2nd Hellenic Conf. on AI, SETN-2002, 11-12 April 2002, Thessaloniki, Greece,
Proceedings, Companion Volume, pp. 3-14

Virtual Director: Visualization of Simple Scenarios

Konstantinos Manos, Themis Panayiotopoulos, George Katsionis

Knowledge engineering Lab, Department of Informatics,
University of Piraeus, Piraeus, 185 34, Greece
manos_konstantinos@yahoo.com, themisp@unipi.gr,
gkatsion@singular.gr,

Abstract. The last years have been a period of significant change and evolution in the Intelligent Virtual Environments (IVE) field, which combines Artificial Intelligence and Virtual Reality. Recently, there have been some efforts on developing interactive storytelling systems with dynamically generated behaviors of the participating virtual characters. Such systems however, lack the capability of a declarative definition of the virtual environment. In this paper we present Virtual Director, an application that combines Intelligent Text Processing and a Virtual Reality Module in order to visualize scenarios. Given a scenario written in almost plain English and based on a Knowledge Base consisted of a Virtual Library, where Items and Rules are described, the Director tries to create a Virtual Scene and produce a Virtual Animation.

1. Introduction

The last ten years have been a period of significant change and evolution in the field of Artificial Intelligence and Virtual Reality Modeling. The use of artificial intelligence techniques for the production of multimedia application software has greatly expanded. Moreover, human – computer interaction environments demanded a more straightforward data input than mouse and keyboard can provide.

With the use of Artificial Intelligence, computers may “understand” (apart from programming languages) natural languages giving users a more convenient way to communicate. This led the technology to the development of Natural Language Understanding (NLU) Models and Modules. The applications that use NLU are divided into two major classes: text-based and dialogue-based applications, [1].

Apart from understanding a written text, a way for visualizing the information gathered is needed. Here the scepter goes to Virtual Reality Modeling and consequently to VRML (Virtual Reality Modeling Language). VRML enables you to create dynamic worlds and sensory-rich virtual environments, including the ability to animate objects, play sounds/video and allow user interaction, [2]. Given these features, plus the ability to run over the Internet, VRML provides the perfect tool for developing Virtual & Dynamic Worlds.

The field of Intelligent Virtual Environments (IVE) was born from the combination of Artificial Intelligence and Virtual Reality, [3]. These environments offer a much more sophisticated approach to user interaction. The user is placed opposite to a

dynamical and interactive environment, which has the ability to "understand" him and modify itself in such a way, that it will be easier to the user to define his actions and commands, [4,5,6]. In this way IVE covers the new needs of Human Computer Interaction (HCI).

Quite recently, attempts have been made for a text based animation generation. The 'Papous' project, [7], is a virtual storyteller, a synthetic character that tells stories in an expressive and believable way. It reads a text enriched with control tags, which customize its visual behavior. 'Papous' however, does not understand the story, it just presents it enriching it with gestures and vocal effects.

In the context of Intelligent Virtual Environments, NL is used as a tool to extract "commands" out of given text or speech. There are numerous projects, which try to accomplish this, such as the Ossa, [15], Jacob, [13] as well as the DenK System, [16]. In the NAUTILUS project, [20], one can use ones' own voice to activate and control many equipments, instead of using a keyboard or mouse. Moreover there are several models being defined which aim in making easier and more flexible the extraction of these "commands", [14], [19].

In this paper we present Virtual Director, a digital scenario visualization system which taking as input a text written in "plain" English it translates it to a three dimensional animation. After "reading" and "understanding" the scenario, the Director creates the objects (actors – items) on the scene and then constructs the relevant animation. Moreover, if it encounters unknown words, which are defined inside the scenario, it expands its Knowledge Base, thus "learning on the job".

2. System Architecture

The application consists of three discrete parts. The *VD-Base*, where everything is stored in a structure "close" to natural language, trying to simulate the way relevant information is stored in our minds. The *Virtual World,* is where everything is at the final stage of process and is ready to be represented in a three dimensional environment. Finally the *Virtual Director*, which is the module that takes care of the actual process of combining the "Knowledge" stored in *VD-Library* and the information given from the Scenario, thus creating the final "Virtual Animation".

The whole project is written in Delphi. Every system's entity is implemented by a Delphi class (object), which has the relevant published properties and services. The various objects of the application can use these services to achieve many different tasks. Using Object Oriented Programming enhances the flexibility of each module while maintenance costs are reduced dramatically, [8].

