Multi-user interactions in the context of concurrent virtual world modelling

Patrice Torguet, Frédéric Rubio,
Véronique Gaildrat, René Caubet

I.R.I.T., Paul Sabatier University,
118, Route de Narbonne, 31062 Toulouse Cedex, France
e-mail: {torguet | rubio}@irit.fr
URL: http://www.irit.fr/ACTIVITES/EQ_SYNTHIM/syntima.html

Abstract: Distributed virtual reality, thanks to newly proposed paradigms, offers interesting perspectives to virtual world modelling. Our main interests are in applications which allow users to edit and deform existing shapes. This paper presents a concurrent virtual world modelling application base on those ideas. As well as a first implementation based on VIPER (VIrtuality Programming EnviRonment) a generic distributed virtual reality development platform.

Keywords: Distributed virtual reality, Interactive modelling, Declarative modelling, Free-form deformations.

1. Introduction

First virtual reality (VR) applications were mainly architectural walk-through with symbolic representations of more complex elements (e.g. a cube instead of a house). As soon as basic virtual reality paradigms were defined, software designers turned their attention towards interactivity problems.

Therefore, recently, proposed interfaces were improved thanks to technology advances (workstation power, new and reliable peripherals...) and virtual reality interactivity studies. Moreover, distributed virtual reality, adds multi-user interaction and co-operation to virtual environments. Those aspects are the main interests of our current works.

We focus on distributed virtual reality applied to interactive world modelling. Our final objective is to offer high level metaphors for co-operative world modelling. We wish to provide users a set of tools used to interact in a virtual environment in the most natural way: they can model shapes by doing natural gestures they would make in a real situation (i.e. hammer an object, dig a hole...).

In this paper we present an application where users model virtual worlds by interacting with modelling tools. Section 2 is an overview of VIPER [Torguet95], our generic distributed virtual reality programming platform. Section 3 presents the modelling application. Section 4 details the distribution of such an application based on VIPER. Finally we conclude our paper with some perspectives on either our modelling application and VIPER.

2. VIPER: a VR application programming environment

Developing distributed virtual environments is a complex time consuming task. In order to develop such environments the programmer has to be proficient in network, graphics, device handlers and user interface programming. Moreover, network based programs are inherently more difficult to program and debug than sequential ones. In order to simplify this task, we have developed VIPER (VIrtuality Programming EnviRonment) which enables the rapid and easy development of distributed VR applications.

2.1. The virtual environment model

VIPER is aimed for the design of every application based on a virtual environment which can be modelled by exchanges (symbolised by **stimuli**) between **entities**, in a **virtual universe** (Figure 1).

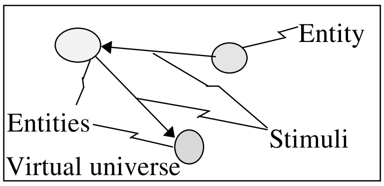

Figure 1: Virtual environment structure

The entity paradigm allows uniform management of virtual worlds scenery, virtual objects and **avatars** (an entity which behaves as an interface between a user [Mouli93], an application [Snowdon94] [Carlsson93] or a robot and the virtual universe). Entities are autonomous and own a set of attributes and behaviours. They are conceptually grouped in **families** (a set of entities which own the same attributes and behaviours). Our system is best suited for homogeneous virtual environments (few families made of many instances).

The purpose of this structure is to simplify the definition of distribution schemes. Autonomous entities leads to a perfect encapsulation of the behaviour and state of an entity, and therefore facilitates distribution of entities: such an entity can execute its behaviour on any site communicating with other entities through well defined stimuli.

Interactions between entities, modelled by the **stimulus** paradigm (phenomenon or event perceptible by an entity), cross media, called **stimuli spaces**, which allow communications between many entities simultaneously. Each stimuli space is in fact a projection of the environment along a specific type of stimulus (3D shape space, sound space...). An entity receives perceptible stimuli (visible shapes, near sounds...) through **sensors** and acts on its environment through **effectors** (producing new stimuli) (Figure 2). Sensors and effectors also manage interaction with the real world (e.g. a glove and a tracker sensors which track user movements and a HMD effector which presents two images to an user).

