

Animation Model to Support Exploratory Analysis of Dynamic Environments

Jorge Campos, Max J. Egenhofer, and Kathleen Hornsby
{jorge,max,khornsby}@spatial.maine.edu
National Center for Geographic Information and Analysis and
Department of Spatial Information Science and Engineering
University of Maine, Orono, ME 04469-5711, USA

Keywords: data model, animation, temporal model, and virtual reality.

Abstract

We present a conceptualization of an animation data model for exploration of multi-dimensional environments. Specifically, we present the dynamic and temporal parts of a framework for exploratory analysis of animations in VR settings. The proposed model is tailored for both modelers and non-expert users to manipulate and analyze the output of a simulation application through animations. The operations of such a model facilitate the manipulation of animations, allowing users to create their own view of a dynamic environment.

1. INTRODUCTION

The use of visualization as a tool in the simulation field has increased with the advances and the decreasing cost/performance ratio of hardware and software. Simulation applications have come a long way from merely presenting the simple digital version of an image and now incorporate complex representations of the modeled environment. Despite the importance of visualization in the entire simulation lifecycle (verification, validation, analysis, and presentation), the two latter phases are the most problematic for incorporating such visualization advances. These phases involve users with little expertise in the simulation process. This kind of user usually requires a more interactive environment and a realistic representation of the information to understand the model's functioning. In this way, Virtual Reality (VR) becomes the natural paradigm for extending and enhancing the presentation capability of simulation applications. VR is a type of human-computer interface that comes close to the way humans perceive information [9], reducing the impedance between the representation of information and people's mental conceptualizations of space and time.

This paper focuses on the conceptualization of a model for *virtual exploration of animations*. By virtual exploration of animation we mean a framework composed of abstract data types (ADTs) and a user interface that allow a non-expert user, immersed in multi-dimensional environment, to compose, manipulate, and analyze dynamic environments. The term virtual suggests that the exploration of the animation takes place in a VR environment, implying that interactions between the observer and the environment need to be consistently represented in the model. The term exploration suggests that the interface and methods used to control the animation moves beyond the mere reproduction of the modeled objects' behaviors. In order to accomplish such goals we propose a representation of objects' behaviors and a small set of operations that give the observer the desired control over the animation. The representation of objects' behaviors refers to the dynamic model. The manipulation of such behaviors is accomplished through a set of operations that changes the temporal aspects of an object's action. These operations are part of the temporal model.

The remainder of this paper is structured as follows: Section 2 discusses the framework used by current three-dimensional animation models. Section 3 describes the structure of the dynamic model. Section 4 presents the structure of the temporal model and the set of operations used to manipulate animations. Section 5 introduces a graphical user interface for the model. Section 6 draws conclusions and presents ideas for future work.

2. ANIMATION FRAMEWORK

The integration of visualization tools and simulation applications has existed for decades [2]. Such tools can be roughly classified in two main groups: applications that use visualization as an interface during the entire simulation process and applications that use visualization as a post-processor of the simulation output for analysis and presentation purposes. The first group of applications runs in parallel with the simulation and is typically used by modelers to visually interact with the simulator at every stage of the simulation lifecycle. The second group of applications runs after the completion of the simulation and is used by the modeler to analyze the result of the simulation and to communicate such results to a broad audience. A model of virtual exploration of animations is part of the second group of applications. The computational demand of rendering VR environments prohibits the sharing of CPU resources. Moreover, the exploratory nature of such a model requires that the entire behavior of an object be known ahead of time. In a model that supports virtual exploration of animations, users need to have a complete control over the animation to take full advantage of this rich representation. It is possible only if the behavior of the object is known in advance. Animations that run in parallel with simulation applications cannot be manipulated, are constrained by the pace of the simulation, and compete with the simulator for CPU usage [8].

The scientific visualization community has proposed many different data models [7,13] to represent time variation of information through animations. The majority of proposed specifications partitions animation into three main logical parts [11], which clearly identify the animated objects (*who*), the actions these objects undergo (*what*), and the times during which the objects undergo the actions (*when*).

The conceptualization of an animation data model based on the who-what-when framework has led to implementations with very low levels of abstraction and without sufficient information and methods to combine and manipulate animations. The focus of these models has been the graphical presentation of objects and an efficient running of animations. However, the animation is difficult to maintain as the number of objects or the complexity of the behavior increases. Likewise, the animation is difficult to manipulate without a reasonable representation.

Another drawback in adopting current three-dimensional animation models to explore virtual environments is that these models do not explicitly represent the objects' semantics. These semantics are almost always assumed and derived from the context of the application. Objects that represent non-existent facilities or conceptual objects, however, can not interact with the observer and other objects in the environment in the same way as objects that represent existent elements or those objects that have physical realization in the world [15]. Thus, we extend the who-what-when framework with a model to capture the evolution of an object's semantics over time [3]. The associated semantics of an object directs its behavior in a virtual environment and defines the object response in the presence of the observer or other objects in the environment.

