STEDEL: A LANGUAGE FOR INTERACTIVE SPATIO-TEMPORAL COMPOSITIONS

Iosif Lazaridis
University of California, Irvine
iosif@ics.uci.edu

Michalis Vazirgiannis
Athens University of Economics & Business
mvazirg@aueb.gr

Timos Sellis
National Technical University of Athens
timos@cs.ntua.gr

ABSTRACT

The two fundamental challenges in Virtual Reality over the WWW are (i) to facilitate the process of creating convincing, non-trivial, highly interactive and maintainable content and (ii) to efficiently deliver this content to the client with QoS guarantees. This paper focuses on the first challenge: we have created a data model for spatio-temporal compositions that allows us to define relative placement of objects in a virtual world by means of their geometric characteristics (Joints). Thus, we model spatial relationships of objects at a higher level of abstraction than simple co-ordinate system transformations. We also define the behavior of objects in a declarative way, by means of Event-Condition-Action rules. Events and Actions can be synthesized by a set of operators for the creation of really complex behavior. Our model has been realized in prototype form in Spatio-Temporal Descriptive Language (STEDEL) that demonstrates the validity and power of our approach in terms of re-usability and authorship effort.

1. INTRODUCTION

Virtual Reality on the Web is linked to Virtual Reality Modeling Language [1] which is the de facto standard for delivery of spatio-temporal content over the Internet. Lately, an alternative to VRML has become available in Sun's Java 3D API [2]. Both of these methods provide a rich set of primitives that allow (in principle) the creation of complex and highly interactive virtual scenes. However, significant programming experience and authorship effort is required to create even the most simple virtual world. This is due to the fact that they are not fully (in the case of VRML) or at all (in that of Java 3D) declarative. Thus users need to know *how* they can achieve a certain outcome rather than focusing on *what* they want to achieve.

To illustrate this problem with a simple example, consider the problem of a ball on a table. To achieve the effect of the ball being precisely *on* the table, some calculations have to be made to define the position of the ball and the table relative to the global co-ordinate system; moreover if later we decide to change the size of either of the two, then these calculations will have to be repeated to ensure that the topological constraint "ball on table" will be maintained. Additionally, if we wanted to have an effect of "ball rolling on the table", then we would have to set up (in VRML) complex interactions between TimeSensor, TranslationInterpolator and RotationInterpolator objects for time keeping, forward and rolling motion respectively.

The motivation for our work is to make this entire process simpler by making it entirely declarative. The user need only specify the simple elements of the virtual scene (e.g., ball, table), how these are composed spatially in a meaningful way ("Ball is *on* table") and how they behave in simple understandable terms ("Move ball north at a speed of 10m/s, rotating it as it goes along"). The advantages are apparent: it's much simpler to write virtual scenes in this way and it's much easier to modify them at a later time; you can change the size of a ball but the semantic constraint "ball *on* table" does not need to be modified.

The present paper is one effort towards making the declarative description proposed in the previous paragraph a reality. In Section 2 we define a model for composing objects in spatial hierarchies using *Joints*. In Section 3 we define an ECA-based model for behavior modeling. In Section 4 we present the prototype STEDEL system. In Section 5 we present some relevant work and conclude with some future directions in Section 6.

2. SPATIAL COMPOSITION: JOINTS

Let O_1 and O_2 be two geometric objects and $CS(O_1)$, $CS(O_2)$ be co-ordinate systems centered on these objects. We can define the relative placement of O_1 with respect to O_2 by specifying a translation *T*, rotation *R* and scaling *S* vector or an equivalent affine transformation matrix [10] that defines the relative placement of their co-ordinate systems $CS(O_1)$ and $CS(O_2)$. This is how spatial composition is done in VRML and Java 3D.

We note that by using just the object-centered co-ordinate systems, we are completely disregarding the object's geometric properties in defining their relative placement. Naturally, using the affine transformation matrix we can place objects in every possible configuration with respect to one another. But we are ignoring the natural way in which objects are usually linked together: by means of their surfaces, i.e., of their geometric attributes.