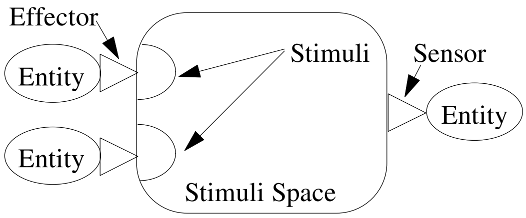

Figure 2: Interaction model

Each entity owns a set of behaviour components which modify its internal state (the set of its attributes) and commands actions to its effectors. Behaviour components are triggered by sensors (there is a time sensor which allows timed behaviours) or by other components.

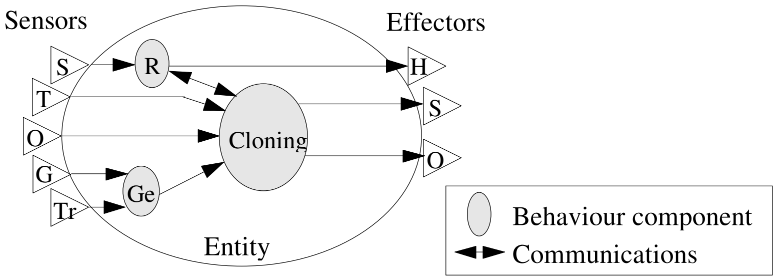

Figure 3: The behaviour of an avatar

We are going to illustrate the previously exposed behavioural model describing a typical avatar entity (Figure 3). Such an entity behaviour is composed of three components. The rendering component (R) creates images of the environment (known through a shape sensor — S — and presented to a user through an HMD effector — H) at a certain frequency (given by the cloning component — typically 30 Hz) and detects collisions with near entities. The gesture component (Ge) gets information from the user peripherals through specific sensors (e.g. a glove — G — an a tracker sensor — Tr) and does some gesture recognition. The cloning component is either triggered by a time sensor (T), the rendering component (e.g. when a collision occurs) and an interaction specific sensor called the order sensor which will be described in part 3.2 of this document. It clones hand and head movements (presented by the shape effector — S — to the environment) and uses gesture recognition to either move the avatar in the environment or command orders like grabbing and releasing to the order effector (O) .

This model borrows a lot from behavioural simulation architectures [Wilhelms90] [Mouli93]. In this previous domain, very complex behaviours were successfully implemented (e.g. animal flocks, ant farms...). And this is one of the reasons why we chose such a model, in order to allow the design of complex dynamic virtual environments.

2.2. The software architecture of VIPER

In order to better suit the structure of our applications, we have adopted an object oriented design for our system. Classical advantages of object oriented languages (encapsulation, reuse...) and most of all, inheritance are very interesting to model our entity families. The C++ programming language has been chosen because of its availability on most hardware platforms (from low-end PCs to multiprocessors).

In order to simplify the developer task we decided to offer the programmer a set of generic classes of concurrent aggregates [Moisan93]. Those aggregates encapsulate object distribution mechanisms for their elements and remote access over a network. A number of distribution models used in distributed VR have been implemented into aggregate classes: active replication [Carlsson93], replication on demand [Singh94], topologically optimised distribution schemes [Macedonia95] [Torguet95]... Virtual universes and stimuli spaces have been defined respectively as concurrent aggregates of entities and stimuli.

The software architecture of VIPER consists in four layers (Figure 4).

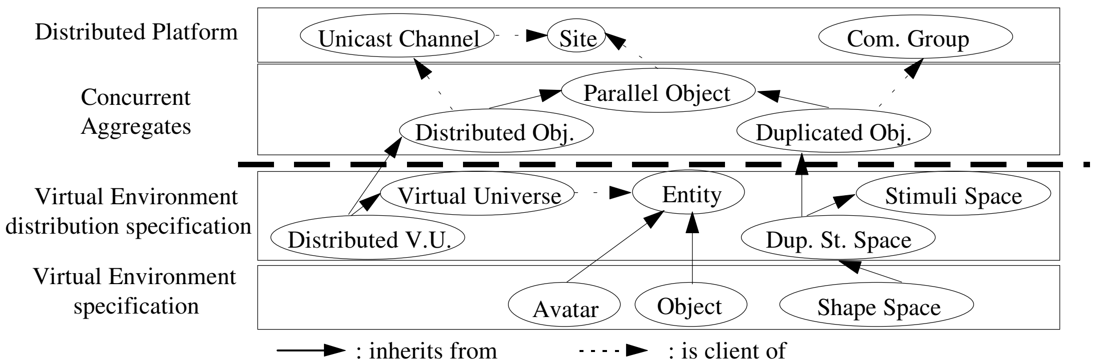

Figure 4: Layers and example classes of VIPER

The two first layers are the kernel of VIPER:

- The first layer, the distributed platform, is composed of a set of sites (workstations, multicomputers...) which create a virtual machine based on a message passing system like PVM [Sunderam90] or RMP [Whetten94]. This layer encapsulates the communication system used by our system. VIPER requires that such a communication system provides reliable (even though unreliable communication can be used in some part of the system) point to point (unicasting) and group (multicasting) communications.
- Using this layer, an object-oriented concurrent programming environment encapsulates data distribution and an SPMD (single program multiple data) model (cf. [Moisan93] for details). This layer proposes aggregates classes (also called parallel object classes).

The two last layers are the API (Application Programming Interface) of VIPER and define two programming levels:

- The third layer allows the definition of specific virtual environment distribution schemes. Distribution and remote access mechanisms can be chosen or redefined from a library of existing classes of parallel objects (from the second layer). Thus, the developer defines new stimuli spaces and new virtual universes. And therefore, he/she is able to optimise thoroughly his/her application.

- Using the last layer, a developer can describe a virtual environment (entities and their interactions) as if his/her application was a sequential object oriented program. He or she defines new classes of entities (or stimuli, stimuli spaces, sensors, effectors) that inherit from existing ones. In fact, this programming level totally hides the distributed aspects of the application.

3. An example application: multi-user world modelling

The first application we have developed is a multi-user world modeller. In this part, we will describe the design and implementation of this application.

We think that interactive world modelling can benefit a lot from distributed virtual reality in term of ergonomics, conviviality and efficiency (two or more users can co-operate to model a large virtual world). In order to better apprehend the problems of this domain, we have studied some existing multi-user modelling applications.

The first application, we know of, is built over Shastra, an architecture for development of collaborative applications [Anupam93]. Several users can co-operate in order to smooth a polyhedral model. The session master (a specific user) distributes certain areas of the object to smooth to other users. However, those tasks are performed through a 2D classical GUI, which is not the best suited environment for 3D manipulations.

Within the Virtuosy project [Benford94], a collaborative design application has been initiated in the domain of the fashion industry. This application allows several designers to visualise and modify clothes, negotiating and discussing their properties. Those designed objects are manipulated and changed in a 3D real-time virtual environment. Moreover, animated mannequins wear those clothes and move in a realistic way.

The European Computer-industry Research Centre (ECRC) has defined a co-operative design oriented platform suited for Augmented Reality applications [Ahlers95]. Several designers can fit out a building interior visualised through video images. Users can add and arrange virtual objects (e.g. chairs, tables) in position through 2D widgets and mouse operations. A VR interface (using 3D peripherals) would certainly enhance the application interactivity.

According to those studies and our previous application domain (declarative multimodal modelling) [Gaildrat93], we have decided to design an interactive multi-user world modelling application.

We wish to introduce high level metaphors for co-operative world modelling. We offer users a set of tools used to interact in a virtual environment in the most natural way: they can model shapes by doing natural gestures they would make in a real situation (i.e. hammer an object, dig a hole...). Our aim is to make modelling more intuitive. Therefore our tools have a double role: their shape gives a hint on both their function (hammer, nippers...) and their use (to hammer, to pinch...).

We have currently developed two deformation tools: a hammer and a hook which can be used to interactively deform surfaces (the user catch some part of an object with the hook and then move or rotate it to produce surface deformations). Tools are available on tables which exist in the environment. Of course, the user can also directly manipulate objects in order to move them, to arrange them...

3.1. The mathematical model of deformations

Let us describe first, the mathematical model used by our deformation tools.

The deformation model we use, is called FFD (Free-Form Deformation) [Sederberg86]. FFD deforming consists in enclosing an object (or a part of it) in a parallelepiped of clear, flexible plastic. The object is imagined to also be flexible, so that it deforms along with the plastic that surrounds it.

