X3D Graphics for Web Authors

Chapter 7

Event Animation

*If it ain't moving, it ain't 3D.*

Andy van Dam, SIGGRAPH Pioneer, Brown University
Contents

Chapter Overview

Concepts

X3D Nodes and Examples

Chapter Summary and Suggested Exercises

References
Chapter Overview
Overview: Event Animation

Behaviors, events, ROUTE connections, animation

Animation as scene-graph modification

Event-animation design pattern: 10-step process

Interpolation nodes

- TimeSensor and event timing
- ScalarInterpolator and ColorInterpolator
- OrientationInterpolator, PositionInterpolator, PositionInterpolator2D and NormalInterpolator
- CoordinateInterpolator2D, CoordinateInterpolator
Concepts
Behaviors

**Behavior** defined as changing the value of some field contained by some node in scene graph.

Animation nodes, user interaction nodes and network updates can produce updated values.

ROUTE statements connect output of one node as an input to field in another node.

**Event** defined as the time-stamped value passed by a ROUTE, from one field to another.

Thus the values held by nodes in scene graph can change as time advances.
Behavior traversal of scene graph

Double buffer: once frame is swapped to update screen image, repeat and update scene values

*Event model* consists of

- Examining clock-driven and user-initiated events
- Updating scene-graph values
- Triggering and updating new events as appropriate
- Continue until all events handled, loops not allowed

Event updates modify the scene graph

- Changing rendering properties, or
- Generating further event outputs
Example behavior event chain

- User clicks button to start a timer clock
- Clock outputs new event at start of each frame,
  - ... which stimulates linear-interpolation function which produces another output value
  - ... which updates some target value in scene graph
- Repeat event traversal after each frame redraw
ROUTE connections

ROUTE connection enables the output field of one node to pass a value that then stimulates the input field of another node

- The passed value also includes a time stamp

Field data type and accessType must both match between node/field of source and target

- Chapter 1, Technical Introduction lists field types
- Also provided in tooltips and specification
- Authors usually must carefully check these
Animation as scene-graph modification

**Behavior** = changing a field value in a node, somewhere in the scene graph

**Event** = time-stamped value going over a ROUTE

**Event cascade** is a series of events, each one triggering the next, before next frame is drawn

- No event loops allowed, guaranteeing completion

Thus all X3D animation can be considered as modification of the scene graph at run time
Event-animation design pattern

X3D can be imposing, there are many nodes. Nevertheless, a simple design pattern is used for nearly every kind of animation.

This consistent event ROUTE pattern enables you to expertly animate most X3D scene behaviors.
Visualizing scenes on paper

It is good practice to sketch out 3D scene drafts
  • Consider what models are needed, and how multiple models might be composed

Consider user experience, from their perspective
  • What tasks and goals, what use cases
  • What might things look like when first seen

Storyboarding can help build long-form content
  • Series of vignettes to tell a larger story
  • Each scene defines needed models and behaviors
  • Build each piece, put them together
Field data types

X3D is a strongly typed language

• Each field in each node (i.e. each XML attribute) has a strictly defined data type

• Data types for boolean, integer, floating point

Types are either single or multiple-value

• Example: SFFloat, SFVec2f, SFVec3f, SFRotation

Also have arrays for all types

SF = Single Field, MF = Multiple Field (array)

Failure to match data types correctly is an error!

• During schema validation, loading or at run time
X3D has strong data typing

Data typing is very important to prevent errors

- *Strong data typing* means that all data types must match (or be converted) exactly
- *Weak data typing* means data types may be promoted or changed by the system automatically without author direction (or quality control)

Data type errors lead to erroneous computations and system crashes, in any computer language

X3D has strong data typing

- Cost: authors must ensure their scene is correct
- Benefit: mysterious run-time errors avoided
<table>
<thead>
<tr>
<th>Field-type names</th>
<th>Description</th>
<th>Example values</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFBool</td>
<td>Single-field boolean value</td>
<td>true or false (X3D syntax), TRUE or FALSE (Classic VRML syntax)</td>
</tr>
<tr>
<td>MFBool</td>
<td>Multiple-field boolean array</td>
<td>true false false true true (X3D syntax), [ TRUE FALSE FALSE TRUE ] (Classic VRML syntax)</td>
</tr>
<tr>
<td>SFColor</td>
<td>Single-field color value, red-green-blue</td>
<td>0.5 1.0</td>
</tr>
<tr>
<td>MFColor</td>
<td>Multiple-field color array, red-green-blue</td>
<td>1.0 0.1 0.0 1</td>
</tr>
<tr>
<td>SFColorRGBA</td>
<td>Single-field color value, red-green-blue alpha</td>
<td>0.5 1.0 0.75</td>
</tr>
<tr>
<td>MFColorRGBA</td>
<td>Multiple-field color array, red-green-blue alpha</td>
<td>1.0 0.25 0.5 0.75 (red green blue, varying opacity)</td>
</tr>
<tr>
<td>SFInt32</td>
<td>Single-field 32-bit integer value</td>
<td>0</td>
</tr>
<tr>
<td>MFInt32</td>
<td>Multiple-field 32-bit integer array</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>SFFloat</td>
<td>Single-field single-precision floating-point value</td>
<td>1.0</td>
</tr>
<tr>
<td>MFFloat</td>
<td>Multiple-field single-precision floating-point array</td>
<td>-1.2 3.14159</td>
</tr>
<tr>
<td>Field-type names</td>
<td>Description</td>
<td>Example values</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>SFDdouble</td>
<td>Single-field double-precision floating-point value</td>
<td>2.7128</td>
</tr>
<tr>
<td>MFDdouble</td>
<td>Multiple-field double-precision array</td>
<td>-1 2.0 3.14159</td>
</tr>
<tr>
<td>SFIImage</td>
<td>Single-field image value</td>
<td>Contains special pixel-encoding values, see Chapter 5 for details</td>
</tr>
<tr>
<td>MFImage</td>
<td>Multiple-field image value</td>
<td>Contains special pixel-encoding values, see Chapter 5 for details</td>
</tr>
<tr>
<td>SFNode</td>
<td>Single-field node</td>
<td>&lt;Shape/&gt; or Shape {space}</td>
</tr>
<tr>
<td>MFNode</td>
<td>Multiple-field node array of peers</td>
<td>&lt;Shape/&gt; &lt;Group/&gt; &lt;Transform/&gt;</td>
</tr>
<tr>
<td>SFRotation</td>
<td>Single-field rotation value using 3-tuple axis, radian angle form</td>
<td>0 1 0 1.57</td>
</tr>
<tr>
<td>MFRotation</td>
<td>Multiple-field rotation array</td>
<td>0 1 0 0, 0 1 0 1.57, 0 1 0 3.14</td>
</tr>
<tr>
<td>SFString</td>
<td>Single-field string value</td>
<td>“Hello world!”</td>
</tr>
<tr>
<td>MFString</td>
<td>Multiple-field string array</td>
<td>“EXAMINE” “FLY” “WALK” “ANY”</td>
</tr>
<tr>
<td>SFTime</td>
<td>Single-field time value</td>
<td>0</td>
</tr>
<tr>
<td>MFTime</td>
<td>Multiple-field time array</td>
<td>-1 0 1 567890</td>
</tr>
<tr>
<td>Field-type names</td>
<td>Description</td>
<td>Example values</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>SFVec2f/SFVec2d</td>
<td>Single-field 2-float/2-double vector value</td>
<td>0 1.5</td>
</tr>
<tr>
<td>MFVec2f/MFVec2d</td>
<td>Multiple-field 2-float/2-double vector array</td>
<td>1 0, 2 2, 3 4, 5 5</td>
</tr>
<tr>
<td>SFVec3f/SFVec3d</td>
<td>Single-field vector value of 3-float/3-double values</td>
<td>0 1.5 2</td>
</tr>
<tr>
<td>MFVec3f/MFVec3d</td>
<td>Multiple-field vector array of 3-float/3-double values</td>
<td>10 20 30, 4.4 –5.5 6.6</td>
</tr>
</tbody>
</table>

**ClassicVRML syntax notes**

- **TRUE** and **FALSE** (rather than XML true and false)
- MF multiple-field array values are surrounded by square brackets, e.g. [ 10 20 30, 4.4 –5.5 6.6 ]
- No special XML escape characters such as `&amp;`
accessType: input, output, initialize

accessType determines if field is data sender, receiver, or holder

- inputOnly: can only receive events
- outputOnly: can only send events
- initializeOnly: cannot send or receive
- inputOutput: can send, receive and be initialized

Failure to match accessType correctly is an error!

- Detected during authoring-tool checks, or run time
- inputOnly and outputOnly values cannot be listed as attributes in .x3d scene file, since they are transient
The accessType names were changed when VRML97 was upgraded to X3D. Functionality remains essentially unchanged.

X3D specification entries for each node use yet another shorthand, as shown here:

<table>
<thead>
<tr>
<th>VRML97 Name</th>
<th>X3D Name</th>
<th>X3D Specification abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventIn</td>
<td>inputOnly</td>
<td>[in]</td>
</tr>
<tr>
<td>eventOut</td>
<td>outputOnly</td>
<td>[out]</td>
</tr>
<tr>
<td>field</td>
<td>initializeOnly</td>
<td>[ ]</td>
</tr>
<tr>
<td>exposedField</td>
<td>inputOutput</td>
<td>[in,out]</td>
</tr>
</tbody>
</table>

VRML, Virtual reality modeling language; X3D, Extensible 3D.
accessType naming conventions 2

Field names often reveal special accessType

- Prefix `set_` indicates inputOnly field
- Prefix `_changed` indicates outputOnly field
- Prefix `is` for outputOnly boolean field (e.g. isActive)

inputOnly, outputOnly fields not allowed in files

Understanding naming conventions helps authors understand ROUTE definitions and results

Looking ahead: we will name our own fields when creating Scripts and prototypes, further underscoring importance of naming
Interpolating animation chains: 10-step design process

The following 10-step process can be used for all animation tasks

Table is also provided in order to look up how to produce typed-value outputs corresponding to each interpolator or sequencer node

A detailed example follows

This 10-step process is a good check to perform each time you create an animation chain
Interpolating animation chains 1-2

1. **Pick target.** Pick node and target field to animate (i.e., field that receives changing animation values)

2. **Name target.** Provide a DEF label for the node of interest, giving it a name
Interpolating animation chains  3-4

3. **Check accessType and data type.**
   - Ensure target field has *accessType* of inputOnly or inputOutput, so that it can receive input events
   - Determine if target field has floating-point type: SFFloat, SFVec3f, SFColor, SFRotation, and so on...
     If so, use an interpolator node as the event source

4. **Determine if Sequencer or Script.**
   - If the target type is an SFBool or SFInt32, use a sequencer node as event source
   - If the target type is an SFNode or MFNode, use a Script node as the event source
5. **Determine which Interpolator.** If you are not using a sequencer or Script node, determine corresponding Interpolator which produces the appropriate data type for `value_changed` output using lookup table

- Example: PositionInterpolator produces `SFVec3f value_changed` events

6. **Triggering sensor.** If desired, add sensor node at beginning, to provide appropriate `SFTime` or `SFBool` trigger to start animation

- Sometimes the triggering event is an output event from another animation chain
Interpolating animation chains  7-8

7. **TimeSensor clock.** Add a TimeSensor as the animation clock, then set its *cycleInterval* field to the desired duration interval of animation
   - Set *loop='false'* if an animation only runs once at certain specific times. (Will need triggering event.)
   - Set *loop='true'* if it loops repeatedly

8. **Connect trigger.** ROUTE sensor or trigger node’s output field to the TimeSensor input in order to start the animation chain
   - Each node in animation chain needs a DEF name, so that ROUTE can connect to/from
Interpolating animation chains 9-10

9. **Connect clock.** ROUTE the TimeSensor `fraction_changed` field to the interpolator (or sequencer or Script) node's `set_fraction` field, in order to drive the animation chain.

10. **Connect animation output.** ROUTE the interpolator, sequencer, or Script node's `value_changed` field to target field of interest in order to complete the animation chain.

