Alternative Reality: a New Platform for Virtual Reality Art

Marc Cavazza Simon Hartley Jean-Luc Lugrin Mikael Le Bras
School of Computing and Mathematics,
University of Teesside,
44 01642 382657
{m.o.cavazza, s.hartley, j.l.lugrin, m.lebras}@tees.ac.uk

ABSTRACT
Virtual Reality Art involves the design of artificial worlds that offer new experiences to spectators. An important aspect for the development of VR Art installations is the principled definition of behaviour for the environment as a whole, which would facilitate experiments with alternative laws of physics, time, and causality. We describe the first results of an ongoing project dedicated to the development of software tools for the use of Intelligent Virtual Environments in VR Art. Using the architecture of a state-of-the-art game engine, we have developed Artificial Intelligence techniques that support the definition of alternative laws of physics. After discussing the principles behind alternative reality we describe two complementary modes of description for alternative behaviour: qualitative physics and causal simulation. This is illustrated by examples integrated into the virtual environment.

Categories and Subject Descriptors
H5.1 [**MultiMedia Information Systems**] Artificial, Augmented and Virtual Reality - *Virtual Reality for Art and Entertainment*

General Terms
Theory, Algorithms, Design and Experimentation

Keywords:
Virtual Reality for Art and Entertainment, Modelling and Simulation, Intelligent Virtual Environments, Qualitative Physics

1. INTRODUCTION
Virtual Reality Art [16] [8] offers the perspective of creating alternative worlds that depart from our everyday experience of the physical worlds. In other words, virtuality need not be modelled after reality: this vision was actually part of the original ideas that pioneered Virtual Reality, e.g. Leary's psychedelic metaphor of Virtual Reality [14]. On the other hand, Virtual Reality Art has retained contact with these original ideas, and VR artists have experimented in their works with fundamental aspects, such as space and the laws of physics [5] or time (Toshio Iwai). To a large extent, VR Art is concerned with the implementation of alternative realities: worlds in which to experiment with time and space, in which laws of physics or even fundamental phenomena like causality can be tailored to the requirements of an artistic experience.

But more interestingly, in recent years, the work of several artists has explicitly addressed *Physics* as a source of inspiration, in particular where it could depart from our everyday physical experience. For instance, the collective exhibition "The Amplitude of Chance" in Nagasaki was entirely devoted to briefs exploring causality [21]. One sophisticated and even more explicit example is constituted by the animation series "The Quarxs™"[1], which features a set of invisible creatures which violate the laws of physics and provide an explanation for the unaccounted phenomena of our everyday world. These "insect-like" creatures are named after the physical laws they twist: for instance the *Reverso Chronocycli* causes time to flow backwards and the *Spiro Thermophage(*figure 1) lives in water pipes interfering with heat transfer processes.

Figure 1. The Spiro Thermophage from the Quarxs [tm] . © Maurice Benayoun and Z-A Productions.

This opens the possibility to support the development of alternative reality installations using Intelligent Virtual Environments [2], which incorporate novel, programmable, high-level behaviour layers in a VR system. In this paper, we describe the development of such a system: after a brief introduction to the baseline technology used, we present two specific and complementary techniques to redefine virtual world behaviour: qualitative physics and causal simulation.

1.1. System Overview and Architecture
In order to maximise user experience as well as making an installation visible to a wider audience, Virtual Reality Art is often presented using immersive displays such as CAVEs™. Our target systems are large-scale virtual reality installations, based on the

[1] The Quarxs™ have been ideated by Maurice Benayoun, one of leading Virtual Reality artists (see e.g., [8]).

SAS Cube™, which is a 4-wall, PC-based, CAVE™-like, immersive visualisation system. The use of a CAVE-like system should facilitate interaction with virtual world objects, which is an essential aspect of the alternative reality experience.

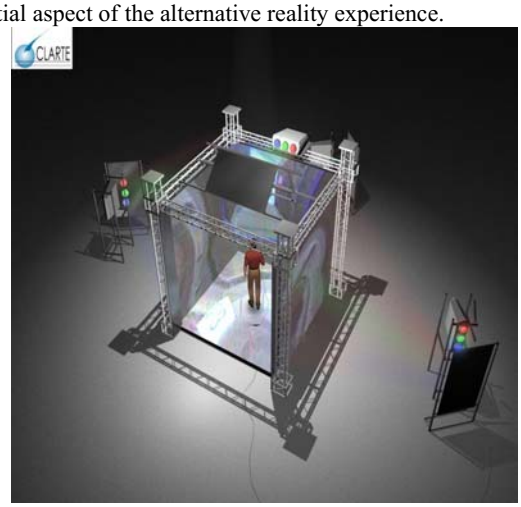

Figure 2. The SAS Cube™

We use a game engine, Unreal Tournament 2003™ (UT), as a visualisation engine and as a development environment. Game engines are now increasingly used for visualisation in scientific research due to their rendering performance and their ability to communicate with external software modules [15], which in the present case is essential to the development of a simulation layer that will override basic physics mechanism to implement alternative reality. Another interesting aspect is the growing use of game engines for 3D Digital Arts. In addition, the engine we are using, UT, has previously been ported to CAVE™ systems [12] and we are currently porting it to the SAS Cube™, using the original approach described in the CAVE-UT implementation.