In this paper we present the conceptualization of the dynamic and temporal models of a framework for exploratory analysis of dynamic environments. We are interested in a reasonable representation of objects' behaviors and a set of operations that allows users to manipulate such behaviors.

3. DYNAMIC MODEL

A dynamic model (Figure 1) represents increasing abstractions of the animated objects' behaviors, which are constructed from the lowest to the highest level of abstraction in the model. *Normalized Act*, *Spanned Act*, and *Course of Actions* represent different stages of the behavior of a single animated phenomenon. *Animation* aggregates pairs of *Course of Actions* and *VR objects*. *VR objects* are abstractions that represent the geometry and appearance of the animated phenomena.

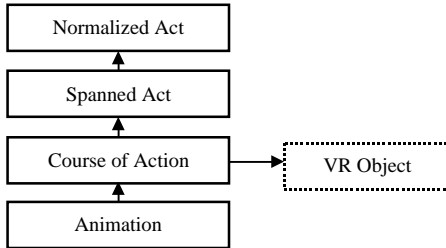


Figure 1. Modules of the dynamic model.

The process of modeling a dynamic environment is accomplished by breaking down the behavior of each object into small pieces of information. *Normalized Act* is an Abstract Data Type (ADT) that represents the building blocks of an object's behavior. Each *Normalized Act* stores a list of key values of every attribute being interpolated and an interpolation function to compute the intermediary attribute's values. Since the observer cannot manipulate this kind of information, we will abstract away from implementation's details of such entities.

Normalized Acts are combined to form an act of the object. An act is the smallest unit of the entire object's behavior that can be manipulated by a user. Depending on the complexity, an act can be modeled by a different number of combinations of *Normalized Acts*. Consider, for instance, the simple act of a ball moving between two spatial locations that continuously changes its size and color while performing its movement. One can imagine a *Normalized Act* representing the entire act of the ball, different *Normalized Acts* for each changing attribute, or any combinations of *Normalized Acts*. The best configuration is an implementation issue and depends on the types of *Normalized Acts* supported by the application.

A *Spanned Act* is an abstraction that represents an object's act. Thus, a *Spanned Act* is a collection of *Normalized Acts* that do not interpolate the same attribute. In the case of the ball, for instance, if one *Normalized Act* is used for each changing attribute (color, position, and size), then multiple *Normalized Acts* can be combined in a single *Spanned Act*. One salient characteristic of *Spanned Acts* is that all attributes of different *Normalized Acts* have an active state during the entire act. If the size or the color of the ball changes only during a fraction of the movement of the ball, for instance, they cannot be grouped in a single *Spanned Act* and must be modeled as separate acts.

Course of Actions is an ADT that represents the entire behavior of an object. If the phenomenon has a complex behavior, including a period of inactivity, or needs different interpolation functions for different periods of object behavior, a simple *Spanned Act* can no longer be used. In order to model complex behaviors, therefore, we need to combine *Spanned Acts* into a *Course of Actions*. Consider, for instance, a more complex movement of a ball (Figure 2). In this example, the color and size of the ball remain the same during its movement. A *Spanned Act* with an interpolation function is able to model the entire movement of the ball. Such a function, although mathematically sound, cannot be readily accomplished. Another approach is to divide the entire movement of the ball into acts, which have a known interpolation function. Those acts are combined forming the *Course of Actions* of the ball.

The final stage required for modeling the dynamic environment is the definition of the entity *Animation*. This entity is the highest level of abstraction in the model. *Animations* is a first class entity, which can be combined and manipulated to form a new *Animation*. *Animations* are built

up constructively in the same fashion as *Courses of Actions* (i.e., *Animations* are formed by a combination of pairs of *Course of Actions* and *VR Objects* or by a combination of *Animations*, creating an even more complex animation). Thus, an *Animation* can represent the behavior of a single object, a group of objects, or all the objects in the environment. Consider, for example, an application running an animation of vehicles in an urban environment (e.g., cars, buses, and trucks). We can have one *Animation* for every vehicle, one *Animation* with all vehicles in the environment, or different *Animations* for each class of vehicle. The first option is possibly unmanageable for humans, considering the large number of objects in the environment. The second option is the approach used by existing applications and has the disadvantage of limiting users to exploring the animation as a whole. The final option gives rise to a more reasonable number of entities (*Animations*) to be manipulated by users on the fly.

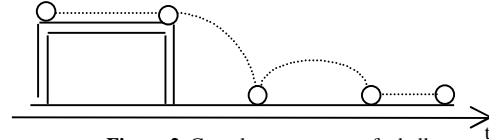


Figure 2. Complex movement of a ball.