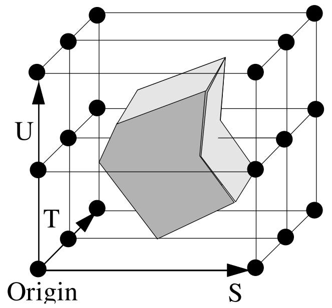

Figure 5: An example of lattice

Deforming consists in a number of steps. First of all, the parallelepiped is defined by providing its origin, main axes and the number of control points on each axis (Figure 5). Then, we compute the object point co-ordinates in the lattice local co-ordinate system. Thereafter, every lattice deformation (moving one or many control points) can create an object deformation according to a Bézier or B-Spline interpolation. We illustrate this with an interpolation based on trivariate Bernstein polynomials:

$$P_{ffd} = \sum_{i=0}^{l} \binom{l}{i} (1-s)^{l-i} s^{i} \left[\sum_{j=0}^{m} \binom{m}{j} (1-t)^{m-j} t^{j} \left[\sum_{k=0}^{n} \binom{n}{k} (1-u)^{n-k} u^{k} P'_{ijk} \right] \right]$$

where (s, t, u) are local co-ordinates of the processed point (P_{ffd}), and P'_{ijk} are control points after deformation of the lattice.

However it is, sometimes difficult to obtain specific effects (such as flattening a bump). We, therefore, decided to design a system which would mask completely the mathematical functions used to deform objects.

We propose a set of tools based on this deformation model [Rubio95]. The only parameters of those tools are natural (the part of the object which is to be modified, the size of a hammer head...). Moreover, those high level parameters are a concise definition of the deformation which, when transmitted over a network, keep the bandwidth usage low.

3.2. Introducing modelling tools in VIPER

Within VIPER all manipulations (grabbing, moving...) are modelled by exchanging orders (a subclass of stimuli) between entities within the framework of a specific stimuli space called an order space. Each virtual object has an order sensor which receives its given orders. Moreover, an entity has to have an order effector in order to manipulate other entities.

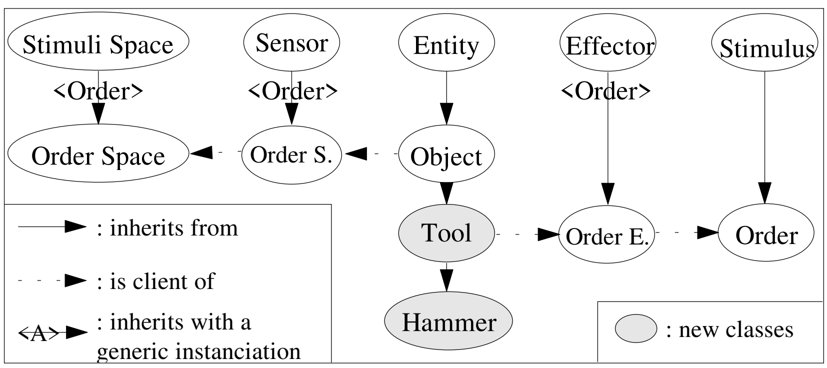

Figure 6: Extending VIPER with new classes

A modelling tool is an entity (in fact it is an object which is in turn an entity). It is manipulated by one or many entities and it, also, manipulates one or many entities. Therefore, a tool owns at least an order sensor (inherited from the object class) and an order effector which allows him to send manipulation orders (Figure 6).

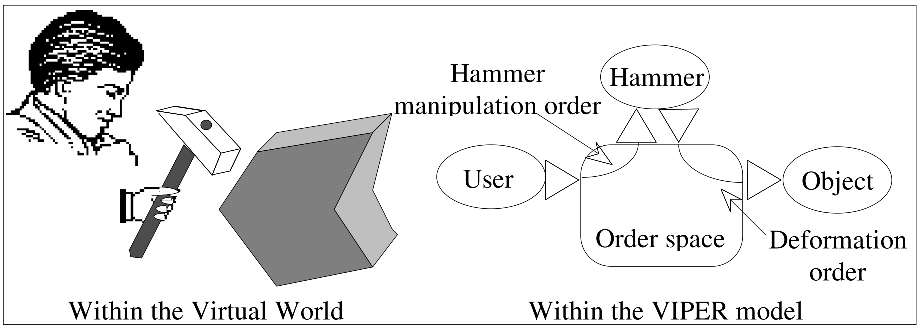

Figure 7: Hitting with a hammer

For example, when a user manipulates a hammer so as to deform an object (Figure 7), the user's avatar entity sends manipulation orders (grabbing, releasing) to a hammer entity. This entity analyses such orders and its surrounding environment (through a shape sensor) in order to detect any collision. Whenever a collision is detected, the hammer entity sends a deformation order to the hit object. As soon as the target object entity receives the deformation order, it builds a lattice (see section 3.1. for details), deforms it as required by the deformation order high level parameters (the last two hammer positions and the size of the hammer head) and then computes the new shape.

