TimeClock - Flexible Animation Control in X3D

Olavo Belloc * Marcio Cabral† Marcelo Zuffo‡
Laboratory of Integrated Systems - University of São Paulo - Brazil

Abstract

In this paper we propose an alternative approach to create animations in X3D. This approach allows extended flexibility to control animations during run-time. Among the extended features, it is possible to: control the speed of the animation; play the animation backwards; repeat any specific time interval of the animation and access any key-frame instantly. In order to illustrate this we propose a new node called TimeClock node. This node implements the same functionalities of a TimeSensor node but with the ability to independently set the time frame, overcoming current X3D limitations in the time model specification for creating animation with interpolators. We think this approach is useful for both animators, developers and users: animators can carefully analyze on the fly their work, users can easily control an animation using a DVD like interface and developers do not need to worry about creating several different interpolators for the same animation. We present our current results along with some examples of usage.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Languages I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation,Virtual reality

Keywords: X3D, animation control, key-frame animation, interaction

1 Introduction

The combined use of Interpolators, TimeSensors and Route nodes in X3D allow animations to be created and used in Virtual Environments. However, once these animations are started, the only level of interaction provided is the start/pause and stop paradigm. It is not possible to randomly access a specific frame or jump to a specific point in the interpolation. Also, it is not possible to play the animation backwards, from any specific point. The speed in which the animations are played are fixed too (at design time) so it is not possible to play them in any desired speed during run-time. This paper proposes an alternative node to the TimeSensor node as a way to overcome these limitations.

This node provides a flexible and interactive scenario at run-time, allowing the user to freely control an animation (play, pause, forward, fast forward, rewind, fast rewind, access any frame). The idea behind this is to overcome the limitation of X3D's time model by providing an alternate TimeSensor node that we call TimeClock node. The TimeClock node mimics the same functionalities of the TimeSensor node with added features. Since it runs independently of the regular "world time", features as variable speed of playback, backwards playback and random access to any frame of the animation are available.

These features can be used in different scenarios and applications such as: only one interpolator is needed for animations that have an object going back and forth on the same path; animations' speed can be interactively changed and visualized at run time; specific portions of animations can be carefully inspected without the need to introduce separate interpolators; an X3D interactive scene can be played with the same features as a DVD movie.

*e-mail: belloc@lsi.usp.br
†e-mail: mcabral@lsi.usp.br
‡e-mail: mkzuffo@lsi.usp.br

2 Motivation

Over the years the X3D standard has been evolving into a highly interactive 3D environment. Lots of features have been added since the introduction of VRML, which opened a wide range of possibilities for applications and uses. The idea of bringing 3D everywhere is bringing more and more users to adopt X3D. However, building interactive animations in X3D are not a fluid process. For every single change in an objects position, color or other attribute, one interpolator node must be created. Even if the animation consists of a single object going back and forth on a unique path, two interpolators are needed.

Moreover, each animation has a cycle time it has to be followed. Currently it is not possible to access a specific frame (or time) during the animation cycle at run-time without using complex scripting, which are time consuming (implementation and debugging) for developers and not easy for regular users.

An animation in X3D is made using the nodes inherited from the abstract `X3DInterpolatorNode`. These nodes usually receive an event from a TimeSensor node in order to perform the interpolation (animation). According to the definition, the TimeSensor node generates events as time passes, which means that an animation is drived continuously by feeding the `set_fraction` fields of the interpolators through pre-defined routes. This restricts the way animations are built. For instance, in the case of animating a notebook being opened and closed, there must be two different interpolators associated with opening and closing actions. Each of these interpolators contains almost exactly the same path (keys and key-values) but they are reversed from one another.

Another example that shows the limitation for building and using animations today is in the case when the user wants to play random portions of it, chosen at run-time. For instance, when using X3D animations in a biology class, it is up to the students to decide when to repeat a specific portion of an animation, based on their doubts. Unless these doubts were predicted at design time, it will not be possible to play just that specific portion of the animation.

