ActiveVRML Comparison

This document is available at http://reality.sgi.com/employees/gavin/vrml/ActiveVRMLResponse.html. It was last updated on 12 Dec 1995.

This document compares Microsoft's proposal for VRML 2.0 (ActiveVRML) with the VRML 2.0 proposal being put forward by SGI, Sony, and WorldMaker (which I will refer to as "our proposal").

Our conclusion is that the two proposals are roughly equivalent in terms of descriptive power, but our proposal better matches both VRML and existing object-oriented tools. We believe the ActiveVRML proposal contains no compelling reasons to require a radical paradigm shift in the way 3D content is specified. Our proposal does not preclude the use of a functional scripting language, and a subset of ActiveVRML used as a scripting language with our proposal might be useful to browser and content creators.

Goals

The goals of the two proposals are similar-- to create multimedia-rich 3D worlds across the World Wide Web. ActiveVRML appears to be focused more on the multi-media (and especially 2D and 2 1/2 D) aspects, while our proposal is focused more on the 3D and World-Wide-Web issues, but in general the two proposals are trying to solve the same problems.

Functionality

The functionality of the two architectures is very similar, even though the approaches to providing that functionality are radically different. For example:

- Our notion of a prototyped node corresponds to the ActiveVRML notion of a parameterized function.
- The ActiveVRML notion of a function changing over time is equivalent to our proposal's idea of a node receiving events over time.
- The notion of composed animations is taken care of by composed functions in ActiveVRML, and is taken care of in our proposal either by the structure of the scene graph hierarchy or via composed Scripts or Interpolators that manipulate the time coming from a TimeSensor.
- Multi-media objects such as sounds with a particular location in the 3D world are supported in both proposals.
- The state-machine operations that are part of ActiveVRML are trivially implemented using the Script nodes of our proposal (all of the operations become Script nodes that update internal state-- a field-- in reaction to events).

There are some minor differences in functionality-- for example, ActiveVRML includes Image and Montage objects, our proposal includes Proximity sensors and a richer set of 3D interaction objects. These could easily be included in either proposal. Some major differences in functionality:

- ActiveVRML supports automatic integration and differentiation of values that are changing over time.
- The scripting API of our proposal supports the modification of an object's behavior; because ActiveVRML is purely declarative, all possible behaviors of an object must be known when the object is created.
- The ActiveVRML proposal does not include constructs equivalent to the `WWWInline` or `EXTERNPROTO` of our proposal to allow very large distributed worlds that are too large to fit into memory.
- Because geometry is opaque, operations such as morphing or changing the color of a vertex of a polygon are not possible in ActiveVRML.

Innovations

The ActiveVRML white paper cites four key technical innovations, each of which are key innovations of our proposal:

1. Time is implicit

   All of the benefits cited for this innovation are also true of our proposal, even though time is more explicit in our proposal (in the form of TimeSensors).

2. Interactivity is built-in

   True of our proposal also. Interactive objects may be arbitrarily combined with geometry and encapsulated using prototypes to create interactive, animated, reactive objects.

3. All media are equal

   Combining time-based media is (arguably) even easier in our proposal, and media objects (Sounds, etc) are treated the same as any other object.

4. Built upon existing formats

   Re-inventing file formats is a bad idea, and our proposal incorporates existing formats just as easily as ActiveVRML.

In addition, our proposal includes the key innovations of VRML 1.0:

1. WWWInline to allow distributed worlds across the World Wide Web
2. LOD (level of detail), that allows these worlds to be scalable.

Benefits

All of the benefits claimed in the ActiveVRML white paper are also true of our proposal:

- Sampling rate, frame generation, image resolution, etc are implicit in both proposals.
- Our proposal provides enough information to the browser so that browsers can perform many optimizations (such as not running behaviors that do not affect anything that can currently be sensed). Theoretically, ActiveVRML is more optimizable because everything that can possibly happen is known to the browser. However, just as the functional programming community has been arguing for years that ML is more optimizable than C, we have serious doubts as to whether or not highly optimized implementations are practical to implement.

- Our proposal is also designed to run on a wide variety of platforms, is open and extensible, and is as secure as the scripting language chosen.
- Our proposal was also designed with distribution in mind, and in most cases the implementation can be just as efficient as ActiveVRML-- in some cases in which Scripts make arbitrary changes to the scene, more information will be need to be transmitted across the network; however, ActiveVRML does not allow that functionality at all!

Functional vs. Object-oriented Programming

If both proposals have similar functionality and similar benefits, what is the real difference between them? Basically, it is mostly a matter of programming paradigm.

ActiveVRML comes from a functional programming heritage. Our proposal is a more traditional event or message-passing system, which we expect to be used with procedural scripting languages such as Java. We believe that our paradigm is better because:

- Humans think in terms of procedures ("DoThis; DoThat; DoLastThing"), not nested functions ("Evaluate: DoItAll = DoLastThing(DoThat(DoThis))"). Procedural programming is much easier for most people to understand than functional programming. The lesson of HTML versus SGML is that a simple, easy-to-create format will be used by "early adopters" before authoring tools are available, which will drive the demand for authoring tools and which provides the energy needed for a standard to become successful.
- Our proposal will be much easier to implement. Microsoft has promised to deliver a sample implementation. However, an efficient implementation of their proposal will require a very sophisticated implementation, and we believe they are likely to omit such optimizations from their sample implementation. Our proposal does not depend on a highly optimized implementation to get acceptable performance.
- Our proposal will work better with existing modeling and animation tools. ActiveVRML encompasses both behaviors and the transformation hierarchy; to create a walking human figure, for example, all of the transformations that are being changed must be specified in ActiveVRML; only the geometric primitives would be specified outside of ActiveVRML (in multiple .wrl VRML files). Our proposal is much less intrusive, allowing behaviors to be attached "from the side", keeping the existing model hierarchy unchanged.
- World creators will be better able to control the performance of their worlds using our proposal. Because ActiveVRML performance relies so much on optimizations implemented in the browser, creating speedy ActiveVRML worlds will be much more difficult; especially for highly dynamic worlds in which the ActiveVRML implementation must constantly re-optimize the parts of the world that are changing. Our proposal gives the world creator much more control over performance, with defaults carefully chosen to allow the browser maximum opportunity for optimization and functionality carefully chosen to encourage the creation of very fast, scalable worlds.
- Our proposal fits in much better with VRML 1.0. It is a natural extension of the VRML 1.0 syntax and ideas, not a radical departure. A radical change could be justified if it included significant functionality or performance advantages; however, there are no clear advantages of the ActiveVRML proposal over our proposal.
- There is a large and well-understood infrastructure for procedural programming. In particular, we believe that Java will be the language of choice for the World Wide Web. There will be a large set of tools and applications to help the developer using Java, there will be a lot of effort on making Java run fast on a wide variety of machines, etc. We do not believe that ActiveVRML will ever have a comparable infrastructure.
- Our proposal allows composition of behaviors involving either entire entities (for example, make the solar system spin around the center of the galaxy) or partial entities (for example, morphing a set of vertices or moving one vertex of a polygon). Functional composition in ActiveVRML is best suited for entire entities. In fact, because geometry is opaque to ActiveVRML, the two examples involving partial entities are not even possible with ActiveVRML.