Let *G* be a geometric type. Let $D^G \subseteq \Re_+^n$ be the set of dimensional attributes for this geometric type. A *simple object* can thus be defined as a pair (*G*, *v*) where *G* is a geometric type and $v \in D^G$ is a vector of its dimensional attributes. Thus the geometric type SPHERE has $D^{SPHERE} = [0,\infty)$ which is the set of values that its radius attribute can possibly take. Subsequently the pair (SPHERE, 5) defines a sphere of radius 5.

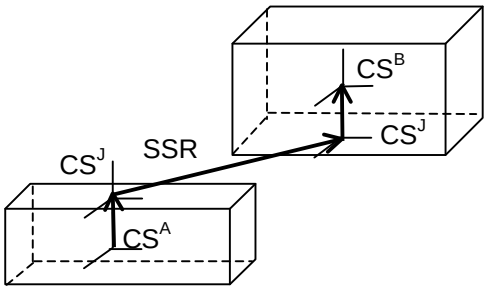

Figure 1. Spatial Relationship using Joints. We move from CS^A to CS^B using the objects' surfaces via the joint co-ordinate systems CS^{JA} and CS^{JB}.

A 3D box with size 3×4×5 could similarly be (BOX, (3, 4, 5)).

For each simple object $O = (G, v)$ we can define an object-centered co-ordinate system CS^O such that if a point's co-ordinates in CS^O are (x, y, z) then the function $f_O(x, y, z)=1$ iff (x, y, z) is part of the object and 0 otherwise.

A *simple spatial relationship* SSR between two objects is defined as a tuple (T, R, S) where $T = (tx, ty, tz)$ is a translation vector, $R = (rx, ry, rz, \alpha)$ is a rotation vector (normal plus rotation angle) and $S = (sx, sy, sz)$ is a scaling vector.

For each geometric type G we define a set of keywords J^G and a set of parameters P^j associated with each keyword $j \in J^G$ such that the pair (*keyword*, *keyword parameters*) defines a unique co-ordinate system (Joint co-ordinate system) associated with a surface feature of each object of geometric type *G*. As an example, the tuple (on_surface, (45^o, 45^o)) defines a unique co-ordinate system CS^J at the given latitude/longitude of a given object of type *G*.

The tuple $J = (direction, keyword, keyword\ parameters)$ where $direction \in \{from, to\}$ defines (for a given object *O*) a unique SSR *from* CS^O to CS^J or conversely *to* CS^J from CS^O. This is the basic definition of a *Joint J*. It is defined for a given geometric type *G* and is independent specific object on which it is instantiated. Therefore, the object's instantiation may be altered, but *J* will still correspond to a co-ordinate transformation to the object's surface feature specified by the (*keyword*, *keyword parameters*) pair.

By using the Joint concept we can now define spatial relationships in the following manner. To place object *B* in a relative position to object *A* we can express their spatial relationship as (J_A, SSR, J_B). The sequence of co-ordinate transformations are presented in Figure 1. We can now state relationships such as *A* is 10m above *B* as "*B* (from top surface, (0, 10, 0), to bottom surface) *A*".

We have defined Joints for the commonly used geometric primitives of VRML, i.e., for BOX, SPHERE, CONE and CYLINDER. These can be seen in Figure 2. Similarly we can also define joints for other geometric primitives that are likely to be frequently used and re-used in an authoring situation. As an example, in some application a parameterized HAND type may be frequently used. It is quite simple to define a *Wrist* Joint on

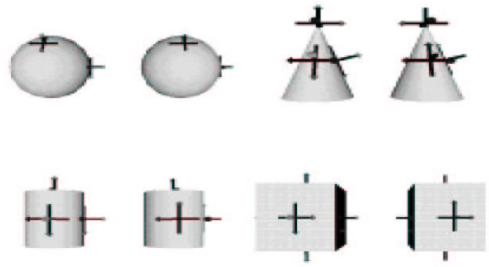

Figure 2. Joint co-ordinate systems for BOX, SPHERE, CONE and CYLINDER geometric types.

this type. Subsequently, we can use this joint to simply attack hands to arms defined separately. One could imagine a library of human parts that could be easily composed with one another to create a variety of human individuals of varying appearance.