2.1. Virtual Director Base (VD-Base)

VD-Base is a special type of Knowledge Base, which represents and stores the knowledge (and maybe experience) an actual director has in his mind when starting working on a new scenario. It consists of a Library, which is a container for a set of Items and Rules.

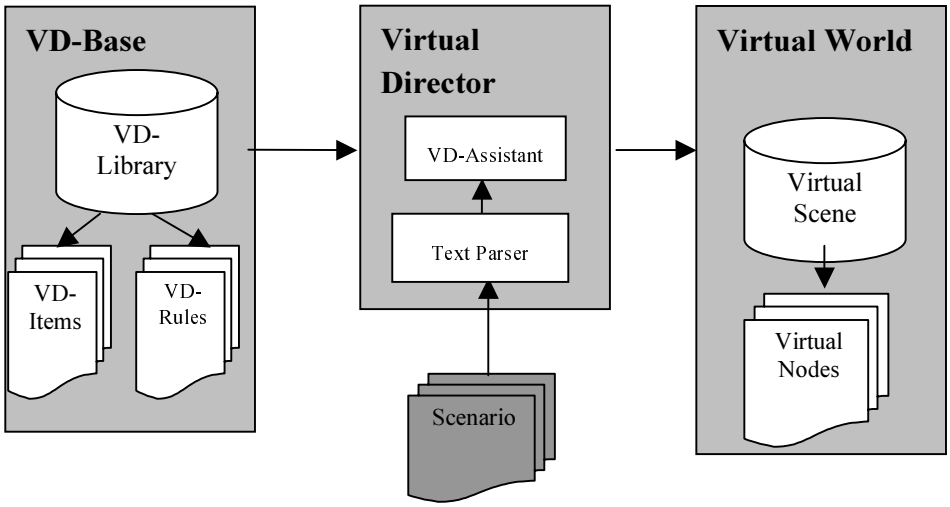

Fig. 1. System Architecture.

2.2. Virtual Director's Library (VD-Library)

Apart from being a simple container, VD-Library supports a wide variety of services helpful for the Director. These services are meant to hide from the Text Parser (TP) the Virtual meaning of the words and expressions it encounters. Let's take for example the next simple sentence: "There is a red box". While parsing, the TP needs not know what the 3D representation of the object "box" is, or what's the impact of assigning the adjective "red" to the specific object. It needs only to know that the first one is a noun and the second one is an adjective. Furthermore, VD-Library maintains a dictionary of known words and expressions, on which the Director is based.

2.3. Virtual Director's Item (VD-Item)

VD-Items are primitive objects, which represent the definition of Virtual Nodes. The information kept in the structure consists of a descriptive word (which is always a noun), and the Virtual Data needed to construct the 3D representation of it accompanied by the default properties that initialize the node's form (e.g.: size, color, texture etc). Examples of VD-Items are: box, sphere, table, chair, man etc.

VD-Items can be created in two different ways. One way is to directly import into the Library any VRML object previously created in any environment. You may, for example, create a table in 3D Studio Max then export it as a VRML File and import it into the Library. That way you need only supply a descriptive word, representing the noun with which the new item is linked. This process simulates the

way we learn new words as kids, we are first shown an item and then we are given its name.

Another way is to create a VD-Item from a Virtual Node from the Scene. This is supported so that the Director can learn and expand its knowledge base while reading a scenario. If a new item is described in the scenario then after creating the relevant node, a new Item is created from it and is added to the Library. For example think of the next simple scenario: “Maria is a white woman”. First the director creates a node “woman” and it applies a new characteristic (“white”) to it. Then it reaches an unknown word (“Maria”), since the rule “is” represents the definition of a new description, a new VD-Item is created under the description “Maria” and its properties are initialized from the relevant properties of the node “Maria”. Finally the new Item is added to the Library, so that it can be used when another reference is found (e.g. “Maria is tall”). If the rule “is” is encountered and the object that it refers to is known, then instead of defining a new Item, a new characteristic is added to the node in reference. For example, for the sentence “The box is red”, a box would be created and the “red” rule would be applied to it.