4. TEMPORAL MODEL

For virtual explorations of animations the exploration of the static content of the environment is accomplished through the usual mechanisms to navigate in VR environments extended with well-known mechanisms to compose different views of the environment (e.g., operations that hide/show individual or group of objects). The exploration of the dynamic content, however, still depends on a small set of operations that resembles the operations of a VCR control (e.g., operations that define the speed/temporal order of the animation). These operations change the temporal configuration of modeled behaviors in a way that is more suitable for a user to explore the dynamic environment. Since all manipulation of the dynamic environment is accomplished through a set of operations performed over the temporal space, entities of the temporal model are critical in a model for exploratory analysis of animations. In such a model the temporal evolution of the information is perceived through a mapping among different conceptualization of time, varying from the *simulated time* domain to the *user time* domain.

Simulated time represents the time when the fact is true in the modeled reality. This definition is similar to the notion of *valid time* coined by the database community [10]. *Simulated time* is time generated by a simulation application. This kind of information cannot be modified by operations of the animation model, but only observed by a user at a special and ephemeral point along the *user time* domain (i.e., the user present). *User time* is the time in which the user senses the facts. Thus, *user time* is time as experienced by a user. This experience of time by the user refers only to the present and "going forward" at a fixed rate. Since *simulated times* are fixed and *user time* cannot be manipulated, a direct mapping from the *simulated time* domain to the *user time* domain constrains the user to explore the information as it "happened" or "will happen" in the modeled world. In order to allow the user the control of the flow of information coming from the *simulated time* domain, we need to represent such information in a temporal domain that can be manipulated (i.e., *animated time*).

The mapping from *simulated times* to *animated times* is a mere conversion of time units made by an application in an automatic fashion. The mapping from *animated times* to *user times*, however, is controlled by a user through a set of operations that changes the *animated time* space. The simplest mapping between these time domains aligns the user present with a certain instant in the *animated time* domain, allowing the user to sense a single snapshot of the modeled reality. Continuously mapping different *animated times* to the present of the user allows a user to sense the evolution of the modeled reality. We call this continuous mapping an animation. Other temporal operations distort the modeled reality allowing a user to create a new view of the dynamic environment. An operation that

performs a scale over the *animated time* space, for instance, slows down or speeds up the evolution of the animated phenomena. An operation that inverts the *animated time* space changes the temporal order of the animation, allowing the user to explore the environment where the evolution of objects' behavior is observed in the reverse chronological order.

In most existing animation models, the *animated time* domain is represented by a single and elementary temporal structure. Such an elementary structure constrains the user to control the animation as a whole, without any means to address the behavior of an individual or a set of individuals. In such a model, for example, it is impossible to observe the behavior of an object being performed in a chronological order while other objects are observed performing their activity in a reverse chronological order. A user using only VCR-like style of operations cannot generate this interesting and possibly insightful view of the dynamic environment.

In order to provide a user with a rich set of operations to manipulate the animation and the ability to control the behavior of a small group of objects or even of a single object in the environment, we need a more sophisticated conceptualization of the temporal domain. In our model, the *animated time* domain is partitioned in a hierarchical representation of time (Figure 3). This representation forms the basis of the temporal model. The elements of the temporal model are strongly related to entities of the dynamic model; that is, they model the temporal aspects of *Normalized Act*, *Spanned Act*, *Course of Actions*, and *Animation*. The tight connection between entities of the dynamic and temporal models enforces the user's understandings and facilitates the implementation of the model.

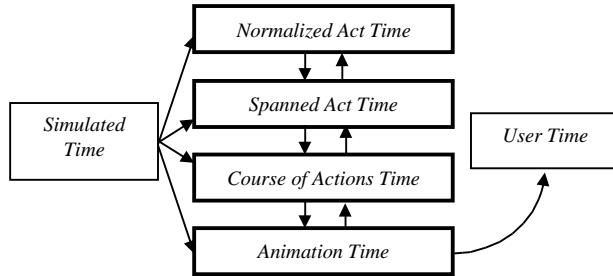


Figure 3. Mappings between different temporal domains.

Arrows coming from the *simulated time* domain represent operations used to convert information between *simulated* and *animated time* spaces. *Simulated times* information is spread among all abstractions in the model. Entities of the *animated time* domain are related through temporal operations. The down arrows in the figure indicate operations used to combine and map entities at a low-level of abstraction. The up arrows indicate that operations performed at high-level of abstractions are consistently propagated to entities at lower levels of abstraction, enforcing the hierarchical nature of the model. Finally, a set of operations performs the mapping between the *animated* and the *user time* domains, allowing the user to observe the modeled animation.

