On the Creation of Dynamic, Interactive Virtual Environments

Kristopher J. Blom[‡]
im/ve, University of Hamburg, Germany

Steffi Beckhaus[§]
im/ve, University of Hamburg, Germany

ABSTRACT

The creation of engaging, interactive virtual environments is a difficult task, but one that can be eased with the development of better software support. This paper proposes that a better understanding of the problem of building Dynamic, Interactive Virtual Environments must be developed. Equipped with an understanding of the design space of Dynamics, Dynamic Interaction, and Interactive Dynamics, the requirements for such a support system can be established. Finally, a system that supports the development of such environments is briefly presented, Functional Reactive Virtual Reality.

Keywords: Virtual Environments, Dynamics, Dynamic Interaction, Interactive Dynamics

Index Terms: I.3.7 [Computing Methodologies]: Computer Graphics—Three-Dimensional Graphics and Realism; I.6.0 [Computing Methodologies]: Simulation and Modeling—General

1 INTRODUCTION

Designing and creating interactive virtual environments is a challenging task. Even basic environments generally require extensive knowledge, spanning multiple sub-areas: programming, modelling, and even interactive design. Building environments that are dynamic and capture the interest of user for extended periods extend the difficulties, requiring domain experts. However, the difficulties of these areas can be reduced through improved software support.

The need for elements of the environment that go beyond static settings is generally accepted. Interactivity is one way to do this. Interactivity has been a cornerstone of the power of Virtual Reality (VR), generally even a defining characteristic. Interactivity envelops the user, making them feel part of the environment. The other way is including aspects that change over time. Such aspects can hold the interest of the user. Having parts of an environment changing over time is not new to VR. However, ever more of the applications of VR explicitly rely on them, often very heavily. For instance, the emerging areas of Virtual Reality Exposure Therapy (VRET) and VR as distraction during physical therapy require the user to be interested in and feel a part of the environment presented, often for longer periods of time.

Environments that achieve these things can be built. Modern computer games demonstrate this well. However, requiring experts, working over extended time periods, to be able to build the environments is limiting. The adoption of VR and related VE based techniques in broader communities depends on solutions that can reduce this work load. The acknowledgement of this can be seen in recent trends in the Web3D community focusing on this and the creation of the special interest group Software Engineering and Architectures for Real-time Interactive Systems (SEARIS) in the VR community.

That support for the creation of such engaging, interactive environments needs to be extended is clear, but it is a problem that has been approached numerous times. In this paper, we want to incite the community to take a broader view and a more formal look at the actual requirements of what needs to be supported. We feel that previous attempts have failed to fully address the issues present in the design and implementation of such environments. In order to support them fully, an understanding of the actual problem has to be developed. Though numerous attempts have been made at support, no works are known to the authors that investigate, formally or even informally, the requirements of such a system in relation to the actual design problem (software design requirements withholding). This paper introduces ongoing research in our group on formally investigating the design space and defining requirements for such a system. Finally, we propose a solution that can fulfill the basic set of requirements presented.

[‡]e-mail: blom@informatik.uni-hamburg.de
[§]e-mail: steffi.beckhaus@uni-hamburg.de

2 DEFINING DYNAMIC, INTERACTIVE VIRTUAL ENVIRONMENTS

The first step that we take is providing a name to our goal to lighten discussion of the area. We are calling the type of environments of interest *Dynamic, Interactive Virtual Environments*. This name is chosen for simplicity and because it expresses the three main points: we are dealing with synthetic (virtual) environments, they are dynamic, and they are interactive. If any of these factors are missing, then the goal is truly something different.

The Virtual Environment concept is likely familiar to readers and the connotations of the term related to VR are implied. More precisely, we mean synthetic computer generated environment with possible real components, i.e. mixed and augmented reality, presented under real-time conditions. A point that needs clarification is that these environments may be unrealistic. Only the imagination of the designer and the prowess of the programmer need limit what is possible, as it is virtual. A common fallacy in the software support design is too strong a focus on environments recreating reality.

The area of interactivity is likewise familiar to readers. By interactivity, we mean that the user can take action in the environment. VEs typically require interactivity in the form of travel. Immersive Virtual Reality requires implicit interactivity, in the form of head tracking. These components are fairly well understood. Interactivity with parts of the VE turns out to be more narrowly understood. We define interactivity as: the user being able to take influence on some part of the environment or its controlling structure.