Construction of animation-chain design pattern is complete, now test whether animation works.
Example animation chains

Each row in Table 7.2 shows commonly authored sequences of nodes in animation chains

<table>
<thead>
<tr>
<th>Triggering Nodes (Optional)</th>
<th>Clock Nodes</th>
<th>Value-Producing Nodes</th>
<th>Value-Consuming Nodes, Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>TouchSensor</td>
<td>TimeSensor</td>
<td>ScalarInterpolator</td>
<td>Material (transparency)</td>
</tr>
<tr>
<td>VisibilitySensor</td>
<td>TimeSensor</td>
<td>ColorInterpolator</td>
<td>Material (color field)</td>
</tr>
<tr>
<td>PrimarySensor</td>
<td>TimeSensor</td>
<td>PositionInterpolator</td>
<td>Transform (translation, scale)</td>
</tr>
<tr>
<td>TouchSensor</td>
<td>TimeSensor</td>
<td>OrientationInterpolator</td>
<td>Transform (rotation)</td>
</tr>
<tr>
<td>MovieTexture (loop complete)</td>
<td>TimeSensor</td>
<td>PositionInterpolator2D</td>
<td>Rectangle2D</td>
</tr>
</tbody>
</table>

Used in Step 5: Determine which Interpolator
# X3D field types and corresponding animation nodes

<table>
<thead>
<tr>
<th>Field type</th>
<th>Description</th>
<th>Interpolator/Sequencer animation nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFBool</td>
<td>Single-field boolean value</td>
<td>BooleanSequencer</td>
</tr>
<tr>
<td>SFColor</td>
<td>Single-field Color value, red-green-blue</td>
<td>ColorInterpolator</td>
</tr>
<tr>
<td>SFInt32</td>
<td>Single-field 32-bit Integer value</td>
<td>IntegerSequencer</td>
</tr>
<tr>
<td>SFFloat</td>
<td>Single-field single-precision floating-point value</td>
<td>ScalarInterpolator</td>
</tr>
<tr>
<td>SFRotation</td>
<td>Single-field Rotation value using 3-tuple axis, radian angle form</td>
<td>ColorInterpolator</td>
</tr>
<tr>
<td>SFTime</td>
<td>Single-field Time value</td>
<td>TimeSensor</td>
</tr>
<tr>
<td>SFVec2f</td>
<td>Single-field 2-float vector value</td>
<td>PositionInterpolator2D</td>
</tr>
<tr>
<td>MFVec2f</td>
<td>Multiple-field 2-float vector array</td>
<td>CoordinateInterpolator2D</td>
</tr>
<tr>
<td>SFVec3f</td>
<td>Single-field vector value of 3-float values</td>
<td>PositionInterpolator</td>
</tr>
<tr>
<td>MFVec3f</td>
<td>Multiple-field vector array of 3-float values</td>
<td>CoordinateInterpolator</td>
</tr>
</tbody>
</table>

*Used in Step 5: Determine which Interpolator*
Animation chain for this example

HelloX3dAuthorsAnimationChain.x3d
is our detailed animation-chain example

TouchSensor            TimeSensor       OrientationInterpolator       Transform

Trigger node           Clock node       Interpolator or Sequencer       Target node

SFTime event           SFFloat event    SFRotation event

touchTime => startTime  value_changed => rotation

fraction_changed => set_fraction
Hello X3D Authors showing ROUTEs

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO/OMS/DOM/DOTX3D 3.0/EN" "http://www.web3d.org/specifications/x3d-3.0.dtd">
  <head>
    <meta name="filename" content="HelloX3DAuthors.x3d"/>
    <meta name="author" content="Don Brutzman"/>
    <meta name="created" content="5 October 2000"/>
    <meta name="revised" content="8 March 2005"/>
    <meta name="description" content="Simple example showing spinning globe and text. Hello!"/>
    <meta name="url" content="http://www.web3d.org/x3d/content/examples/course/HelloX3DAuthors.x3d"/>
    <meta name="generator" content="X3D-Edit, http://www.web3d.org/x3d/content/examples/README.X3D-Edit.html"/>
    <meta name="license" content=".././license.html"/>
  </head>
  <Scene>
    <World Info title="Hello X3D Authors" info : an introductory scene
    <Viewpoint description="Hello, world" position="0 0 -8" orientation="0 1 0 3.14159"
    <Navigation Info type="EXAMINE" "ANY"
    <TimeSensor DEF="OrbitalTimeinterval cyclicInterval"="12.0", loop="true"
    <OrientationInterpolator DEF="SpinThoseThings" key="0.0 0.25 0.50 0.75 1.00", keyValue="0 1.0 0 1.0 1.5708, 0 1 0 3.14159, 0 1 0 4.7123889, 0 1 0 6.2831852"
    <ROUTE fromNode="OrbitalTimeinterval", fromField="traction_changed", toNode="SpinThoseThings", toField="set_traction"
    <Transform DEF="EarthCoordinateSystem"
    <ROUTE fromNode="EarthCoordinateSystem", fromField="value_changed", toNode="EarthCoordinateSystem", toField="rotation"
    <Group DEF="MiniWorld"
    <Shape>
      <Appearance>
        <ImageTexture url="earth-topo.png", "earth-topo.gif", "earth-topo-small.gif", "http://www.web3d.org/x3d/content/examples/course/earth-topo.png"
        <Shape>
          <Transform DEF="SimpleGeoStationarySatellite", translation="0 0 5", rotation="1 0 0 .3", scale="0.1 0.3 0.1"
          <Shape>
            <Appearance>
              <Material diffuseColor="0.9 0.1 0.1"
              <Text string="Hello NPS X3D Authors II"
              <FontStyle size="3"
```

Hello X3D Authors 10-step process
Hello X3D Authors 10-step process

1. **Pick target.** The target node is a Transform, and the target field is *set_rotation*.

2. **Name target.** The Transform is named `DEF='EarthCoordinateSystem'`.

3. **Check accessType and data type.** As shown by the Transform node field-definition table in Chapter 3 and the X3D-Edit tooltip, the *set_rotation* field has type SFRotation.

4. **Determine whether Sequencer or Script.** These special node types are not applicable to this example, because the data type for *set_rotation* is SFRotation which is a floating-point type.

5. **Determine which Interpolator.** The animating OrientationInterpolator is named `DEF='SpinThoseThings'` and placed just before the Transform.

6. **Triggering sensor.** A triggering TouchSensor is added next to the geometry to be clicked, and then named `DEF='ClickTriggerTouchSensor'`.

7. **TimeSensor clock.** The TimeSensor is added at the beginning of the chain, named `DEF='OrbitalTimeInterval'` and has both the *cycleInterval* and *loop* fields set.

8. **Connect trigger.** Add ROUTE to connect the triggering TouchSensor node’s `touchTime` output field to the clock node’s `startTime` input field.

9. **Connect clock.** Add ROUTE to connect the clock node’s *fraction_changed* output field to the interpolator node’s *set_fraction* input field.

10. **Connect animation output.** Add ROUTE to connect the interpolator node’s *value_changed* output field to the original target input field, *set_rotation*. 
ROUTE editor examples

<ROUTE
    fromNode='OrbitalTimeInterval'
    fromField='fraction_changed'
    toNode='SpinThoseThings'
    toField='set_fraction' />

<ROUTE
    fromNode='ClickTriggerTouchSensor'
    fromField='touchTime'
    toNode='OrbitalTimeInterval'
    toField='startTime' />
Interpolation

Interpolation is the estimation of intermediate values from other values

Computing averages is computationally efficient and highly optimizable

Linear approximation is thus well suited for high-performance graphics animation

X3D provides interpolation nodes for each of the floating-point data types

• including multiple-value types: Color, Vec3f, etc.
Interpolation node type

X3DInterpolationNode is the formal name for the interpolation node type.

Each interpolation node includes the following common fields and naming conventions:

- SF, MF <type> definition must be consistent for node in order to properly define response function.

<table>
<thead>
<tr>
<th>Type</th>
<th>accessType</th>
<th>Name</th>
<th>Default</th>
<th>Range</th>
<th>Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFFloat</td>
<td>inputOutput</td>
<td>key</td>
<td>[]</td>
<td>(−∞, ∞)</td>
<td>Interchange</td>
</tr>
<tr>
<td>MF&lt;type&gt;</td>
<td>inputOutput</td>
<td>keyValue</td>
<td>[]</td>
<td>(type dependent)</td>
<td>Interchange</td>
</tr>
<tr>
<td>SFFloat</td>
<td>inputOnly</td>
<td>set_fraction</td>
<td></td>
<td></td>
<td>Interchange</td>
</tr>
<tr>
<td>[SF, MF]&lt;type&gt;</td>
<td>outputOnly</td>
<td>value_changed</td>
<td></td>
<td></td>
<td>Interchange</td>
</tr>
<tr>
<td>SFNode</td>
<td>inputOutput</td>
<td>metadata</td>
<td>NULL</td>
<td>[X3DMetadataObject]</td>
<td>Core</td>
</tr>
</tbody>
</table>
Common interpolator fields

- *key, keyValue* hold the point values defining the characteristic function
- *key* array always has type MFFloat
- *keyValue* array data type matches the named type of the parent Interpolator node
  - final value must equal first value in *keyValue* array if smooth looping is desired
- Lengths of *key, keyValue* arrays must be equal
- Note that *keyValue* array can hold values which are themselves MF (multi-field) array type
- Function output *value_changed* always has same name, but data type matches the Interpolator node
Linear interpolation

Piecewise-linear curve fitting can approximate any curve with arbitrary accuracy.

Multi-field (MF) values are individually interpolated proportionately.

`key='0 0.3333 0.666 1'`

`keyValue='1 0 0, 0 1 0, 0 0 1, 1 0 0'`
Step-wise linear interpolation

Step functions are created by repeating time values and corresponding output

key='0 0.25 0.25 0.5 0.5 1'
keyValue='1 1 2 2 3 4'

Note that time-fraction key array must always be monotonically (steadily) increasing
Double linear-interpolation averaging

Matched *key, keyValue* arrays define the points for a linear-interpolator approximation function.

Two-way weighted averaging is used to compute interpolated-input, interpolated-output results.
X3D Nodes and Examples
TimeSensor

TimeSensor is the heartbeat of an animation
• provides pulse that triggers event cascades
• initiates computations for drawing next frame
• Outputs values as fraction_changed, from 0 to 1

TimeSensor samples elapsed time based on the computer clock, rather than screen update rate
• Ensures that animations are smooth and realistic
• Fixed (constant) frame rate is typically not feasible since computation varies for screen-image updates
**TimeSensor output**

Output time is an SFTTime ramp function ranging [0,1] that repeats every `cycleInterval` seconds

- Sometimes called a 'sawtooth' function
- SFFloat output field `fraction_changed` used as input to other interpolators, sequencers

```plaintext

time = now
temp = (now - startTime) / cycleInterval
f = fractionalPart(temp)