The rationale for using a game engine is not just its graphic rendering abilities or the built-in mechanisms for user interaction with world objects. The UT 2003 engine incorporates a sophisticated physics engine (the Karma™ system from Mathengine™) together with a sophisticated API and programming features supporting the modification of baseline physics. As it is through the redefinition of physical behaviour that alternative reality can be implemented, this feature provides an essential path to the implementation of new behavioural layers overriding basic mechanisms.

2. THE ELEMENTS OF ALTERNATIVE REALITY

As the examples from the Quarxs™ illustrated, the systematic modification of physical laws, by defining new rules of behaviour for physical objects, is a principled way of creating alternative universes. Of course, to implement alternative reality as we have defined it, the newly defined physical laws should take place in real-time in an interactive environment, rather than an offline animation. The essence of Alternative Reality, like virtual reality is its user-centred nature and its interactivity. The alternative nature of physical phenomena is only perceived because the user is part of them and constitutes an experience insofar as he can be at the origin of some of the unusual transformations. The main technical innovation of this research is to establish principles for redefining the virtual environment physical behaviour.

However, redefining physical processes in a consistent fashion is a challenging task, which should be supported by appropriate formalisms. In particular it would be extremely difficult to produce consistent sets of low-level equations defining the new behaviour of objects. Some high-level description of the physical processes and their consequences is required instead. Two factors contribute a solution to this problem: the first one is the discretisation of physical processes into events which supports the implementation of physics in most games engines including UT (and is also developed within more traditional VR systems [12]) and the second is the existence of computing formalisms for high-level, qualitative descriptions of physical processes that have been developed in Artificial Intelligence [6]

Another aspect consists in modifying the laws of causality [20], which relate the occurrence of macroscopic events in the world, and play a fundamental role in our perception of reality. While previous work has addressed the replay events in a virtual environment [10], redefining causality requires more sophisticated processing of the co-occurring events in order to form a new causal chain and, in addition, needs to be compatible with real-time user interaction.

2.1. Event-based Architectures for Behaviour Description

Most interactive systems are based on the notion of event for their implementation. Highly interactive systems, such as VR systems [13] and game engines in particular intensively exploit this notion for their implementation. On the other hand the notion of event is also the basis for the high-level description of physical behaviour, as events discretise the continuous motion of objects (in terms of positions, trajectories, contacts with other objects) into meaningful high-level actions.

This section describes the event management mechanisms, which support the redefinition of alternative behaviours for the virtual environment. As these mechanisms underlie the implementation of both alternative causality and alternative physics, we will describe them first. After introducing the native mechanisms provided by the UT game engine, we will show how these can be used to define complex events corresponding to object behaviour as well as a control strategy to override the basic mechanisms supporting the world physical behaviour. This will then open the possibility of extensive redefinitions of the virtual world behaviour, supporting alternative views on physics and causality.

The Unreal engine extensively relies on event generation to support many of its interaction aspects and, most importantly, the mechanism for event generation is accessible to redefine specific behaviours for the environment. Formally, an Event can be characterised as an encapsulated message, which is generated by an *Event Source* and sent to one or more *Event Consumers*, these being both objects of the environment. The transmission of an Event to an Event Consumer triggers some specific action in response to that event. It should be noted that the actions triggered by Event Consumers can further be detected by other Event Sources, and this accounts for the possibility of Event propagation as a kind of “forward chaining”, propagating the consequences of an action.

The Unreal Engine implements two different kinds of event sources: the basic events, which are primitive low-level events defined within the game engine and the programmed events. The latter are events whose definition is scripted (i.e. can be programmed by the system developer) and are originally used to

customise the interactions between objects, by defining which objects will trigger (or react to) specific events. The basic events can be classified into six major categories, of which two are mostly used in our implementation to redefine environment behaviour: the interaction events and the time notification events. Interaction events are generated by the native mechanisms that control low-level graphical events such as collision detection. The interaction event category is further refined into several sub-categories, the most important being: *physics and world interaction event*, *player input event*, and *trigger event*, which is the basic event class supporting the definition of high-level events.