4.1 Normalized and Spanned Act Time

Normalized Act Time (NAT) refers to the temporal aspects of *Normalized Acts*. The temporal structure of NAT is a continuous, linear, and closed interval defined over a normalized temporal space. Each NAT stores a list of *normalized times* for each attribute being interpolated. *Normalized times* is an ADT that represents real values between 0 and 1, inclusive. For each *normalized time* there exists an associated attribute value. The pairs (*normalized time*, *attribute value*) are known as keyframes of the animation.

Spanned Act Time (SAT) is a temporal structure resulting from the mapping of NATs over the positive space. The main purpose of such a mapping is to define the duration, the pace, and the temporal order of *Spanned Acts*. NATs are mapped onto the *Spanned Act* space through a composition of four operations: *duration* • *pace* • *flow* • *order* (Figure 4). Such a composition of operations preserves the temporal

structure of NATs used as an argument (i.e., the resulting SAT's temporal structure is still a continuous, linear, and closed interval, but defined over the positive space).

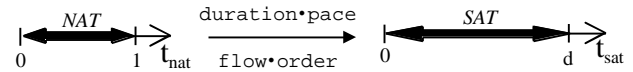


Figure 4. Mapping between *Normalized Act Time* and *Spanned Acts Time*.

The collection of *Normalized Acts* that are combined to form a *Spanned Act* of an object is mapped using the same composition of operations (i.e., all NATs have the same duration, pace, order, and flow). Thus, an act of an object can be redefined as a collection of non-related *Normalized Acts* that share the same temporal space. The composition of operations, depending on the value of the argument of each operation, produces different types of mappings (Figure 5).

- The operation *duration* performs a scale of the normalized temporal space. Such a scale has the effect of defining the duration of the act. This operation has a scale factor as argument.
- The operation *pace* defines the speed of the mapping of *normalized times* onto the positive space. Changing the speed of such a mapping affects the way the user perceives the evolution of attributes' values. The operation *pace* controls this speed by performing four kinds of mappings: *constant*, *accelerated*, *decelerated*, and *accelerated/decelerated*. An argument with a value *constant* promotes a linear mapping between the two domains (Figure 5.a). Such a mapping imposes the constraint that animated attributes change their values at a constant speed along the entire act. An argument with a value *accelerated*, *decelerated*, or *accelerated/decelerated* yields a non-linear mapping between the two domains. A non-linear mapping imposes the constraint that animated attributes change their values at different speeds during the act. The argument *accelerated*, for instance, changes the value of the attributes at a low speed at the beginning of the act and a high speed at the end (Figure 5.b).
- The operation *flow* defines how attributes' values between keyframes are computed. This operation has three kinds of arguments: *stepwise*, *continuous*, and *none*. The argument *stepwise* indicates that the interpolation function does not generate intermediate attribute values between two *normalized times*. Such a mapping imposes the constraint that only the discrete movement of the object will be presented, highlighting the keyframes of the animation (Figure 5.c). The argument *continuous* indicates that the attribute values are continuously interpolated with the evolution of time. The argument *none* indicates that the object will not perform its associated activity.
- The operation *order* defines the temporal order of the presentation of the act. This operation can be performed with the *reverse* or *normal* argument. The *reverse* argument inverts the mapping of *normalized times* causing the act to be perceived in the reverse chronological order (Figure 5.d). The argument *normal* preserves the natural evolution of *normalized times* (i.e., from 0 to 1).

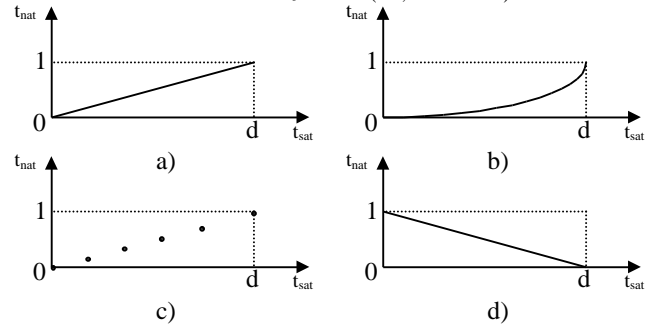


Figure 5. Mappings between *Normalized Act* and *Spanned Acts Times*.

A user operating at this level of abstraction has finer control of the animation's content. A user can manipulate temporal aspects of small pieces of the entire object's behavior without the necessity of dealing with type or values of attributes.

4.3 Course of Actions Time

Course of Actions Time (CAT) is the temporal model of *Course of Actions*. *CAT* is a collection of *SAT* intervals positioned along the *Course of Actions* timeline. *SAT* intervals become periods in the *Course of Action* temporal space. A period is an anchored version of a temporal interval [6]. Thus, the temporal structure of *CATs* is a multiple period structure defined over the real space. *SATs* temporal intervals are positioned along the *Course of Actions* timeline by specifying the start point of each interval in the target temporal space (Figure 6).

