Formal Integral Model for Creating Virtual Worlds

Gabriel López-García, Rafael Molina-Carmona, and Javier Gallego-Sánchez

Abstract—Virtual Reality (VR) and graphical systems have achieved a wide importance in many fields. The processes of rendering, the physics engines and the artificial intelligence modules have reached a certain maturity. Nevertheless, they are usually defined as separated modules, and there is not a complete model that allows the definition of a whole VR system, including complex interaction.
In this paper, an integrated model to create Virtual Worlds is presented. It integrates the diversity of the VR systems and the main modules: Graphics, Physics and Artificial Intelligence engines. Through the use of a formal grammatical model, a system with the following features is defined: independent from the interaction devices, flexible, complete and reusable. The main elements of the system are primitives, transformations and actors, which use events to communicate.
A case of study is also included: a simulation of fires in forests caused by lightning. This example illustrates the modelling, expansion, and portability of the system, as well as the independence from hardware and the main aspects of graphical representation.

Index Terms—Virtual Reality, Virtual Worlds generation, Formal model, Interaction

1 Introduction

In recent years, the development of Virtual Reality (VR) and graphical systems has been amazing. This progress has contributed to the creation of new ways to analyze and work with information, such as graphics, animations or simulations. Computer games industry is one of the elements which has contributed most to the advance of graphical systems and to the emergence of new VR systems. However, these developments have opened up new challenges.

One of the most important problems is the wide variety of visual devices (monitors, PDAs, cell phones, VR goggles), each one with different features. For that reason, it is necessary to develop adaptive systems which generate the images according to the specific characteristics of the device (resolution, size, ...).

While the processes of visualization and renderization of images have reached a certain maturity in the past decades, in the field of interaction a similar evolution has not been produced [10], [2]. Keyboard and mouse still dominate the world of interactive devices. High degree-of-freedom input devices are complex to manage, and other possibilities of interaction are only used in research and graphic art [10]. However, there has been tentative progress in recent years. Among them, it can be highlighted the input device of Wii [19].

It is necessary to develop a model which unifies the whole range of devices of interaction and visualization. For example, the action of moving an object in the virtual world, should not be connected with the push of a specific button on a mouse, a keyboard, or any other device. Instead, this diversity of events should be unified in one action: "move the object", removing this way all the specific characteristics of the different devices.

In the development of VR systems, there are three major modules which have had an uneven evolution. These are the Graphics, the Physics and the Artificial Intelligence (AI) engines. The first one handles the display of data into visual devices. The second one simulates all the physical processes, providing more coherent actions and visual effects to the virtual world. The last one tries to simulate an intelligent and independent behaviour of the actors in the scene.

There are multiple tools and solutions used to develop these three modules. However, there is not any unified criteria to deal with the different development challenges. There are some common points used to define certain structural designs, but there is no study about the connections between the different modules.

This paper presents a new model which integrates the diversity of the VR systems and the aforementioned modules (Graphics, Physics and AI engines). The major goal is to separate the scene definition from the hardware-dependent characteristics of the system devices. It uses a grammar definition which integrates activities, visualitation and interaction with users.

The following section describes the background of VR systems. Section 3 shows the main objectives to achieve. Section 4 defines the proposed model, and Section 5 presents a case of study. Finally, the conclusions and the lines for future work are shown.

2 Background

A VR system can be considered to be composed of three main modules: render system, physics engine and AI system. Next, the background of these modules are reviewed briefly.

• *The authors are with the Group of Informática Industrial e Inteligencia Artificial, Universidad de Alicante, Ap.99, E-03080, Alicante, Spain*
Tel.: +34 96 590 3400
Fax: +34 96 590 3902
E-mail: {glopez, rmolina, ajgallego}@dccia.ua.es

There are two main libraries which are used to develop components of graphics engines: Direct3D [12] and OpenGL [13]. Both libraries are defined as a layer between the application and the graphics card. Different systems which try to unify these two APIs (*Aplication Program Interfaces*) into a single interface have appeared. This is the case of OGRE [15] and VTK [16]. There are also libraries which are used to manage the interactive hardware. Two examples are SDL [14] and DirectX [12]. In general, they have similar features, but SDL is open source and cross-platform.

There are a large variety of tools which implement physics engines, such as Working Model [3] (used mainly in the area of education to perform simulations), Newton Game Dynamics [5], Physics Abstraction Layer (PAL) [11] (it is an open source and cross-platform API for physical simulation), Open Dynamics Engine (ODE) [6] (it is an implementation of a part of the PAL specifications), and so on. There are many other API among the proprietary software, such as PhysX [1] (developed by NVidia and used in the PlayStation 3) and Havok [8] (it belongs to the company Havok and it is implemented for Windows, MacOS, Linux, Xbox 360, Wii, Playstation3 and PlayStation Portable).

The last type are the tools used to develop the AI engine. There are very few libraries which can be used as a generic solution because they are designed specifically for particular applications. One of the reasons why the AI systems have been developed more slowly than the rest, may be that the effort being done to improve the graphical aspects of games is not comparable to the one done for the improvement of the intelligence of its characters [21]. Nevertheless, some types of systems can be found, such as evolutive algorithms or agent-based systems.

There are some tools which facilitate the use of evolutive algorithms, such as EO Evolutionary Computation Framework [17] (it is implemented in C++ and it has all the necessary to work with evolutive algorithms), CILib [18] (it is a development environment implemented in Java). There are also a lot of papers about genetic algorithms and about the development of evolutive algorithms for specific cases [22], [23], [24].