2.4. VD-Rule

VD-Rules represent actions the Director can take while working on a scenario. These actions when interpreted may result to the production of an animation or the modification of some Virtual Nodes’ characteristics. Examples of VD-Rules are: red, big, bouncing, on, in front of, morning, dark etc. To fully define a rule you need to supply the following properties:

- Description: this can be a sole word or an expression. For example: “on”, “in front of”, “rotating around”
- Rule Type: this can be Adjective, Verb, Numeric, or General. The first three categories are self-explanatory. General Rules are whatever rules do not fall into any of the previous categories.
- Position Type: defining whether the rule is Standalone, Pre/In/ Postfix. This property shows the number and the position of any arguments expected by the rule. For example, the rule “red” is a Postfix rule needing one argument after it (“a red *box*”), on the other hand an Infix rule would be the “on” rule (*something* “on” *something*).
- Animation Rule Flag: defining whether the result of applying this rule onto the scene objects creates any kind of animation. This results to specific actions the director needs to do before applying the rule.
- Script: Using a simple script language you can specify what is the impact of applying the rule to either a group of Virtual Nodes or to the Scene itself (a rule is applied directly to the Scene if its Rule Type is Standalone, for example “morning”).

During the direction, the Director (depended on the Rule Type and Position Type) gathers the necessary objects, which will be passed as arguments when the rule is applied. After all the arguments are collected, the Animation Rule flag is checked. Then it is decided whether the rule can be applied immediately or should be put in a stack and wait for some preconditions to be met. For example consider the next

scenario: “There is a small bouncing ball on the table”. The rule “bouncing” is an animation rule and should be applied after the rule “small” and that’s because the animation path has to be calculated when all the objects have reached the final form. If we first apply the “bouncing” rule and then the “small” rule, the resulting animation path would not be correct and we would see the ball bouncing over the table without touching it since the calculations would have been made on a normal sized ball and not on a small one.

A couple of very powerful characteristics emerge from the flexibility VD-Rules gain through the use of the scripting language. First of all it is easy to combine VD-Rules together by using logical operators (AND/OR/XOR) on their scripts, thus giving the ability to the Director to create new rules by using the ones it has in its VD-Library.

3. Virtual Director and Virtual Worlds

Virtual Director is the core of the application. It is consisted of all the necessary modules, needed for the actual work of directing to take place. This includes a Text Parser and a VD-Assistant.

3.1. Text Parser (TP)

While reading the text, TP constantly checks the VD-Library for the information it has on the words it encounters. Remember the fact that the actual dictionary that the Director “understands” is stored (and exported) from the VD-Library. For each word TP encounters in the text, it asks the VD-Library the type and the object itself. Now depended on the type a different procedure is followed. There are three types of known words: Rule, Item and Reserved. “Reserved” words represent rules that cannot be described via the simple scripting language of the director. These are words that have a special role during the text parsing such words are: “the”, “is”, “and”, “that”, “these” etc. The other two types we have already explained. We are going to describe in more details the way TP works using a simple scenario later on.

3.2. VD-Assistant

The TP doesn’t directly communicate with either the VD-Library or the Virtual Scene. It first contacts the VD-Assistant and asks the relevant information. It is the VD-Assistant that decides whether it should retrieve the information from the Scene or the Library. To better understand the functionality of the VD-Assistant we should give an example. Lets check the following scenario: “There is a table. There is a vase on the table”. Reading the first sentence the director asks from the VD-Assistant a “table”. VD-Assistant checks the Library and retrieves a VD-Item matching the description. Then the director finishes the sentence so it gives the nodes it has created to the VD-Assistant to put them on the scene. Next the second sentence follows, so the director again asks for a table, but this time the reserved rule “the” follows. This

makes the VD-Assistant to first check on the scene if there is a matching node already placed on it. If so, it retrieves this node and passes it to the TP, else it checks the VD-Library. That way we end up with one table on the scene and one vase on it.

The VD-Assistant's main reasons of existence are: first to hide as many "Virtual aspects" as it can from the TP, and second to provide a mid-level process between TP and the rest of the application's parts, giving the whole application a more flexible structure. With the use of the VD-Assistant as a "referee" the role each part is playing during the direction phase is strictly defined.

3.3. Virtual World

Virtual World represents the actual environment where the story described in the scenario is taking place. It has a Virtual Scene on which the Director is called to apply his version of the scenario, creating new Virtual Nodes, animation and sound modules, lights, background etc. Every item put on the scene has a VRML representation, whether it is an object or an animation or special effect module. Consequently Virtual Scene can be exported as a VRML World and be viewed using an external VRML Browser.