4. Application distribution

We will now describe, the distribution of this application using VIPER. Showing how distributed virtual universes and distributed image spaces can solve groupware problems of the application.

4.1. Managing distributed entities

Within a Distributed VR application, obviously, the first distributed data are entities. This distribution is managed by distributed virtual universes (DVU). Each DVU is a concurrent aggregate of entities. It defines a naming function for entities (an entity Id is composed of its DVU Id followed by its creation site Id and a sequential number), a distributing function which tells where (on which site) each entity is stored and possibly some remote access functions for entities.

The simplest class of DVU is the passive distributed virtual universe. This universe holds entities that can't move from site to site. New entities can be added dynamically on specific sites and new sites can join the DVU. Avatars obviously belong to this type of universe. Such a DVU only provides a simple distribution function which extracts the site Id from an entity Id.

However, in order to better deal with the high latency of wide area networks, we have designed another type of DVU called the active distributed virtual universe (Figure 8). This DVU tries to minimise the distance between interacting entities. For example, when an user manipulates a virtual object which happen to reside on a distant site, the DVU will as soon as possible move the virtual object entity to the manipulating user site. Such a DVU provides a distribution function (implemented by a replicated synchronised table which contains for each entity of the DVU: an entity Id and the Id of the site which currently owns the entity) and remote access methods used to send and receive entities (using point to point communication).

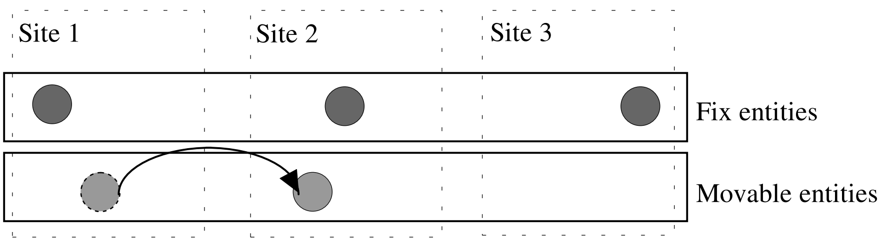

Figure 8: Distributed virtual universes

4.2. Interactions between distributed entities

As soon as two or more users enter the modelling space, a problem arises. Each user must see other users in order to interact with them. VIPER solves this problem with a distributed stimuli space, called the shape space, which uses an active replication distribution model. Each entity owns a **shape** which is composed of one or many geometric 3D shapes (which are used as different *level of details* or as multiple views of a same entity), a position, an orientation and a scaling factor in relation to a father object co-ordinate system (the shape space is a hierarchy of shapes). Each time an entity modifies its shape (movement, change of parent or deformation) the stimuli space replicates those changes on any site (Figure 9) using a stimuli space dedicated communication group. Then each entity which owns a shape sensor can use this hierarchy in order to present a view of the world to its user (avatar) or to detect collisions with other objects (avatars, hammer entities...).

It is important to note that all communications are hidden in the stimuli space and therefore an entity, as well as the programmer of the entity class, isn't concerned with communication management.