On the other hand, *time notification events* are related to the engine internal time management system and can be used to define the control cycle of any new event-based layer, for instance to programme the sampling rate of event management, which determines the temporal course of alternative reality phenomena.

From another perspective, basic events can also be classified as discrete or continuous events. Discrete events notify instantaneous actions, such as "bumping", while continuous events signal the beginning and ending of durative actions, for instance touch/untouch, attach/unattach (these are used for instance for carrying or manipulating physical objects).

The redefinition of event mechanism comprises three main aspects: i) the attachment of events to specific classes of objects, ii) the overriding of native event generation mechanisms and iii) the definition of ad hoc complex events from basic system events.

Relating events to objects is implemented using the native UT *mutator* system, which in UT supports the redefinition of object behaviours. This confers a new behaviour to the environment objects, which is the ability to enter into an Event Interception (EI) state. Once an object is set into the EI state, every event fired for this object will not trigger any corresponding action (via the procedure described above) *i.e. the message coming from the event source is intercepted at the event consumer level*. But it will use a procedure to signal the event call and arguments to an Event Controller module. This is a basic, though generic, mechanism by which events can be "frozen", i.e. detected and recorded but with their associated actions temporarily inhibited.

The Unreal Engine relies on a fixed set of basic events. However, for most applications it is necessary to define high-level events, whose semantics is dictated by the application. Such complex events are often called *context event*. The rationale for the use of context events is that they capture the semantics of an application scenario, which facilitates the formalisation of alternative rules for causality that operate on properties or parameters of the context events. Because they are aggregates of basic events, context events can be recognised by parsing a stream of lower-level events using a template for the (high-level) event to be recognised. This is a standard approach in event recognition, which has been used previously in computer vision and VR alike [1] [4]

3. IMPLEMENTING ALTERNATIVE REALITY

We have developed an example scenario to support the development of our platform as well as the required experiments. This environment was inspired from some of the Quarxs™ episodes, which tend to take place in everyday settings such as kitchens and bathrooms. Actually these areas concentrate many potential physical processes, such as the fall of objects, flows of liquids, heat transmission, surface contacts, formation of bubbles and foam, etc. They are also familiar places where the affordances for interaction tend to be obvious, and in which unexpected behaviour becomes immediately salient, precisely because of their familiar nature. This is why we have developed several test scenarios in such environments.

3.1. Alternative Laws of Causality

Causality is an important mode of apprehension of reality, whenever interaction and dynamic processes are involved [20] [17,18][19]. This mode equally applies to interactive environments such as virtual environments. Users will attribute causal relationships to events that follow their interactions with the virtual world's objects, which they will consider as consequences of their actions, or to events showing some form of correlation, the most frequent being co-occurrence (i.e. temporal, and to some extent spatial, correlation). Manipulating the perception of causality thus offers a great potential to leverage new forms of experience. Consider a normal VR scene: it comprises several background physical processes, such as water flows and particles getting in contact with objects; the scene may contain autonomous agents, which interact between them or with objects, and finally the user. An ordinary scene is thus hosting many forms of interaction, on small and large scales. All the corresponding event occurrences can be re-arranged and, given the correct emphasis, this will contribute to the definition of a totally different world.

In an environment in which all interactions are mediated through events representations, it is thus possible to modify the user's perception of his/her environment by modifying the dynamics of event management by the system. At the heart of causality reprogramming is the notion of event interception. This interception "freezes" the activation of the events post-conditions whenever an event is generated. The basis for alternative causality is that frozen events in the stack can be subject to several formal transformations, which will produce the desired effects once the event is re-activated.

The alternative causality system is based on a set of procedures operating on the events continuously generated by the system, as described in the previous section. These procedures operate a certain frequency, which determines the "sampling rate" of alternative causality, i.e. the time scale over which events are intercepted, transformed and re-activated to produce the desired causal effects. The Figure below reproduces the Event Control software architecture.

A procedure called *Event Interception Process* continuously intercepts and freezes the in-coming event produced by world objects (set in an Event Interception state) as they occur in the virtual environment, as a result of user interaction or other background phenomena.

The frozen basic events intercepted during a given time (corresponding to the sampling rate) are stored into a corresponding data structure. This structure of basic events is in turn parsed to extract higher-level events (context events) corresponding to the application semantics, which are the events onto which the causal simulation module will operate. For instance, from a set of basic events describing the movement of a stone and its contact with a glass surface, Context Events such as *throwing(stone, glass)* can be generated.

The set of Context Events obtained in this way is time stamped and sent to the causal simulation module, acting as a server,

through a TCP/IP socket. The causal engine uses explicit knowledge to carry out transformations on the events it receives.

