the ICEA plug-in for Virtual Reality, Immersive Creation and Edition of Animation

Emmanuel Bernier
TERA lab/Heudiasyc
Italian Institute of Technology
Université de Technologie de Compiègne

Indira Mouttapa Thouvenin
Heudiasyc
Université de Technologie de Compiègne

Ryad Chellali
TERA lab
Italian Institute of Technology

Kristopher Blom
Event Lab
Universitat de Barcelona

ABSTRACT
This paper reports on ICEA, the Immersive Creation and Edition of Animation platform we developed in order to support easy computer animations. This plug-in allows users to create new animations and to modify existing ones by interacting intuitively within Virtual Environments. The aim of our system is to enable people without computer animation background to design objects’ motions and animations through simplified and immersive recording, editing and modification processes. Indeed, ICEA uses the immersive authoring principles, and as such, it offers an alternative to classic 3D animation solutions: users have only to perform actions in the virtual environment using any available interfaces to create animations.

ICEA’s core is coded in Haskell on top of the FRVR framework. It builds on a functional reactive paradigm, thus allowing easy scaling of the plug-in usage. ICEA is a fully C++ compatible plug-in supporting most Virtual Reality applications.

To validate the concept ICEA, we implemented and tested it on a "sale’s man-like" problem: users were asked to generate (e.g. create, play, and edit) the movement of a character on what they considered the shortest path linking predetermined points. Our preliminary results show that users familiarize quickly with immersive paths manipulation.

Index Terms: I.3.8 [Computing Methodologies]: COMPUTER GRAPHICS—Application I.6.8 [Computing Methodologies]: SIMULATION AND MODELING—Types of Simulation|animation

1 INTRODUCTION
Objects and characters animation impacts greatly Virtual Reality experiences [19]. It is part of the realism every VR developer is seeking for in order to let users feel immersed. Unfortunately, designing such animations is a complex task and programmers use powerful and complex software, like Blender, Maya, or 3DSmax. The later require advanced skills in computer graphics and extensive training. Indeed, most of the time, developers specify key-frames by hand a long and burdensome work. For some applications, it is possible to use full body motion capture systems to generate a finite set of human-like characters animations. Yet, such solutions require expensive hardware, and the recording sessions are time consuming.

Beside the complexity of the animation creation, the integration of animation in Virtual Environment (VE) is also a complicated matter. Programmers need to export their model to a format that suits their application. Then they use computer animation library like CAL3D, the native Ogre3D classes or OpenSceneGraph manipulator classes to generate the aimed animations. Using such libraries implies coding issues: understanding the platforms, coding compliances, etc. This leads inevitably to difficulties and greatly increases costs of VR applications.

To overcome the previous issues we developed the ICEA platform (Immersive Creation and Edition of Animation). The aim of this platform is to support easy computer animations. We built it as a plug-in to allow naive users to create new scenarios and to modify existing ones by interacting intuitively within Virtual Environments.

We define animation as the modification of position and orientation of one or more entities over the course of time. Varying values over time can be seen as a dynamic phenomenon as defined in Blom and Beckhaus work [3]. Here dynamic is not defined as the well known physical sense, but as "a change over time that affect perceivable changes to the VE, either directly or indirectly". They propose a framework, FRVR, to deal more easily with such dynamic phenomenons. We propose to extend this framework to deal easily with animations. This extension take the form of a set of Haskell functions, to record, replay and edit animation. We hypothesize that this should offer an accessible and efficient alternative for VR designers. To validate our hypothesis we implemented the ICEA plug-in, for Immersive Creation and Edition of Animation, this plug-in is intended to provide a solution for rapid prototyping of animation.

In this paper, we will first assess the usefulness of such a tool. We then specify the technical requirements of our plug-in, before discussing our implementation of the ICEA plug-in. Finally, we will present and integrate how the ICEA plug-in was integrated in a Virtual Reality application, and how users interact with it.

2 RELATED WORK
A lot of applications in VR deals primarily with animation of virtual objects. Manipulation and navigation are two basic interactions in VR, that can be seen as animations creation task. Those interactions are still studied extensively [5].