if (now ≤ startTime)
    fraction_changed = 0.0
if ((f == 0.0) && (now > startTime))
    fraction_changed = 1.0
else fraction_changed = f
```
TimeSensor fields

- *enabled* controls whether node enabled or disabled
- *loop* is an SFBool indicating whether to continue looping indefinitely after first cycle is complete
- *cycleInterval* defines total loop duration in seconds, either for single-shot animation or looped repetition
- *cycleTime* field is sent an SFTime output value upon completion of each loop
TimeSensor fields 2

- **startTime**, **stopTime** are provided (or contain) SFTime values for when to start, stop respectively
  - ROUTE an SFTime value to **startTime** or **stopTime**
  - **isActive**, **isPaused** are output SFBool true/false events sent whenever the TimeSensor is set to run or paused

- **pauseTime**, **resumeTime** are SFTime values for current clock time whenever paused or resumed
  - Corresponding boolean **isPaused** event is also sent, with value of true when paused and false when resuming

- **elapsedTime** output provides cumulative number of seconds since TimeSensor was activated and began running, without including paused time
<xml version="1.0" encoding="UTF-8">
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1.1//EN" "http://www.web3d.org/specifications/x3d-3.1.dtd">
<X3D profile='Immersive' version='3.0' xmlns:x3d = "http://www.w3.org/2001/XMLSchema-instance"
    x3d:noNamespaceSchemaLocation = "http://www.web3d.org/specifications/x3d-3.1.xsd">
    <head>
        <meta content='ColorInterpolatorExample.x3d' name='filename'/>
        <meta content='Demonstrate basic design pattern for animating a node.' name='description'/>
        <meta content='Don Brutzman' name='creator'/>
        <meta content='17 April 2005' name='created'/>
        <meta content='17 January 2008' name='modified'/>
        <meta content='ColorInterpolatorExampleSceneGraphWithRoutes.png' name='drawing'/>
        <meta content='ColorInterpolatorExample4Colors.png' name='image'/>
        <meta content='Animation ColorInterpolator' name='keywords'/>
        <meta content='&amp;apos;http://X3DGraphics.com/examples/X3DForWebAuthors/Chapter07-EventAnimationInterpolation/ColorInterpolatorExample.x3d' name='identifier'/>
        <meta content='X3D-Edit, http://www.web3d.org/x3d/content/README.X3D-Edit.html' name='generator'/>
        <meta content='/' name='license'/>
    </head>

    <Scene>
        <!-- Place triggering text above sphere of interest --><!--
        <Transform translation='0 2 0'>
            <Shape>
                <Text string='"Touch text to " start animation..."'/>
                <FontStyle justify='MIDDLE' size='MIDDLE'/>
            </Text>
        </Shape>
        </Transform>
        <!-- This TouchSensor only reacts to user clicking on the sibling Shape and Text, because it is under a parent Transform grouping node -->
        <TouchSensor DEF='TextTriggerTouchSensor' description='Touch text to start...'/>
        </Transform>
        <!-- Here is Sphere with accompanying Material that will get animated -->
        <Transform translation='0 -1 0'>
            <Shape>
                <Appearance>
                    <!-- SphereMaterial diffuseColor gets overridden by Interpolator output -->
                    <Material DEF='SphereMaterial' diffuseColor='0.5 0.5 0.5'/>
                </Appearance>
            </Shape>
            <Transform>
                <!-- TimeSensor is triggered to start by TouchSensor, then sends animating values to Interpolator -->
                <TimeSensor DEF='AnimationClock' cycleInterval='0' loop='false'/>
                <ROUTE fromField='touchTime' fromNode='TextTriggerTouchSensor' toField='startTime' toNode='AnimationClock'/>
                <!-- Interpolator: ColorChanger interpolates evenly between red, green, blue and then back to red -->
                <ColorInterpolator DEF='ColorChanger' key0='0.3333 0.6666 1' key1='1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1' />
                <!-- ROUTE 2: the ColorChanger interpolator gets stimulated by AnimationClock TimeSensor fraction to compute a color -->
                <ROUTE fromField='fraction_changed' fromNode='AnimationClock' toField='set_fraction' toNode='ColorChanger'/>
                <!-- ROUTE 3: Interpolator output is sent to target node of interest. Changed color value is routed to SphereMaterial -->
                <ROUTE fromField='value_changed' fromNode='ColorChanger' toField='diffuseColor' toNode='SphereMaterial'/>
            </Transform>
        </Scene>
    </X3D>
</xml>
TimeSensor continuously generates events as time passes. Typical use: ROUTE thisTimeSensor.fraction_changed TO someInterpolator.set_fraction. Interchange profile hint: TimeSensor may be ignored if cycleInterval < 0.01 second.

**DEF**

[DEF ID #IMPLIED]

DEF defines a unique ID name for this node, referencable by other nodes. Hint: descriptive DEF names improve clarity and help document a model.

**USE**

[USE IDREF #IMPLIED]

USE means reuse an already DEF-ed node ID, ignoring _all_ other attributes and children. Hint: USEing other geometry (instead of duplicating nodes) can improve performance. Warning: do NOT include DEF (or any other attribute values) when using a USE attribute!

**enabled**

[enabled: accessType inputOutput, type SFBool (true|false) "true"]

Enables/disables node operation.

**cycleInterval**

[cycleInterval: accessType inputOutput, type SFT ime CDATA "1.0"]

cycleInterval is loop duration in seconds. Interchange profile hint: TimeSensor may be ignored if cycleInterval < 0.01 second.

**loop**

[loop: accessType inputOutput, type SFBool (true|false) "false"]

Repeat indefinitely when loop=true, repeat only once when loop=false.

**startTime**

[startTime: accessType inputOutput, type SFT ime CDATA "0"]

When time now >= startTime, isActive becomes true and TimeSensor becomes active. Absolute time: number of seconds since Jan 1, 1970, 00:00:00 GMT. Hint: usually receives a ROUTEd time value.

**stopTime**

[stopTime: accessType outputOnly, type SFT ime CDATA "0"]

When stopTime becomes <= time now, isActive becomes false and TimeSensor becomes inactive. Absolute time: number of seconds since Jan 1, 1970, 00:00:00 GMT. Hint: usually receives a ROUTEd time value.

**cycleTime**

[cycleTime: accessType outputOnly, type SFT ime CDATA #FIXED ""]

cycleTime sends a timeOnly outputAt startTime, and also at the beginning of each new cycle (useful for synchronization with other time-based objects).

**isActive**

[isActive: accessType outputOnly, type SFBool (true|false) #FIXED ""]

isActive true/false events are sent when TimeSensor starts/stops running.

**isPaused**

[isPaused: accessType outputOnly, type SFBool (true|false) #FIXED ""]

isPaused true/false events are sent when TimeSensor is paused/resumed. Warning: not supported in VRML97.

**pauseTime**

[pauseTime: accessType inputOutput, type SFT ime CDATA "0"]

When time now >= pauseTime, isPaused becomes true and TimeSensor becomes paused. Absolute time: number of seconds since Jan 1, 1970, 00:00:00 GMT. Hint: usually receives a ROUTEd time value. Warning: not supported in VRML97.

**resumeTime**

[resumeTime: accessType inputOutput, type SFT ime CDATA "0"]

When resumeTime becomes <= time now, isPaused becomes false and TimeSensor becomes inactive. Absolute time: number of seconds since Jan 1, 1970, 00:00:00 GMT. Hint: usually receives a ROUTEd time value. Warning: not supported in VRML97.

**elapsedTime**

[elapsedTime: accessType outputOnly, type SFT ime CDATA #FIXED ""]

Current elapsed time since TimeSensor activated/running, cumulative in seconds, and not counting any paused time. Warning: not supported in VRML97.
<table>
<thead>
<tr>
<th>fraction_changed</th>
<th>[\text{fraction_changed: accessType outputOnly, type SFFloat CDATA #FIXED ''}]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fraction_changed continuously sends value in range [0,1] showing time progress in the current cycle.</td>
</tr>
<tr>
<td>time</td>
<td>[\text{time: accessType outputOnly, type SFTime CDATA #FIXED ''}]</td>
</tr>
<tr>
<td></td>
<td>Time continuously sends the absolute time (since January 1, 1970) for a given simulation tick.</td>
</tr>
<tr>
<td>containerField</td>
<td>[\text{containerField: NMTOKEN &quot;children&quot;}]</td>
</tr>
<tr>
<td></td>
<td>containerField is the field-label prefix indicating relationship to parent node. Examples: geometry Box, children Group, proxy Shape. containerField attribute is only supported in XML encoding of X3D scenes.</td>
</tr>
<tr>
<td>class</td>
<td>[\text{class CDATA #IMPLIED}]</td>
</tr>
<tr>
<td></td>
<td>class is a space-separated list of classes, reserved for use by XML stylesheets. class attribute is only supported in XML encoding of X3D scenes.</td>
</tr>
</tbody>
</table>
ScalarInterpolator node

Generates a scalar (single-valued) SFFloat for 
value_changed output 
key and keyValue arrays contain SFFloat values

set_fraction determines input value to piece-wise linear function

• Percentage between bracketing key[i], key[i+1] values used to compute corresponding output value_changed as weighted average between keyValue[i], keyValue[i+1]
• Which is same algorithm for all interpolators
<X3D version="1.0" encoding="UTF-8">
  <!DOCTYPE X3D PUBLIC "ISO//web3d//DTD X3D 3.1//EN" "http://www.web3d.org/specifications/x3d-3.1.dtd">
  <X3D profile='Immersive' version='3.1' xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance' xmlns:nav='http://www.web3d.org/specifications/x3d-3.1.xsd'>
    <head>
      <meta content='ScalarInterpolatorExample.x3d' name='title'/>
      <meta content='Demonstrate use of ScalarInterpolator to animate transparency.' name='description'/>
      <meta content='Don Brutzman' name='creator'/>
      <meta content='28 January 2008' name='created'/>
      <meta content='28 January 2008' name='modified'/>
      <meta content='http://X3DGraphics.com' name='reference'/>
      <meta content='Copyright Don Brutzman and Leonard Daly 2007' name='rights'/>
      <meta content='X3D book, X3D graphics, X3D-Edit, http://www.x3dgraphics.com' name='subject'/>
      <meta content='X3D-Edit, https://savage.nps.edu/X3D-Edit' name='generator'/>
      <meta content='license.html' name='license'/>
    </head>
    <Scene>
      <Shape>
        <Sphere radius='2'/>
        <Appearance>
          <Material DEF='SphereMaterial' diffuseColor='0.941176 0.027451 0' transparency='0'/>
        </Appearance>
      </Shape>
      <TimeSensor DEF='AnimationClock' cycleInterval='5' loop='true'/>
      <!-- note that final value equals first value in keyValue array in order to support smooth looping -->
      <ScalarInterpolator DEF='TransparencyAnimator' Key='0 0.5 1' keyValue='0 1 0'/>
      <ROUTE fromField='fraction_changed' fromNode='AnimationClock' toField='set_fraction' toNode='TransparencyAnimator'/>
      <ROUTE fromField='value_changed' fromNode='TransparencyAnimator' toField='set_transparency' toNode='SphereMaterial'/>
      <!-- notice that Text appears later in scene although it is located above Sphere -->
      <Transform translation='0 1.5 0'/>
      <Shape>
        <Text string='Animating transparency using ScalarInterpolator'/>
        <FontStyle justify="MIDDLE" MIDDLE/>  
      </Text>
    </Shape>
    <Shape>
      <Text string='Animating transparency using ScalarInterpolator'/>
      <FontStyle justify="MIDDLE" MIDDLE/>  
    </Text>
  </Scene>
</X3D>
**ScalarInterpolator**

ScalarInterpolator generates piecewise-linear values that can be ROUTEd to other Float attributes. Typical input: ROUTe someTimeSensor.fraction_changed TO someInterpolator.set_fraction Typical output: ROUTe someInterpolator.value_changed TO destinationNode.set_attribute.

### DEF

[DEF ID #IMPLIED]
DEF defines a unique ID name for this node, referencable by other nodes.

**Hint:** descriptive DEF names improve clarity and help document a model.

### USE

[USE IDREF #IMPLIED]
USE means reuse an already DEF-ed node ID, ignoring _all_ other attributes and children.

**Hint:** USEing other geometry (instead of duplicating nodes) can improve performance.

**Warning:** do NOT include DEF (or any other attribute values) when using a USE attribute!

### key

[key: accessType inputOutput, type MFFloat CDATA #IMPLIED]
Definition parameters for linear-interpolation function time intervals, in increasing order and corresponding to keyValue.

**Hint:** number of keys must match number of keyValue!

### keyValue

[keyValue: accessType inputOutput, type MFFloat CDATA #IMPLIED]
Output values for linear interpolation, each corresponding to time-fraction keys.

**Hint:** number of keys must match number of keyValue!

### set_fraction

[set_fraction: inputOnly type SFFloat CDATA #FIXED "" ]
set_fraction selects input key for corresponding keyValue output.

### value_changed

[value_changed: accessType outputOnly, type SFFloat CDATA #FIXED "" ]
Linearly interpolated output value determined by current key time and corresponding keyValue pair.

### containerField

[containerField: NMTOKEN "children"]