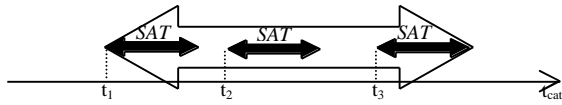


Figure 6. Mappings of SATs defining the CAT temporal structure.

The process of combining and positioning temporal intervals along a temporal axis by specifying the start point of each interval is straightforward. This process is usually performed by an application during the production of the animation (i.e., after the completion of the simulation). In an environment tailored for users to manipulate animations, however, this simple mechanism to position temporal entities is time consuming, susceptible to errors, and hard to maintain. Exploring temporal relations between time periods [12,14] is a more natural way to position intervals. Consider, for instance, two temporal intervals *a* and *b*. There exist thirteen possible relations that may hold between those intervals [1] (Figure 7). Although all temporal relations have a significant role in the animation model, only some of them can be used to position a new interval unambiguously. An operation used to position an interval needs to constrain at least one of the end point of the new interval. In this way, we divide the set of temporal relations in two sub-groups: tight and loose. The tight group of temporal relations constrains at least one of the end points of the interval. This group is composed by the relations *meets*, *starts*, *finishes*, *equals* and their converses. The temporal relation *equals* represents the strongest relation in the group since this operation constrains the start and end points of the new interval, thus constraining the duration of the new interval. The loose group of temporal relations limits the range of possible values of the interval's end points, but does not unambiguously position the new interval.

Relation		Converse
before(a,b)		after(b,a)
meets(a,b)		metBy(b,a)
overlaps(a,b)		overlapped(b,a)
contains(a,b)		during(b,a)
starts(a,b)		startedBy(b,a)
finishes(a,b)		finishedBy(b,a)
equals(a,b)		equals(b,a)

Figure 7. Temporal relations between intervals.

A set of operations based on the tight group of temporal relations (i.e., *makeMeets*, *makeStarts*, *makeFinishes*, and *makeEquals*) is used to position a new temporal interval along the *Course of Actions* timeline. These operations have two *SATs* as arguments. The first argument is an interval used as reference, and the second argument is the interval to be repositioned. These operations act over individual acts of the object's behavior. For example, for cases where an object has a behavior composed of two acts with a long temporal gap between them (i.e., a long period of inactivity), currently available animation techniques require a user to wait a considerable amount of time to observe the entire behavior of the object. But, by performing a *makeMeets* operation between these two acts, a user

can now produce an animation where one act of the object follows the other, eliminating the temporal gap.

Existing animation models have explored the use of temporal relations between intervals [4,5]. These models, however, are primarily concerned with the production of the animation. The set of operations of such models acts over low-level abstractions of the objects' behavior. Thus, they are not suitable for manipulation by users on the fly.

The process of combining and positioning *SATs* intervals along the *Course of Actions* timeline, however, is insufficient to represent complex temporal structures. The linear temporal structures of *SATs*, for instance, do not support the representation of objects' behavior with a cyclic pattern. A cyclic behavior implies that the object repeats its entire behavior a certain number of times or indefinitely. The operation *cycles* models cyclic behaviors. This operation acts over an abstraction that encompasses all *SATs* intervals of the *Course of Actions* of the object (i.e., the cycles has a *CAT* as argument). The *CAT* argument represents the reference period (*CAT_R*). For the sake of simplicity, we represent the multiple periodic structure of *CATs* by a single period in the *Course of Actions* temporal space.

The second argument of the operation *cycles* defines the type of cyclic pattern developed by the object. The type of cyclic behavior can be *infinite*, *finite*, *semi-infinite positive* or *semi-infinite negative*. The argument *infinite* models endless cyclic behaviors (Figure 8.a). This type of behavior implies that the object is always observed performing some activity. The argument *finite* models a behavior that repeats a certain number of times after the reference interval (Figure 8.b). The number of cycles needs to be provided in this case. The argument *semi-infinite positive* implies that the object starts its behavior with the reference period and repeats such behavior indefinitely after that (Figure 8.c). The *semi-infinite negative* type of behavior implies that the object finishes its behavior with the reference period but is observed performing its behavior at every instant before that (Figure 8.d). In current animation models, cyclic behaviors are either finite or positive semi-infinite. Negative semi-infinite and infinite cycles are not supported.

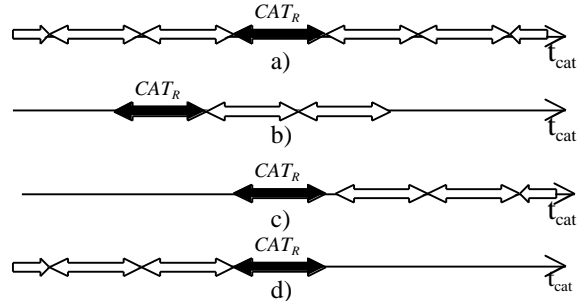


Figure 8. Types of cyclic behaviors.