Some research has been conducted on the best way to store animations. Luciani et al. [13] proposed a format to record such data by saving the animations as simple monodimensionnal scalars organized in coherent geometric sets. Those geometric sets are then regrouped to form correlated groups. Olive et al. built on such format, to propose a VR application for training in maintenance operations on complex machinery. In this application, sequential animations of mechanical parts is the content to be learned [15]. That is to say, animation is considered as content that need to be created and consulted. In this application, the animations are generated by

---

*e-mail: emmanuel.bernier@iit.it
†e-mail: ryad.chellali@iit.it
‡e-mail: indira.thouvenin@utc.fr
§e-mail: blomk@acm.org
skilled operators with no background on computer science, but they possess unique knowledge of the modelised machinery. However, there is no possibility to modify the existing content.

There is a lot of VR applications that need tools to deal with animations generation and replay. In the field of entertainment we can cite guided tour of Virtual Environments [9], [2] where the user are discharged of the navigation task. The Immersive Music Painter is another example where the user can draw music in 3 dimensions [1]. In the field of VR-assisted learning, trainees can learn complex tasks, such as karate moves [4], calligraphy [22] or even industrial procedures [17]. We can also cite the work on visualization of time time varying phenomenons [21]. All of these applications either use predetermined animation created through complicated and costly procedures or live performance by a trainer.

In recent years the concept of Immersive Authoring was defined for AR applications, as "an authoring method that allows the direct specification and testing of the content within the execution environment" [11]. This method enable AR applications designers to create their application following the concept of "What You Get Is What You Experience" [12]. This is particularly convenient for 3D environments where it is not possible to embrace the whole experience in just one glance. Immersive Authoring was also experimented in VR, by Steed and Slater [20]. Even if they did not use the term authoring, they propose to edit VE while immersed into them. The main objective of this tool was to rapidly prototype VR application, by trying various metaphors. However none of these systems are performant when it comes to creating the animation of virtual objects. Another AR immersive authoring tool [7] allows to define animation by using free floating hand movement, a much more natural way. However, drawing 3D curves is complicated, even if some solutions exist. Some of these solutions target graphic artists [6], other allow to merge multiple curves [8], or use dynamic dragging to smooth input [10]. However, such curves editing methods cannot be used for animation since they neglect the timing of the movements.

3 DEFINING AN IMMERSIVE ANIMATION AUTHORING TOOL

3.1 Sample target application

To illustrate the need for a platform such as ICEA, we describe the 2 following VR applications that would greatly benefit from animation edition tool.

3.1.1 Use case 1: Surgery training application

An instructor surgeon works in a virtual environment corresponding to a given surgery problem. In this environment, the instructor performs a "perfect" operation. This operation is recorded to create an animation. That animation is then used for demonstrations for the surgeon trainees. First, the animation is replayed to students. Then students are asked to perform the surgery, eventually with skilled professionals to generate, evaluate and modify the animations. A tool that allows to do this in an efficient way does not exist yet. Our goal is to create the foundation for such a tool.

3.2 Technical requirements

Our objective is to develop a VR solution to easily manage animations from creation to exploitation. The system we want to design should have the following set of features: recording, replaying and modifying animations immersed into a Virtual Environment. We need this system to be easily integrated into existing VR system with limited efforts. We also want our system to be expanded with limited efforts by programmers. To this aim we define the following set of feature

- Record new animation,
- Interactive replay a given animation,
- Easy exportation and importation of animation,
- Immersed edition of animation,
- Fine tuning of animation.

First, our solution should be able to record new animation. Any virtual object movement in the environment could be recorded, no matter how it is generated, or how many articulations it possesses. To this end we use a simple keyframe format, composed of a global reference, an articulation reference, the key’s number, 7 floating point values for position and orientation of the joint and the sampling rate as last value.

Once we have stored an animation into the system, we need to replay it. We need to support the replay of the animation as it was recorded, by actually moving an object in the VE. We also want the system to display the whole trajectory at once to allow a better understanding of the animation in space. Moreover, we want the replay to be interactive, interactivity can be achieved by adjusting the passing of time for each animation.

Our final and most important goal is to offer the possibility to modify recorded animation. Animation editing can take place either during the replay or in asynchronous fashion. Asynchronous edition is achieved by defining subparts of the animation to remove and new animation fragment to replace them. Our system can also alter an animation while it is replayed, for finer tuning purposes.

4 THE ICEA EXTENSION: IMMERSIVE CREATING AND EDITING OF ANIMATION

4.1 The ICEA architecture

The most obvious choice for our system was to develop it using only C++, that way it could be integrated with C++ VR applications whenever the source code is available. However we decided instead to base the ICEA plug-in on top of Kristopher Blom’s FRVR system. From a technical point of view, the FRVR framework (for Functional Reactive VR) [3] is an open source framework designed especially to ease the manipulation of time sensitive phenomenons.