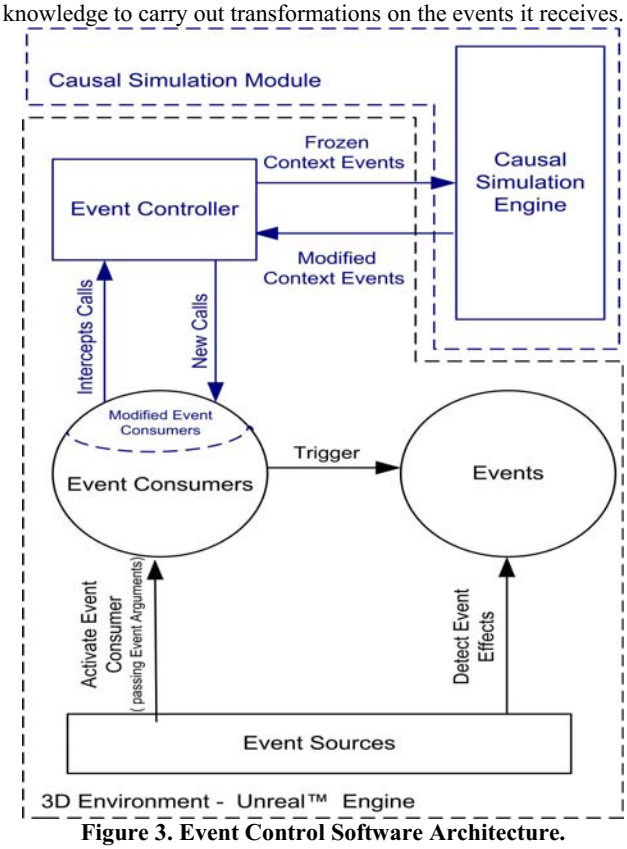

Figure 3. Event Control Software Architecture.

This knowledge encodes the new definition of causality that should be implemented: for instance, swapping the object of a given action to displace the cause-effect relationship from one object to another. One example of this triggers the effect of an intercepted action on an object other than the one on which the action was originally executed, but of a compatible nature: for instance when throwing a stone at the glass of a mirror, a glass other than the one that has been hit is broken instead. Depending on the explicit knowledge redefining causality (which can be expressed in a declarative fashion for each application) various transformations are carried: swapping objects action's, re-ordering events (including reversing the flow of time on some small-size causal chains), changing the spatial occurrence of events in the environment, etc.

As a result of these transformations, a new ordered set of Context Events is generated and sent back to the Event Controller module in the game engine's environment. The Event Controller is "listening" for incoming event data frames from the Causal Engine.

From a formal perspective, the process by which a new sequence of events is produced from the original set of Context Events bears similarities with some AI planning techniques, in particular search-based planning [9] [3]. The context events can be associated to STRIPS-like representation. A representation that contains the objects, pre-conditions and post-conditions which define the context event. These representations can be re-ordered to produce the final sequence of events, this re-ordering being formally equivalent to the generation of a plan. Re-ordering is an important aspect of alternative causality, as from the user's perspective causality is essentially derived from co-occurrence. However, as previously described, many different transformations can take place, such as swapping of semantically compatible parameters between two events. Various mechanisms can be used to govern these transformations, such as the definition of macro-operators operating on the STRIPS representation, for a more expressive formalisation of alternative laws of causality.

```
Class ECC_OBJECT extends Decoration
...
var() bool EI_Active;
 // if True Event Interception is Activated
var() string  CausalObjectType;
  // Type of Causal Propagation Object
var ECC_EventController EventControllerRef;
 // Reference to the Event Controller Instance
....
Event Bump( Actor Other ){

 if ( EI_Active) // if Object set to event Interception State then...
 {
    EventControllerRef.Record_BasicEvent(

       Level.TimeSeconds,
// event Triggered Time
       "Bump",
// Basic event Signature
       self,
// pointer to event consumer object instance
       self.CausalObjectType,
// Causal Object Type identifier of the event consumer object
       Other,
// Pointer to event instigator Object instance
       ECC_OBJECT(Other).CausalObjectType
 // Causal Object Type identifier of the event Instigator object
    );

 }  else super.Bump( Other );

// Otherwise, it calls the "normal" Bump
//  method of parent class : Decoration

}
....
```

Figure 4. Example of Unreal Script Interception Event Code

The final step consists in re-activating the transformed intercepted events within the environment: this comprises two different aspects. First the target objects of these events (the event consumers in UT's terminology) should be set to an event execution (rather than event interception) state. Second, the consequence (post-condition) part of the high-level event operating on that object should be executed in terms of basic events corresponding to it. For instance the consequences of a "breaking" event will consist in the destruction of the glass, the replay of a breaking-glass animation and the generation of a set of glass fragments in the world (which in turn will generate more events to be processed during the next cycle). Once the actions associated to these basic events have been executed, the objects are put back into Event Interception mode and a new cycle of event sampling is resumed.