Virtual Scene can either be empty at the start of the direction, or you can previously initialize it using an external Virtual Scene Editor. This application provides you with every tool you may need to fully construct a background scene for the Director to work with. The need for such an application arouse from the fact that however well formed and detailed a description may be you can never construct the exact duplicate of the scene. Let's take for example the next description: "There are two boxes on the table". There are numerous ways you can place the two boxes on the table, so you need a tool to specify the exact form of the background scene. Thinking of an actual director's job, this is exactly the way they work, they first construct the actual environment and then put the actors inside and start getting everything in motion.

4. Additional Virtual Director Modules

Apart from the basic modules, that have already been described, many smaller but vital modules have been developed. These modules take care of various processes, taking the load off the core modules. Some of them are:

- **String Tokenizer / Parser**: This module is used to parse written text. It provides a set of functions that can divide the text into sentences, divide the sentences into tokens (based on a token list) and search for words or expressions throughout the text. This module is used: from the director to parse the scenario before passing the words to TP, from the VD-Rules' Script Interpreter to parse the relevant script, and from the VRML & Poser Parser to parse the given text file.

- **VRML Parser**: This module is used to read a text file where a VRML World is stored and then extract all the information needed to create a VD-Item out of it. As previously mentioned, it uses the String Tokenizer to parse the text.
- **Poser Parser**: This module works like the VRML Parser, but it parses Poser Files instead of VRML. Poser is a graphical environment for the construction of Virtual Worlds.
- **Mathematical Expression Evaluator**: This is a very useful and vital module of the application. It can take as an input a string representing a mathematical expression and calculate the result. For example: "4+3/2-5*3+(8/2-1)" would give –6,5. This module is mainly used from the VD-Rules' Script Interpreter to evaluate any expressions found inside the script. The implementation of the module is based on a string tokenizer and an automaton, [8].
- **VD-Rules' Script Interpreter**: We have already mentioned that the rule's functionality is described mainly by the use of a simple script. In order for the application to use this script an interpreter had to be implemented. This interpreter reads the script and evaluates any mathematical expressions found using the Expression Evaluator. It then applies the actions, described in the script, on the various Virtual Nodes changing their published properties (e.g. color, position, orientation etc).

Moreover some external third-party controls have been used. These controls are:

- **VRML Active-X Control**: This control can be embedded in the application's forms and make the presentation of VRML Worlds possible. [10]
- **Text Editor Control**: This control is an enhanced text editor. It includes a string tokenizer, used by the application to highlight special words (e.g. unknown, reserved etc), while the scenario or the script of a VD-Rule is being edited. [9]

Almost every module and functionality we have described is currently implemented in Virtual Director. Some of them are fully implemented and some partly. A simple user-interface provides an easy way to input the scenario and the parameters the Director needs to get the job done. (Fig. 2) A text editor is provided for writing the scenario along with an object inspector to check whether the direction's output is the intended one.

Moreover, you can find many services to help you customize the Knowledge Base of the Director. There is a form with which you can create/import/modify VD-Items. You may choose and modify every virtual property they have, this includes: size, position, rotation center, transparency, color, and texture mapping. From the same form you may call the VRML or Poser parser to import any external files.

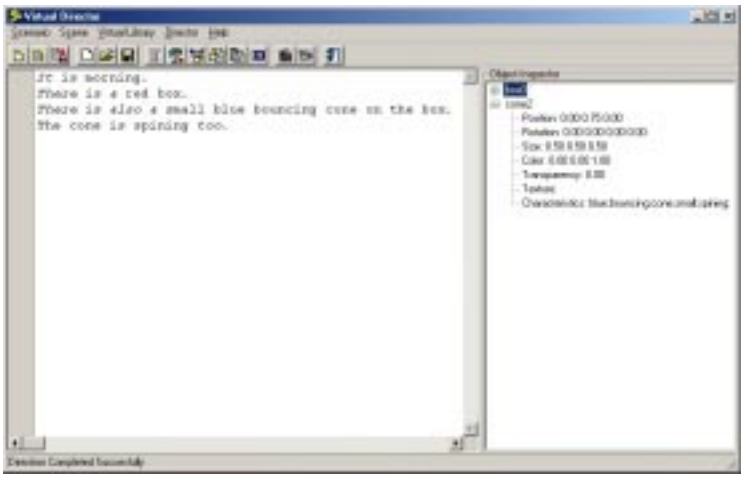
Fig. 2. Main Form

There is also a form for the creation or modification of VD-Rules. There you may choose the description of a rule, the rule type, the position type, whether the rule takes part in an animation and give the relevant script describing the rule's functionality.