Choosing at run time the speed an animation will play, at any point, or playing it backwards, allows added flexibility for both users and programmers. Accessing and playing repeatedly specific portions of it, without having to pre-define changes at design time, stimulates developers and users by not restricting them to only a few actions that can be performed.

The intention of this research work is to provide more flexibility to create and interact with animations as well as facilitating the creativity process of animators, by making them concentrate on what is important. We try to accomplish these by providing an alternate approach to the TimeSensor node.

3 Previous Work

Literature on this subject has not been extensively found although it seems to be a very desirable feature for X3D. Most of the work found is restricted to simple X3D examples available at the World Wide Web.

Some of these examples have been done by Bruztman *et al.*[Brutzman 2006] and made available to the public at the *Savage authoring tools*. Some of them try to simulate a DVD player interface. However, none of them achieved the desired features described in this paper. Usually behind the menu GUI, there is a poor X3D node interface with a hugh set of scripts and programming supporting it, making it difficult to just use it in any scene. Unfortunately we could not successfully test it with either one of the standard players available such as BS Contact [Bitmanagement 2006], Octaga [Octaga 2006], Flux [Medial Machines 2006] or CosmoPlayer [Karmanaut 2006].

Another approach that simulates one of the desired features, the backwards playback, can be partially achieved with a combination of interpolators where the key values are cleverly defined [LightHouse 3D 2006]. This is accomplished by defining a scalar interpolator with the key value $keyValue = [0,1,0]$. These keyValues generates `value_changed` events ranging from 0 to 1 and then back to 0 again, completing the loop. This scalar interpolator is then used to feed the others interpolators used for animation, simulating a TimeSensor node going back and forth. However, this implementation is limited to a continuous playback: It is not possible to play an specific portion of the animation nor to alter between forward playback and backward playback at run-time, with different speeds.

Very interesting work have been done by Stocker [Stocker 2006]. Using filters theory, stocker proposed a set of nodes that smoothly creates transitions from one value to another. The proposed features improve the quality of the interpolation algorithms over those already included in the X3D standard (linear interpolation). However it does not introduce any time controlling related characteristic. In the example of closing and opening a notebook, some script programming are still needed to switch between destination values. However, if used in conjunction with our set of nodes, no programming would be necessary, resulting in a better and clear design.

4 Proposed Node

The most common approach for creating animations within the X3D standard goes through the TimeSensor node. The suggestion of a more flexible node, the TimeClock, intends to provide a new model for driving animations over time. With some new fields added to its interface, despite of also generating events as time passes, the user is able to control how these events are going to be generated. Although it's not possible to set how frequently these events are generated, it's desired to have more control over their generation and evaluation over the time.

Actually, the TimeSensor has a very simple event generation model. The most important event thrown by this node is the `fraction_changed` event. The `fraction_changed` is thrown for every simulation tick, with values representing the fraction of the cycleInterval time already elapsed. The TimeClock node is able to control the behavior of the `fraction_changed` event generation, thus, enabling the user to have a more flexible control over their animations.

4.1 TimeClock Node

According to the definition of X3D's Time component [Web 3D Consortium 2006], browsers that conform to the specification generates `fraction_changed` events for TimeSensors nodes as time passes. The *time* stated here is the computer's time relative to $00:00:00$ GMT January 1^{st}, 1970. All events are generated with a timestamp t, and regardless of what happens, any processing occurring to timestamp t will generate an event with a timestamp greater than t.

So, by definition, it is not possible to have the `fraction_changed` field with values smaller than the ones already processed. This due to the fact that the `fraction_changed` field is calculated with the elapsed time from the beginning of the cycle, using the system clock as its reference.

Each TimeClock node instance actually implements its own clock, that runs parallel to the system clock. This implementation allows the user to setup the speed and the direction of its built-in clock, thus changing the behavior of the `fraction_changed` event generation. Among other features, the user is able to configure the speed of the clock and if the time should go forth or back.