3.1.1 An Example: The Mirror Causality

The example we use to illustrate alternative causality we call mirror causality. This consists of propagating the effect of throwing a projectile at mirror on to the objects being reflected by the mirror. The consequence being that, following the missile impact, the mirror will not break but the object whose reflection in the mirror is hit will behave as if it were struck by the projectile.

In our 3D environment, the Event controller controls each graphics object instance. Then we have defined a list of Context event associated with this environment. Context events correspond to high-level Events each expressed with Pre-conditions and Post-condition, which can be illustrated through the following declarative form:

Context Event: Break_glass_object (?target, ?missile)

Pre-conditions:
Frozen_Event (BE_Bump(?target, ?missile))
Breakable(?target)
Moving_object(?missile)

Post-conditions:
Active_Event ((BE_Bump(?target, ?missile))
Active_Event (explode (? target))

This is the Context Event defined called Break_Glass_Object, describes the behaviour of a "breakable" object when hit by a missile. A Context Event can be instantiated when its pre-conditions are satisfied. CE preconditions are conjunctions of static and dynamic predicates: static predicates correspond to object properties (such as the fact that an object is breakable), while dynamic predicates check the occurrence of a Basic Event involving objects that can be arguments of that Context Event. In most cases the BEs appearing in the pre-condition list have been frozen by the event interception system. Triggering the CE post-conditions will re-activate relevant BEs (see example above).

The sequence of events transformations underlying alternative causality can thus be described as follows: i) objects that are in event interception mode freeze BEs affecting them at regular intervals ii) each time a new set of BEs have been produced, the system identifies CE corresponding to them by, testing the CEs' pre-conditions iii) this set of CEs sampled over that time unit is passed to the causal engine, which operates various transformation on the population of CEs (re-ordering, parameter swapping between CEs, parameter modification). These modifications affect the CEs' post-conditions in terms of modification of parameters, addition of new BEs to the post-condition list or deletion of BEs (in the latter case, the BE deleted will have to be explicitly "neutralised"). In our "causal mirror" example, the ?target parameters for the Active_Event(explode ?target) post-conditions are modified. iv) after having been transformed by the causal engine, the modified CEs are activated by triggering their post-conditions. The figure 5: Event Control for Causal Mirror, illustrates the pipeline of the Causal Control System with the causal Example.