The whole library can be exported and imported, giving the opportunity to the user to share his knowledge base with others thus enriching it.

Finally, a form with a VRML Client ActiveX Control [10] is provided to view the Virtual Animation created after the completion of the direction. To better understand the functionality and the use of the Virtual Director, let's present a couple of simple scenarios.

5. Illustrative Examples

5.1. A "bouncing" model

Let's give a simple example of a scenario and follow the direction steps. The words written in *italic* represent words unknown to the Director.

"It is morning. There is a red box. There is also a small blue bouncing cone on the red box. The cone is spinning too."

The sentences are parsed in the order they are written. So the direction starts with the phrase "It is morning". The standalone rule "morning" is applied to the scene creating the relevant background effect. There is nothing else in this sentence so we proceed to the next one. Reading from right to the left the direction does the following:

“box”: TP asks from the VD-Assistant the object representing the word “box”. It returns a VD-Item so a new temporary Virtual Node is created and kept in memory.

“red”: Now the VD-Assistant returns a postfix rule so it is immediately applied to the current temporary Virtual Node, changing it’s color and appending “red” to its characteristic list.

“a”: is an unknown word and it is not processed

“There is”: It is a reserved rule so TP gives the temporary nodes to VD-Assistant to be inserted into the scene.

Until know we have a red box inside our scene. We proceed to the following sentence.

“box”: like the first time, a temporary node is created.

“red”: like the first time, the rule is applied to the temporary node.

“the”: Virtual Scene (through the VD-Assistant) is queried for an item matching the characteristics of the temporary node (“box”, “red”). One is found so the temporary node is discarded and its place takes the node from the scene.

“on”: VD-Assistant returns an Infix rule. Because all the arguments needed for its appliance are not still gathered, the rule is placed inside a stack to be processed later.

“cone”, “bouncing”, “small”: result to the creation of a cone and the corresponding rules “small” and “bouncing” are applied to it.

“a”, ”also”: are unknown words so they are not processed.

“There is”: TP flushes the rule stack, processing the rule “on”. Now it has both arguments needed so it applies the rule and empties the rule stack. Then it processes the animation stack. There it finds rule “bouncing”. Now all the object of the scene has reached its final form so the animation path, described by the rule, is calculated correctly.

At this point the director has successfully created a new item (“cone”), has placed it on another object of the scene and finally has created an animation path for it. With the same logic the last sentence will be parsed changing the characteristics of the “cone” and creating another animation effect.

After the direction, a Virtual Scene would have been created, consisted of two Virtual Nodes. One represents a box while the other a cone. The background of the scene will have changed to represent the fact that it is morning. An animation clock and path will have been created and they will be linked with the cone to create the “bouncing” and “spinning” effects. Furthermore the characteristics of both nodes will have changed accordingly to suite the descriptions provided (“red”, “blue”, “small”). The result can be seen in the snapshots taken.

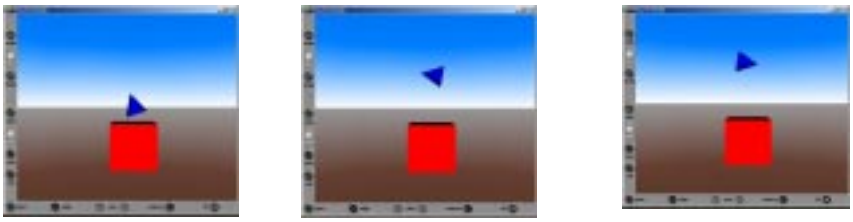

Fig. 3. A ‘Bouncing Model’

5.2. A Solar Model

The scenario for this model is the following:

"It is dark. We are in space. There is a big yellow self-illuminated sphere. The sphere is producing light. A spinning blue sphere is rotating around the yellow sphere. Another small green sphere is rotating around the blue sphere."

Following the steps described in the previous example a Virtual Scene is created, containing 3 spheres. The three spheres have different colors and sizes, while two of them have an animation path applied to them. You can see screenshots of the outcome in the following snapshots.