Although the TimeClock has a built-in clock, it's important to emphasize that every event generated by the node, mainly those associated with the fields inherited by the abstract node `X3DTimeDependentNode` like startTime, stopTime, pauseTime and resumeTime receive and generate values containing the time value for the system clock and not the value of the built-in clock. It's also important to note that the time value of the built-in clock is not used for event input/output or event timestamp values, it's only used for the purpose of evaluating the value of the `fraction_changed` field.

4.2 Interface Description

The TimeClock node is designed to work exactly the same as the TimeSensor node. The TimeClock nodes possess the same fields as the TimeSensor, carrying the same functionalities. Thus, if you simple substitute a TimeSensor node by a TimeClock node, the animation will continue to work as it have been previously working.

However further fields have been added to the TimeClock when compared to the TimeSensor node. These fields add the desired flexibility of "controlling time".

These are the fields and inheritance of a TimeClock node:

A TimeClock node has the following extra fields: `SFBool [in,out] pitch`, `SFFloat [in,out] forward` and `SFTime [in,out] fraction`.

The `pitch` inputOutput field specifies a multiplicative factor for the speed of the built-in clock node. It determinates the speed that the TimeClock node will take to run through all the `cycleInterval` time. Valid values for this field are float numbers greater than zero. The default speed is $pitch = 1$, which will play the animation according to the browser's current capability of

```
TimeClock : X3DTimeDependentNode, X3DSensorNode  {
  SFTime   [in,out] cycleInterval    1      (0,∞)
  SFBool   [in,out] enabled          TRUE
  SFBool   [in,out] forward          TRUE
  SFFloat  [in,out] fraction         0.0
  SFBool   [in,out] loop             FALSE
  SFNode   [in,out] metadata         NULL
  SFTime   [in,out] pauseTime        0      (−∞,∞)
  SFFloat  [in,out] pitch            1.0
  SFTime   [in,out] resumeTime       0      (−∞,∞)
  SFTime   [in,out] startTime        0      (−∞,∞)
  SFTime   [in,out] stopTime         0      (−∞,∞)
  SFTime   [out]    cycleTime
  SFTime   [out]    elapsedTime
  SFBool   [out]    isActive
  SFBool   [out]    isPaused
  SFTime   [out]    time
}
```

playing that animation. Increasing this number to 2 will make the animation play twice as faster as the default, thus taking half of the time to run through all the `cycleInterval` time.

The same effect available through the `pitch` field can be achieved by changing the `cycleInterval` value of a TimeSensor node. However, the `pitch` field can be used to synchronize with other audio nodes like AudioClip, thus allowing the sound to always follow the animation speed and time.

The `forward` inputOutput field indicates the direction that the built-in clock will run through the `cycleInterval` time. If `forward` field holds the TRUE value, the clock will go forth, from the current time to the end of the cycle and, therefore, the `fraction_changed` event will always throw increasing values, from 0.0 to 1.0. If `forward` field holds the FALSE value, the clock will go back, from the current position to the beginning of the cycle, therefore, the `fraction_changed` event will always throw decreasing values, from 1.0 to 0.0. When a `set_forward` event is received while the TimeClock node is paused or stopped, the TimeClock node will resume from the current time in the direction specified by the field.

Actually, the `forward` field can be used as if it were the "sign" of the `pitch` field. If `forward` is FALSE then `pitch` will be used as if it were negative.

The `cycleTime` outputOnly field event is generated with system clock time at the beginning of the current cyle. As the `cycleTime` field is frequently used for synchronization purposes such as sound with animation, It will only be generated at the beginning of every cycle if the `forward` is with the TRUE value. If the built-in clock reaches the beginning of the cycle going backward, the `cycleTime` event is not generated.