containerField is the field-label prefix indicating relationship to parent node. Examples: geometry Box, children Group, proxy Shape. containerField attribute is only supported in XML encoding of X3D scenes.

### class

[class CDATA #IMPLIED]
class is a space-separated list of classes, reserved for use by XML stylesheets. class attribute is only supported in XML encoding of X3D scenes.
ColorInterpolatoator node

Generates a 3-tuple (triple-valued) SFColor for continuous \textit{value\_changed} output

\textit{key} array contains SFFloat values

\textit{keyValue} array contains SFColor values

Linear interpolation of red, green, blue (RGB) values is respectively performed for each bracketing \textit{keyValue} pair

\begin{verbatim}
key='0 0.3333 0.666 1'
keyValue='1 0 0, 0 1 0, 0 0 1, 1 0 0'
\end{verbatim}
ColorInterpolator animation chain

Each node's output field matches data type of next node's input field

accessType outputOnly to inputOnly, initializeOnly also match

Touch Sensor → Time Sensor → Color Interpolator → Material

- **TextTriggerTouchSensor**
  - output: touchTime

- **AnimationClock**
  - input: startTime
  - output: fraction_changed

- **ColorChanger**
  - input: set_fraction
  - output: value_changed

- **SphereMaterial**
  - input: diffuseColor
ColorInterpolator example output

Using the pointing device to select the text triggers the ColorInterpolator animation

• Colors vary gradually, by linear interpolation of each of the component red-green-blue RGB values
ColorInterpolator scene graph illustration

Scene
  Group
    Place triggering text above sphere of interest
    Transform: translation: 0 2 0
    Shape
      Text: string: "Touch text to" "start animation..."
        FontStyle: justify: "MIDDLE" "MIDDLE"
      TouchSensor: DEF: TextTriggerTouchSensor, description: Touch text to start...
      Here is Sphere with accompanying Material that will get animated
      Transform: translation: 0 -1 0
      Shape
        Sphere
        Appearance
          SphereMaterial diffuseColor gets overridden by interpolator output
          Material: DEF: SphereMaterial, diffuseColor: 0.5 0.5 0.5
          TimeSensor is triggered to start by TouchSensor, then sends animating values to Interpolator
          TimeSensor: DEF: AnimationClock, cycleInterval: 6, loop: false
          ROUTE 1: TouchSensor trigger to TimeSensor clock
          ROUTE: fromNode: TextTriggerTouchSensor, fromField: touchTime, toNode: AnimationClock, toField: startTime
          Interpolator: ColorChanger interpolates evenly between red, green, blue and then back to red
          ColorInterpolator: DEF: ColorChanger, key: 0 0.3333 0.6666, keyValue: 1 0 0, 0 1 0, 0 0 1, 1 0 0
          ROUTE 2: the ColorChanger interpolator gets stimulated by AnimationClock TimeSensor fraction to compute a color value
          ROUTE: fromNode: AnimationClock, fromField: fraction_changed, toNode: ColorChanger, toField: set_fraction
          ROUTE 3: Interpolator output is sent to target node of interest. Changed color value is routed to SphereMaterial color.
          ROUTE: fromNode: ColorChanger, fromField: value_changed, toNode: SphereMaterial, toField: diffuseColor
ColorInterpolatorator scene graph with ROUTEs

Suggested exercise:
label each link with field names, types, and accessType.
<Group>
  <!-- Place triggering text above sphere of interest -->
  <Transform translation="0 2 0">
    <Shape>
      <Text string="Touch text to " startAnimation="..." />
        <FontStyle justify="MIDDLE" middle/>
      </Text>
    </Shape>
  </Transform>
  <!-- This TouchSensor only reacts to user clicking on the sibling Shape and Text, because it is under a parent Transform grouping node -->
  <TextTriggerTouchSensor ROUTE: [from touchTime to AnimationClock startTime] -->
    <TouchSensor DEF='TextTriggerTouchSensor' description="Touch text to start..."/>
  </TextTriggerTouchSensor>
  <!-- Here is Sphere with accompanying Material that will get animated -->
  <Transform translation="0 -1 0">
    <Shape>
      <Sphere/>
      <Appearance>
        <!-- SphereMaterial diffuseColor gets overridden by interpolator output -->
        <SphereMaterial ROUTE: [from ColorChanger.value changed to diffuseColor] -->
          <Material DEF='SphereMaterial' diffuseColor="0.5 0.5 0.5"/>
        </Material>
      </Appearance>
    </Shape>
  </Transform>
  <!-- TimeSensor is triggered to start by TouchSensor, then sends animating values to Interpolator -->
  <TimeSensor ROUTE: [from TextTriggerTouchSensor touchTime to AnimationClock startTime] [from fraction changed to ColorChanger set_fraction] -->
    <animationClock DEF='AnimationClock' cycleInterval="6"/>
    <!-- ROUTE 1: TouchSensor trigger to TimeSensor clock -->
    <ROUTE fromNode='TextTriggerTouchSensor' fromField='touchTime' toNode='AnimationClock' toField='startTime'/>
    <!-- Interpolator: ColorChanger interpolates evenly between red, green, blue and then back to red -->
    <ROUTE fromNode='TextTriggerTouchSensor' fromField='fraction changed' toNode='ColorChanger' toField='set_fraction'/>
  </TimeSensor>
  <!-- ColorChanger ROUTEs: [from AnimationClock fraction changed to set_fraction] [from value changed to SphereMaterial diffuseColor] -->
    <ColorChanger ROUTE: [from AnimationClock fraction changed to set_fraction] [from value changed to SphereMaterial diffuseColor] -->
      <ColorInterpolator DEF='ColorChanger' key='0 0.3333 0.6666 1' keyValue='1 0 0 1 0 0 0 1 1 0 0'/>
    </ROUTE>
    <!-- ROUTE 2: the ColorChanger interpolator gets stimulated by AnimationClock TimeSensor fraction to compute a color value -->
    <ROUTE fromNode='AnimationClock' fromField='fraction_changed' toNode='ColorChanger' toField='set_fraction'/>
    <!-- ROUTE 3: Interpolator output is sent to target node of interest. Changed color value is routed to SphereMaterial color. -->
    <ROUTE fromNode='ColorChanger' fromField='value changed' toNode='SphereMaterial' toField='diffuseColor'/>
  </ColorChanger>
</Group>
<Group>
  <!-- Place triggering text above sphere of interest -->
  <Transform translation="0 2 0">
    <Shape>
      <Text string="Touch text to " startAnimation...">
        <FontStyle justify="MIDDLE" middle/>
      </Text>
    </Shape>
    <!-- This TouchSensor only reacts to user clicking on the sibling Shape and Text, because it is under a parent Transform grouping node -->
    <!-- TextTriggerTouchSensor ROUTE: [from touchTime to AnimationClock startTime] -->
    <TouchSensor DEF="TextTriggerTouchSensor" description="Touch text to start..."/>
  </Transform>
  <!-- Here is Sphere with accompanying Material that will get animated -->
  <Transform translation="0 -1 0">
    <Shape>
      <Sphere/>
      <Appearance>
        <!-- SphereMaterial diffuseColor gets overridden by interpolator output -->
        <!-- SphereMaterial ROUTE: [from ColorChanger.value_changed to diffuseColor] -->
        <Material DEF="SphereMaterial" diffuseColor="0.5 0.5 0.5"/>
      </Appearance>
      <!-- TimeSensor is triggered to start by TouchSensor, then sends animating values to Interpolator -->
      <!-- TimeSensor ROUTEs: [from TextTriggerTouchSensor touchTime to startTime] [from fraction_changed to ColorChanger set_fraction] -->
      <TimeSensor DEF="AnimationClock" cycleInterval="6"/>
      <!-- ROUTE 1: TouchSensor trigger to TimeSensor clock -->
      <ROUTE fromNode="TextTriggerTouchSensor" fromField="touchTime" toNode="AnimationClock" toField="startTime"/>
      <!-- Interpolator: ColorChanger interpolates evenly between red, green, blue and then back to red -->
      <!-- ColorChanger ROUTEs: [from AnimationClock fraction_changed to set_fraction] [from value_changed to SphereMaterial diffuseColor] -->
      <Interpolator DEF="ColorChanger" key="0 0.3333 0.6666 1" keyValue="1 0 0 1 0 0 0 1 1 0 0 0"/>
      <!-- ROUTE 2: the ColorChanger interpolator gets stimulated by AnimationClock TimeSensor fraction to compute a color value -->
      <ROUTE fromNode="AnimationClock" fromField="fraction_changed" toNode="ColorChanger" toField="set_fraction"/>
      <!-- ROUTE 3: Interpolator output is sent to target node of interest. Changed color value is routed to SphereMaterial color. -->
      <ROUTE fromNode="ColorChanger" fromField="value_changed" toNode="SphereMaterial" toField="diffuseColor"/>
    </Shape>
  </Transform>
</Group>
<xml version="1.0" encoding="UTF-8">
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1//EN" "http://www.web3d.org/specifications/x3d-3.1.dtd">  
<head>  
<meta content="ColorInterpolatorExample.x3d" name="filename"/>  
<meta content="Demonstrate basic design pattern for animating a node." name="description"/>  
<meta content="Don Brutzman" name="creator"/>  
<meta content="27 April 2005" name="created"/>  
<meta content="27 January 2008" name="modified"/>  
<meta content="ColorInterpolatorExampleSceneGraphWithRoute.png" name="drawing"/>  
<meta content="ColorInterpolatorExample.png" name="image"/>  
<meta content="Animation ColorInterpolator" name="keywords"/>  
<meta content="&dquo;http://X3DRelated.com/examples/X3DForWebAuthors/Chapter07-EventAnimationInterpolation/ColorInterpolatorExample.x3d"" name="generator"/>  
<meta content="//.../licence.html" name="license"/>  
</head>  
<Scene>  
<!-- Place triggering text above sphere of interest -->  
<Transform translation='0 2 0'>  
  <Shape>  
    <Text string='"Touch text to " start animation..."'/>  
    <FontStyle justifying='"MIDDLE"'/>  
  </Text>  
  <Shape>  
    <TouchSensor DEF='TextTriggerTouchSensor' description='Touch text to start...'/>  
  </Shape>  
</Transform>  
<!-- Here is Sphere with accompanying Material that will get animated -->  
<Transform translation='0 -1 0'>  
  <Shape>  
    <Appearance>  
      <Material DEF='SphereMaterial' diffuseColor='0.3333 0.6666 1'/>  
    </Appearance>  
  </Shape>  
</Transform>  
<!-- TimeSensor is triggered to start by TouchSensor, then sends animating values to Interpolator -->  
<TimeSensor DEF='AnimationClock' cycleInterval='0' loop='false'/>  
  <ROUTE fromField='touchTime' fromNode='TextTriggerTouchSensor' toField='startTime' toNode='AnimationClock'/>  
  <ROUTE fromField='timeFraction' fromNode='AnimationClock' toField='set_fraction' toNode='ColorInterpolator'/>  
  <ROUTE fromField='value_changed' fromNode='ColorInterpolator' toField='ColorChanger' toNode='SphereMaterial'/>  
</Scene>  
</X3D>
# ColorInterpolator

ColorInterpolator generates a range of Color values that can be ROUTEd to a `<Color>` node's color attribute. Typical input: ROUTE someTimeSensor.fraction_changed TO someInterpolator.set_fraction. Typical output: ROUTE someInterpolator.value_changed TO destinationNode.set_attribute.

## DEF

[DEF ID #IMPLIED]
DEF defines a unique ID name for this node, referencable by other nodes.

*Hint:* descriptive DEF names improve clarity and help document a model.

## USE

[USE IDREF #IMPLIED]
USE means reuse an already DEF-ed node ID, ignoring _all_ other attributes and children.

*Hint:* USEing other geometry (instead of duplicating nodes) can improve performance.

*Warning:* do NOT include DEF (or any other attribute values) when using a USE attribute!

## key

[key: accessType inputOutput, type MFFloat CDATA #IMPLIED]
Definition parameters for linear-interpolation function time intervals, in increasing order and corresponding to keyValues.

*Hint:* number of keys must match number of keyValues!

## keyValue

[keyValue: accessType inputOutput, type MFColor CDATA #IMPLIED]
Output values for linear interpolation, each corresponding to time-fraction keys.

*Hint:* number of keys must match number of keyValues!

## set_fraction

[set_fraction: accessType inputOnly, type SFFloat CDATA #FIXED ""]
set_fraction selects input key for corresponding keyValue output.

## value_changed

[value_changed: accessType outputOnly, type SFColor CDATA #FIXED ""]
Linearly interpolated output value determined by current key time and corresponding keyValue pair.

## containerField

[containerField: NMTOKEN "children"]
containerField is the field-label prefix indicating relationship to parent node. Examples: geometry Box, children Group, proxy Shape. containerField attribute is only supported in XML encoding of X3D scenes.

## class

[class CDATA #IMPLIED]
class is a space-separated list of classes, reserved for use by XML stylesheets. class attribute is only supported in XML encoding of X3D scenes.
OrientationInterpolator node

Generates a 4-tuple (four-valued orientation) SFRotation for `value_changed` output

- `key` array contains SFFloat fraction values
- `keyValue` array contains SFRotation output values
  - As always: same number of `key`, `keyValue` entries

OrientationInterpolator animates along shortest path between the two normal vectors, also computes linear average between two corresponding angles, in `keyValue` array
OrientationInterpolator example

This animation-chain example can be added to any scene (via cut and paste) to create a look-around Viewpoint. This bound camera view rotates about a fixed position.