Allowing an observer to modify the temporal structure of *Course of Actions* is useful in exploring animations. When an object is modeled with a cyclic behavior but the observer is not interested in the repetition of such behavior, the observer can deactivate the repetition of the object's behavior and examine only the reference period. In addition, when an object does not have a cyclic behavior, but the observer wants to simulate this kind of behavior (i.e., the observer want to examine the object performing its activity a certain number of times, while other objects are performing their modeled behavior), the user can select a specific type of cyclic behavior and observe the desired configuration of the animation.

4.4 Animations

Courses of Actions are combined to form *Animations*. The combination of *Course of Actions* to form an *Animation* is accomplished in a similar fashion as the combination of *Spanned Acts* to form a *Course of Actions*. In the temporal domain this process is accomplished by positioning *CATs* temporal intervals along the *Animation* timeline. *CATs* are mapped onto the animation space forming *Animations Times (AT)* temporal structures. *ATs*

preserve the temporal structure of their composing *CATs* (i.e., a linear or cyclic, multiple periodic structure defined over the real space).

Different from existing data models, an *Animation* can represent the behavior of a single object, a group of object, or all objects in the environment. In our model *Animations* are first-class entities that can also be combined, forming an even more complex *Animations* (Figure 9). At this level of abstraction users also see *Animations* as a single interval. Thus, the operations *makeMeets*, *makeStarts*, *makeFinishes*, and *makeEquals* can still be used to compose new *Animations* by imposing a temporal relation between their arguments.

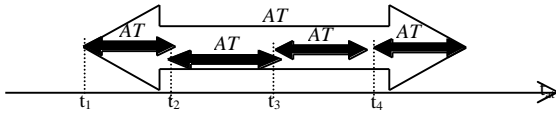


Figure 9. Combinations of Animations forming a complex animation.

At the Animation level of abstraction, however, users are not restricted to manipulate *Animations* by changing the temporal configurations of temporal intervals. A group of operations that resembles operations over mathematical sets (i.e., intersection, difference, union, and concatenation) can also be used to compose a new *Animation*.

- The intersection of two animations generates an animation defined over an interval obtained from the intersection of the intervals used as the arguments of the operation. The result is an animation defined over an interval when only the simultaneous occurrences of both animations will be present. With this operation a user can verify mutual interference between the movement of different objects.
- The difference of two animations generates an animation defined over an interval when the second animation is not defined (i.e., an interval when only the first animation occurs). This operation permits the user to isolate the behavior of the first object during the period when the second object does not occur.
- The union of two animations is a simple combination of two animations while preserving the temporal configuration of individual arguments. This operation is useful for combining animations to form complex animations.
- The concatenation of animations produces a new animation when the second animation follows the first (i.e., the second animation starts immediately after the first). The result is that temporal gaps are eliminated. This operation is equivalent to performing the *makeMeet* operation.

The final stage in the process of manipulation of animations is defining the presentation configuration. A group of presentation operations (*start*, *play*, and *stop*) map *animated time* structures to the *user time* domain. Operations of such a group acts over the set of all animations in the model. The operation *start* aligns a point in the animation timeline with the user present. This operation has the effect of defining the point of insertion of the user in the animated temporal domain (i.e., this operation defines the start point of the animation). The operation *play* performs a continuous mapping from the *animated time* domain to the *user time* domain, allowing the user to sense the evolution of the modeled dynamic environment (i.e., allowing the user to explore the animation). The operation *stop* suspends the flow of information coming from the animated time domain, which allow a user to observe a static version of the environment.

5. AN EXAMPLE

In order to highlight the level of control that a user has over dynamic environments, we present an example based on the movement of a small number of objects. This example is kept intentionally simple to emphasize major features of the model and does not pretend to cover all possible configurations of an animation that can be accomplished by a user. The example discussed in this paper can be seen as a movie at the following URL: www.spatial.maine.edu/~jorge/scs03/animations.html. This example

shows the movement of two balls and a wall. Figure 10 depicts key positions and paths of these objects. The wall is initially at rest above the ground, then starts its movement falling down and reaching the ground somewhere in the path of the balls. The balls have the same pattern of movement (i.e., the balls run over the table, bounce once on the floor, roll in a straight line and then stop).

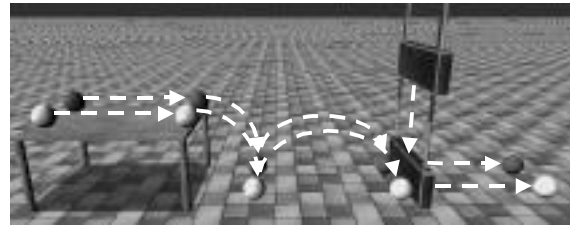


Figure 10. Positions and paths of the movement of the wall and the balls.