The FRVR framework enable to compute part of the VR simulation using Haskell programming [16] while using classic C++
for generating graphics and all the other aspects. Haskell is mostly known for being a lazy language, hence trying not to compute as long as it is not needed. This allows programmers to produce efficient code, without spending much time optimising it. FRVR uses specifically a particular extension, called AFRP or Yampa [14]. In the Yampa extension, programs run as continuous function of time that can be affected by "events", those functions are called Signal Function. The ICEA extension is composed of Haskell parametric functions generating FRVR compatible signal functions.

This framework was chosen for multiple reasons. First the internal representation of time allows easier manipulation of time sensitive phenomena. In FRVR, a function runs as a continuous signal, developer do not have to handle any time delta. FRVR also offers an event based architecture, that allows efficient control of the signal functions states. Since events can be issued easily through a simple C++ function, the programmer does not need to know Haskell to interact with FRVR signal function. Since FRVR is a functional programming language, it is easy to generate functions through parametrization and composition, allowing for efficient expansions. Last FRVR has built in key-frame animation : FRVR transforms a set of key-frames into a continuous signal of coordinates over time. Naturally, FRVR can be integrated easily into any C++ based VE, even in preexisting ones.

The ICEA plug-in make 4 elements working together: a C++ VR application, an Haskell thread, a shared memory blackboard and a database. The C++ thread generates the environment, updates the environment’s geometry, gets input from the interfaces and does all the computation not directly related to animation. The Haskell thread is used to save, generate and modify animations. The shared blackboard is used to pass object coordinates and events between the VR application and the Haskell thread. The animations are stored in the database. This allows safe concurrent access of the data by the two different threads and external applications. The database also allows to easily manage meta data attached to the animations.

The ICEA animation format is close enough to the VRML format to convert easily from each other. Both are based on keyframes. The only two significant difference is that in VRML/X3D, the time associated with each frame is the absolute time since the beginning of the animation, whereas in ICEA it is the time passed since the last frame. The other difference is that in VRML/X3D, position and orientation are handed by two different node. This might be a problem if in a VRML animation to be converted, position and orientation frames are not synchronised. Beside that the conversion is straightforward.

FRVR provide the architecture of our plug-in, however ICEA’s functions and mechanisms are completely new. The plug-in is written in Haskell.

4.2 Implemented functionalities of the ICEA system

ICEA is composed of Haskell functions. Those functions, during compilation, create the signal functions run by FRVR that allow to manipulate animations. An ICEA function always take a string identifier as parameters and eventually additional parameters to generate a signal function.

Each signal function listen to specific Event. Each Event name is created by concatenating the signal function identifier, like editing, and the event name, like start-edit. This mechanism allows programmer to add as many animated objects as they need using only a reduced set of ICEA functions.

The following function have been created in ICEA : recording, replaying, off line editing, on-line editing using spring damper model, time skewed replay and a function to rewind or fast forward an animation.

4.3 Integration with C++

A programmer has two things to do to use ICEA in C++: put and retrieve data from the blackboard, generate and listen to events.

The C++ can then exchange coordinates, integers and floats using the functions insertCoordData(string sharedName, float inCoordinate) and retrieveFloatData(string sharedName, float outCoordinate).

Event are created with the C++ function insertEvent(string eventName). This event is then detected by the Haskell thread. The programmer also need to monitor what happen on the Haskell side using the C++ function isEvent(string eventName) and clearEvent(string eventName).

4.4 Example of ICEA architecture in a generic application

This section explain the content of Figure 3. In this configuration, the animations of 2 virtual objects (A and B) can be recorded, while object C can reproduce a preexisting animation.

For the recording of object A and B, the C++ thread puts the coordinates of the objects to be recorded into the blackboard. It can control independently which animation is recorded or not by issuing events to the corresponding Signal Function.

To move object C, the C++ thread puts a float value in the blackboard. This value will control the speed of the replayed animation. The C++ thread also reads the updated coordinate of object C in the blackboard. Last, the C++ thread trigger the replay through an event and listen to a specific event in order to know when the replayed animation has reached its end.