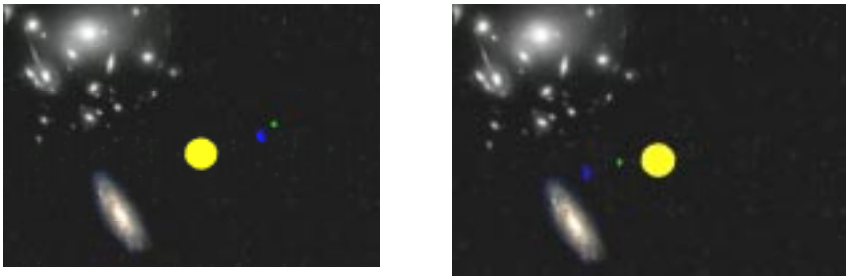

Fig. 4. A 'Solar Model'

6. Future work : Virtual Actors and Intelligent Agents

It is relatively easy to expand VD-Rules so that they can be applied to other VD-Rules (not only to VD-Items). For example, if we had the following scenario: "I can see a man running fast", then after applying the rule "running" to the VD-Item "man", the Director should apply the rule "fast" to the rule "running" resulting to the acceleration of the animation. That way we can end up we a great number of different effects/animations/etc from a very small number of "primitive" VD-Rules.

Such dynamic behavior is needed for the introduction of Virtual Actors in the scene. Virtual Actors are intelligent agents capable of understanding commands given to them using Natural Language. Such entities take as input the environment created with the help of the Assistant and the acting commands of the director. With this information the agents can "decide" and form a plan on how to act-behave, while at the same time following the commands of the Director. Due to the fact that VD-Rules have the ability to interact and drastically change anything inside the environment, by implementing a virtual agent with the ability to use them, we can create much more dynamic environment.

We are also planning to expand the Assistant using Intelligent Agent technology. Such an Intelligent assistant, could easily expand its KB by "learning" how to produce better animations and backgrounds, based on the corrections the actual director does on its choices. Moreover, it could expand its knowledge on different items, visual and audio special effects, new animations etc. This could be done either by manually upgrading its Knowledgebase or by "teaching" it, using NL to describe those new effects. The same could apply to the actors.

The current core of the application could be made the agent's brain. For the Knowledge Base, the use of VD-Library (with VD-Items and VD-Rules) is a very effective one. Moreover the NL Parser (the future version of the Text Parser) will be the same for both the Assistant and the Actors. So, it's very easy to define an abstract agent's brain and to create from it the assistant and the actors (using OOP inheritance). More modules could be implemented to enhance the whole project, like a behavior module for the actors, making each actor a unique entity.

7. Conclusions

By combining Scenario parsing and Virtual Reality modeling a brand new area of research and development unfolds. Virtual Director tries to use these new technologies and unite them under a common application, aiming the construction of Virtual & Interactive Films.

The originality of the Virtual Director is that it uses NL as the source of it's information and not only as a dialog interface, [18]. Through the given text, the Director, not only understands specific commands, but has also the ability to dynamically update its Knowledge Base by interpreting various definitions it encounters. Thus, the Director uses NL not just as a communication tool but as a teaching tool too. Furthermore, the Director does not "specialize" it's Text Processing in a given field, but uses an appropriate vocabulary of the objects and their activities, appearing on a scenario. This is much different than many NLP approaches which need to narrow down their vocabulary in order to make the "understanding" process easier, [21].

From the NLP point of view, Virtual Director falls into the class of text-based applications, since its input is a written text-scenario. It also uses some type of special purpose Knowledge Base, which combines some simple conceptual modeling along with some linguistic and visualization information. Finally, it contains a visualization module for the presentation of the scenario in a 3D scene.

Virtual Director can be very useful for educational purposes. It can be used as a tool for teaching a new language to small children, since they can learn while "playing", [12]. Moreover, Virtual Director could become a Director's Virtual Assistant. The actual director can use it to generate the environment of any given scene she/he is about to shoot. With the use of Natural Language to communicate with the Virtual Assistant one could easily create all the background one needs for the scene.

Acknowledgements

Special thanks to G. Gravos, S. Marntirosian (Singular Research & Development Department) as well as S. Vosinakis (University of Piraeus) for their contribution and support.