The lack of temporal information about the objects' behaviors, however, does not allow any mental picture of the animated environment. To have a rough idea of how these objects develop their behavior over time, we need to know, at least, the duration and the start point of each movement. Additional information, such as the acceleration of the movement, contributes to the cognitive process of analyzing the result of simulations. However, exploring the dynamic environment and witnessing objects performing their associated activity is still the best way to understand multi-dimensional environments.

The presentation of animations represents a cognitive gain in interpreting simulations' result, but the ability to produce new views of the pre-orchestrated objects' movement leads to an even better understanding of the modeled world. Thus, we implement a graphical user interface that gives a user information about the temporal configuration of animations and the ability to manipulate and control the presentation of such animations. This interface allows a user to manipulate the animation at distinct levels of abstraction. At each level of abstraction, the user dealing with different granularities of the animation and a set of operations that can be performed over these representations can produce new views of the environment, gaining insights and discovering relationships among dynamic processes.

In this paper we discuss only manipulations that can be performed at a high-level of abstractions (i.e., *Course of Actions* and *Animations*). Figure 11 shows a snapshot of the interface depicting the temporal configuration of the balls and the wall, and the set of operation that can be used to change this configuration. Even in a simple example with a small number of objects, the number of different configurations is large. In our example, for instance, the movement of the balls and the wall objects can be modeled as three *Animations* (i.e., an *Animation* representing each moving object) or as two *Animations* (i.e., one *Animation* representing a group with the balls and one *Animation* representing the wall). In this paper we use the latter configuration. The configuration of the modeled animation depends basically on the strategy employed by the application. Here, the maxim "garbage-in-garbage-out" is still valid. The configuration of the animation does not change the content of its presentation, but a poor configuration of the animation limits users' creativity and flexibility for manipulating the dynamic environment.

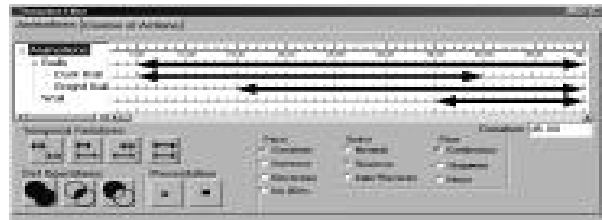


Figure 11. Operations and the temporal configurations of *Animations*.

The operations of the model are grouped and depicted in the interface accordingly their functionality. Set operation allows the user to combine two animations based on operations that resemble operations over mathematical sets. Temporal relations allow the user to change the temporal configuration of the animation by imposing a temporal relation between any two temporal intervals as arguments (i.e., *Course of Actions x Course of Action*, *Course of Actions x Animation*, or *Animation x Animation*). Consider, for example, that a user wants to explore an environment in which the wall starts to move as the first ball start its movement. Thus, the user can perform a `makestarts` operation using the intervals representing the animation of the balls and the animation of the wall as arguments. Figure 12 shows the graphical representation that corresponds to the modified behavior of the wall after such a manipulation.

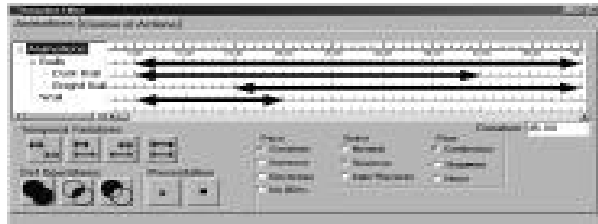


Figure 12. The modified behavior of the wall.

The analysis of the graphical representation of the temporal configuration of the modified movement of the wall and the balls do not allow a user to reason about interactions among these objects. The users do not have any spatial information about the objects' activity. In this way, users cannot infer if the object interacts with others simply by analyzing this kind of information. Although, the animation model does have all necessary information that allow an application to anticipate any kind of interaction, we decide to leave to the user the task of exploring the environment and verifying all interactions among objects.

By exploring the dynamic environment with the balls and the modified version of the wall the user can verify that the wall blocks the path of both balls; that is, due to a collision with the wall, the balls stop their modeled behavior sooner than expected. There is no guarantee, however, that a simulation in which the wall starts to move sooner will generate the same result accomplished by the manipulation of the animation. The model for virtual exploration of animation does not pretend to substitute any phase of the simulation process but only to incorporate information that allow users to explore the potential of VR environments as an analytical tool. Such a model is a mere presentation mechanism that offers a user the means to manipulate the output of a simulation application. The insights and understandings achieved by users and modelers exploring the modified dynamic environment must be used to re-feed the simulator, creating a virtuous cycle.