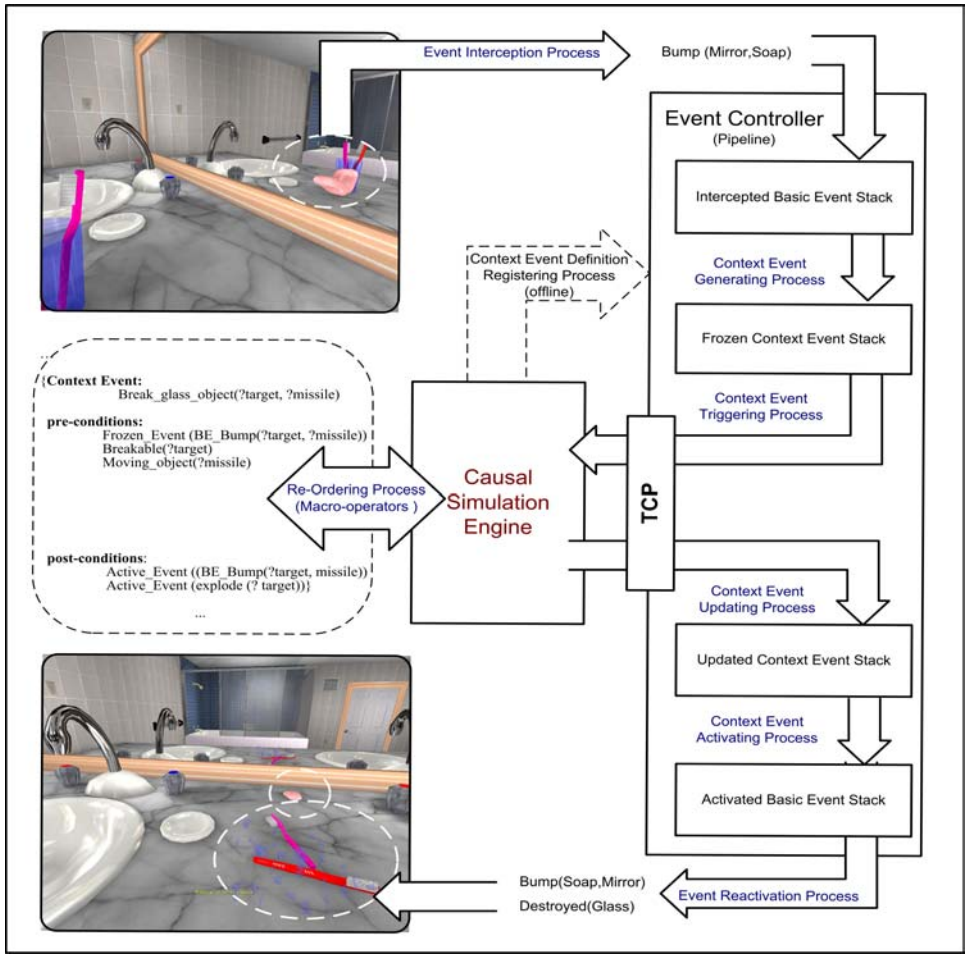

Figure 5. Event Control for Causal Mirror

3.2 Defining Alternative Laws of Physics

As we have previously introduced, the redefinition of alternative physical behaviour for objects is based on an AI technique called qualitative physics [11] [6,7]. The principle behind qualitative physics is to make discrete the variation of physical properties and to model all physical transformations through processes that encapsulate the relation between physical variables, through the notion of influence equations (see details of process described below).

The qualitative physics engine is implemented in an external C++ program that communicates with the UT 3D environment, when events occur. The implementation of the qualitative physics system relies on the same event-processing layer as the causal simulation module, which provides a unified approach for all alternative behaviours.

The virtual world objects that are manipulated by Qualitative Processes (QP) are, in fact, derived of a special class of objects that owns an event interception system. Qualitative Processes also operate through an event-based approach. For instance, certain QP events will launch specific animations, while other QP events will potentially update physical variables of the object (such as temperature, amount of matter contained, state, etc.).

The infrastructure of the Process Controller is comparable to the Causal Event Controller. Except for the Context Event Modeller which converts the event list coming from the QP Engine into a QP event list, instead of a list of Basic Events. The QP Event Modeller role is to dispatch those QP events to the targeted virtual world object instances.

In order for the qualitative physics engine (QP engine) to produce the events, for physical processes within the virtual environment, the engine needs to receive data about the list of available processes and the data for the objects in the world. The qualitative physics engine then uses this data to generate a list of potential processes that can occur in the world. The generation of the list of potential qualitative processes within the external C++ module is performed by analysis of the individual objects in the world. As processes are defined in part by the individual objects they apply to only the processes that have their individuals present can occur. Below is an example of the fluid flow process that we implement to show the alternative behaviour of a glass, which can hold an infinite amount. This alternative behaviour leads to the glass becoming too heavy for the user to move. The description of the fluid flow process which describes the filling of the Glass is:

Process: Fluid-flow (?source?sub ?dst ?path)
Individuals: ?source a contained liquid
?destination a contained liquid
?sub a substance
?path a fluid-path
Preconditions:Connects(?path,?source,?dst)
Aligned(?path)
Quantity Conditions
A[Pressure(C-S(?sub, liquid, ? source))] > A[Pressure(?dst)]
Relations Quantity(flow-rate)
Flow-rate=Pressure(C-S(?sub,liquid,source)) - Pressure(?dst)
Influences:I+(Amount-of-in(?sub, liquid, ?Source), A[flow-rate])
I-(Amount-of-in(?sub, liquid, ?dst), A[flow-rate])

In our example we have defined three Objects for the qualitative physics simulation. The details for these objects are then sent to the Qualitative Physics engine which creates instances of the Processes fluid flow process. The fluid flow process is instantiated