REFERENCES

[1] James Allen, “Natural Language Understanding”, Benjamin/Communings Publishing Company, 1995, ISBN 0-8053-0334-0

[2] Andreal L.Ames & David R.Nadeau & John L.Moreland, “VRML 2.0 Sourcebook”, Wiley, ISBN 0-471-16507-7

[3] "*Intelligent Virtual Environments*", T. Panayiotopoulos, S. Vosinakis, N.Avradinis, Seminar/Workshop on Intelligent Agents and Virtual Reality, Artificial Intelligence Human Networks, Athens University of Business and Economics, pp.3-8, June 29, 2001.

[4] "*Multi-agent Systems as Intelligent Virtual Environments*", G. Anastassakis, T. Ritchings, T. Panayiotopoulos, KI 2001 : Advances in Artificial Intelligence, F. Baader, G. Brewka, T. Eiter (Eds.), Lecture Notes in AI, (LNAI 2174), Springer-Verlag, pp.381-395, 2001.

[5] "*Virtual Agent Societies with the mVITAL Intelligent Agent System*", G. Anastassakis, T. Panayiotopoulos, T. Ritchings, Intelligent Virtual Agents, A. de Antonio, R. Aylett, D. Ballin (Eds.), Lecture Notes in AI, (LNAI 2190), Springer-Verlag, pp.112-125, 2001.

[6] "*SimHuman : A Platform for Real-Time Virtual Agents with Planning Capabilities*", S. Vosinakis, T. Panayiotopoulos, Intelligent Virtual Agents, A. de Antonio, R. Aylett, D. Ballin (Eds.), Lecture Notes in AI, (LNAI 2190), Springer-Verlag, pp.210-223, 2001.

[7] "*Papous: The Virtual Storyteller*", A. Silva, M. Vala, A. Paiva, Intelligent Virtual Agents, A. de Antonio, R. Aylett, D. Ballin (Eds.), Lecture Notes in AI, (LNAI 2190), Springer-Verlag, pp.171-180, 2001.

[8] Robert Sedgewick, “Algorithms”, Addison Wesley, Second Edition 1998, ISBN 0-201-06673-4

[9] Danny Thorpe, “Delphi Component Design”, Addison Wesley Developer’s Press, 1997, ISBN 0-201-46136-6

[10] “Cortona VRML Client”, Parallel Graphics, http://www.parallelgraphics.com

[11] “SynEdit”, Open Source Editor, http://synedit.sourceforge.net

[12] Lester, J.C., Converse, S.A., Kahler, S.E., Barlow, S.T., Stone, B.A., Bhogal, R.S.: The persona effect: affective impact of animated pedagogical agents. In: CHI’97, Proceedings (1997) pp. 359-366

[13] “Jacob – An Animated Instruction Agent in Virtual Reality”, T.Tan, Y.Shi, and W.Gao (Eds): ICMI 2000, LNCS 1948, pp 526-533, 2000.

[14] “A Task Oriented Natural Language Understanding Model”, T.Tan, Y.Shi, and W.Gao (Eds): ICMI 2000, LNCS 1948, pp 526-533, 2000.

[15] “Ossa – A Conceptual Modelling System for Virtual Realities”, H. Delugach and G. Stumme (Eds): ICCS 2001, LNAI 2120, pp. 333-345, 2001.

[16] “Multi Modal Cooperation with the DenK System”, Harry Bunt, Rene Ahn, Robbert-Jan Beun, Tijn Borghuis, and Kees van Overveld

[17] “An Agent model for NL dialog interfaces”, F. Giunchiglia (Ed.): AIMSA ’98, LNAI 1480, pp. 14-27, 1998

[18] “An Agent model for NL dialog interfaces”, F. Giunchiglia (Ed.): AIMSA ’98, LNAI 1480, pp. 14-27, 1998

[19] “Connectionist Analysis and Creation of Context for Natural Language Understanding and Knowledge Management”, P. Bouquet et al. (Eds): CONTEXT’99, LNAI 1688, pp. 479-482, 1999.

[20] ”NAUTILUS (Navy AUTomated Intelligent Language Understanding System) - a general purpose natural language processing system”, http://www.aic.nrl.navy.mil/~severett/vr.html

[21] ”Learning to Generate CGs from Domain Specific Sentences”, H. Delugach and G. Stumme (Eds): ICCS 2001, LNAI 2120, pp. 44-57, 2001.