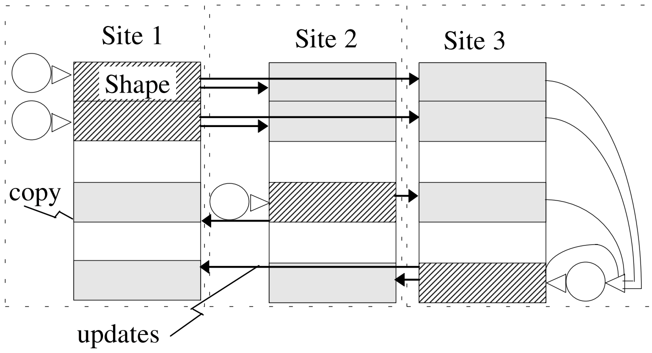

Figure 9: The shape space replication mechanisms

Direct interaction (i.e. when an entity manipulates other entities) in a distributed context is another problem: interacting entities may happen to reside on different sites. We have decided to model those interactions by exchanging orders (a new class of stimuli) between entities within the framework of a specific distributed stimuli space called the order space. We could have solved this problem with the previously exposed replicated stimuli space. However we can optimise the application by defining a new stimuli space which sends only orders to interested entity sites using point to point communications (Figure 10). Effectively, as each entity Id can be translated to a site Id where the entity is stored (thanks to DVU distribution functions), we can only send orders to those sites.

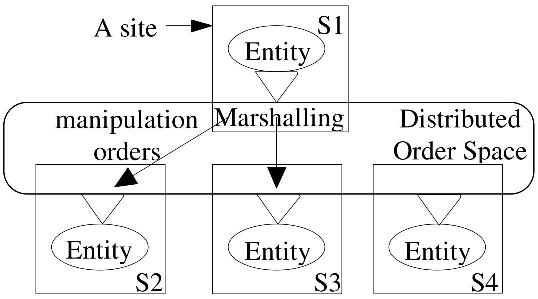

Figure 10: A distributed order space

4.3. Access management

In our application, access management problems may occur when two or more users want to access the same object in order to manipulate it. Those problems can either be physical problems (two users want to deform the same part of an object at the same time) or higher level problems (e.g. one user wants to edit a shape in order to create a bottle while another one wants to model a vase).

In order to solve physical problems for deformations, we state that a new deformation can be started on one object if and only if the lattice to create doesn't intersect with an existing lattice. Note, that however multi-user concurrent deformation (twisting, bending, stretching, flattening...) can still be done because two users can modify the same lattice through multi-user tools. For other manipulations (e.g. moving an object, rotating it), a manipulation can only be started if no other manipulations are being performed.

Moreover, a user can put an explicit graphical lock on an object (or a part of it), in order to get an exclusive right to modify the object (Figure 11). Thereafter the lock can only be removed by this user. This is the way we solve the higher level conflicts. This type of explicit locks can be seen as very restricting and unnatural to users, however those locks were actually demanded by users of our application in first demonstrations. They thought that the advantage of a virtual environment over a real modelling environment is that you can add such artificial signs and constraints in order to aid their task.

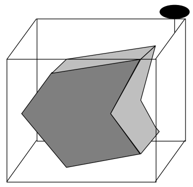

Figure 11: An explicit lock on an object

5. Conclusion and future works

We have implemented the ideas presented here within the context of a countryside modelling application (Figure 12). Users can do terrain modelling like digging lakes, growing hills... They can also place scenery elements (trees, houses...) on the terrain. Multi-user aspects are very interesting in this application, because they accelerate a lot large terrain modelling.

We are further developing VIPER with interpreted behaviours. Those behaviours will allow us to migrate entities from site to site as required by the application (for load balancing or in order to gather interacting entities on a site for performance reasons). Moreover we will then be able to dynamically add behaviours to objects in our virtual world modelling application. Thus, using an enchanter's wand metaphor, users will be able to select objects and add interpreted behaviours to them.

We are also very interested in the VRML (Virtual Reality Modelling Language) [Bell95] architecture and plan to support its language and protocols in the near future.

On the other hand, we are still developing other interaction tools (multi-user deforming tools, multimodal modelling tools...).