We model the relative placement of objects in a scene by a connected directed acyclic graph (SpatialGraph). Vertices in this graph are simple objects and edges are spatial relationships between simple objects. A *complex object* is simply a SpatialGraph that is considered as an atom of the scene. A complex object consists of simple objects physically linked together to form a unique entity, e.g., human body parts form a human being. On the other hand, some of the spatial relationships in the SpatialGraph of a virtual scene are *convenience relationships* that simply indicate relative placement, e.g., a ball on a table is such a relationship; during the course of the application, the ball/table complex may be broken since the two do not inherently form a unity. The position of an object (simple or complex) in the global co-ordinate system can be determined by following the spatial relationship edges of the SpatialGraph from the root to each object. Motion of a complex object can be achieved by changing its root's placement in its parent's co-ordinate system, i.e., the edge linking it to the rest of the SpatialGraph.

To summarize, Joints allow us to declare the spatial relationship between objects in a more natural and understandable way. This is done by exploiting the geometric features of objects, e.g., their surfaces and conforms with the way that complex artificial objects are usually composed in reality, by linking them via such features. Joints are maintainable, since they capture semantic information from the objects' unchangeable geometric type. Thus, if the objects' sizes in a composition change, the specification of their spatial relationships does not need to be updated. Thus, we achieve our twin goals of reducing authorship effort and increasing virtual scene maintainability.

3. BEHAVIOR MODELING

We model behavior via a set of Event-Condition-Action rules of the form:

If event *E*, under condition *C*, do action *A*

An event *E* is an instantaneous occurrence in the context of the spatio-temporal application. It triggers action *A*, iff the condition *C* of the rule evaluates to true. simple action *A* is defined for a given object (spatial/multimedia) within the context of the application. It corresponds to a transformation of either its state

or its placement. As an example, the action might resize an object, move it in virtual space or start playing a sound clip object. A transformation $T:[0,t_f] \rightarrow S$ is a law that defines the state or placement of an object in a parametric space *S* for any instance of time in the interval [0, t_f]. This space could e.g., be $\Re^3$ for the position of an object or [0,∞) for the radius attribute of a sphere. Also note that the temporal interval is not linked to real time. The state of the transformation can be arbitrarily moved by the application to any position of this interval using a *control*. A control is defined for a transformation and allows us to move to different temporal positions within that transformation. We have defined the following generic controls:

- *Play*: if t< t_f start increasing time
- *Pause*: *t* is frozen
- *Kill*: transformation is invalidated, *t* is frozen to last value
- *Rewind*: *t*=0

In Figure 3 we see the state diagram for a transformation as different controls are activated on it:

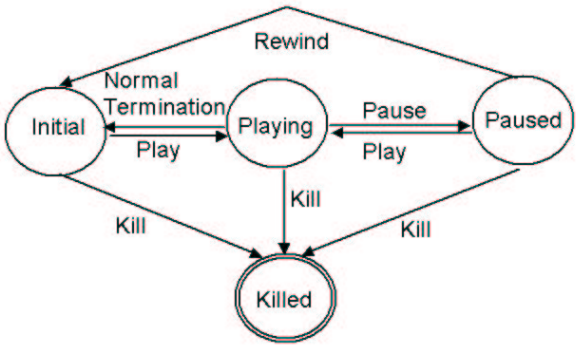

Figure 3. State diagram of transformation as different controls *play*, *kill*, *pause*, *rewind* are applied to it.