A wide research has been done in the area of agent systems, such as in [25], [26]. Currently, these systems are widely used in research for the development of AI, game characters (called *bots*, especially in first person games), social simulations and mobile robots [9], [27]. There are different frameworks which simplify the task of *bots* development, but they are often oriented to specific games: QuakeBot for Quake game and FlexBot for Half-Life game [21], [28], for instance. There are other libraries for the development of generic agents, such as Jade [20] (used to develop multi-agent-based applications).

3 OBJECTIVES

Our objectives are primarly focused on achieving a model which unifies all the modules of a VR system. This system must be independent of hardware devices, both visual and interaction devices. It means that one device can be replaced by another one, or even by a simulation, without affecting the internal mechanisms of the system. To achieve this, the following objectives are proposed:

1) Define a graphics engine which eliminates the diversity in visual devices. That is, only one description is needed to process the scene for any graphical device with a level of detail according to its features.
2) Provide flexibility to the graphics system in order to be able to change the elements of representation. This way, the system can provide different types of information without the need to modify the scene description.
3) Design a graphics system which can be used to extract and use the elements that are defined in the scene. In addition, it is intended to handle other non-visual elements like sound.
4) Define a physics engine which models the whole system activity and can be adapted to different hardware devices. If there are hardware components which implement physics algorithms, the system must exploit them, but if not, it should be implemented with software.
5) Design a physics engine which can provide the information needed by the graphics system at any time. It is intended to perform functions of simulation and analysis of the elements that compose the scene.
6) Provide to the physics engine the ability to modify the scene elements through simulations, but without the need to know the architecture of these elements in detail.
7) Unify the AI system with the physics engine. The limitations imposed by the physics engine must be considered by the AI system.
8) Make the interaction independent from the hardware. The system has to abstract the origin of the action and process the orders directly from the user. Therefore, the interaction could be handled as an internal process of the system (such as the calculation of collisions).
9) Design and implementing the elements so that they can be reused. If, for example, an item is designed for a specific virtual world, it should be able to be reused in another virtual world or VR application.
10) Control the activity of the system through events. However, the origin of these events could not be known.

For the fulfillment of these objectives, mathematical models are used to formalize the different components of the system, abstracting the characteristics of the three major modules of a VR system.

4 Model for Virtual Worlds Generation

A virtual world can be described as an ordered sequence of primitives, transformations and actors. The concept of primitive is considered as an action executed in a certain representation system. It is not, for instance, just a primitive to draw a sphere or a cube. Instead, it could implement any action which could be carried out in a representation device, such as a sound, for example. Transformations modify the behavior of primitives. Actors are the components which define the activities of the system in the virtual world. They will be displayed through primitives and transformations. To model the different actor's activities, the concept of event is used. Events are not generated by any certain input device. They are considered as an action that represents the activation of a certain activity and they can be run by one or more actors.

Each element in the scene is represented by a symbol from the so called "set of symbols of the scene". They are used to build strings to describe scenes. These strings are formed using a language syntax, which is usually presented as a grammar [4]. This grammar (hereafter denoted by M) is defined by the tuple $M = < \Sigma, N, R, s >$, where Σ is a finite set of terminal symbols used to build strings, N is a finite set of non-terminal symbols or variables which represents the substrings of this language, R is a finite set of syntactic rules used to describe how a non-terminal symbol can be defined as a function of terminal and non-terminal symbols (a syntactic rule is an application r: $W^* \rightarrow W^*$, where $W = \Sigma \cup N$) and $s \in N$ and it is the initial symbol or axiom of the grammar.

Considering the previous definition, M can be defined, in our case, as the following grammar:

1) Let $\Sigma = P \cup T \cup O \cup A^D$ be the set of terminal symbols, where:
   - P is the set of symbols which defines primitives.
   - T is the set of symbols which defines transformations.
   - $O = \{\cdot()\}$ is the set of symbols of separation and operation. () indicates the scope of the previous symbol, and $\cdot$ the concatenation of symbols.
   - A^D is the set of symbols which represents actors. D is the set of all the types of events which are generated by the system. Actors will carry out activities when they receive one of these events. For example, the actor a^d will carry out the activity d when it receives the event e^d.
2) Let N = {WORLD, OBJECTS, OBJECT, ACTOR, TRANSFORMATION, FIGURE } be the set of non-terminal symbols.
3) Let s = WORLD be the initial symbol of the grammar.
4) Grammar rules R are defined in Table 1.

TABLE 1
Grammar rules R

Rule 1.	**WORLD** $\rightarrow$ OBJECTS
Rule 2.	**OBJECTS** $\rightarrow$ OBJECT | OBJECT OBJECTS
Rule 3.	**OBJECT** $\rightarrow$ FIGURE | TRANSFORMATION | | ACTOR
Rule 4.	**ACTOR** $\rightarrow a^d$ (OBJECTS), $a^d \in \mathbf{A}^D$
Rule 5.	**TRANSFORMATION** $\rightarrow t$ (OBJECTS), $t \in T$
Rule 6.	**FIGURE** $\rightarrow p^+$, $p \in P$

A string $w \in \Sigma^*$ is generated by the grammar M if it can be obtained starting with the initial symbol WORLD and using the rules of the grammar. The language generated by the grammar M is the set of all the strings which can be generated by this method. This language is called $L(M)$ and is defined as:

$$L(M) = \{w \in \Sigma^* \mid \text{WORLD} \xrightarrow{*} w\} \quad (1)$$

The grammar M is a context-independent grammar (or a type-2 grammar, according to the Chomsky hierarchy). Therefore, there is a procedure which verifies if a scene is correctly described or, in other words, if a string belongs to the language $L(M)$ or not. For example, it can be determined that the string $a_1^d(p_1 \cdot p_2) \cdot p_3 \cdot t_1(p_1 \cdot p_2) \in L(M)$ where $p_i \in P$, $t_i \in T$, $a_i^d \in A^D$, $d \in D$, but, on the contrary, $a_1^d \cdot p_1 \notin L(M)$.