```xml
<Viewpoint DEF='DizzyViewpoint' description='Rotating viewpoint' position='[somewhere you want it]' orientation='0 1 0 0'/>
<OrientationInterpolator DEF='Spinner' key='0 0.25 0.5 0.75 1'
  keyValue='0 1 0 0, 0 1 0 1.57, 0 1 0 3.14, 0 1 0 4.71, 0 1 0 6.28'/>
<TimeSensor DEF='SpinClock' cycleInterval='12' loop='true'/>
<ROUTE fromField='fraction_changed' fromNode='SpinClock'
  toField='set_fraction' toNode='Spinner'/>
<ROUTE fromField='value_changed' fromNode='Spinner'
  toField='orientation' toNode='DizzyViewpoint'/>
```
<X3D version="1.0" encoding="UTF-8">
    <head>
        <meta content="PositionOrientationInterpolatorsExample.x3d" name="title"/>
        <meta content="Demonstrate use of PositionInterpolator and OrientationInterpolator to animate object motion." name="description"/>
        <meta content="Don Brutzman" name="creator"/>
        <meta content="19 January 2008" name="created"/>
        <meta content="20 January 2008" name="modified"/>
        <meta content="http://X3DGraphics.com" name="reference"/>
        <meta content="http://www.webidl.org/x3d/examples/help.html" name="reference"/>
        <meta content="Copyright Don Brutzman and Leonard Daly 2007" name="rights"/>
        <meta content="X3D book, X3D graphics, X3D-Edit, http://www.x3dgraphics.com" name="subject"/>
        <meta content="X3D-Edit, https://savage.nps.edu/X3D-Edit" name="generator"/>
        <meta content="../license.html" name="license"/>
    </head>

    <Scene description="Animation Demo: orientation=1 0 1 -0.2 position=0 4 10"/>

    <Transform DEF='Animation'>
        <Rotation rotation='1 0 0 1.57'>
            <Shape>
                <Cube bottomRadius='0.5' height='1.5'>
                    <Appearance>
                        <Material DEF='ConeMaterial' diffuseColor='0.427451 1 0.167784'/>
                    </Appearance>
                </Shape>
            </Rotation>
        </Shape>
    </Transform>

    <Transform DEF='Floor'>
        <Box size='10 0.05 10'>
            <Appearance>
                <Material diffuseColor='0 0.262745 0.941176'/>
            </Appearance>
        </Box>
    </Transform>

    <TimeSensor DEF='AnimationClock' cycleInterval='10' loop='true'/>
</X3D>
**OrientationInterpolator** generates a series of rotation values. Results can be ROUTEd to a `<Transform>` node's 'rotation' attribute or another `Rotations` attribute.

Typical input: ROUTE someTimeSensor.fraction_changed TO someInterpolator.set_fraction

Typical output: ROUTE someInterpolator.value_changed TO destinationNode.set_attribute.

**DEF**

[DEF ID #IMPLIED]
DEF defines a unique ID name for this node, referencable by other nodes.

**Hint:** descriptive DEF names improve clarity and help document a model.

**USE**

[USE IDREF #IMPLIED]
USE means reuse an already DEF-ed node ID, ignoring _all_ other attributes and children.

**Hint:** USEing other geometry (instead of duplicating nodes) can improve performance.

**Warning:** do NOT include DEF (or any other attribute values) when using a USE attribute!

**key**

[key: accessType inputOutput, type MFFloat CDATA #IMPLIED]
Definition parameters for linear-interpolation function time intervals, in increasing order and corresponding to keyValues.

**Hint:** number of keys must match number of keyValues!

**keyValue**

[keyValue: accessType inputOutput, type MFRotation CDATA #IMPLIED]
Output values for linear interpolation, each corresponding to time-fraction keys.

**Hint:** number of keys must match number of keyValues!

**set_fraction**

[set_fraction: inputOnly type SFFloat CDATA #FIXED '"
set_fraction selects input key for corresponding keyValue output.

**value_changed**

[value_changed: accessType outputOnly, type SFRotation CDATA #FIXED '"
Linearly interpolated output value determined by current key time and corresponding keyValue pair.

**containerField**

[containerField: NMTOKEN "children"]
containerField is the field-label prefix indicating relationship to parent node. Examples: geometry Box, children Group, proxy Shape. containerField attribute is only supported in XML encoding of X3D scenes.

**class**

[class CDATA #IMPLIED]
class is a space-separated list of classes, reserved for use by XML stylesheets. class attribute is only supported in XML encoding of X3D scenes.
PositionInterpolator node

Generates a 3-tuple (three-valued floating point) SFVec3f for value_changed output

- key array contains SFFloat fraction values
- keyValue array contains SFVec3f output values
  - As always: same number of key, keyValue entries

PositionInterpolator computes weighted average between corresponding x, y and z pairs in the keyValue array
  - ROUTE to Transform, either translation or scale
<X3D profile="Immersive" version="3.1" xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance">
  <head>
    <meta content='PositionOrientationInterpolatorsExample.x3d' name='title'/>
    <meta content='Demonstrate use of PositionInterpolator and OrientationInterpolator to animate object motion.' name='description'/>
    <meta content='Don Brutzman' name='creator'/>
    <meta content='29 January 2006' name='created'/>
    <meta content='29 January 2006' name='modified'/>
    <meta content='http://X3DGraphics.com' name='reference'/>
    <meta content='http://www.web3d.org/x3d/content/examples/help.html' name='reference'/>
    <meta content='Copyright Don Brutzman and Leonard Baily 2007' name='rights'/>
    <meta content='X3D-Edit, https://savage.nps.edu/X3D-Edit' name='generator'/>
  </head>
  <Scene description='Animation demo' position='0 4 13' orientation='1 0 1 0.2'>
    <Transform translation='1 0 1' DEF='Pointer'>
      <Transform rotation='1 0 0 1.57'>
        <Shape box='0.5 0.5 0.5' height='1.5'>
          <Appearance>
            <Material DIFF='ConeMaterial' diffuseColor='0.42745 1 0.160784'/>
          </Appearance>
        </Shape>
      </Transform>
    </Transform>
    <Shape DEF='Floor'>
      <Appearance>
        <Material DIFF='BoxMaterial' diffuseColor='0.26575 0.94175'/>
      </Appearance>
    </Shape>
    <TimeSensor DEF='AnimationClock' cycleInterval='10' loop='true'>
      <!-- note that final value equals first value in keyValue array in order to support smooth looping -->
      <!-- first drive around the location -->
      <ROUTE fromField='fraction_changed' fromNode='AnimationClock' toField='set_fraction' toNode='PositionInterpolator'/>
      <ROUTE fromField='value_changed' fromNode='PositionAnimator' toField='set_translation' toNode='Pointer'/>
      <ROUTE fromField='value_changed' fromNode='OrientationAnimator' toField='set_rotation' toNode='Pointer'/>
      <!-- notice that explanatory Text appears later in scene although it is located above driving plane -->
      <Transform translation='0 3.5 0'>
        <Text string='Animation using PositionInterpolator and OrientationInterpolator' ...
          <FontStyle justify='MIDDLE' size='7'/>
        </Text>
      </Transform>
    </TimeSensor>
  </Scene>
</X3D>
Event tracing

X3D-Edit author-assist feature provides support for tracing output events

- “Trace” checkbox on editing pane adds some extra X3D source to your scene
- Captures all output events, routes them to a Script
- Script outputs event values to X3D browser console at run time so that you can trace execution logic

Can be helpful for selective debugging when animation chains are not behaving as expected

- Available for ROUTE, most event-producing nodes
- Simply remove when done troubleshooting
Event tracing example

PositionOrientationInterpolatorsExampleTraced.x3d
PositionInterpolator generates a series of triplet values. Results can be ROUTEd to a <Transform> node's 'translation' attribute or another Vector3f attribute. Typical input: ROUTE someTimeSensor.fraction_changed TO someInterpolator.set_fraction Typical output: ROUTE someInterpolator.value_changed TO destinationNode.set_attribute.

<table>
<thead>
<tr>
<th>DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>[DEF ID #IMPLIED]</td>
</tr>
<tr>
<td>DEF defines a unique ID name for this node, referencable by other nodes.</td>
</tr>
<tr>
<td>Hint: descriptive DEF names improve clarity and help document a model.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>[USE IDREF #IMPLIED]</td>
</tr>
<tr>
<td>USE means reuse an already DEF-ed node ID, ignoring <em>all</em> other attributes and children.</td>
</tr>
<tr>
<td>Hint: USEing other geometry (instead of duplicating nodes) can improve performance.</td>
</tr>
<tr>
<td>Warning: do NOT include DEF (or any other attribute values) when using a USE attribute!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>[key: accessType inputOutput, type MFFloat CDATA #IMPLIED]</td>
</tr>
<tr>
<td>Definition parameters for linear-interpolation function time intervals, in increasing order and corresponding to keyValue values.</td>
</tr>
<tr>
<td>Hint: number of keys must match number of keyValue values!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keyValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>[keyValue: accessType inputOutput, type MFVec3f CDATA #IMPLIED]</td>
</tr>
<tr>
<td>Output values for linear interpolation, each corresponding to time-fraction keys.</td>
</tr>
<tr>
<td>Hint: number of keys must match number of keyValue values!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>set_fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[set_fraction: inputOnly type SFFloat CDATA #FIXED &quot;&quot;]</td>
</tr>
<tr>
<td>set_fraction selects input key for corresponding keyValue output.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>value_changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>[value_changed: accessType outputOnly, type SFVec3f CDATA #FIXED &quot;&quot;];</td>
</tr>
<tr>
<td>Linearly interpolated output value determined by current key time and corresponding keyValue pair.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>containerField</th>
</tr>
</thead>
<tbody>
<tr>
<td>[containerField: NMTOKEN &quot;children&quot;]</td>
</tr>
<tr>
<td>containerField is the field-label prefix indicating relationship to parent node. Examples: geometry Box, children Group, proxy Shape. containerField attribute is only supported in XML encoding of X3D scenes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>[class CDATA #IMPLIED]</td>
</tr>
<tr>
<td>class is a space-separated list of classes, reserved for use by XML stylesheets. class attribute is only supported in XML encoding of X3D scenes.</td>
</tr>
</tbody>
</table>
PositionInterpolator2D node

Generates a 2-tuple (two-valued floating point) SFVec2f for `value_changed` output

- `key` array contains SFFloat `fraction` values
- `keyValue` array contains SFVec2f output values
  - As always: same number of `key, keyValue` entries

PositionInterpolator2D computes weighted average between corresponding (x, y) pairs in the `keyValue` array
<xml version="1.0">
<!DOCTYPE X3D PUBLIC "ISO/Web3D/DTD X3D 3.1/EN" "http://www.web3d.org/specifications/x3d-3.1.dtd">
<X3D profile='Immersive' version='3.1' xmlns:x3d='http://www.w3.org/2001/XMLSchema-instance'>
  <x3d:namespaceSchemaLocation 'http://www.web3d.org/specifications/x3d-3.1.xsd'/>

  <head>
    <meta content='PositionInterpolator2DExample.x3d' name='name'/>
    <meta content='PositionInterpolator2D Example to interpolate using PositionInterpolator2D - click geometry to activate animation loop.' name='description'/>
    <meta content='Don Brutzman' name='creator'/>
    <meta content='16 October 2001' name='created'/>
    <meta content='PositionInterpolator2D' name='subject'/>
    <meta content='http://X3DGraphics.com/examples/X3DForWebAuthors/Chapter07-EventAnimationInterpolation/PositionInterpolator2DExample.x3d' name='identifier'/>
    <meta content='http://www.web3d.org/x3d/content/examples/x3d/Basic/development/PositionInterpolator2D Example.x3d' name='reference'/>
    <meta content='X3D-Edit, http://www.web3d.org/x3d/content/README.X3D-Edit.html' name='generator'/>
    <meta content='.' name='license.html'/>
  </head>

  <Scene description='Click to activate animation' position='0 0 3'/>
  <TimeSensor DEF='Clock' cycleInterval='10' enabled='false' loop='true'/>
  <PositionInterpolator2D DEF='InterpolateTScale' key='0 0.35 0.45 0.8 0.9 1' keyValue='1.0 1.0 1.0 3 3 3 0.8 0.8 0.8 1.0 1.0 1.0 1.0 1.0'/>
  <ROUTE fromField='fraction_changed' fromNode='Clock' toField='set_fraction' toNode='InterpolateTScale'/>
  <Transform DEF='ImageAspectRatio' scale='1.5 1 1'/>
  <TouchSensor DEF='Toucher' description='click and hold to animate TextureTransform'/>
  <ROUTE fromField='isActive' fromNode='Toucher' toField='enabled' toNode='Clock'/>
  <Shape>
    <IndexedFaceSet coordIndex='0 1 2 3 0 -1' solid='true'/>
    <Coordinate DEF='TwoByTwoSquare' point='-1 -1 0 1 0 0 0 0 0 1 1 0 -1 -1 0'/>
  </IndexedFaceSet>
  <Appearance>
    <ImageTexture DEF='ContactImage' url='http://X3DGraphics.com/examples/X3DForWebAuthors/Chapter07-EventAnimationInterpolation/InvisibleObjectForTheCamera.jpg'/>
    <TextureTransform DEF='TextureTransformNode'/>
  </Appearance>
  </Shape>
  <ROUTE fromField='value_changed' fromNode='InterpolateTScale' toField='set_scale' toNode='TextureTransformNode'/>
</Scene>
</X3D>
Selecting the texture with the mouse pointer starts the TextureTransform *scale* animation, deselecting the texture stops the animation.
PositionInterpolator2D generates a series of Vector2Float values that can be ROUTEd to a Vector2Float attribute. Typical input: ROUTE someTimeSensor.fraction_changed TO someInterpolator.set_fraction. Typical output: ROUTE someInterpolator.value_changed TO destinationNode.set_attribute.

<table>
<thead>
<tr>
<th>DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>[DEF ID #IMPLIED]</td>
</tr>
<tr>
<td>DEF defines a unique ID name for this node, referencable by other nodes.</td>
</tr>
<tr>
<td>Hint: descriptive DEF names improve clarity and help document a model.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>[USE IDREF #IMPLIED]</td>
</tr>
<tr>
<td>USE means reuse an already DEF-ed node ID, ignoring <em>all</em> other attributes and children.</td>
</tr>
<tr>