The final term to define is dynamics, the part that gives "life" to the environment. We define dynamics as:

> any changes over time that affect perceivable changes to the VE, either directly or indirectly.

This definition is purposefully large and encompassing. This is because, we believe, any dynamic will influence the user and is likely to capture the interest of the user, not just virtual humans.

3 DESIGN SPACE AND REQUIREMENTS

Armed with a definition of what we are interested in enabling, the creation of Dynamic, Interactive Virtual Environments, the next step is to understand the design space that they entail. An investigation of the possibilities within the design space provides many insights and shows the astounding depth which can be created. A taste of the results of a study we have conducted is provided here. A more complete review of this will be provided in a future venue.

The design space of interaction is fairly well covered, for instance by Bowman's Taxonomy [3]. However, we will contend later that a major aspect is missing from that analysis. The area of dynamics is well known to us, but not well understood in the context

of virtual environments. The most common and readily thought of dynamics of a virtual environment are the 3D movement of objects. When one considers all of the possibilities, spatial behavior is only a single category of dynamic that can occur.

We have built taxonomies of the dynamics possible in order to better understand the design space and in hopes of drawing out insight for the design of support. The sheer breadth of design space indicates that solutions must reach beyond spatial behavior. This is in stark contrast to the traditional approach of embedding the dynamics and interaction systems into the scene graph - a spatial mechanism. Out of the large design space, classes of types of dynamics can be identified. This means structured support for those individual categories of dynamics can be built.

A second method of categorization is even more informative. If the possibilities are analyzed by how the dynamic is perceived in time, clear requirements on the support for programming dynamics emerge. Time representations are important to the approaches used in computer science [15] and need be considered for building good support. Using this approach, one find three categories:

- continuous infinite, $[-\infty$
- continuous over an interval, $[-)$
- discrete, $(-)$

When placing the possibilities into these categories it is informative to see that almost all identified dynamics fall into the one of the two continuous categories. The difference lies in their boundedness in time, or in the expectation of how long the dynamic lasts. For both the continuous over an interval and discrete dynamics an event semantic can be used. Although events can be understood in many ways [15], we support a view, where events mean the possibility and the occurrence of events being distinct, countable incidents of that event type.

In defining Dynamic, Interactive Virtual Environments, we have striven to make clear that the point of interest is the confluence of all components. The conjunction of interaction and dynamics needs explicit consideration. When considering the combination, two basic concepts can be identified: *Dynamic Interaction* and *Interactive Dynamics*. These form fundamentally different ideas and need to be individually considered.

When putting the stress on the interaction portion of the combination, a concept that deals with the time aspect of the interaction is developed. Formally, we define it as:

> Dynamic Interaction is any interaction (the user taking influence over the environment), where the interaction takes place over time.

This style of interaction is not new to VR. As a matter of fact, almost all of the basic interactions considered in Bowman's taxonomy fit into this category. This aspect of defining time as a fundamental element of the interaction is, unfortunately, rarely done. When time is considered it is typically in relation to filtering of input data. Where this view of time is important is in considering how the interaction should work and how it is implemented. Categorizing the design space of Dynamic Interactions by the time representation introduces additional representations:

- continuous infinite, $[-\infty$
- continuous over an interval, $[-)$
- ordered sets of intervals $[-)\ldots[-)$
- ordered sets of discrete events $(-)\ldots(-)$

Support structures for interaction are not uncommon [9, 14], but supporting the defining characteristic, time, of this very common class of interactions is rare. Those that do are mostly concerned with input filtering. We believe that the time factor needs to be taken into consideration both in the conceptual design of interaction and implementation in order to create effective interfaces.

Conversely, the combination of interaction and dynamics can be made to stress the dynamics component. This represents an often ignored aspect of interaction. Formally we define this as:

> Interactive Dynamics are any interaction (the user taking influence over the environment), where the "object" of interaction is a dynamic.

Interactive Dynamics are an interesting area for exploration. Examples of interactive dynamics exist, though are generally very limited in scope. Often these involve interactions with semi-autonomous entities, such swarming bees, or with virtual humans. Interaction research has largely shunned the topic, with only limited special exceptions (character/avatar interaction and interactive storytelling). Unfortunately, because this area has been little explored, it is difficult to draw good conclusions on good representations. The one area that has addressed interaction with dynamics is animation. The basic methodology that is developed is defining different transition functions that enable the movement from one animation (dynamic) to another when (discrete event) interaction occurs.