Apart from the language syntax, it is necessary to define the functionality of the strings, that is, the semantics of the language. It can be denoted through three methods: operational, denotational and axiomatic. The first one uses abstract machine language operations. The second one uses mathematical functions to describe the meaning of the strings. The third one defines the meaning of the string through mathematical logic terms. In our case, the denotational method is used.

4.1 Semantics of the Language $L(M)$

In this section, a mathematical function is assigned to each rule of Table 1.

4.1.1 Semantic Function for Primitives (Rule 6)

Rule 6 defines the syntax of a figure as a sequence of primitives. Primitive's semantics is defined as a function α. Each symbol in the set P carries out a primitive on a given geometric system G. Therefore, the function α is an application defined as:

$$\alpha : P \rightarrow G \quad (2)$$

So, depending on the definition of the function α and on the geometry of G, the result of the system may be different. G represents the actions which are run on a specific geometric system.

A usual example of geometric system are graphical libraries, such as OpenGL or Direct3D. In this case, each symbol of P would have associated an action of the

render system. For instance, given the symbol of a sphere ($sphere \in P$), it would implement a function which draws this object in the corresponding graphics system (Direct3D or OpenGL):

$$\alpha_{opengl}(sphere) = glutSolidSphere$$
$$\alpha_{direc3d}(sphere) = drawSphereDirect3D$$

These functions are examples of visual geometric systems. But this definition can also be extended to non-visual geometric systems. For example, a non-visual system could be the calculation of the boundaries of a figure, which could be used to calculate collisions between elements. So, a new function $\alpha_{bounds}(sphere) = boundSphere$ can be defined for the same alphabet P, where "*boundSphere*" is a function which calculates the limits of spheres.

Previous examples show that the function α provides the abstraction needed to homogenize the different implementations of a render system. Therefore, only a descriptive string is needed to run the same scene on different graphics systems.

So far, only Euclidean geometric systems have been considered. But, the function α has no restrictions on the geometric system that can be applied as long as there is a definition for this primitive. Thus, it could also implement a sound, the movement of a robot, the reflection of a material and so on. Moreover, the definition of the function α could also describe systems to write different file formats (VRML, DWG, DXF, XML, etc.), to send strings in a network, etc.

4.1.2 Semantic Function for Transformations (Rule 5)

Rule 5 defines the syntax for transformations. The scope of a transformation is limited by the symbols "()". Two functions are used to describe the semantics of a transformation. These functions are:

$$\begin{array}{l} \beta : T \rightarrow G \\ \delta : T \rightarrow G \end{array} \quad (3)$$

β represents the beginning of the transformation. It is carried out when the symbol "(" is processed and it has to take into account the previous symbol of the set T. The function δ defines the end of the transformation which has previously been activated by the function β. It is carried out when the symbol ")" is found.

These two functions have the same features that the function α, but they are applied to the set of transformations T, using the same geometric system G.

A new function φ is defined using the set of primitives P, the set of transformations T and the functions α, β and δ. Given a string $w \in L(M)$ and using only symbols of P and T, this function φ runs the sequence of primitives and transformations in the geometric system G. It is defined as:

$$\varphi(w) = \left\{ \begin{array}{ll} \alpha(w) & if\ w \in P \\ \beta(t); \varphi(v); \delta(t) & if\ w = t(v) \wedge v \in L(M) \wedge \\ & \wedge\ t \in T \\ \varphi(s); \varphi(v) & if\ w = s \cdot v \wedge s, v \in L(M) \end{array} \right\} \quad (4)$$

One of the most important features of this system is the independence from a specific graphics system. The definition of the functions α, β and δ provides the differences in behaviour. These functions encapsulate the implementation details which may differ for different systems, such as a render system, a geometric calculation system, a command execution system in a production chain, and so on. Therefore, in the development of strings to define virtual worlds is not necessary to consider the special features of the geometric system. In addition, these strings may be used on all the systems which implement these functions.

4.1.3 Semantic Function for Actors (Rule 4)

Rule 4 of the grammar M refers to actors, which are the dynamic part of the system. The semantics of the actor is a function which defines its evolution in time. For this reason, the semantic function is called *evolution function* λ and it is defined as:

$$\lambda : L(M) \times E^D \rightarrow L(M) \quad (5)$$

where E^D is the set of events for the device D and the strings $w \in L(M)$ fulfill the syntactic rule 4. The function λ can be interpreted as the evolution of the actor in time. λ has a different expression depending on its evolution over time. However, the general expression can be defined as:

$$\lambda(a^d(v), e^f) = \left\{ \begin{array}{ll} u \in L(M) & if\ \ f = d \\ a^d(v) & if\ \ f \neq d \end{array} \right\} \quad (6)$$

It means that an actor $a^d(v)$ is transformed into another string u when the actor is prepared to respond to the event d (that is, $f = d$) and it remains unchanged when it is not prepared to respond to this event.

The result of the function λ may contain or not the own actor, or it can generate other event for the next stage (or the next frame). So, an actor can become a generator of events. The new events generated by actors are accumulated to be passed to the next stage of the system evolution. In this way, a feedback process is generated. Such process is very important for the physics engine and the AI system.