<td>Hint: USEng other geometry (instead of duplicating nodes) can improve performance.</td>
</tr>
<tr>
<td>Warning: do NOT include DEF (or any other attribute values) when using a USE attribute!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>[key: accessType inputOutput, type MFFloat CDATA #IMPLIED]</td>
</tr>
<tr>
<td>Definition parameters for linear-interpolation function time intervals, in increasing order and corresponding to keyValue.</td>
</tr>
<tr>
<td>Hint: number of keyValue must be an integer multiple of the number of keys!</td>
</tr>
<tr>
<td>Hint: keyValue/key integer multiple defines how many coordinates are sent in value_changed outputOnly.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keyValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>[keyValue: accessType inputOutput, type MFVec2f CDATA #IMPLIED]</td>
</tr>
<tr>
<td>Output values for linear interpolation, each corresponding to time-fraction keys.</td>
</tr>
<tr>
<td>Hint: number of keyValue must be an integer multiple of the number of keys!</td>
</tr>
<tr>
<td>Hint: keyValue/key integer multiple defines how many coordinates are sent in value_changed outputOnly.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>set_fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[set_fraction: inputOnly type SFFloat CDATA #FIXED ]</td>
</tr>
<tr>
<td>set_fraction selects input key for corresponding keyValue output.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>value_changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>[value_changed: accessType outputOnly, type SFVec2f CDATA #FIXED ]</td>
</tr>
<tr>
<td>Linearly interpolated output value determined by current key time and corresponding keyValue pair.</td>
</tr>
<tr>
<td>Hint: keyValue/key integer multiple defines how many coordinates are sent in value_changed outputOnly.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>containerField</th>
</tr>
</thead>
<tbody>
<tr>
<td>[containerField: NMTOKEN &quot;children&quot;]</td>
</tr>
<tr>
<td>containerField is the field-label prefix indicating relationship to parent node. Examples: geometry Box, children Group, proxy Shape. containerField attribute is only supported in XML encoding of X3D scenes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>[class CDATA #IMPLIED]</td>
</tr>
<tr>
<td>class is a space-separated list of classes, reserved for use by XML stylesheets. class attribute is only supported in XML encoding of X3D scenes.</td>
</tr>
</tbody>
</table>
NormalInterpolator node

Generates a 3-tuple (three-valued floating point) SFVec3f for \textit{value_changed} output

\textit{key} array contains SFFloat values

\textit{keyValue} array contains SFVec3f values

- As always: same number of \textit{key}, \textit{keyValue} entries
- SFVec3f outputs: unit-normal vectors, magnitude=1

NormalInterpolator animates along shortest path between the pair of normal vectors currently being referenced in \textit{keyValue} array

Normal vectors used for special shading effects

- see Chapter13 - Geometry Triangles Quadrilaterals
NormalInterpolator editor

Rows correspond to each key
Correct results have same number of MFVec3f values in each row
Normal-to-normal interpolation occurs on each column, allowing easier comparison
NormalInterpolator generates a series of normal (perpendicular) vector sets along the surface of a unit sphere. ROUTE values to vector attribute of a `<Normal>` node or another `Vector3FloatArray` attribute.

**Typical input:**
```
ROUTE someTimeSensor.fraction_changed TO someInterpolator.set_fraction
```

**Typical output:**
```
ROUTE someInterpolator.value_changed TO destinationNode.set_attribute.
```

<table>
<thead>
<tr>
<th>DEF</th>
<th>[DEF ID #IMPLIED]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF defines a unique ID name for this node, referencable by other nodes.</td>
<td></td>
</tr>
<tr>
<td>Hint: descriptive DEF names improve clarity and help document a model.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE</th>
<th>[USE IDREF #IMPLIED]</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE means reuse an already DEF-ed node ID, ignoring <em>all</em> other attributes and children.</td>
<td></td>
</tr>
<tr>
<td>Hint: USEing other geometry (instead of duplicating nodes) can improve performance.</td>
<td></td>
</tr>
<tr>
<td>Warning: do NOT include DEF (or any other attribute values) when using a USE attribute!</td>
<td></td>
</tr>
</tbody>
</table>

### key
- **key**: `accessType inputOutput`, `type MFFloat CDATA #IMPLIED`
- Definition parameters for linear-interpolation function time intervals, in increasing order and corresponding to `keyValues`.
- **Hint**: number of keys must match number of `keyValues`!

### keyValue
- **keyValue**: `accessType inputOutput`, `type MFVec3f CDATA #IMPLIED`
- Output values for linear interpolation, each corresponding to time-fraction keys.
- **Hint**: number of keys must match number of `keyValues`!

### set_fraction
- **set_fraction**: `inputOnly type SFFloat CDATA #FIXED ""`
- `set_fraction` selects input key for corresponding `keyValue` output.

### value_changed
- **value_changed**: `accessType outputOnly`, `type MFVec3f CDATA #FIXED ""`
- Linearly interpolated output value determined by current key time and corresponding `keyValue` pair.

### containerField
- **containerField**: `@NMToken "children"`
- `containerField` is the field-label prefix indicating relationship to parent node. Examples: geometry Box, children Group, proxy Shape. `containerField` attribute is only supported in XML encoding of X3D scenes.

### class
- **class**: `CDATA #IMPLIED`
- `class` is a space-separated list of classes, reserved for use by XML stylesheets. `class` attribute is only supported in XML encoding of X3D scenes.
CoordinateInterpolator2D node

Generates 2-tuple (two-valued floating point) array, MFVec2f for `value_changed` output

`key` array contains SFFloat values

`keyValue` array contains MFVec2f values

- As always: same number of `key`, `keyValue` entries
- Counting is very important for arrays of arrays!

CoordinateInterpolator2D computes weighted average between corresponding x and y pairs for each subarray in the `keyValue` array.
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoordinateInterpolator2D</td>
<td>Generates a series of Vector2FloatArray values that can be ROUTEd to a Vector2FloatArray attribute. Typical input: ROUTE someTimeSensor.fraction_changed TO someInterpolator.set_fraction. Typical output: ROUTE someInterpolator.value_changed TO destinationNode.set_attribute.</td>
</tr>
</tbody>
</table>

### DEF

- **[DEF ID #IMPLIED]**
  - Defines a unique ID name for this node, referencable by other nodes.
  - **Hint:** Descriptive DEF names improve clarity and help document a model.

### USE

- **[USE IDREF #IMPLIED]**
  - USE means reuse an already DEF-ed node ID, ignoring all other attributes and children.
  - **Hint:** USEing other geometry (instead of duplicating nodes) can improve performance.
  - **Warning:** Do not include DEF (or any other attribute values) when using a USE attribute!

### key

- **[key: accessType inputOutput, type MFFloat CDATA #IMPLIED]**
  - Definition parameters for linear-interpolation function time intervals, in increasing order and corresponding to keyValues.
  - **Hint:** Number of keyValues must be an integer multiple of the number of keys!
  - **Hint:** keyValue/key integer multiple defines how many coordinates are sent in value_changed outputOnlys.

### keyValue

- **[keyValue: accessType inputOutput, type MFVec3f CDATA #IMPLIED]**
  - Output values for linear interpolation, each corresponding to time-fraction keys.
  - **Hint:** Number of keyValues must be an integer multiple of the number of keys!
  - **Hint:** keyValue/key integer multiple defines how many coordinates are sent in value_changed outputOnlys.

### set_fraction

- **[set_fraction: inputOnly type SFFloat CDATA #FIXED ""]**
  - set_fraction selects input key for corresponding keyValue output.

### value_changed

- **[value_changed: accessType outputOnly, type MFVec2f CDATA #FIXED ""]**
  - Linearly interpolated output value determined by current key time and corresponding keyValue pair.
  - **Hint:** keyValue/key integer multiple defines how many coordinates are sent in value_changed outputOnlys.

### containerField

- **[containerField: NMTOKEN "children"]**
  - containerField is the field-label prefix indicating relationship to parent node. Examples: geometry Box, children Group, proxy Shape. containerField attribute is only supported in XML encoding of X3D scenes.

### class

- **[class CDATA #IMPLIED]**
  - class is a space-separated list of classes, reserved for use by XML stylesheets. class attribute is only supported in XML encoding of X3D scenes.
CoordinateInterpolator node

Generates $n$-tuple (multiple-valued floating point) array, MFFloat for `value_changed` output

- `key` array contains $n$ SFFloat values
- `keyValue` array contains $n$ MFFloat values
  - As always: same number of `key`, `keyValue` entries
  - Counting is very important for arrays of arrays!

CoordinateInterpolator computes weighted average between corresponding element pairs for each subarray in the `keyValue` array
Morphing images in 2D

Morphing 2D is the smooth transformation of one image into another using digital in-betweening

• Not a feature built into X3D

Typically this is done by identifying control points and gradually changing individual pixel colors from one 2D image into another

• Initial and final 2D images are already defined
• Corresponding control points are carefully chosen
• Special algorithms are applied
• Now a common photographic editing technique
Morphing models in X3D

Morphing 3D is the smooth transformation of one model into another using digital in-betweening

• Supported in X3D by CoordinateInterpolator node

Typically this is done by identifying control points and gradually changing coordinates from one set of values to another

• Linear interpolation value-by-value for each point
• Also usable for colors, normals, texture coordinates

We will use this technique to morph between 3D coordinate sets using CoordinateInterpolator
Creating a morphable model

1. **Create baseline model geometry**
   - typically an IndexedFaceSet node

2. **Create alternate poses in key positions**
   - Ensure that same geometric layout is maintained
   - These are referred to as “key frames”
   - Can observe proper sequencing via a Switch node

3. **Use each set of point position values as part of a CoordinateInterpolator keyValuePair array**
   - Corresponding key array holds fraction durations, similar to any other interpolator node
CoordinateInterpolator node X3D-Edit

Case-study example: morphing a dolphin model

- Chris Lang, author
- Monterey High School class of 2008
- Models online at

https://savage.nps.edu/Savage/Biologics/Dolphin
Creating a morphable dolphin

1. **Create baseline model geometry**
   - typically an IndexedFaceSet node

Chris Lang first built a dolphin model using Maya
- advanced 3D modeling tool, commercial product
- [http://autodesk.com > Products > Maya](http://autodesk.com)

Such models tend to be complex, perhaps saved in proprietary or perhaps-confusing formats
- But can nevertheless be saved as X3D or VRML, making this approach reasonably repeatable
- Can also use Aaron Bergstrom's Rawkee tool
Creating a morphable dolphin 2

2. Create alternate poses in key positions
   - Ensure that same geometric layout is maintained
   - These are referred to as “key frames”
   - Can observe proper sequencing via a Switch node

The original model was then modified to match other poses, making sure each time that no new coordinate points were added or deleted
   - Each individually saved as X3D or VRML scenes
   - Switch cycler allows direct comparison of each
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d-3.0.dtd">
<X3D profile='Immersive' version='3.0' xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance'
xsd:noNamespaceSchemaLocation='http://www.web3d.org/specifications/x3d-3.0.xsd'>

<Scene>
  <Background groundColor='1 1 1' skyColor='1 1 1'/>
  <Viewpoint description='Dolphin switcher, 2m away' position='0 0 2'/>
  <!-- Modify the whichChoice value in this Switch to 0, 1 or 2 to see the various versions of the model. Select whichChoice=-1 to show nothing. -->
  <Switch DEF='Switch' whichChoice='2'>
    <!-- whichChoice values are 0, 1, 2 for these three children -->
    <Group DEF='CurvedUpwardPose'>
      <Inline url='DolphinPoseO2.x3d' 'https://savage.nps.edu/Savage/Biologics/Dolphin/DolphinPoseO2.x3d'/>
    </Group>
    <Group DEF='NeutralPose'>
      <Inline url='DolphinPoseO1.x3d' 'https://savage.nps.edu/Savage/Biologics/Dolphin/DolphinPoseO1.x3d'/>
    </Group>
    <Group DEF='CurvedDownwardPose'>
      <Inline url='DolphinPoseO3.x3d' 'https://savage.nps.edu/Savage/Biologics/Dolphin/DolphinPoseO3.x3d'/>
    </Group>
  </Switch>
  <IntegerSequencer DEF='Sequencer' key='0 0.25 0.5 0.75 1' keyValue='0 1 2 1 0'/>
  <TimeSensor DEF='Time' cycleInterval='4' enabled='true' loop='true'/>
  <ROUTE fromField='value_changed' fromNode='Sequencer' toField='whichChoice' toNode='Switch'/>
  <ROUTE fromField='fraction_changed' fromNode='Time' toField='set_fraction' toNode='Sequencer'/>
</Scene>
</X3D>
DolphinPose02.x3d  DolphinPose01.x3d  DolphinPose03.x3d
Creating a morphable dolphin

3. Use each set of point position values as part of a CoordinateInterpolator keyValue array

   Corresponding key array holds fraction durations, similar to any other interpolator node

Define pose sequence for CoordinateInterpolator

- 02 CurvedUpwardPose
- 01 NeutralPose
- 03 CurvedDownwardPose
- 01 NeutralPose
- 02 CurvedUpwardPose completes loop, then repeat
CoordinateInterpolator key array lists all five at equal time-fraction intervals:

- key='0 0.25 0.5 0.75 1'

Now need to build keyValue array

- Brute-force approach is then to copy set of array values from each <Coordinate point='...'/> pose

But note that each point sequence is quite long

- 508 points, meaning 1524 x-y-z values for each!!