A further area emerges when considering interactive dynamics, dynamic interactions with dynamics. We previously identified that most of the classical VR interaction involve dynamic interactions; therefore, it reasons that the interaction involved in the interactive dynamic could be a dynamic interaction. How this works, even at a conceptual level is uncertain and a very interesting question. Unfortunately, building software support beyond enabling the exploration of this space cannot yet be directly performed. Our suggested requirement for this is that the support of continuous time functionality needs to be pervasive throughout the support, available at all levels.

4 Requirements

Based on an analysis of the design space, a set of requirements for software support can finally be captured. Additional requirements should be taken into consideration, particularly common wished for functionality and some standard software design requirements for good practice. Here, the most crucial of the requirements that we have identified are presented:

- support time representations: continuous infinite, continuous over interval, discrete time, and ordered sets
- existence of an input "event" type
- interaction induced "event" dynamics
- interaction induced continuous dynamics (dynamic interactions)
- representation of dynamics better matching human understanding
- user should not deal directly with δt
- time's flow changing itself, i.e. time skewing
- multiple independent time "speeds"
- run-time changeable structure
- support for transitions between behaviors
- undo, including over all time representations
- (soft) real-time capable
- user extensible
- scalable

This core set of requirements should be understood as the basic set needed, further requirements are relevant in a larger scope. The requirements stem from the analysis discussed previously and those after "undo" come from standard practices. Undo is the one issue that requires additional explanation. An undo capability is commonly seen as necessary for any interactive system, as errors

```
flying_la_vache :: a -> SF b Point3f
flying_la_vache _ = proc _ -> do
  velocity <- arr ((Vector3 (-1) 20 0) + ) <<< integral -< Vector3 0 (-9.8) 0
  (Vector3 x y z) <- arr ((Vector3 0 0 0) + ) <<< integral -< velocity
  returnA -< (Point3 x y z)

run_world :: ((Float, Quatf), String) -> SF Double ((Quatf, String), Point3f)
run_world init_newton = proc skew_factor -> do
  let save_event = tag (consumeEvent "saveEvent") Push
      undo_event = tag (consumeEvent "undoEvent") (Pop 1)
      collect_event = merge save_event undo_event
  ball_orient <- undoSF (timeSkewSFd (newtonsCradle init_newton))
                 -< (((), skew_factor), collect_event)
  lavache <- switch (arr (\ e -> (Point3 0 0 0, e)) <<< (after 30 ()))
             flying_la_vache -< ()
  returnA -< (ball_orient, lavache)
```

Listing 1: An example of FRVR Yampa code.

do occur. The standard undo method of storing states becomes intractable, particularly when dynamic interaction is considered. Storing interaction "messages" is likewise impractical. A method needs to be considered that takes into account all the dynamics and interaction that has occurred up to that point, at least in as far as they influence the development of the environment over time.

5 Functional Reactive Virtual Reality

Meeting even this basic set of requirements for a system in support of Dynamic, Interactive Virtual Environments is difficult. Few systems support a continuous representation of time and, more importantly, allow their implementation. The systems that come closest to achieving this are constraint network based systems [7, 17]. However, these fail on other requirements, notably real-time capabilities, scalability, runtime configurability, and events are difficult. The Web3D community has investigating the issue, but has focused primarily on modeling such environments [6, 11]. Those works have not supplied any help for the actual implementation of dynamics and interaction.

Alternatively, a class of systems have focused on the ease of programming/creation of such such environments. Examples of these are Alice [12] and Virtools [16]. The basic premise is to provide the user with building blocks of functionality to allow them to build up what they need by applying and combining those basics. To go beyond the predefined behaviors and interactions, the user must once again program everything by hand, with little specific support for the problem. We propose the use of a different approach, one that can handle the all of the requirements listed above.

5.1 System Introduction

Our proposed system is based on the programming paradigm, Functional Reactive Programming (FRP). FRP was originally conceived for programming animation in a manner that matched more closely the way movement is understood by people [8]. The "functional" aspect of the paradigm is the creation of continuous time "behaviors." To include interaction, the system is also reactive; the reactive nature is performed by changing which behaviors are running, based on discrete events. This design, although first later formally recognized, directly enables the creation of hybrid systems, i.e. combining continuous functions and discrete events [13]. VEs, particularly those in immersive technologies, are prime examples of hybrid systems. Our system, Functional Reactive Virtual Reality (FRVR), builds on top of the FRP paradigm, combining it with existing VR frameworks. The details of how FRVR achieves this combination can be found in [1, 2].