The function λ can define a recursive algorithm which, given a set of events and a string, describes the evolution of the system at a given point in time. This function is called *function of the system evolution* and it is represented by η. Its definition requires a set of events $e^i, e^j, e^k, \ldots, e^n$, which in short is denoted as e^v, where $v \in D^+$. This algorithm is defined as:

$$\eta(w,e^v)=\left\{\begin{array}{lll} w & if & w\in P \\ t(\eta(v,e^v)) & if & w=t(v) \\ \prod\limits_{\forall f\in v}(\lambda(a^d(\eta(y,e^v)),e^f)) & if & w=a^d(y) \\ \eta(s,e^v)\cdot\eta(t,e^v) & if & w=s\cdot t \end{array}\right\} \quad (7)$$

The operator $\prod\limits_{\forall f\in v}(\lambda(a^d(\eta(y,e^v)),e^f))$ concatenates the strings generated by the function λ. It is important to note that the strings which are only composed of transformations and primitives are not processed. Therefore, these strings are not part of the system evolution. This fact is important for the system optimization, because the evolution is carried out only for the strings which contain actors, that is, the dynamic part of the system.

The functions λ can be classified in accordance with the strings which are obtained during the evolution of the system. One important type of λ functions are those which return strings with only primitives and transformations, because they are essential to define the visualization of strings. As it was explained before, the functions α, β and δ do not admit strings with actors. Therefore, actors must be first converted into strings made up only of primitives and transformations. This conversion is carried out by a type of function λ called *visualization function*, or function θ, which is defined as:

$$\theta : L(M)\times E^V \rightarrow L(M') \quad (8)$$

where $V\subseteq D$, E^V are events created in the visualization process, and $L(M')$ is the language $L(M)$ without actors. The expression of the visualization function is defined as:

$$\theta(a^d(v),e^f)=\left\{\begin{array}{lll} w\in L(M') & if & f=d\wedge d\in V \\ \epsilon & if & f\neq d \end{array}\right\} \quad (9)$$

There are small differences between λ and θ. The first one is that θ returns strings belonging to $L(M')$. The second is that θ returns an empty string if the event does not match. So, the actor will not have representation for that event.

The type of event V (visualization event) does not correspond to any specific input device. It is used to create different views of the system. For example, to design a system which filters out certain elements in order that they become invisible, it is only necessary to avoid reacting to those events. Moreover, they can also be used to change the visualization type, depending on the type of window, or on the output device and its features.

As with the function λ, an algorithm is defined for θ. It returns a string $z\in L(M')$, given a string $w\in L(M)$ and a set of events e^v, where $v\in V^+$ and $V\subseteq D$. This function is called *function of system visualization* π and it is defined as:

$$\pi : L(M)\times E^V \rightarrow L(M')$$

$$\pi(w,e^v)=\left\{\begin{array}{lll} w & if & w\in P^+ \\ t(\pi(y,e^v)) & if & w=t(y) \\ \prod\limits_{\forall f\in v}(\theta(a^v(\pi(y,e^v)),e^f)) & if & w=a^v(y) \\ \pi(s,e^v)\cdot\pi(t,e^v) & if & w=s\cdot t \end{array}\right\} \quad (10)$$

This function is the same as the function η unless that λ is replaced by θ, and that events can only belong to V. If the event does not belong to V, the function π returns an empty string.

4.1.4 Semantic Functions for OBJECT, OBJECTS and WORLD (Rules 1, 2 and 3)

The semantic function of these rules breaks down strings and converts them into substrings. The functions specified above are carried out depending on whether the symbol is an actor, a transformation or a primitive. Therefore, the semantic function of WORLD is a recursive function which breaks down the string of the WORLD and converts it into substrings of OBJECTS. Then, these substrings are in turn broken down into substrings of OBJECT. And for each substring of OBJECT, depending on the type of the object, the semantic function of ACTOR, PRIMITIVE or TRANSFORMATION is run.

4.2 Activity and Events

In VR systems some mechanisms must be established to model the activity of the graphics system. This activity can appear during the action of an actor, between several actors with a certain relationship (for example, an actor which is composed of other actors -which are their children- and there is some activity between them), and between input devices and scene's strings. Each type of activity is different, but they have some features in common, and they are going to be established.

As it has been said before, actors activity is carried out when a certain type of event is produced with some specific data. But not all activities are directly run when an event is received. There are events that only run its activity when certain conditions are satisfied, depending on the actor state. This condition is not defined at the object that runs the activity, but at the event. The following event definition is established: *e_c^d is defined as an event of type $d\in D$ with data e, which is carried out only when the condition c is fulfilled.*

For simplicity, when there is not any condition, that is, *c=TRUE*, the event is represented by e^d.

Let us notice that the origin of events is not identified. In fact, the origin is not important, but the event type and its data.

4.3 Input Devices and Event Generators

It is necessary to establish the independence between the system and the input devices. So, the events needed to make the system respond to a set of input devices must be defined. It means that there may be actions on input devices (buttons, mouse wheel, movements and so on) which generate one or more events, and others which do not generate any. The input devices need not be only physical devices, they may be software devices which, depending on the system state, may or may not generate events. For example, a mouse could be used as input device to move the objects in the scene. However, there has not to be a mouse event associated to the object; instead, there should be a generic event of movement associated to the object and the mouse has an event generator which creates events of movement. On the other hand, an example of software device may be an algorithm for detecting collisions. In this case, an event can be generated when a collision is detected.

A new function called *event generator* is defined as: *Let $C^d(t)$ be a function which creates events of type d at the time instant t, where $d \in D$ and D is the set of event types which can be generated by the system.*

In the previous definition, it should be noticed that events are generated in the time instant t. It is due to synchronization purpose. It is also possible that C does not create any event at a given moment.

