WebTOP Training: Interacting with an X3D Scene Using Xj3D

Resources:

- X3D Language Specification: Although you won't actually be writing any code in this tutorial, this is a good resource for later. The Web3D Consortium maintains an online version of the spec at http://www.web3d.org. Go to X3D Development -> Specifications and select "X3D Architecture and base components with Amendment 1."
- VRML 2.0 Reference: As of this writing, the WebTOP project has two hard copies of this book. Since X3D is basically a better revision of VRML, this book is a pretty good resource for early learning.
- Xj3D Browser: This is the open-source stand-alone X3D viewing program created by Xj3D. We use the libraries created by Xj3D to create a very similar viewer for our modules. You won't need it for this lesson, but you will later.
    - http://www.xj3d.org

Assignment

This is a reading assignment. It is intended to give you the very basics of X3D concepts to get you started so you can figure out the other tutorial labs. You'll learn more about the language from doing the labs and making modules than you ever will from reading, so this is designed to give you just enough information to start coding the rest of the labs in this series.

X3D Language Characteristics

X3D is a sort of object-oriented programming language. I say "sort of" because I'm not sure if it truly is in the academic sense, but it certainly has objects. It is not an imperative language like C/C++ or Java, where the whole gist of the language is executing one statement after another. Instead it creates a series of objects called "Nodes" and describes their properties and relationships to one another. The combination of the nodes and their relationships is called the Scene Graph, or just the Scene.

How X3D Creates 3D Scenes

As mentioned, the basic element of a X3D Scene is a Node. A Node can be many things, from a Shape with some specified geometry (e.g., a box), to moving that Shape about, to a description of the

navigation properties of the camera in the Scene. Basically, as far as X3D is concerned, all objects are Nodes. Nodes have properties, including the ability to group together other Nodes. Nodes with this property are called Grouping Nodes. The Scene is a hierarchical graph of these Nodes. The scene is actually a tree graph, with any number of root Nodes, and if any of the root Nodes are Grouping Nodes, they create branches, where the children of the Grouping Nodes become, appropriately, Children Nodes. And the process continues...

Fields

Nodes, as objects, have some member data, called "fields." Fields can be either individual atomic data items like numbers or character strings. Fields have three sub-properties: access type, data type, and value. Access type will be discussed last. Data type and value are described together.

Data type can be various things, including intergers, floats, and even other Nodes. Data type encompasses several things. It specifies whether the field holds one value (Single Field, or SF) or multiple values (Multiple Field, or MF). It also specifies the conventional concept of data type (integer, character string, float, Node, etc.), as well as whether each individual value is a scalar or a vector (a.k.a. array). For example, take two data types: SFBool and MFVec3f. SFBool holds a single boolean value. MFVec3f contains a vector of vectors, each of which hold three floating point values. Think of MFVec3f as a two-dimensional array of floating points. As stated, fields can also contain one or more Nodes (SFNode or MFNode). For instance, the Shape Node has a field called "geometry," which has as its value a Geometry Node, which is an abstract class of Nodes. Some specific Nodes derived from Geometry Node are Box, Cone, Cylinder, etc.

A field's access type basically defines the read/write permissions that other Nodes have to the field. It also specifies when in the process of loading the Node into the Scene that the field's value can be set, but that is outside the scope of this simple discussion. There are four access types:

- `initializeOnly`: Writable only by hard-coding. Readable only by the X3D interpreter.
- `inputOutput`: Readable and writable.
- `inputOnly`: Writable only. Used to send events to the Node (more detail in a later tutorial).
- `outputOnly`: Readable only. Used to send events from the Node (see above).=

It is worth noting that from an object-oriented perspective, all fields are public **in respect to**

inheritance. For purposes of other Nodes accessing their values, while Nodes don't do so directly in the C/C++ sense, they can access fields as limited by their access type described in the above list.

Encodings and Syntax Example

So you conceptually understand the Scene Graph. So what's it look like? Well, that depends. When manually authoring X3D content, you have two choices of language syntax: XML (often just called X3D) and Classic (a.k.a. VRML encoding, VRML 3.0 or Classic encoding). Since WebTOP historically used VRML and its predecessors, and since at the time of the conversion from VRML and Applet-based delivery to X3D and application-based delivery there were no good VRML to X3D-XML conversion programs, it was decided to stay with the Classic encoding. It is this writer's opinion that Classic encoding is much more readable with scenes as complex as we have, anyway, considering the amount of fine-tuned control we need over every node.

The basic syntax for a Classic Scene looks something like this, using previous examples.

```
#X3D V3.0 utf8

SomeRootNode {
    outputOnly  SFBool  someFieldName      TRUE
    inputOutput MFVec3f someOtherFieldName [0.5 1 2.3,
                                            1 1 1]
}
```

The # creates a one-line comment, which is the only comment type in X3D. Nodes begin with the name of the node, and their definitions are contained within curly braces. Note that X3D does not have a line delimiter like C. Other than newlines, whitespace is generally unimportant. However, newlines do not always start a new statement. For instance, in the example above, the newline in the middle of the MFVec3F's main vector dimension does not cause a syntax error. The interpreter is waiting for the ']' to end that dimension. Also note the odd syntax of using spaces for individual vector items, but commas to separate vectors themselves.