- 5 arrays hold 7620 points total, 40% duplication
- Prefer 3 arrays, 4572 points, no array duplication
CoordinateInterpolator editor

- Rows correspond to each key
- Correct results have the same number of MFVec3f values in each row
- Point-by-point interpolation occurs on each column, allowing easier comparison
Modifying TimeSensor outputs

Two ways to achieve the same pose sequence

- Five poses: forward 02, 01, 03, 01, 02, repeat
- Three poses: forward 02, 01, 03, backwards, repeat

Build the second approach by modifying the TimeSensor fraction_changed output via a ScalarInterpolator

- TimeSensor output ramps from 0..1 linearly
- Five-pose CoordinateInterpolator uses 0..1 directly
- ScalarInterpolator output ramps 0 to 1 then back to 0
- Three-pose CoordinateInterpolator uses 0..1..0 instead
Modifying TimeSensor outputs

Can modify TimeSensor output by ROUTE connections through a ScalarInterpolator

- Change timing characteristic from 0..1 to 0..1..0

Resulting sequence of five poses:

- forward 02, 01, 03, backwards 03, 01, 02, repeat
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d-3.0.dtd">

<X3D profile=' Immersive ' version='3.0' xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance'
xsd:noNamespaceSchemaLocation='http://www.web3d.org/specifications/x3d-3.0.xsd'>

<head>
<meta content='DolphinMorpher.x3d' name='title'/>
<meta content='Switch among 3 different dolphin poses' name='description'/>
<meta content='Chris Lang' name='creator'/>
<meta content='13 August 2007' name='created'/>
<meta content='13 April 2008' name='modified'/>
<meta content='https://savage.nps.edu/Savage/Biologics/Dolphin/DolphinMorpher.x3d' name='identifier'/>
<meta content='X3D-Edit, https://savage.nps.edu/X3D-Edit' name='generator'/>
<meta content='X3D-Edit, http://www.web3d.org/x3d/content/README.X3D-Edit.html' name='generator'/>
<meta content='..//license.html' name='license'/>
</head>

<Scene>

  <Viewpoint description='Dolphin morpher, 2m away' position='0 0 2'/>
  <Background groundColor='1 1 1' skyColor='1 1 1'/>
  <Transform rotation='0 1 0 1.57' scale='0.1 0.1 0.1' translation='0.12 -0.22 0'>

    <Shape>
      <Appearance>
        <Material diffuseColor='0 0.5 1'/>
      </Appearance>

      <!-- default setup is neutral pose J1. Note that coordIndex values (and thus mesh construction) is identical for all 3 poses -->
      <IndexedFaceSet coordIndex='78 62 145 144 -1 160 82 39 -1 196 161 163 -1 157 164 166 -1 287 173 172 -1 183 298 181 -1 185'>
        <Coordinate DEF='SkinCoordinates' point='0.406 0.431 7.729 0.595 0.574 1.581 -10.422 0.592 0.769 -10.422 1.246 2.580 5.322 1.'/>
      </IndexedFaceSet>
    </Shape>

    <!-- three Coordinate arrays from poses 2, 1, and 3 are pasted into the keyValue array -->

    <CoordinateInterpolator DEF='MorphInterpolator' key='0 0.5 1' keyValue='0.406 1.049 7.905 0.595 2.957 -10.332 0.592 2.263 -10'>
    <ScalarInterpolator DEF='AnimationAdapter' key='0 0.25 0.5 0.75 1' keyValue='0.5 0 0.5 1 0.5'/>
    <TimeSensor DEF='Clock' cycleInterval='1' enabled='true' loop='true'/>
    <ROUTE fromField='value_changed' fromNode='MorphInterpolator' toField='point' toNode='SkinCoordinates'/>
    <ROUTE fromField='value_changed' fromNode='AnimationAdapter' toField='set_fraction' toNode='MorphInterpolator'/>
    <ROUTE fromField='fraction_changed' fromNode='Clock' toField='set_fraction' toNode='AnimationAdapter'/>
  </Transform>

</Scene>
</X3D>
**CoordinateInterpolator** generates a series of Coordinate values that can be ROUTEd to a `<Coordinate>` node's 'point' attribute or another Vector3FloatArray attribute. Typical input: ROUTE someTimeSensor.fraction_changed TO someInterpolator.set_fraction. Typical output: ROUTE someInterpolator.value_changed TO destinationNode.set_attribute.

<table>
<thead>
<tr>
<th>DEF</th>
</tr>
</thead>
<tbody>
<tr>
<td>[DEF ID #IMPLIED]</td>
</tr>
<tr>
<td>DEF defines a unique ID name for this node, referencable by other nodes.</td>
</tr>
<tr>
<td>Hint: descriptive DEF names improve clarity and help document a model.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>[USE IDREF #IMPLIED]</td>
</tr>
<tr>
<td>USE means reuse an already DEF-ed node ID, ignoring all other attributes and children.</td>
</tr>
<tr>
<td>Hint: USEing other geometry (instead of duplicating nodes) can improve performance.</td>
</tr>
<tr>
<td>Warning: do NOT include DEF (or any other attribute values) when using a USE attribute!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
</tr>
</thead>
<tbody>
<tr>
<td>[key: accessType inputOutput, type MFFloat CDATA #IMPLIED]</td>
</tr>
<tr>
<td>Definition parameters for linear-interpolation function time intervals, in increasing order and corresponding to keyValue.</td>
</tr>
<tr>
<td>Hint: number of keyValue values must be an integer multiple of the number of keys!</td>
</tr>
<tr>
<td>Hint: keyValue/key integer multiple defines how many coordinates are sent in value_changed outputOnlys.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>keyValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>[keyValue: accessType inputOutput, type MFVec3f CDATA #IMPLIED]</td>
</tr>
<tr>
<td>Output values for linear interpolation, each corresponding to time-fraction keys.</td>
</tr>
<tr>
<td>Hint: number of keyValue values must be an integer multiple of the number of keys!</td>
</tr>
<tr>
<td>Hint: keyValue/key integer multiple defines how many coordinates are sent in value_changed outputOnlys.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>set_fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[set_fraction: inputOnly type SFFloat CDATA #FIXED '']</td>
</tr>
<tr>
<td>set_fraction selects input key for corresponding keyValue output.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>value_changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>[value_changed: accessType outputOnly, type MFVec3f CDATA #FIXED '']</td>
</tr>
<tr>
<td>Linearly interpolated output value determined by current key time and corresponding keyValue pair.</td>
</tr>
<tr>
<td>Hint: keyValue/key integer multiple defines how many coordinates are sent in value_changed outputOnlys.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>containerField</th>
</tr>
</thead>
<tbody>
<tr>
<td>[containerField: NMTOKEN &quot;children&quot;]</td>
</tr>
<tr>
<td>containerField is the field-label prefix indicating relationship to parent node. Examples: geometry Box, children Group, proxy Shape. containerField attribute is only supported in XML encoding of X3D scenes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>[class CDATA #IMPLIED]</td>
</tr>
<tr>
<td>class is a space-separated list of classes, reserved for use by XML stylesheets. class attribute is only supported in XML encoding of X3D scenes.</td>
</tr>
</tbody>
</table>
Chapter Summary
Chapter Summary: Event Animation

Behaviors, events, ROUTE connections, animation

Animation as scene-graph modification

Event-animation design pattern: 10-step process

Interpolation nodes

- TimeSensor and event timing
- ScalarInterpolator and ColorInterpolator
- OrientationInterpolator, PositionInterpolator, PositionInterpolator2D and NormalInterpolator
- CoordinateInterpolator, CoordinateInterpolator2D
Suggested exercises

Illustrate and annotate ROUTE connections in an animation scene graph (documenting 10 steps)

• Print out one of these scenes in landscape mode, either using the X3dToXhtml.xslt stylesheet version or Netbeans 'Save as HTML' option.
• Then draw all ROUTE connections, label beginning and end of each by name, type and accessType

Draw animation chain diagrams to document behaviors in your own example scenes

• Add use-case summaries about user intent
References
References

**X3D: Extensible 3D Graphics for Web Authors**
by Don Brutzman and Leonard Daly, Morgan Kaufmann Publishers, April 2007, 468 pages.

- Chapter 7, Event Animation and Interpolation
- http://x3dGraphics.com
- http://x3dgraphics.com/examples/X3dForWebAuthors

**X3D Resources**

- http://www.web3d.org/x3d/content/examples/X3dResources.html
References 2

X3D Scene Authoring Hints
• http://x3dgraphics.com/examples/X3dSceneAuthoringHints.html

X3D Graphics Specification
• http://www.web3d.org/x3d/specifications
• Also available as help pages within X3D-Edit
References 3


- http://www.wiley.com/legacy/compbooks/vrml2sbk/cover/cover.htm
- http://www.web3d.org/x3d/content/examples/Vrml2.0Sourcebook
- Chapter 08 – Animating Position Orientation Scale
- Chapter 19 – Normals Shading

References  4

Wikipedia

• Double buffering

• Interlacing
  http://en.wikipedia.org/wiki/Interlace

• Event-based computing
Contact

Don Brutzman

brutzman@nps.edu

http://faculty.nps.edu/brutzman

Code USW/Br, Naval Postgraduate School
Monterey California 93943-5000 USA
1.831.656.2149 voice
CGEMS, SIGGRAPH, Eurographics

The Computer Graphics Educational Materials Source (CGEMS) site is designed for educators

- to provide a source of refereed high-quality content
- as a service to the Computer Graphics community
- freely available, directly prepared for classroom use
- http://cgems.inesc.pt

*X3D for Web Authors* recognized by CGEMS! ☺

- Book materials: X3D-Edit tool, examples, slidesets
- Received jury award for Best Submission 2008

CGEMS supported by SIGGRAPH, Eurographics
Creative Commons open-source license

http://creativecommons.org/licenses/by-nc-sa/3.0
Open-source license
for X3D-Edit software and X3D example scenes

http://www.web3d.org/x3d/content/examples/license.html

Copyright (c) 1995-2013 held by the author(s). All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

• Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

• Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

• Neither the names of the Naval Postgraduate School (NPS) Modeling Virtual Environments and Simulation (MOVES) Institute nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Andries van Dam founded SIGGRAPH. He is one of the four authors of the “bible” (or at least “Old Testament”): Computer Graphics, Principles and Practice, coauthored with J.D. Foley, S.K. Feiner, and J.F. Hughes, published in 1990 by Addison Wesley.

Dr. van Dam's home page is http://www.cs.brown.edu/~avd
Chapter 7 - Event Animation and Interpolation

Contents

Chapter Overview

Concepts

X3D Nodes and Examples

Chapter Summary and Suggested Exercises

References
Chapter Overview
Overview: Event Animation

Behaviors, events, ROUTE connections, animation

Animation as scene-graph modification

Event-animation design pattern: 10-step process

Interpolation nodes

- TimeSensor and event timing
- ScalarInterpolator and ColorInterpolator
- OrientationInterpolator, PositionInterpolator, PositionInterpolator2D and NormalInterpolator
- CoordinateInterpolator2D, CoordinateInterpolator
Chapter 7 - Event Animation and Interpolation

Concepts
Behaviors

*Behavior* defined as changing the value of some field contained by some node in scene graph

- Animation nodes, user interaction nodes and network updates can produce updated values
- ROUTE statements connect output of one node as an input to field in another node

*Event* defined as the time-stamped value passed by a ROUTE, from one field to another

Thus the values held by nodes in scene graph can change as time advances

Fun definition of time: “time is what keeps everything from happening at once!”

This is motivation for why events include timestamps. The timestamp values allow ordering of events so that an earlier behavior doesn't mistakenly override a later behavior.
Behavior traversal of scene graph

Double buffer: once frame is swapped to update screen image, repeat and update scene values

Event model consists of

- Examining clock-driven and user-initiated events
- Updating scene-graph values
- Triggering and updating new events as appropriate
- Continue until all events handled, loops not allowed

Event updates modify the scene graph

- Changing rendering properties, or
- Generating further event outputs

Double buffering is an approach used by most 3D programs to incrementally draw each frame in the background, then swap it with the front buffer when ready. When this repeats at a frame rate of 10 Hz or better, smooth motion is perceived by the user.
Example behavior event chain

- User clicks button to start a timer clock
- Clock outputs new event at start of each frame,
  - ... which stimulates linear-interpolation function
    which produces another output value
  - ... which updates some target value in scene graph
- Repeat event traversal after each frame redraw
ROUTE connections

ROUTE connection enables the output field of one node to pass a value that then stimulates the input field of another node

- The passed value also includes a time stamp

Field data type and accessType must both match between node/field of source and target

- Chapter 1, Technical Introduction lists field types
- Also provided in tooltips and specification
- Authors usually must carefully check these
Animation as scene-graph modification

*Behavior* = changing a field value in a node, somewhere in the scene graph

*Event* = time-stamped value going over a ROUTE

*Event cascade* is a series of events, each one triggering the next, before next frame is drawn
  • No event loops allowed, guaranteeing completion

Thus all X3D animation can be considered as modification of the scene graph at run time
Event-animation design pattern

X3D can be imposing, there are many nodes. Nevertheless, a simple design pattern is used for nearly every kind of animation. TouchSensor is optional. Some other triggering event may be provided to start the animation chain, or the TimeSensor may be looping indefinitely.

There are many interpolator nodes. The choice of which interpolator to utilize is determined by the data type of the target field in the target node.

A sequencer node is used instead of an interpolator node if the target field is boolean or integer. Sequencer nodes are described in Chapter 9, Event Utilities and Scripting.

X3D for Web Authors, Figure 7.1, p. 189.
Visualizing scenes on paper

It is good practice to sketch out 3D scene drafts