The `fraction` inputOutput field is responsible for holding the percentage of the cycle already completed by the built-in clock. The improved feature over the TimeSensor node is that instead of being an outputOnly field, the `fraction` field has input/output capabilities. The `fraction_changed` event can also be used to feed the interpolators, but besides of that, the user is also able to use the `set_fraction` event to change the current position of the built-in clock in the cycle. This is a key feature of the TimeClock node: it allows access to any instant of time over the interval determined by the `cycleInterval` field.

The fields inherited by the abstract nodes: `X3DTimeDependentNode` and `X3DSensorNode` works exactly the same way. For instance, the `time` outputOnly event is generated with absolute times of the system clock for each tick.

The behaviour of a TimeClock node could be implemented as an extension over the TimeSensor node just by adding the necessary fields to its interface. However, the solution of adopting a new node with fairly similar interface was chosen because the name of a node really suggests its purpose and functionalities. The name TimeSensor suggests a node that generates events as time passes, constrained to the system time. On the other hand, the name TimeClock suggests the creation of a flexible clock, detached from the system time, thus allowing the user to control a few parameters of it (see figure 1).

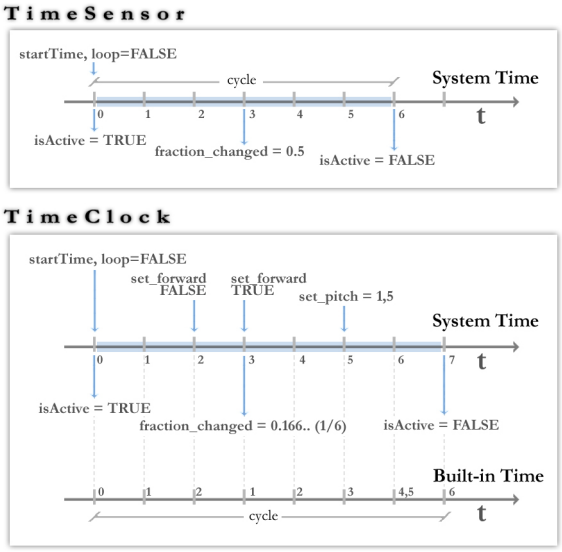

Figure 1: TimeSensor and TimeClock comparison

4.3 Implementation

The TimeClock node has a function `clock_tick(.)` which is evaluated each simulation tick. This function is responsible for updating the built-in clock and generating the main events for each tick. The operation of the built-in clock can be represented with a time line that has the range determined by the `cycleInterval` field, in such a way that the time associated with the built-in clock is always in the range of $[0, \texttt{cycleInterval}]$.

Our implementation of the TimeClock node considers the instantiation of a TimeSensor node inside its Prototype declaration. This approach tremendously facilitates the TimeClock implementation in such a way that every field inherited by the abastract node `X3DTimeDependentNode` is directly connected to the TimeSensor instance. The TimeSensor instance is accessed through the `timer` variable inside the script implementation.

As X3D browsers don't necessarily provide a constant frame rate, the function `clock_tick(.)` is not called on a regular basis, so the first thing it does when entered is evaluate the elapsed time since the last clock tick. This value is stored at the `elapsed` variable and it is further used to calculate the new time of the built-in clock.

```
function clock_tick( now ) {

    elapsed = now - lastTick;
```

The current time of the built-in clock is stored at the `timeCycleInterval` variable and its basically used to generate the `fraction_changed` event. However, the `forward` field and the `pitch` field must be properly taken into account to correctly update the `timeCycleInterval` value. The code below updates the built-in clock for a TRUE value of the `forward` field.

```
timeCycleInterval += (pitch * elapsed);
if ( timeCycleInterval > cycleInterval ) {
    if ( loop ) {
        timeCycleInterval = 0.0;
        cycleTime = now;
    }
    else {
        timeCycleInterval = cycleInterval;
        timer.stopTime = now;
        return;
    }
}
```

With the example above, its not difficult to consider the case for a FALSE value of the `forward` field. Among other things, it's necessary to decrement the value of the `timeCycleInterval` considering a negative sign for the `pitch` field and do the approprite checkings when the `timeCycleInterval` reaches zero. It's also important to note that its not recommended to generate the `cycleTime` event in the case of having the `forward` field as FALSE.

