländer and I. Assenmacher 2007)
- Risk prevention will be one of the main applications of future simulators. Avatars are already very well driven by realistic physical engines, but still we miss a collaborative behaviour to develop some risky tasks.
- New numerical methods for simulating multi-body systems will come, including multi-rate integrators (M. Arnold 2006), new distributed models (J Wang, Z. Ma and G. Hulbert 2003) and the use of level of details in physics (S.Redon, N.Galoppo, and M.Lin 2005).

6. CONCLUSIONS

Along this paper the most important technologies, both hardware and software, that are involved in the design of a real-time training simulator have been reviewed. Although the basic simulator building blocks are independent from the hardware used, the design of the software architecture will be determined by the number of displays that the simulator will provide to the user.

Serious games technologies and embedded display systems will lead the future high-end training simulation systems. Personal supercomputers are replacing mostly medium sized cluster-based simulators, however low cost web browser-based simulators will spread around, thanks to the new graphics engines able to render graphics using the GPU.

Physics engines already use parallel algorithms executed in a single or a cluster of GPUs, however new developments will have to come in order to solve the distribution of dynamic scene graphs driven by GPU computations into a set of several displays.

REFERENCES

M. Arnold. Multi-Rate Time Integration for Large Scale Multibody System Models. *IUTAM Symposium on Multiscale Problems in Multibody System Contacts*. p. 1-10. 2006

J. Bartlett. Programming high-performance applications on the Cell BE processor. Jan, 2007. https://www.ibm.com/developerworks/power/library/pa-linuxps3-1/

D. Blythe. The Direct3D 10 system. *ACM Transactions on Graphics*, vol. 25, nº 3, pp. 724-734, August 2006.

R. Borgo, K. Brodlie. State of the Art Report on GPU Visualization. School of Computing – University of Leeds. VizNET Report 2009.

C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon and J. C.Hart. The Cave Audio Visual Experience Automatic Virtual Environment, Communication of the ACM, vol. 35, no. 6, pp. 64-72, 1992.

Z. Fan, F. Qiu, and A. E. Kaufman. Zippy: A framework for computation and visualization on a GPU cluster. *Computer Graphics Forum*, 27(2), June 2008.

R. Furse. Building an OpenAL Implementation using Ambisonics. *AES 35th Int. Conference*, 2009, London, UK.

T. Lentz, D. Schröder, M. Vorländer and I. Assenmacher, Virtual Reality System with Integrated Sound Field Simulation and Reproduction, *EURASIP Journal on Advances in Signal Processing*, 2007.

E. Lindholm, J. Nickolls, S. Oberman, J. Montrym. NVIDIA Tesla: A Unified Graphics and Computing Architecture. *Micro, IEEE*, Vol. 28, No. 2. (2008), pp. 39-55.

A. Moerschell and J. Owens. Distributed texture memory in a multi-GPU environment. *Computer Graphics Forum*, 27(1):130–151, March 2008.

NVidia, 2008. Compute Unified Device Architecture Programming Guide Version 2.0, http://www.nvidia.com/object/cuda develop.html

V. Pulkki "Compensating displacement of amplitude-panned virtual sources." *Audio Engineering Society 22th Int. Conf. on Virtual, Synthetic and Entertainment Audio*. pp. 186-195. 2002 Espoo, Finland.

S. Redon, N. Galoppo, and M. Lin. 2005. Adaptive dynamics of articulated bodies. *In ACM SIGGRAPH 2005*. p. 936 - 945. 2005.

R. Rost, 2006 OpenGL Shading Language 2nd edn. Reading, MA: Addison-Wesley.

T. Sheridan (1992). Musings on telepresence and virtual presence. *Presence: Teleoperators and Virtual Environments*, 1, 120–126. 1992

M. Slater, P. Khanna, J. Mortensen and Y. Insu. Visual Realism Enhances Realistic Response in an Immersive Virtual Environment. *Computer Graphics and Applications*, IEEE Volume 29, Issue 3, May-June 2009. Page(s):76 – 84.

M. Strengert, C. Mller, C. Dachsbacher, and T. Ertl. CUDASA: Compute Unified Device and Systems Architecture. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV08)*, pages 49–56, 2008.

J. Stuart, J. Owens. Message Passing on Data-Parallel Architectures. *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*, pp.1-12, May 2009.

S. St-Laurent, 2005 The Complete Effect and HLSL Guide. Redmond, WA: Paradoxal Press

S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, A. Singh, T. Jacob, A10, S. Jain, A11, S. Venkataraman, A12, Y. Hoskote, A13, N. Borkar. “An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS”. *Digest of Technical Papers. IEEE International In Solid-State Circuits Conference*, 2007. pp. 98-589.

J Wang, Z. Ma and G. Hulbert. A Gluing Algorithm for Distributed Simulation of Multibody Systems. *Nonlinear Dynamics*, 34 (1-2). p. 159-188. 2003

AUTHORS BIOGRAPHY

Rafael J. Martínez is an assistant professor at the University of Valencia in the Computer Architecture knowledge area. He received a PhD on Computer Engineering from the University of Valencia in 1997. He teaches Operating Systems and Fault Tolerant Computing. He belongs to the Robotics Institute since eighteen years ago and is the head of the Simulation & Modelling Laboratory (LSyM). His main interest research topics are computer graphics and simulation for training. He has experience both in management, research activities and consulting. He has coordinated and participated in more than 20 research projects, mostly of them related with simulation activities and technology transfer to enterprises. He is the main inventor of two patents related with harbour crane simulators and has published more than 40 papers in several congresses and research periodicals.