In our approach, we use *law-based transformations*. The transformation *T* is not given as a set of values as in the case of interpolation. It is rather given as parameters for a predefined functional form. The main advantage is that simple motions (e.g., linear translation) can be described with parameters understandable by humans. An application developer can now simply state that he wants object *A* to move with a velocity of 10m/s along the *x*-axis, without worrying about how this is to be achieved. A supplemental benefit is one of compression in that a transformation can be described with only a few parameters rather than a set of intermediate points over which interpolation is performed.

We have additionally proposed two mechanisms for composing simple actions into complex ones. The *ActionChain* mechanism is useful when we want a series of simple actions to be performed exactly one after another. Each action is performed after the previous one has *finished*. Thus, a single action in the chain is active at any instance of time. We can apply controls on

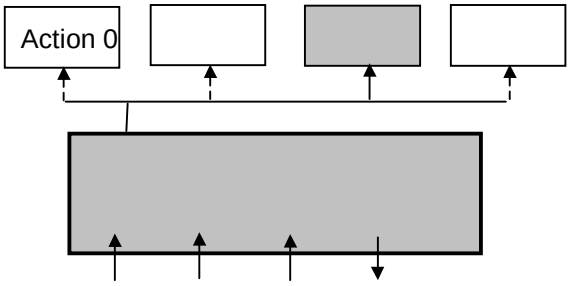

Figure 4. ActionChain. Controls on the action chain are directed to the active action (here Action 2). Each action starts *play*-ing after the previous one has *finished*.

an ActionChain just as in any other type of action. A schematic diagram for an ActionChain can be seen in Figure 4.

An ActionChain keeps track of the currently *play*-ing transformation and directs control activations on itself to that transformation. An example of an action chain is informally the following: "Move the rocket with acceleration 10m/s^2 upwards for 10sec, then move its second stage upwards with acceleration 15m/s^2 and its first stage downwards with acceleration 9.81m/s^2 for 100 sec".

The second construct we have proposed is that of a *Sequence*. A sequence is an arbitrary series of control activations on transformations in fixed temporal relationship to one another. Informally, we could give the following example: "Start playing song 3, after 10s start moving ball upwards at 10m/s, after 5 s pause the song, after 1s add a rotation to the ball at 50^o/s and after 3s increase the volume of the song by 50%, hide the ball and present in its place fragments F1-F10 and play explosion noise N1".

4. STEDEL PROTOTYPE

The ideas we have described above are some of the features incorporated in the STEDEL language prototype. Spatio-Temporal Descriptive Language is a fully declarative, event-based language that is suitable for rapid development of VR scenes with dynamic interactive content and for quick prototyping of VR scenes. It is compiled into equivalent VRML97 code, combined with a series of PROTO nodes which we have written which implement the various constructs of the language, corresponding to the primitives of our model outlined in the previous section. The language is not specifically linked to VRML in any way and it can be compiled to other 3D modeling languages. However, due to the prevalence of VRML we have designed STEDEL so as to facilitate the inclusion of VRML code within a STEDEL description. Additionally the VRML content exported by our compiler can be used in a larger project that is done in pure VRML with minimal effort.

STEDEL was designed with two goals in mind. The first and most significant goal is to make life easier for the content developer who is not necessarily either knowledgeable or interested in computer programming. We have thus abstracted away the *how*'s of creating virtual scenes and went fully declarative in the language definition. Users of STEDEL only

need to be familiar with basic geometric principles and simple concepts like speed, acceleration, rate of rotation, etc. Programming effort is thus significantly reduced and trivial syntactic or semantic errors which are frequent in VRML leading to faulty or inconsistent virtual scenes are avoided. Furthermore, in the limited scope of the few virtual scenes that we have implemented ourselves in STEDEL, we have noted that the size of descriptions in our language (in terms of size of code in bytes) is at least a factor of 3 less than the equivalent VRML code. A full grammar of the language can be obtained from [11].