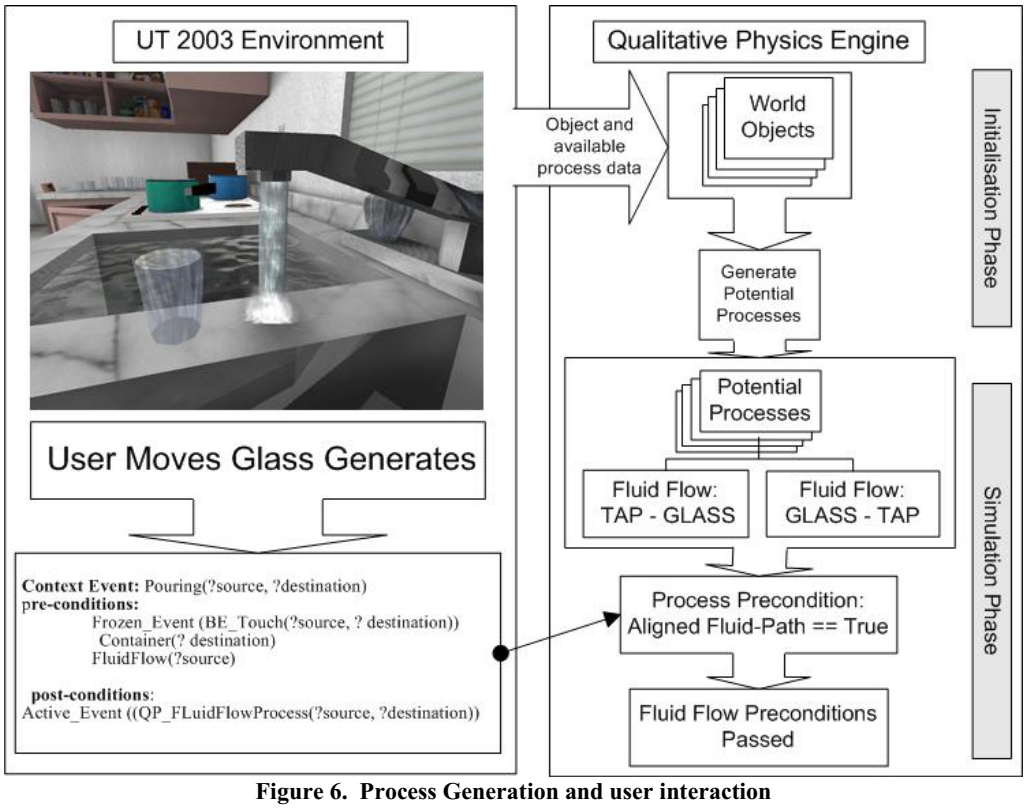

Figure 6. Process Generation and user interaction

as the three objects to which it applies to exist i.e. the Tap (a contained liquid source), the Glass (a contained liquid destination.) and a fluid path (shown as a column of water). This then generates for the list of potential processes two fluid flow processes these processes represent water flowing processes from the tap to the glass and from the glass to the tap.

Next we detail the simulation loop for the qualitative physics engine is given for the example of the water flowing into the glass.

The definition of a process determines the conditions under which it can become active. The conditions that the user can directly affect are known as preconditions for instance aligning a glass/container under a tap is a precondition for water flow to occur. (as shown in Figure 6. Process Generation and user interaction). Context Events are included within the system by adding the fluid flow Mutator to the Event controllers list of Active Context Events. Note: The pouring Context Event is defined by a Flow-Path Object instance in the world (i.e.: Tap Water flow) and a destination Container object instance (i.e. Glass). It is the alignment precondition for the fluid flow process that the context event pouring fulfils. The pouring context event is generated and sent to the QP engine when the user moves a container into a flow path.

The conditions that apply to the quantities within the objects, whose values and changes are governed by the qualitative process theory, are known as quantity conditions. An example of quantity conditions for the water flow process is that the pressure of the individuals needs to be different for the process to occur. To determine which processes are active we have to test the preconditions and the quantity conditions if these are both active we place these objects into the active process stack. In our example, we had two potential processes which both had passed the quantity conditions. Now for the process to occur the precondition and quantity conditions for the processes need to be tested. For both processes the preconditions are that there must be an aligned path between the two contained liquid individuals. If the user then moves the glass under the flow we have the context event aligned path which is sent form the environment to the QP engine. This event aligns the flow and allows both processes to pass the preconditions. The next stage is the quantity conditions for the processes since we are comparing the pressure for the two objects only one process will pass this stage, the process from tap to glass. The process will then become active and begin altering the quantities as described by the influence equations.

In the case of water flow the amount of water in glass will increase.

The next stage involves calculating the relations between the individuals for the process and then determining changes in the quantity via resolving influence equations.

When determining changes in the quantity via resolving influence equations the first stage is to resolve the direct influences of the processes. In our example the directly influenced values are the amount of water in the glass and the amount of water in tap. Then the process applies the relations for the changes in these quantities, this is to alter the indirectly influenced quantities. In our case we are changing amount of water in the glass but by our alternate laws the glass can hold an infinite amount so the water will not over flow but the glass will get heavier so the user will be unable to move the glass. Eventually a limit point for the glass/container mass will be reached and this will generate an event that will be sent from the qualitative physics engine to the UT 3D virtual environment. This event informs the virtual environment that the mass has passed a certain value. For alternative laws the virtual environment could respond in many ways to this event such as breaking the glass or starting different processes any of which would affect the user experience.