FRVR is based on the most recent incarnation of the FRP paradigm, Yampa. Yampa is programmed in the Haskell language - a strictly typed, pure functional language. Haskell's syntax is based on mathematics, particularly that of mathematical proofs. This makes much of the structure familiar and also enables formal proofs of the validity of code. Yampa relies heavily on the Arrows extensions to Haskell, which enables a procedural style of programming based on computations. Various books provide coverage of the Haskell language and the book from one of FRP's architects, Hudak, covers the the full spectrum of Yampa's Haskell usage [10]. An introduction to the Yampa system can be found in [5].

FRP is designed on a building block style of programming and hierarchical building of functionality. The basic types that make up the Yampa implementation are: Signal Functions, Events, switches, and "parallelizers." Signal Functions (SFs) are the implementations continuous time functions, implemented as signals. The basic functionality for continuous time functions is the integral function, though not the only method. Events are discrete occurrences of a specific Event type and may carry additional information. The basic functionality for reaction, are the switches. Switches are triggered by Event occurrences, causing a change to a new behavior (SF). Finally, the parallelizing functionalities allow grouping of functionalities together, for instance individual Boids into a flock. Complex functionalities are built by aggregating SF functionalities together.

Listing 1 provides an excerpt of FRVR Yampa code. The code simulates two different objects. The full simulation code for a catapulted cow is present; the cow is first "launched" 30 seconds after the start of the simulation. The other component is the simulation of an interactive Newton's Cradle, originally presented in [1]. Here, that functionality is simply used, but extended to allow the user to skew time, for instance slow it done to make selection of the balls easier, and the addition of an undo function. Most of the functionalities used in the example code are explained in following paragraphs.

5.2 FRVR's Functionalities

Yampa provides functionalities that fulfill a portion of the requirements. The basic functionalities provided were presented above. Using them, all of the time representations required can be created. The programmer does not deal with the time delta when programming dynamics in Yampa. Utilizing the connection to existing VR systems, interactions, both discrete and continuous style interactions, are possible; for details of this aspect and for an example see [1]. Unfortunately, Yampa falls short on various of the remaining requirements. Those are addressed in this section.

One of the most pervasive issues with Yampa is that it is designed to hide the internals from the programmer; This is done so that the programmer can not create "space-time leaks," but also inhibits the extension of the system. FRVR contains a revised version of Yampa

that restructures and opens the implementation up to allow the extension of the functionality by the advanced user. This access is only required in cases where the provided building blocks are not sufficient and should rarely be needed. For the remaining functionalities FRVR takes advantage of the FRP concept and, in particular, the implementation method used by Yampa, to create extensions to the Yampa language that meet the requirements listed above. The key to being able to create many of them is Yampa's continuation based signal implementation of the continuous behaviors. Details of the continuation based signal approach can be found in [4].

The details surrounding time's progression are handled internally by Yampa code. However, many aspects of time are desirable to control. Standard timing functionalities (e.g. after 30 seconds) are included in Yampa. FRVR introduces further time based functionalities. Control of the time resolution of the simulation is one of the most fundamental. Time skewing is another. The time resolution, that is the size of the δt, is directly tied to the graphics frame loop. This update rate is often insufficient for simulation. Special extensions allow the simulation, either component-wise or complete, to be sampled at a higher time resolution. Along with this, switching is improved to find a more exact time of internally generated events, an issue that previously caused even simple higher-order time dependent functions to become unstable.

An often considered extension is changing the flow of time. Time skewing functionalities allow this, based on a scaling factor. Time skewing can be performed on the entire simulation or on individual portions of the simulation independently. The time skew can be any non-negative value; skewing time by a factor of zero effectively freezes time. Two versions of this functionality allow both fixed run-time established skew factors as well as run-time dynamic skewing of time.

FRVR also adds an "undo" functionality available at every level. By exploiting Yampa's continuation based stream implementation, the stand of the system at any moment can be saved. The system can jump back to the old time of the simulation as desired. What makes this powerful is that the frozen behavior effectively captures the system "state" consisting of all inputs and decisions up to that point, by "freezing" the continuation. This is achieved simply by wrapping the behavior with a single function.