This definition makes the system independent from the input devices. Thus, event generators can be created for a mouse, a VR glove, a keyboard or any other input device, and all of them can generate the same type of events. The most interesting fact about the implementation of the event generator is that the device-dependent code is encapsulated and separated from the rest.

One problem is that different event generators can create the same type of events. So, a priority order among event generators must be established to avoid ambiguities. Given two generators C_i and C_j which create the same event, if $i < j$, then the events generated by C_i will have a higher priority. For example, the following priority order could be established among input devices: $1st$ joystick, $2nd$ mouse and $3rd$ keyboard. This ambiguity is due to the fact that the system must be implemented for all the possible devices. If there are several devices, only one prevail. The priority order can be established using different criteria: the level of immersion into the system, the ease of use and so on.

The process which obtains the events produced by input devices and their associated event generators is defined as follows: *Let C^* be the set of all the event generators which are associated with input devices.*

The function $e(z, e^i)$ which concatenates all the events in a list is defined as:

$$e(z, e^i) = \left\{ \begin{array}{ll} z \cdot e^i & if \quad e^i \notin z \\ z & if \quad e^i \in z \end{array} \right\} \tag{11}$$

The function $E(C^*, t)$ collects all the events from all the generators and accumulates them in a list. If the event e^i already exists, then it is not inserted in the list. This function is defined as:

$$E(C^*, t) = \left\{ \begin{array}{ll} e(z, C_i(t)) & if \quad z = E(C^* - C_i, t) \\ \epsilon & if \quad C^* = \emptyset \end{array} \right\} \tag{12}$$

4.4 System Algorithm

Once all the elements involved in the model have been defined, the algorithm which carries out the entire system can be established. It defines the system evolution and its visualization at every time instant 't' or frame. The algorithm of virtual worlds generation is defined as:

Step 1.	$w = w_o$
Step 2.	$t = 0$
Step 3.	$e^* = E(C^*, t)$
Step 4.	$e^v =$ events of e^* where $v \in V^+$
Step 5.	$e^u = e^* - e^v$
Step 6.	$w_{next} = \eta(w, e^u)$
Step 7.	$v = \pi(w, e^v)$
Step 8.	$g = \varphi(v)$
Step 9.	$w = w_{next}$; $t = t + 1$
Step 10.	If $w = \epsilon$ then go to step 12
Step 11.	Go to step 3
Step 12.	End

Where:

- D = { Set of all the types of possible events in the system }.
- V = { Set of all the types of visual events } where $V \subseteq D$.
- C^* = { All the event generators which generate events of type D }.
- g is the output device.
- e^* are all the events generated by the system.
- e^v are all the events from visual devices. These events are the input of the visual algorithm π.
- e^u are all the events from non-visual devices. These events are the input of the evolution algorithm η.
- w_o is the intial string of the system.

The first two steps initialize the system. The initial state w_o is introduced to the system and the first frame is set to 0.

Steps 3, 4 and 5 manage the system events. In Step 3, generators are called to insert all the events in a list e^*. In Steps 4 and 5, events are divided into visual events (e^v, which are the input of the visual algorithm π) and non-visual events (e^u, which are the input of the evolution algorithm η).

In Step 6, the evolution algorithm η is called with the current string w and the non-visual events (e^u). The output is the string for the next frame or instant.

In Steps 7 and 8, the visualization of the system is performed. First, actors are transformed into primitive and transformations. Next, the visualization algorithm π is called with the visual events (e^v). Finally, the function φ is called to display the current state of the system into the render engine g.

In Step 9, the next iteration is prepared. w is assigned to w_{next} and the time instant (or current frame) is increased in 1.

Step 10 checks if the current string satisfies the condition of completion, that is, if the following string is empty, then the algorithm ends (Step 12). Otherwise, the algorithm goes to Step 3. Therefore, to finish the main algorithm the function η must just return an empty string. This empty string can be generated by a special event which is created when the system must stop.

It must be noticed that Step 6 can be exchanged with Steps 7 and 8, because they do not share any data. This feature is very important for the parallel implementation of the algorithm. So, Step 6 and Steps 7 and 8 can be divided into two parallel tasks. This type of optimization has a significant impact on the final graphics system, which leads to a faster system performance.

The diagram of the virtual world generation algorithm is shown in Figure 1.

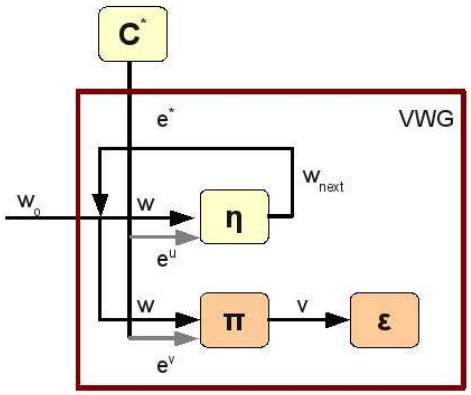


Fig. 1. Virtual worlds generation algorithm

This formalization of the system has two main consequences. First, the scene definition is separated from the hardware-dependent characteristics of components. The functions α, β and δ provide the independence from the visualization system, and the event generators provide the independence from the hardware input devices. Secondly, due to the fact that there is an specific scheme to define the features of a system, the different system elements can be reused easily in other areas of application.

5 Case of Study

This example is an application that simulates fires in forests caused by lightning. It could be used to analize the best distribution of trees to reduce the number of burnt trees [9]. It is important to notice that we are not actually interested in the problem of fire simulation, but rather in defining a graphic system using the proposed model and all the concepts presented in this paper. Therefore, it studies the ease of system modeling, expansion, portability and system independence from hardware and other aspects of graphical representation.