4. CONCLUSIONS

Modifying the laws of physics in Virtual Reality to create alternative behaviours on a principled basis is certainly a challenge in terms of conception, implementation and authoring. This research should support the creation of VR artistic installations which support recent ideas in the field for which no software tools are yet available. Using symbolic representations should also facilitate the translation of the artistic ideas in terms of computer implementation. We have shown that state-of-the art AI techniques such as qualitative physics can be adapted to virtual environments, taking advantage of the discretisation of physics events in game engines. In a similar fashion, the explicit description of new rules for causality is probably one of the highest levels of description that can be envisioned to describe behaviours. Current work is dedicated to building libraries of qualitative physics processes so as to scale up the platform beyond its current prototype status, as well as integrating the interaction mechanisms that would be required for immersive installations.

ACKNOWLEDGEMENTS

This work has been funded in part through the ALTERNE project (IST-38575), funded by the European Union under the Information Society Technologies programme (Cross-Programme Action 15). Maurice Benayoun is thanked for his introduction to the Quarxs™ and for authorising the use of Figure 1. SAS-Cube™ picture courtesy of CLARTE/Laval Mayenne Technopole. Unreal Tournament™ is a trademark of Epic™ Ltd. We would like to acknowledge VSK clan as the creators of the original UT Map. We would also like to acknowledge the work of Fabrizio Morbini for the planning and event systems.

REFERENCES

[1]Andre, E, Herzog, G and Rist, T,. On the Simultaneous Interpretation of Real-world images and Natural Language Descriptions: the SOCCER system. Proceedings of the 8th European Conference on Artificial Intelligence, Munich, 1988.

[2]Aylett, R. and Cavazza, M., Intelligent Virtual Environments: a State-of-the-Art report, Eurographics STAR reports, Eurographics, Manchester, UK, September 2001.

[3] Bonet, B. and Geffner, H. Planning as Heuristic Search, Artificial Intelligence, vol. 129, n. 1-2. pp. 5-33.
[4]Cavazza, M. and Palmer, I.J., High-level Interpretation in Virtual Environments, Applied Artificial Intelligence, 1999.
[5]Char Davis :"Osmose: Notes on Being in Immersive Virtual Space", in Digital Creativity, Vol. IX (2) [Preliminary version published in ISEA '95 Conference Proceedings. ISEA: Sixth International Symposium on Electronic Arts Montreal (1995).] (1998), pp. 65-74.
[6]Forbus, K, Qualitative Process Theory. Artificial Intelligence, 1984, 24, 1-3, pp. 85-168
[7]Forbus, K.D.,. Qualitative Reasoning. In A.B. Tucker, editor, The Computer Science and Engineering Handbook, pages 715--733. CRC Press, 1996.
[8]Grau, O.,Virtual Art : From Illusion to Immersion ,Cambridge (Massachussets), MIT press. ISBN: 0262072416,2002
[9] Gerevini A., Saetti A., Serina I Planning through Stochastic Local Search and Temporal Action Graphs, Jounal of Artificial Intelligence Research (JAIR). (2002)
[10] Greenhalgh, C., Purbrick, J., Benford, S., Craven, M., Drozd, A. and Taylor, I., "Temporal links: recording and replaying virtual environments", *ACM Multimedia 2000*, L.A. , October 2000.
[11]Hayes, P.,. The Naïve Physics Manifesto. In: D. Michie (Ed.), Expert Systems in the Micro-electronic Age, Edinburgh, Scotland: Edinburgh University Press,1978
[12]Jacobson, J. and Hwang, Z. Unreal Tournament for Immersive Interactive Theater. Communications of the ACM, Vol. 45, 1, pp. 39-42.
[13]Jiang, H., Kessler, G.D and Nonnemaker, J., DEMIS: a Dynamic Event Model for Interactive Systems. ACM Virtual Reality Software Technology 2002, Hong Kong
[14]Leary, T. Chaos and Cyberculture, Ronin Press, 1994.
[15]Lewis, M and Jacobson, Games Engines in Scientific Research. Communications of ACM, Vol. 45, No. I, January 2002. pp27-31.
[16]Moser, M.A. (Ed.), Immersed in Technology: Art and Virtual Environments, Cambridge (Massachussets), MIT Press.,1996
[17]Pearl, J.,Reasoning with Cause and Effect, Proceedings of IJCAI'99,1999, pp. 1437-1449.
[18]Pearl, J.,. Causality. Cambridge University Press, 2000.
[19]Price, H., Agency and Causal Asymmetry. Mind, 1992. ,101, 501-520,
[20]Riedl, R., 1984. The Consequences of Causal Thinking. In P. Watzlawick (Ed.), The Invented Reality. New York: Norton, 1984
[21]Sato., M. and Makiura, N. Amplitude of Chance: the Horizon of Occurrences, Kinyosya Printing Co., Kawasaki, Japan, 2001.