Finally, a series of transitions functions have been created to help support interactive dynamics, inspired by the methods of computeranimation. Transitioning switches are set up to handle the standard "keyframe" animation style transitions (e.g. linear interpolation between the values of old and new for 5 seconds) and also for the relations of interval algebra developed by Allen [15]. This functions form a framework of support mechanisms, assisting the exploration of interesting area.

6 Conclusions

This paper has focused the generation of support for the creation of what we are terming, Dynamic, Interactive Virtual Environments. We have approached this by suggesting that the design space and requirements of such systems need to be better and more thoroughly defined in order to be successful. Our initial insights into the actual design space of these environments, focusing on particularly on: Dynamics, Interactive Dynamics, and Dynamic Interaction. Coupled with interaction, these form the basic points of interest of what must be enabled by a system of support. Based on that investigation, a list of essential requirements for support of creating Dynamic, Interactive Virtual Environments was developed.

Finally, we have presented a system that fulfills the requirements laid out, Functional Reactive Virtual Reality. The FRVR system is based on the Functional Reactive Programming paradigm and is loosely coupled with existing VR frameworks. Leveraging the power and flexibility of the Yampa FRP implementation, FRVR expands its functionality to handle things like transition functions, independent time skewing, and even undo.

References

[1] K. J. Blom and S. Beckhaus. Functional Reactive Virtual Reality. In *Short Paper Proceedings of the IPT/EuroGraphics workshop on Virual Environments (IPT-EGVE '07)*. EuroGraphics, June 2007.

[2] K. J. Blom and S. Beckhaus. Supporting the Creation of Dynamic, Interactive Virtual Environments. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual Reality Software and Technology*, pages 51–54, New York, NY, USA, 2007. ACM.

[3] D. A. Bowman, E. Kruijff, J. J. Laviola Jr., and I. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison-Wesley, 2005.

[4] A. Courtney. *Modeling User Interfaces in a Functional Language*. PhD thesis, Yale University, May 2004.

[5] A. Courtney, H. Nilsson, and J. Peterson. The Yampa Arcade. In *ACM SIGPLAN Haskell Workshop*, pages 7–18. ACM SIGPLAN, 2003.

[6] R. Dachselt and E. Rukzio. Behavior3D: An XML-Based Framework for 3D Graphics Behavior. In *Proceedings of the ACM Web3D 2003 Conference*. ACM Press, 2003.

[7] L. Deligiannidis. *DLoVe: A specification paradigm for designing distributed VR applications for single or multiple users*. PhD thesis, Tufts University, 2000.

[8] C. Elliott. An Embedded Modeling Language Approach to Interactive 3D and Multimedia Animation. *IEEE Transactions on Software Engineering*, 25(3):291–308, 1999.

[9] P. Figueroa, M. Green, and H. J. Hoover. InTml: a description language for VR applications. In *Web3D '02: Proceeding of the seventh international conference on 3D Web technology*, pages 53–58, New York, NY, USA, 2002. ACM.

[10] P. Hudak. *The Haskell School of Expression*. Cambridge University Press, New York, 2000.

[11] B. Mesing and C. Hellmich. Using Aspect Oriented Methods to Add Behaviour to X3D Documents. In *Proceedings of the ACM Web3D 2006 Conference*, Columbia, Maryland, April 2006. ACM.

[12] R. Pausch, T. Burnette, A. C. Capehart, M. Conway, D. Cosgrove, R. DeLine, J. Durbin, R. Gossweiler, S. Koga, and J. White. Alice: Rapid prototyping for virtual reality. *IEEE Computer Graphics and Applications*, 15(3):8–11, 1995.

[13] I. Pembeci and G. Hager. Functional Reactive Programming as a Hybrid System Framework. In *International Conference on Robotics & Automation*. IEEE, September 2003.

[14] G. Reitmayr and D. Schmalstieg. An Open Software Architecture for Virtual Reality Interaction. In *Virtual Realtiy Software Technology (VRST) '01*, Banff, Alberta, Canada., Nov. 2001. ACM.

[15] F. A. Schreiber. A breviary to time concepts for information systems. *Rivista di Informatica*, XX(1):59–58, 1990.

[16] Virtools. http://www.virtools.com/. Last accessed May 24 2007.

[17] A. Witkin, M. Gleicher, and W. Welch. Interactive Dynamics. In *SI3D '90: Proceedings of the 1990 Symposium on Interactive 3D Graphics*, pages 11–21, New York, NY, USA, 1990. ACM Press.