After the update, the output events must be generated with the simple instructions below:

```
fraction = timeCycleInterval / cycleInterval;
time = now;
```

In this example, the `fraction` inputOutput field must be connected to `fraction_changed` field in the prototype declaration of the TimeClock node. For the complete implementation, see the TimeClock home page [TimeClock 2006].

5 Results and Applications

The TimeClock node was first tested in already existent X3D files. Inserting the extern prototype declaration in the beginning of the file and simply substituting the TimeSensor node by a TimeClock node worked well: the X3D files behaved exactly the same. The TimeClock was also used to enrich common animations available over the internet, like simple doors or windows that open and close, elevators that goes up and down, and the like.

Most simple animations available over the internet happens to go through all the animation cycle with a single user interaction. For instance, the simple example of a door open/close is normaly developed with only one interpolator (the simplest case), and in this scenario, the user is able to watch the door opening and just after closing, by issuing only one click. To allow the user to open the door with only one click and further close the door with another, the designer would probably need two interpolators and also some more work for event routing. This can be easily done with the `forward` field available in the TimeClock node.

The TimeClock node itself provides enough functionalities to accomplish the features proposed above, as going forth and back, choosing the current time at run-time and setting the speed of the built-in clock. But in order to make these features easily accessible to a wide variety of applications, a simple DVD like interface was developed following the X3D standard. This interface can be used as an X3D node through the extern prototype declaration and it is called the TimeMenu node. The TimeMenu node encapsulates a TimeClock node instance and allows the user to have full control over a specified animation. The TimeMenu can be easily configured as a HUD (Head Up Display), to be constantly available to the user.

The TimeMenu node displays a pretty easy user interface that can be shown or hidden. This interface has the same functionalities of a DVD player interface: play/pause, stop, fast forward, fast rewind and a time-slider to access any frame of the TimeClock cycle time.

All the nodes developed for this paper, including the TimeMenu node, are available at the TimeClock home page [TimeClock 2006]. They were developed and tested using the BS Contact X3D/VRML player, from Bitmanagement [Bitmanagement 2006].

6 Conclusion and Future Work

In this work we proposed an alternative approach to play animations in X3D. This approach extends current features for using and creating animations in the standard. Most important, it permits flexible control of animations in run-time, allowing an animation to be played with a DVD player like interface. A set of nodes were developed to show the desired features. The core node developed was the TimeClock node that mimics the TimeSensor node with the added flexibility of controlling the time of the built-in clock.

All these resources were created for the purpose of easily having a more flexibel control over time available in the X3D standard. These improved features can become a rich tool for both novice and expert users to explore animations in X3D. We plan to implement a practical and easy to use set of animation tools for novice users using the features availables in the TimeClock node.

References

BITMANAGEMENT, 2006. Bs contact vrml/x3d player. http://www.bitmanagement.de, December.

BRUTZMAN, D., 2006. Authoring and visualization for advanced graphical environments. https://savage.nps.edu/Savage/, December.

KARMANAUT, 2006. Cosmo player. http://www.karmanaut.com/cosmo/player/, December.

LIGHTHOUSE 3D, 2006. VRML interactive tutorial, interpolator examples. http://www.lighthouse3d.com/vrml/tutorial/index.shtml?intex, December.

MEDIAL MACHINES, 2006. Flux player. http://www.mediamachines.com/index.php, December.

WEB 3D CONSORTIUM, 2006. Extensible 3D. http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification. ISO/IEC 19775-1:2004.

OCTAGA, 2006. Octaga player. http://www.octaga.com/, December.

STOCKER, H. 2006. Linear filters: animating objects in a flexible and pleasing way. 119–129.

TIMECLOCK, 2006. Timeclock files - http://www.realidadevirtual.org.br/timeclock, December.