The operations duration, pace, order and flow are also represented in the interface. These operations act over individual *Course of Actions*, over a combination of *Course of Actions*, or over the entire animation. A user can create an animation in which the movement of the balls are accelerated in the beginning and decelerated at the end, the wall moves at a constant speed, the dark ball performs its movement in a reverse chronological order, the bright ball's movement has a small duration, and so on. The group of presentation operations simply starts or suspends the presentation of the manipulated animation.

6. CONCLUSIONS AND FUTURE WORK

We have presented an animation data model and a graphical user interface to support exploratory analysis of dynamic environments. For the manipulation of the animation temporal intervals are used as the basic level of abstraction representing objects' behaviors. In this way, the representation of these behaviors is reduced to the representation of a time interval when the objects are performing their associated activity. Such an abstraction has the advantage of hiding information that cannot be

manipulated by a user on-the-fly (e.g., type and values of object attributes and interpolations functions). Moreover, the set of operations used to combine and manipulate animations having temporal intervals as arguments are effective abstractions even for naive users and have the convenience of being easily represented with a simple graphical user interface, letting users to identify frequency, durations and synchronization between objects' activities.

At this point, the animation model deals only with the representation of time-dependent behaviors. We also consider important to extend the framework with a model to represent causal relations. In a model tailored for exploration of animations, it is important to isolate objects affected by a specific change in the configuration of the environment or objects that lie in a cause-and-effect chain.

ACKNOWLEDGMENTS

This work was partially supported by the National Institute of Environmental Health Sciences, NIH, under grant number 1 R 01 ES09816-01; the National Imagery and Mapping Agency under grant numbers NMA202-97-1-1023 and NMA201-00-1-2009, and NMA201-01-1-2003; by the National Science Foundation under grant numbers IIS-9613646, IIS-9970123, and EPS-9983432; and by the Brazilian Research Council, CNPq, under grant number 200020/00-5.

REFERENCES

- [1] J.F. Allen. "Maintaining Knowledge about Temporal Intervals." *Communications of the ACM*. **26**(11):822-843,1983.
- [2] P.C. Bell and R.M. O'Keefe. Visual Interactive Simulation - History, Recent Developments, and Major Issues. *SIMULATION*. **49**(3):109-116,1987.
- [3] J. Campos, K. Hornsby, and M. Egenhofer. "A Temporal Model of Virtual Reality Objects and Their Semantics." *(DMS'02)-Workshop on Visual Computing*. San Francisco. pp. 581-588, 2002.
- [4] J. Dollner and K. Hinrichs. Object-Oriented 3D Modeling, Animation and Interaction. *Journal of Visualization and Computer Animation*. **8**(1):33-64,1997.
- [5] E. Fiume, D. Tschritzis, and L. Dami. A Temporal Scripting Language for Object-Oriented Animation. *Eurographics 1987*. Holland. pp. 283-294, 1987.
- [6] A.U. Frank. *Diferent Types of "Times" in GIS*, in *Spatial and Temporal Reasoning in Geographic Information Systems*. Oxford University Press, New York. pp. 40-62. 1998.
- [7] M. Green and S. Halliday. A Geometric Modeling and Animation System for Virtual Reality. *Communications of the ACM*. **39**(5):46-53, 1996.
- [8] J. Henriksen. General-Purpose Concurrent and Post-Processed Animation with PROOF. *1999 Winter Simulation Conference*. pp. 176-181, 1999.
- [9] R. Jacobson, *Virtual Worlds, Inside and Out*, in *Cognitive and Linguistic Aspects of Geographic Space*. Kluwer Academic, Netherlands. pp. 507-514. 1991.
- [10] C.S. Jensen, J. Clifford, S.K. Gadia, A. Segev, and R.T. Snodgrass. "A Glossary of Temporal Database Concepts." *Workshop on Infrastructure for Temporal Databases*. Arlington, Texas. pp. 25-29, 1992.
- [11] G.S. Lee. A General Specification for Scene Animation. *International Symposium on Computer Graphics, Image Processing, and Vision - Sibgrapi'98*. Rio de Janeiro, Brazil. pp. 1-8, 1998.
- [12] T.D.C. Little and A. Ghafoor. Interval-Based Conceptual Models for Time-Dependent Multimedia Data. *IEEE Transactions on Knowledge and Data Engineering (Special Issue on Multimedia Information System)*. **5**(4):551-563,1993.
- [13] P. Strauss and R. Carey. An Object-Oriented Graphics Toolkit. *Computer Graphics*. **26**(2):341-349,1992.
- [14] R. Weiss, A. Duda, and D. Gifford. Composition and Search with a Video Algebra. *IEEE Multimedia*. **2**(1):12-25,1995.
- [15] S. Wenzel and U. Jensen. The Integration of 3-D Visualization into the Simulation-based Planning Process of Logistics Systems. *SIMULATION*. **77**(3-4):114-127,2001.