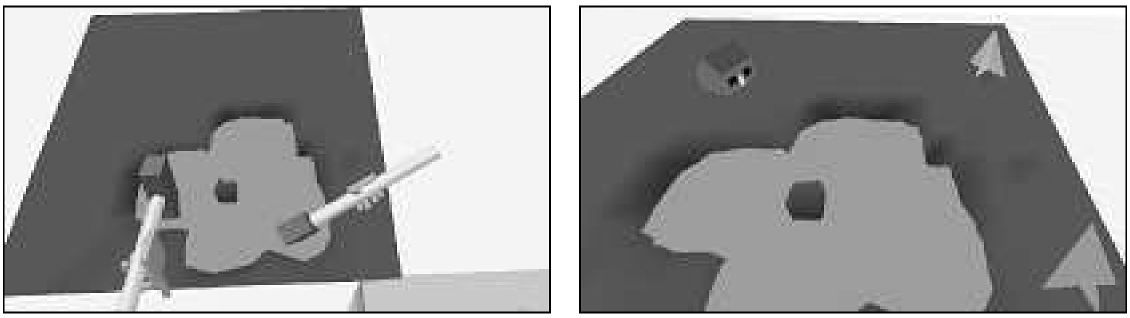

Figure 12: Countryside modelling

References

[Ahlers95] Ahlers K. H., Kramer A., Breen D., Chevalier P.Y., Crampton C., Rose E., Tuceryan M., Whitaker R. T., Greer D., "Distributed Augmented Reality for Collaborative Design Applications", Eurographics'95 Proceedings, Maastricht, NL, 1995.

[Anupam93] Anupam V. and Bajaj C., "SHASTRA - An Architecture for Development of Collaborative Applications", Proceedings of the Second IEEE Workshop on Enabling Technologies : Infrastructures of Collaborative Enterprises, Morgantown, West Virginia, IEEE Computer Society Press, p. 155-166.

[Benford94] Benford S., Bowers J., Gray S., Leevers D., Rodden T., Rygol M., Stanger V., «The Virtuosi Project», Proceedings VR'94 (London Virtual Reality Expo 1994), Meckler, London, Feb. 1994.

[Bell95] Bell G., Parisi A., Pesce M., "VRML 1.0 Final Specification", available through URL: "`http://vrml.wired.com/vrml.tech/vrml10-3.html`", May 1995.

[Carlsson93] Carlsson C. and Hagsand O. "DIVE - a Platform for Multi-User Virtual Environments", Computer & Graphics, Vol. 17, N° 6, November/December 1993.

[Gaildrat93] Gaildrat V., Caubet R., Rubio F., "Declarative Scene Modelling with Dynamic Links and Decision Rules Distributed Among the Objects", IFIP International Conference on Computer Graphics ICCG93 Bombay, February 1993, p. 165-178.

[Macedonia95] Macedonia M.R., Zyda M.J., Pratt D.R., Barham P.T., "Exploiting Reality with Multicast Groups: A Network Architecture for Large-Scale Virtual Environments". In proceedings of VRAIS'95. IEEE Computer Society Press, Los Alamitos, CA, March 1995.

[Moisan93] Moisan B., Duthen Y., Caubet R., "Tools for SPMD object-oriented programming" EUROMICRO'93 Barcelona 6-10 September 1993.

[Mouli93] Mouli R., Duthen Y., Caubet R., "In VitrAm (In Vitro Animats, a behavioural simulation model)" 2nd IEEE International Workshop RO-MAN'93, Tokyo, 3-5 November 1993.

[Rubio95] Rubio F., Torguet P., Gaildrat V., Caubet R., "Déformations de formes libres à l'aide d'outils de réalité virtuelle", Interface to Real and Virtual Worlds, Montpellier, France, June 1995.

[Sederberg86] Sederberg G. T. W., Parry S. R., "Free-Form Deformation of Solid Geometric Models", Siggraph 86 Conference Proceedings, Vol. 20, N_ 14.

[Singh94] Singh G., Serra L., Png W., Ng H., "BrickNet: A Software Toolkit for Network-Based Virtual Worlds", Presence, Vol. 3, No. 1, Winter 1994.

[Snowdon94] Snowdon D. N., West A. J., "The AVIARY VR-System. A Prototype Implementation", 6th ERCIM workshop, 1-3 June 1994, Stockholm.

[Sunderam90] Sunderam V., "PVM : A framework for Parallel Distributed Computing", Concurrency: Practice & Experience Vol. 2 N° 4, December 1990.

[Torguet95] Torguet P., Caubet R., "VIPER (VIrtuality Programming EnviRonment): A virtual reality applications design platform", 2nd Eurographics Workshop on Virtual Environments, Jan. 1995.

[Whetten94] Whetten B., Montgomery T., Kaplan S., "A High Performance Totally Ordered Multicast Protocol", In Theory and Practice of Distributed Systems, N° 938 in LNCS, Springer Verlag 1994.

[Wilhelms90] Wilhelms J., Skinner R., "A *Notion* for Interactive Behavioral Animation Control" IEEE Computer Graphics and Applications, May 1990, pp 14-22.