On the FRVR side, for this application, the programmer decided to use 3 signal functions, corresponding to the 3 objects in the Virtual Environment. The light red squares represent each a signal function, in this example the signal functions are named recorder1, recorder2 and SkewedReplayer1.

These signal functions are parametrized before compilation, this allow the Haskell compiler to optimize them. In this example, the 3 Signal Functions are generated by 2 ICEA functions record and skewedReplay. In this example two signal function (recorder1 and recorder2) are created using the same ICEA function ; beside the signal function identifier, different parameters may differentiate them, like sample rate or database’s table. Every signal function keeps looking for its events in the blackboard. The Skewed Replayer1 gets key-frames from the database and puts the updated coordinate of object C in the blackboard. It also generate events when the animation has been replayed.

The FRVR side of this application would be implemented in 7 lines of code. The C++ counterpart would be 10 lines long.

In this example we do not specify how the user interacts with the environment. This is because our plug-in does not impose any constraints on the way events are generated. It could be by key press, button press, voice recognition or any interface deemed fit by the programmer.

4.5 ICEA original features

In this section, we will present the most advanced and interesting functions implemented in ICEA.

4.5.1 Skewed replay

We call skewed replay a replay function where the time course is scaled by a positive value f. If f = 0.5 the animation takes twice as much time ; if f = 2, the animation is twice as fast. This is a really simple function, however, depending on how f is calculated, this function can have many uses.

For example if we give the user the ability to directly affect f, the user can fast forward the portions of the animation that is not interesting in, pause the animation, or slow the time to analyze more closely a specific part of the animation.
We can also use this skewed replay function to animate a virtual object that show the animation to the user as he gets closer to the object. This would be done by making \( f \) an invert function of the user’s distance to the virtual object. This can be used to guide a surgeon trainee using a visual ghost, as mentioned in section 3.1.1.

4.5.2 On-line animation editing: dynamic deformation

Using FRVR strengths, it is possible to modify animations as they are being replayed. Such an approach allow for a more direct manipulation of the animation in the sense of Schneiderman[18]. We offer the possibility to deviate the object of his planned trajectory by adding a force to a mass-spring-damper pseudo-physic model. One end of the spring is attached to the original animation and moves along it. The other end of the spring is the modified position of the object. This is illustrated in Figure 1.

This model is implemented in Haskell as shown in listing 1. This is simulated by simply applying the second Newton’s law of motion, summing the force applied to the system and integrate it twice to obtain the new relative position of the mass spring system. Since we simulate, this system we can change the applied force at any moment. The modifications made possible by such a system are shown in Figure 4.

The most interesting point about this function is that the force can be generated by basically anything. We can use: a force fields, depending on obstacle proximity; a force function of the user proximity in such way that the object is attracted or repelled by the user; using an haptic interface, users can feel the force they apply to influence the animation. Depending on the mass, spring and damper parameters, the virtual object can follow loosely the animation.

By composing this dynamic deformation function and a time skewing function, we can also guarantee that the animation speed will not exceed a certain level. This could be done by slowing the original animation according to the velocity of the mass spring model alone.

Those two examples show that each ICEA feature can have different uses, depending on how the inputs it receive.

5 Integrating and preliminary testing the ICEA system

5.1 Development and integration in an existing Virtual Environment

The ICEA system was first used in a VE handled by VRJuggler using OpenSceneGraph. To test the core functionalities of our system, we focused at first on a simple task: navigation. This allowed to integrate and test all ICEA functions in a preexisting Virtual Environment picturing a castle. In this environment users interact through a Nintendo Wii mote. They move the avatar by using the directional cross and edited animations by using the remaining buttons.

The first ICEA functions tested were the record function, and the replay function. We then focused our attention on the edit function.

5.2 Off-line editing experiment

To test the efficiency of our immersive editing plug-in, we designed a simple application. Subjects were asked to explore a VE and define a path that connect a set of points of interest. Those set were comprised of 5 to 9 points. The users had to create the shortest animation both in time and walked distance that visits all the points. They controlled their avatar from a third person perspective. This experiment carried out with 12 subjects using a non stereoscopic wide-screen (2.65 m by 1.95 m).

5.2.1 Animation edition

We tested three authoring conditions: Just Record, Validated Editing and Quick Editing. In the Just Record condition, the edition function was disabled. In the Validated Editing condition, subject had to see the modified animation and validate it before being able to modify it further. In the Quick Editing condition this validation phase was skipped. This difference allows us to see whether the subject modifies successfully his animation. If the users cancel their modified animations, it would show that the editing process is too complicated. If the subjects do not cancel their modifications, it would mean that this particular immersive authoring method allows subject to modify their animation as they intended.