5.1 Problem description

Let (i, j) be the cells of a 2D grid representing a squared world. In each cell, a tree can grow with a given probability g. Bolts of lightning can also fall on a (i, j) position with a probability f. In this case, if there is a tree, it will burn as well as the trees around it, in a chain reaction. An example can be seen in Figure 2.

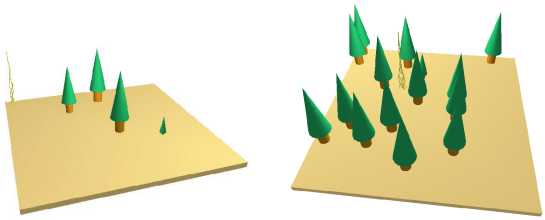


Fig. 2. Examples of different simulation states

To model this example with the Virtual World Generator (VWG), four main elements need to be defined. First, it is necessary to define events for the system to run actor activities. These events include the necessary information to develop their activity. Second, it is necessary to design event generators that separate the interactive system from the origin of the events (mouse, keyboard, and so on). Third, it is necessary to define the actors included in the scene. These elements are required for the system to evolve over time. Last, the graphic primitives must be defined to show the visual elements on the render system.

5.2 Formalization of the system

Following the steps described above, the different elements of the system are defined.

5.2.1 Events

Events are used to produce the necessary activity of system. They are described by an identifier and a set of data. Their aim is to produce certain activities of the actors that compose the scene. The events defined for this example are shown in table 2.

5.2.2 Event generators

The next step is to define the event generators:

1) Time event generator: This generator, named by C_{time}, has the responsibility of animating the system. Every time instant t, it generates an event e^t that is usually associated with some type of primitive transformation to change the appearance of an actor over time.
2) Forest event generator: This generator produces events to create trees and lightning. Taking into account the previous specification, trees are created with a probability g and lightning with a probability f. This generator is defined as:

$$C_{forest} = \begin{Bmatrix} e^c & with\ probability\ g \\ e^f & with\ probability\ f \end{Bmatrix}$$

TABLE 2
Events definition

Type of event	*Meaning*	*Associated data*
t	Event generated every time instant t	Increasing time since previous event
c	Create a tree at a given position	Position (i,j) where the tree is created
f	Create a bolt of lightning at a given position	Position (i,j) where the bolt of lightning is created
e	Eliminate the tree at a given position	Position (i,j) where the tree is eliminated
b	Burn the tree at a given position	Position (i,j) where the tree to be burned is
v	Draw using a graphics library (e.g. OpenGL)	Void

3) Visualization event generator: This generator is necessary to visualize all the elements in the scene. This means that every instant time a drawing order is received, the generator captures this order and produces an event, sending the draw elements of the scene to the graphics system. Its mathematical expression is given as:

$$C_{visualization} = \{e^o \ each \ drawing \ cycle\}$$

5.2.3 Primitives and transformations

The primitives and transformations that make up the scene are shown in tables 3 and 4:

TABLE 3
Definition of primitives

Primitive	*Description*
TR	Draw a tree
TR_b	Draw a burning tree
FA	Draw a bolt of lightning
BO	Draw a grid of NxN

TABLE 4
Definition of transformations

Transformations	*Description*
$D_{i,j}$	Translate (i,j)
S_s	Scale (s)

The functions α, β and δ define these primitives and transformations and they are implemented with a graphics library (in this case OpenGL). This way, the number of primitives and transformations can be easily extended just implementing them in the functions α, β and δ.

5.2.4 Actors

The last step is to specify the actors which compose the dynamic part of the system. As explained in previous sections, an actor is defined by its evolution function λ. If it has a graphical representation, it also uses the function π to generate the primitives and the transformations needed to create that representation. Table 5 shows the actors defined for this example and their evolution function.

TABLE 5
Actors defined for this example

Actor	*Function* λ
B Represents the forest	$\lambda(B^{cfe}, e^i) = \left\{ \begin{array}{ll} TG_c^{t=1} \cdot B^{cfv} & i = c \\ F^{t=1} \cdot B^{cfv} & i = f \\ B^{cfe} & i = e \\ B^{cfe} & i \neq c, f, e \end{array} \right\}$
TG Represents the growth of a tree	$\lambda(TG^t, e^i) = \left\{ \begin{array}{ll} TG^{t+1} & i = t \wedge t+1 \leqslant N_{frames} \\ T^b & i = t \wedge t+1 > N_{frames} \\ TG^t & i \neq t \end{array} \right\}$
T Represents a tree	$\lambda(T^b, e^i) = \left\{ \begin{array}{ll} TB^{t=1} & i = b \\ T^b & i \neq b \end{array} \right\}$
F Represents the animation of a bolt of lightning	$\lambda(F^t, e^i) = \left\{ \begin{array}{ll} F^{t+1} & i = t \wedge t+1 \leqslant N \\ \Delta e^b & i = t \wedge t+1 > N_{frames} \\ F^t & i \neq t \end{array} \right\}$
TB Represents a burning tree	$\lambda(TB^t, e^i) = \left\{ \begin{array}{ll} TB^{t+1} & i = t \\ \Delta e^e & i = t \wedge t+1 > N_{frames} \\ TB^t & i \neq t \end{array} \right\}$

Actor B^{cfe} represents the forest. This actor generates a tree or a bolt of lightning if it receives the event e^c or the event e^f respectively. The event e^e only changes the internal state of the actor B^{cfe}. It frees its position (i, j), so a new tree may grow in the same position.