5. RELATED WORK

A full survey of related work is beyond the scope of this paper and can't be included for lack of space. There are several tools aimed to facilitate the process of development of 3D worlds, e.g., AC3D [13]. In [4], SCORE, a distributed object-oriented system for 3D real time visualization is presented. It achieves a re-usable design that can be used to develop virtual buildings. In [5] the concept of a Virtual Environment is introduced. The main feature of this system is its spatial navigation means and its interactivity features. In [6] methods to create complex scenarios for the IOWA driving simulator are presented. A scenario specifies the behavior of synthetic vehicles, pedestrians, traffic control devices and variations in weather and lighting. In [7] the authors specify separately geometry and behavior in two directed acyclic graphs, the geometry and behavior graphs. In [8] an approach for specifying virtual words is described, The approach is interesting in that it allows for the temporal evolution of objects. In [9] an integrated effort for building 3D animations is presented.

6. CONCLUSIONS & FUTURE WORK

We have developed a data model for the design of interactive spatio-temporal compositions. We model the composition of geometric objects by taking into account the way that objects are linked in the real world: by means of their surfaces. To achieve this, we have introduced the concept of a Joint which specifies a geometric feature of a given geometric type. We model behavior by means of expressive ECA rules and augment the interactivity of the virtual scene by allowing the composition of actions into sequences and chains.

Our goals were to test and validate the belief that the creation of virtual reality scenes by non experts can be achieved with minimal learning effort and without significant programming experience. We have implemented aspects of our data model in STEDEL, a fully declarative prototype language for spatio-temporal compositions. STEDEL is compiled to equivalent VRML code and reduces programming effort considerably by requiring less and easier coding. Furthermore, STEDEL declarative code is much easier to maintain and extend.

In the future we will extend STEDEL by incorporating more features of our data model not presently supported, like complex events/conditions. We will extend STEDEL by incorporating multimedia and graphical features supported by VRML and rendering engines. We acknowledge the two major challenges of VR in the coming years: scaling to large virtual worlds both in terms of performance and authorability. We will build on the STEDEL experience towards achieving that goal.

7. REFERENCES

[1] International Standard, ISO/IEC 14772-1:1997, "The Virtual Reality Modeling Language (VRML)"

[2] Sun Microsystems, Inc. "Java 3D API Specification"

[3] M. Vazirgiannis, Susanne Boll, "Events in Interactive Multimedia Applications: Modeling and Implementation Design", *IEEE International Conference on Multimedia Computing and Systems*, June 1997, Ottawa, Canada

[4] R. Melster, A. Diaz and B. Groth, "SCORE-The virtual museum, development of a distributed, object-oriented system for 3D real-time visualization". *Technical Report 1998-15*, TU Berlin, October 1998

[5] A. Diaz, R. Melster, "Patterns for Modeling Behavior in Virtual Environment Applications", *Second Workshop on Hypermedia Development: design patterns in hypermedia*. Damstadt, Germany, February, 1999.

[6] J.F.Cremer, J.K. Kearney, "Scenario Authoring for Virtual Environment", *Proc. of Image VII Conference*, Tucson, AZ, June 12-17, 1994

[7] J. Dollner, K. Hinrics, "Object-Oriented 3D Modeling, Animation and Interaction", in *The Journal of Visualization and Computer Animation*, vol. 8:33-64, 1997.

[8] A. Vakaloudis, Y. Theodoridis, "The Storage and Querying of 3D objects for the Dynamic Composition of VRML worlds", *Chorochronos Workshop*, Aalborg, Denmark, June 19-20, 1998

[9] M. Najork, M. Brown, "Obliq-3D: A High-Level, Fast Turnaround 3D Animation System", IEEE Transactions on Visualization and Computer Graphics, 1(2), June 1995

[10] J.D.Foley, A. van Dam, S.K. Feiner, J.F.Hughes, "Computer Graphics: Principles and Practice", Second Edition in C, Addison-Wesley, 1997

[11] STEDEL Grammar: http://www.ics.uci.edu/~iosif/stedel

8. APPENDIX

A few simple examples of STEDEL in action.. The X-Wing object was written in 50 lines of STEDEL code. The screenshots are from a 60s space-battle visualization written in 194 lines of STEDEL code and involving around 15 motion elements and 3 camera views