Animation editing, in this implementation, was performed in 7 steps. The replay consisted in reproducing the virtual camera and user’s avatar animation.

1. subject starts replaying the animation,
2. subject specifies a first frame,
3. subject specifies a second frame,
4. subject automatically got back to the first frame,
5. subject starts recording the new part,
6. subject stops recording the new part,
7. subject verifies the newly generated animation.

At each step, excluding the third, the user had to press a button to move on to the next step. After step 6, the fraction of the animation between the 2 selected frames is then deleted and the new part is blended into the whole animation.

5.2.2 Results

All our subjects completed the trials successfully. We then verified that the subjects did not cancel often a modified animation (5 canceled modification for 85 trials). Moreover, the subject tested under the Quick Editing condition did not complain of their modifications. This proved step 7 of the animation edition process was useless. Moreover, implementing an history of modified animation would favorably replace this step, since user would be able to cancel their modification afterwards.

Analyzing the subjects activity allowed us to highlight the most critical part of this immersive authoring protocol. Thus, defining the part of the animation to be suppressed proved to be both critical and tricky, because selecting a specific frame while the system is replaying imply to remember perfectly the animation. To solve this problem, we developed a more flexible replay function that allows to replay an animation backward or forward at any user defined speed. This should allow users to select more easily the fragments they want to modify.

![Diagram of a spring attachment during on-line animation editing. One end on the old animation, the other on the new animation. When no forces are applied, the old and the modified animation blend into one another.](image)
use a walk animation created in a 3D modeler while the character is not incompatible with classic animation methods. The designer can use very generic tools to be integrated in various applications. It is possible from the programmer point of view. Using FRVR mechanism, even a Haskell novice can make use of the ICEA functionalities for his VR application. The ICEA platform is conceived as a walk animation, even a Haskell novice can make use of the ICEA functionalities for his VR application. The ICEA platform is conceived as easy to be animated.

5.3 Integration of on-line animation edition

We then integrated the on-line animation edition, using the mass-spring-damper model. In this case, the edition was triggered by a button press. During the animation’s replay, the directional cross of the Wiimote was used to generate a force in the horizontal plan. This allows subjects to modify their animation as shown in Figure 2 by combining both edition methods.

6 Conclusion

In this article, we presented a plug-in to create and edit displacement while being immersed in a VE. The Immersive Creation and Edition of Animation (ICEA) allows people with no experience in computer animation to produce some through intuitive interaction in VE instead of complicated animation softwares. The system not only records and replays animations, but it also enable the modification of those animations using the same intuitive interactions that were used to create them or intuitive metaphors like mass-spring-damper model.

Moreover, the integration of the ICEA plug-in, is made as easy as possible from the programmer point of view. Using FRVR mechanism, even a Haskell novice can make use of the ICEA functionalities for his VR application. The ICEA platform is conceived as very generic tools to be integrated in various applications. It is not incompatible with classic animation methods. The designer can use a walk animation created in a 3D modeler while the character walks along an animation generated by ICEA.

The next step is to extend the ICEA plug-in, to edit more complex animation. We believe that such an approach would be profitable for complex animation task such as full body motion capture. It would be an enhancement to be able to review the motion capture data after the first capture, and then select a subset of the animation to be modified without capturing the whole animation again. For example, modify only the movement of one arm, while keeping the rest of the animation unchanged. Some new mechanism needs to be created to support such a feature, like selecting a subset the model to be animated.

Acknowledgements

The authors would like to thank Steffi Beckhaus, for her support during the earliest phase of this project. With the rest of the IMM.VE lab staff, they provided very valuable hindsight. We also want to thanks the subject for their participation and feedback during the testing phase.

References


Figure 3: ICEA plug-in architecture: The four parts of a VR application using the ICEA plug-in: C++ program, associated with a shared memory blackboard, a FRVR thread and a database. This Figure is explained in detail in section 4.4.

Figure 4: Modification of a sinusoidal and an hyperbolic curve by applying varying forces to the Mass-Spring-Damper model. The system is critically damped, with mass = 1 kg, spring constant = 1 N.m^{-1}, damping coefficient = 2 N.s.m^{-1}. Speed along the X-axis is constant.