TG^t represents the growth animation of a tree. When the animation ends, that is $(t + 1 > N_{frames})$, the actor T^b that represents the new tree is placed in the position (i, j). F^t is similar to TG^t, unless it creates a bolt of lightning. When the lightning animation ends, it generates an event e^b that burns the tree. This is an example of an actor who becomes an event generator.

T^b represents a tree. This actor waits until it receives the event e^b and then it turns itself into the actor TB^t. TB^t represents the animation of a burning tree. When the animation ends, that is $(t+1 > N_{frames})$, it generates an event e^e to free the space occupied by the tree.

Table 6 shows the definition of the drawing function

π. This function is used to obtain the set of primitives and transformations representing the visual aspect of an actor, which is shown on the display.

TABLE 6
Definition of the drawing function

Actor	*Function* π
B^v	$\pi(B^v, e^i) = \begin{cases} BO & i = v \\ \epsilon & i \neq v \end{cases}$
TG^v	$\pi(TG^v, e^i) = \begin{cases} D_{(i,j)}(S_s(TR)) & i = v \\ \epsilon & i \neq v \end{cases}$
T^v	$\pi(T^d, e^i) = \begin{cases} D_{(i,j)}(TR) & i = v \\ \epsilon & i \neq v \end{cases}$
F^v	$\pi(F^d, e^i) = \begin{cases} D_{(i,j)}(FA) & i = v \\ \epsilon & i \neq v \end{cases}$
TB^v	$\pi(TB^v, e^i) = \begin{cases} D_{(i,j)}(S_{s^{-1}}(TRB)) & i = v \\ \epsilon & i \neq v \end{cases}$

S represents a growing scale factor which depends on the current state of the actor TG. This state is modified by the event e^t. It causes the effect of tree's growth. The scale value is defined by the expression $s = t/N$, where N is the total number of frames and t is the current instant of time. However, in the case of TB the scale factor is defined as a decreasing scale s^{-1}. It causes the effect of a burning tree, which is getting smaller.

The animation of a bolt of lightning is implemented by an algorithm which depends on the instant of time t. In general, actors' animations always depend on t. It modifies the current state of the actor to change its representation and thus perform the animation.

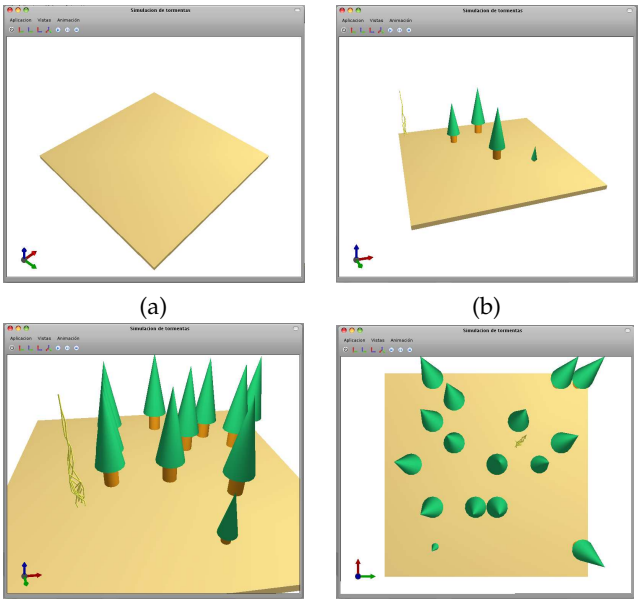
Finally, the initial string is defined. It is the first string to be processed in the algorithm. From this initial state, the system evolves over time. In this example, it is defined as: $w_0 = B^{cfe}$

5.3 View generated by VWG

This section shows the application implemented for the case of study. Figure 3(a) is the initial state of the application, without any frame calculation. However, all the elements (actors, static figures and generators) are ready and with their initial configurations. Figure 3(b) shows the animation running. Figures 3(c) and 3(d) show different system views, they are obtained using the interface's buttons. As can be seen, the application has different menus and action buttons. Theses buttons are used to configurate the visualization view, to manipulate the simulation and to other general options, as for example, close the application.

6 Conclusions and Future Work

A new model of Virtual Worlds generation has been presented. The major goal of this system is to separate the scene definition from the hardware-dependent characteristics of the system devices.



(a) (b) (c) (d)

Fig. 3. (a, b) Initial state of the application, (c) Lateral view, (d) XY-view

First, a language which details the elements of the system using context-free grammars is defined. Moreover, new functions used to represent the evolution of the elements in time and to extract their graphic representation are established. This representation is sent to visual devices using functions which separate the hardware-dependent implementation of the visual device from the graphic description of the element. This way, it removes the dependence of the system from the graphics devices.

The separation of system definition from input devices is made by event generators. These generators create a layer between the input hardware and the representation of the system. The actions generated by input devices are linked using the model of events.

In general, the whole model tries to separate the hardware-dependent parts of devices from the formal definition of a VR system. In order to carry out this separation, different mathematical models are used to transform device actions, both visual and input devices, into more general actions. The system must be able to identify these general actions regardless of the origin of the action. It is achieved using abstraction.

This model has several advantages: firstly, input devices can be replaced by other devices or even software-simulated devices. Secondly, the different elements of the system can be easily reused. Thirdly, the representation of the elements can be visual or non-visual, or even it may be different depending on the display device or the user needs. Thus, if a device has specific characteristics, the representation can always be adapted to the device. Finally, actors can interact between them by sending events to each other.

Using the proposed system, new physics engines which use the elements of the scene can be easily de-

signed. It is achieved by setting up different types of event generators. Depending on the physical features of the device, it activates the activity of the actor needed to react to that physical process. For example, if an actor has to react to collisions, the event generator of this type calculates the collisions between elements by extracting the scene geometry from the graphics engine (it uses the implementation of the functions α, β and δ to calculate the bounding box of the elements). Then, it generates the events needed to react to such collisions. This event generator could be implemented with hardware if the system allows it.

The AI engine can be implemented in the evolution function of the actors. Actors use these functions to make decisions according to their current status. Moreover, events carry out activities which change the status and the behaviour of actors. As actors can generate events, it is possible to implement the feedback processes used in AI. The integration between the physics engine and the AI engine is guaranteed because both engines are connected by events.

The model presented in this work is currently under development. It is pretended to continue developing several issues. One point to investigate - which is also introduced in this article - is the optimization of the algorithm and its parallelization. The definition of the system through strings facilitates the possibility of parallel algorithms. From another point of view, strings represent the states of the system and its evolution. This evolution may change through mutations, so different evolutive solutions may be conceived to design new systems. We also consider the possibility of a new type of events which are activated with a certain probability. For example, if an actor is defined as r^{ab} and it receives the event e^a, then the function associated with this event will be carried out only with a certain probability.

In conclusion, the main aim has been to design a reusable and generic system, which can be easily increased, adapted, and improved. It is also important that the core of the system (the evolution in time) is independent of its representation and the elements which it interacts with.

REFERENCES

[1] *PhysX by AGEIA*, http://physx.ageia.com
[2] David J. Kasik, William Buxton, D. R. Ferguson, *Ten cad challenges*, IEEE Computer Graphics and Applications 25 (2005), 81–90
[3] *Working Model: Simulación de Sistemas*, http://www.design-simulation.com
[4] Davis Martin D., Sigal R., Weyuker E. J.: *Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science*, 2nd ed. San Diego: Elsevier Science, 1994
[5] *Newton Game Dynamics*, http://www.newtondynamics.com
[6] *Open Dynamics Engine*, http://www.ode.org
[7] Wikipedia - Physics engine: http://en.wikipedia.org/wiki/physics_engine
[8] Havok: http://www.havok.com
[9] John H. Miller S. E. P.: Complex Adaptative Systems. Princeton University Press, 2007
[10] Joshua Strickon J. A. P.: Emerging technologies. Siggraph (2004)
[11] PAL: Physics Abstraction Layer: http://www.adrianboeing.com/pal/
[12] Pgina oficial de DirectX: http://www.microsoft.com/windows/directx/default.mspx
[13] Pgina oficial de OpenGL: http://www.opengl.org/
[14] Simple DirectMedia Layer (SDL): http://www.libsdl.org
[15] OGRE 3D: Open source graphics engine: http://www.ogre3d.org
[16] The Visualization ToolKit (VTK): http://public.kitware.com/vtk
[17] EO Evolutionary Computation Framework: http://eodev.sourceforge.net
[18] CILib (Computational Intelligence Library): http://cilib.sourceforge.net
[19] Wii-Nintendo: http://www.nintendo.es
[20] Jade - Java Agent DEvelopment Framework: http://jade.tilab.com
[21] Laird J. E.: Using a computer game to develop advanced ai. Computer 34 (7) (2001), 70–75
[22] Georgios N. Yannakakis John Levine J. H.: An evolutionary approach for interactive computer games. In Proceedings of the Congress on Evolutionary Computation (2004), 986–993.
[23] Chris Miles Juan Quiroz R. L. S. J. L.: Co-evolving influence map tree based strategy game players. IEEE Symposium on Computational Intelligence and Games (2007), 88–95
[24] Robert G. Reynolds Ziad Kobti T. A. K. L. Y. L. Y.: Unraveling ancient mysteries: Reimagining the past using evolutionary computation in a complex gaming environment. IEEE transactions on evolutionary computation 9 (2005), 707–720
[25] Wooldridge M.: Agent-based software engineering. IEEE Proceedings Software Engineering 144 (1997), 26–37.
[26] Wood M. F., DeLoach S.: An overview of the multiagent systems engineering methodology. AOSE (2000), 207–222
[27] Kenyon S. H.: Behavioral software agents for real-time games. IEEE Potentials 25 (2006), 19–25
[28] Aaron Khoo R. Z.: Applying inexpensive ai techniques to computer games. IEEE Intelligent Systems 17(4) (2002), 48–53

Gabriel Lpez-Garca received the B.S. degree in Computer Science from University of Valencia, Spain, in 1990. After that, he received the M.S. degree in Computer Science Engineering from the University of Alicante, Spain, in 2005. He is a PhD student in the Department of Computer Science and Artificial Intelligence in the same university. He has worked as software engineer since 1990 in a CAD-CAE software company. His research interests are in computer graphics and AI Systems, particularly in visualization, virtual reality, interactive system, Multi-Agent system and social behaviour.

Rafael Molina-Carmona is a professor in the Department of Computer Science and Artificial Intelligence in the University of Alicante, Spain, which he joined in 1995. He received a B.S. & M.S. degree in Computer Science from the Polythecnical University of Valencia, Spain, in 1994, and the PhD from the University of Alicante in 2002. His research interests are in computer graphics, computer aided design and manufacturing, artificial vision, 3D reconstruction and virtual reality.

Javier Gallego-Snchez received the B.S. & M.S. degree in Computer Science Engineering from the University of Alicante, Spain, in 2004. He is a PhD student in the Department of Computer Science and Artificial Intelligence in the same university. His research interest is in computer graphics, particularly in artificial vision, visualization, 3D reconstruction and virtual reality.