  • Consider what models are needed, and how multiple models might be composed

Consider user experience, from their perspective
  • What tasks and goals, what use cases
  • What might things look like when first seen

Storyboarding can help build long-form content
  • Series of vignettes to tell a larger story
  • Each scene defines needed models and behaviors
  • Build each piece, put them together

Storyboarding
  • http://en.wikipedia.org/wiki/Story_board
  • http://www.mcli.dist.maricopa.edu/authoring/studio/guidebook/storyboard.html
  • http://en.wikiversity.org/wiki/Lesson:Thumbnail_Storyboard

Previsualization
  • http://en.wikipedia.org/wiki/Previsualization
Field data types

X3D is a strongly typed language

- Each field in each node (i.e. each XML attribute) has a strictly defined data type
- Data types for boolean, integer, floating point

Types are either single or multiple-value

- Example: SFFloat, SFVec2f, SFVec3f, SFRotation

Also have arrays for all types

SF = Single Field, MF = Multiple Field (array)

Failure to match data types correctly is an error!

- During schema validation, loading or at run time

Data type and accessType information is available for each node in the X3D Tooltips and X3D Specification.

When speaking about data types, you can substitute “array of” for the “MF” prefix. Example: “MFColor is an array of Color values.”

For full review see Chapter 1, Technical Overview.
X3D has strong data typing

Data typing is very important to prevent errors

- *Strong data typing* means that all data types must match (or be converted) exactly
- *Weak data typing* means data types may be promoted or changed by the system automatically without author direction (or quality control)

Data type errors lead to erroneous computations and system crashes, in any computer language

X3D has strong data typing

- Cost: authors must ensure their scene is correct
- Benefit: mysterious run-time errors avoided

Strong data typing, XML validation and a number of other X3D quality-control checks prevent the dreaded errors which arise from Garbage In Garbage Out (GIGO).

GIGO errors can be quite difficult to detect, debug and correct. Thus they are best avoided in the first place. Strong typing, XML validation and tools that report errors are an X3D author's best friend.
<table>
<thead>
<tr>
<th>Field-type names</th>
<th>Description</th>
<th>Example values</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFBool</td>
<td>Single-field boolean value</td>
<td>true or false (X3D syntax), TRUE or FALSE (ClassicVRML syntax)</td>
</tr>
<tr>
<td>MFBool</td>
<td>Multiple-field boolean array</td>
<td>true false true (X3D syntax), [ TRUE FALSE FALSE TRUE ] (ClassicVRML syntax)</td>
</tr>
<tr>
<td>SFCOLOR</td>
<td>Single-field color value, red-green-blue</td>
<td>0 0.5 1.0</td>
</tr>
<tr>
<td>MFColor</td>
<td>Multiple-field color array, red-green-blue</td>
<td>1 0 0, 0 1 0, 0 0 1</td>
</tr>
<tr>
<td>SFCOLORRGBA</td>
<td>Single-field color value, red-green-blue alpha (opacity)</td>
<td>0 0.5 1.0 0.75</td>
</tr>
<tr>
<td>MFColorRGBA</td>
<td>Multiple-field color array, red-green-blue alpha (opacity)</td>
<td>1 0 0 0.25, 0 1 0 0.5, 0 0 1 0.75 (red green blue, varying opacity)</td>
</tr>
<tr>
<td>SFInt32</td>
<td>Single-field 32-bit Integer value</td>
<td>0</td>
</tr>
<tr>
<td>MFInt32</td>
<td>Multiple-field 32-bit Integer array</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td>SFFloat</td>
<td>Single-field single-precision floating-point value</td>
<td>1.0</td>
</tr>
<tr>
<td>MFFloat</td>
<td>Multiple-field single-precision floating-point array</td>
<td>-1 2.0 3.14159</td>
</tr>
</tbody>
</table>

*X3D for Web Authors*, Table 1.4, pp. 19-20.
<table>
<thead>
<tr>
<th>Field-type names</th>
<th>Description</th>
<th>Example values</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFD</td>
<td>Single-field double-precision floating-point value</td>
<td>2.7128</td>
</tr>
<tr>
<td>MFDouble</td>
<td>Multiple-field double-precision array</td>
<td>-1.2.0 3.14159</td>
</tr>
<tr>
<td>SFImage</td>
<td>Single-field image value</td>
<td>Contains special pixel-encoding values, see Chapter 5 for details</td>
</tr>
<tr>
<td>MFImage</td>
<td>Multiple-field image value</td>
<td>Contains special pixel-encoding values, see Chapter 5 for details</td>
</tr>
<tr>
<td>SFNode</td>
<td>Single-field node</td>
<td>&lt;Shape/&gt; or Shape {space}</td>
</tr>
<tr>
<td>MFNode</td>
<td>Multiple-field node array of peers</td>
<td>&lt;Shape/&gt; &lt;Group/&gt; &lt;Transform/&gt;</td>
</tr>
<tr>
<td>SFRotation</td>
<td>Single-field rotation value using 3-tuple axis, radian angle form</td>
<td>0 1 0 1.57</td>
</tr>
<tr>
<td>MFRotation</td>
<td>Multiple-field rotation array</td>
<td>0 1 0 0, 0 1 0 1.57, 0 1 0 3.14</td>
</tr>
<tr>
<td>SFString</td>
<td>Single-field string value</td>
<td>&quot;Hello world!&quot;</td>
</tr>
<tr>
<td>MFS</td>
<td>Multiple-field string array</td>
<td>&quot;EXAMINE&quot; &quot;FLY&quot; &quot;WALK&quot; &quot;ANY&quot;</td>
</tr>
<tr>
<td>SFTime</td>
<td>Single-field time value</td>
<td>0</td>
</tr>
<tr>
<td>MFTime</td>
<td>Multiple-field time array</td>
<td>-1 0 1 567890</td>
</tr>
</tbody>
</table>

*X3D for Web Authors*, Table 1.4, pp. 19-20.
### Field data types

<table>
<thead>
<tr>
<th>Field-type names</th>
<th>Description</th>
<th>Example values</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFVec2f/SFVec2d</td>
<td>Single-field 2-float/2-double vector value</td>
<td>0 1.5</td>
</tr>
<tr>
<td>MFVec2f/MFVec2d</td>
<td>Multiple-field 2-float/2-double vector array</td>
<td>1.0, 2.2, 3.4, 5.5</td>
</tr>
<tr>
<td>SFVec3f/SFVec3d</td>
<td>Single-field vector value of 3-float/3-double values</td>
<td>0 1.5 2</td>
</tr>
<tr>
<td>MFVec3f/MFVec3d</td>
<td>Multiple-field vector array of 3-float/3-double values</td>
<td>10 20 30, 4.4 –5.5 6.6</td>
</tr>
</tbody>
</table>

**ClassicVRML syntax notes**

- TRUE and FALSE (rather than XML true and false)
- MF multiple-field array values are surrounded by square brackets, e.g. [ 10 20 30, 4.4 –5.5 6.6 ]
- No special XML escape characters such as &amp;

-X3D for Web Authors, Table 1.4, pp. 19-20.
accessType: input, output, initialize

accessType determines if field is data sender, receiver, or holder

- inputOnly: can only receive events
- outputOnly: can only send events
- initializeOnly: cannot send or receive
- inputOutput: can send, receive and be initialized

Failure to match accessType correctly is an error!
- Detected during authoring-tool checks, or run time
- inputOnly and outputOnly values cannot be listed as attributes in .x3d scene file, since they are transient

Data type and accessType information is available for each node in the X3D Tooltips and X3D Specification.

TODO explain 3D graphics rationale for accessType: performance
accessType naming conventions

The accessType names were changed when VRML97 was upgraded to X3D

- Functionality remains essentially unchanged

X3D specification entries for each node use yet another shorthand, as shown here

<table>
<thead>
<tr>
<th>VRML97 Name</th>
<th>X3D Name</th>
<th>X3D Specification abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventIn</td>
<td>inputOnly</td>
<td>[in]</td>
</tr>
<tr>
<td>eventOut</td>
<td>outputOnly</td>
<td>[out]</td>
</tr>
<tr>
<td>field</td>
<td>initializeOnly</td>
<td>[ ]</td>
</tr>
<tr>
<td>exposedField</td>
<td>inputOutput</td>
<td>[in,out]</td>
</tr>
</tbody>
</table>

VRML, Virtual reality modeling language; X3D, Extensible 3D.

*X3D for Web Authors*, Table 1.6, p. 28.
accessType naming conventions

Field names often reveal special accessType

- Prefix *set_* indicates inputOnly field
- Prefix *_changed* indicates outputOnly field
- Prefix *is* for outputOnly boolean field (e.g. *isActive*)

inputOnly, outputOnly fields not allowed in files

Understanding naming conventions helps authors understand ROUTE definitions and results

Looking ahead: we will name our own fields when creating Scripts and prototypes, further underscoring importance of naming

Script nodes allow an author to define the input and output fields of interest. Each Script node can react and respond to input events by performing computations and then sending output events. The contained Script code may be written in EcmaScript (i.e. Javascript) or in Java.

For more on this subject, see Chapter 9, Event Utilities and Scripting.
Interpolating animation chains: 10-step design process

The following 10-step process can be used for all animation tasks.

Table is also provided in order to look up how to produce typed-value outputs corresponding to each interpolator or sequencer node.

A detailed example follows.

This 10-step process is a good check to perform each time you create an animation chain.
Interpolating animation chains 1-2

1. **Pick target.** Pick node and target field to animate (i.e., field that receives changing animation values)

2. **Name target.** Provide a DEF label for the node of interest, giving it a name

---

*X3D for Web Authors*, section 2.5, pp. 192-193
Interpolating animation chains 3-4

3. **Check accessType and data type.**
   - Ensure target field has `accessType` of `inputOnly` or `inputOutput`, so that it can receive input events
   - Determine if target field has floating-point type: `SFFloat`, `SFVec3f`, `SFColor`, `SFRotation`, and so on... If so, use an interpolator node as the event source

4. **Determine if Sequencer or Script.**
   - If the target type is an `SFBool` or `SFInt32`, use a sequencer node as event source
   - If the target type is an `SFNode` or `MFNode`, use a Script node as the event source

---

*X3D for Web Authors*, section 2.5, pp. 192-193

When checking data type:

- The target field can either be singleton SF type or array MF type
- SF means Single Field, MF means Multiple Field (i.e. an array) in the X3D type-naming convention
Interpolating animation chains 5-6

5. **Determine which Interpolator.** If you are not using a sequencer or Script node, determine corresponding Interpolator which produces the appropriate data type for `value_changed` output using lookup table
   • Example: PositionInterpolator produces SFVec3f `value_changed` events

6. **Triggering sensor.** If desired, add sensor node at beginning, to provide appropriate SFTime or SFBool trigger to start animation
   • Sometimes the triggering event is an output event from another animation chain

*X3D for Web Authors*, section 2.5, pp. 192-193

for Step 5, Determine which Interpolator:
• See Table 7.2 for example interpolator chains
• See Table 7.5 for list of candidate interpolators to use, based on data type
7. **TimeSensor clock.** Add a TimeSensor as the animation clock, then set its `cycleInterval` field to the desired duration interval of animation
   - Set `loop='false'` if an animation only runs once at certain specific times. (Will need triggering event.)
   - Set `loop='true'` if it loops repeatedly

8. **Connect trigger.** ROUTE sensor or trigger node’s output field to the TimeSensor input in order to start the animation chain
   - Each node in animation chain needs a DEF name, so that ROUTE can connect to/from

---

*X3D for Web Authors*, section 2.5, pp. 192-193
Interpolating animation chains  9-10

9. *Connect clock.* ROUTE the TimeSensor *fraction_changed* field to the interpolator (or sequencer or Script) node's *set_fraction* field, in order to drive the animation chain

10. *Connect animation output.* ROUTE the interpolator, sequencer, or Script node's *value_changed* field to target field of interest in order to complete the animation chain

Construction of animation-chain design pattern is complete, now test whether animation works

*X3D for Web Authors*, section 2.5, pp. 192-193
Example animation chains

Each row in Table 7.2 shows commonly authored sequences of nodes in animation chains

<table>
<thead>
<tr>
<th>Triggering Nodes (Optional)</th>
<th>Clock Nodes</th>
<th>Value-Producing Nodes</th>
<th>Value-Consuming Nodes, Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>TouchSensor</td>
<td>TimeSensor</td>
<td>ScalarInterpolator</td>
<td>Material (transparency)</td>
</tr>
<tr>
<td>VisibilitySensor</td>
<td>TimeSensor</td>
<td>ColorInterpolator</td>
<td>Material (color field)</td>
</tr>
<tr>
<td>PrimarySensor</td>
<td>TimeSensor</td>
<td>PositionInterpolator</td>
<td>Transform (translation, scale)</td>
</tr>
<tr>
<td>TouchSensor</td>
<td>TimeSensor</td>
<td>OrientationInterpolator</td>
<td>Transform (rotation)</td>
</tr>
<tr>
<td>MovieTexture (loop complete)</td>
<td>TimeSensor</td>
<td>MovieTexture</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PositionInterpolator2D</td>
<td>Rectangle2D</td>
</tr>
</tbody>
</table>

*Used in Step 5: Determine which Interpolator*

**X3D for Web Authors, Table 7.2, p. 196.** Example Animation Chains: Each Row Shows a Commonly Authored Sequence of Nodes
X3D field types and corresponding animation nodes

<table>
<thead>
<tr>
<th>Field type</th>
<th>Description</th>
<th>Interpolator/Sequencer animation nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFBool</td>
<td>Single-field boolean value</td>
<td>BooleanSequencer</td>
</tr>
<tr>
<td>SFCOLOR</td>
<td>Single-field Color value, red-green-blue</td>
<td>ColorInterpolator</td>
</tr>
<tr>
<td>SFInt32</td>
<td>Single-field 32-bit Integer value</td>
<td>IntegerSequencer</td>
</tr>
<tr>
<td>SFFloat</td>
<td>Single-field single-precision floating-point value</td>
<td>ScalarInterpolator</td>
</tr>
<tr>
<td>SFRotation</td>
<td>Single-field Rotation value using 3-tuple axis, radian angle form</td>
<td>ColorInterpolator</td>
</tr>
<tr>
<td>SFTime</td>
<td>Single-field Time value</td>
<td>TimeSensor</td>
</tr>
<tr>
<td>SFVec2f</td>
<td>Single-field 2-float vector value</td>
<td>PositionInterpolator2D</td>
</tr>
<tr>
<td>MFVec2f</td>
<td>Multiple-field 2-float vector array</td>
<td>CoordinateInterpolator2D</td>
</tr>
<tr>
<td>SFVec3f</td>
<td>Single-field vector value of 3-float values</td>
<td>PositionInterpolator</td>
</tr>
<tr>
<td>MFVec3f</td>
<td>Multiple-field vector array of 3-float values</td>
<td>CoordinateInterpolator</td>
</tr>
</tbody>
</table>

**Used in Step 5: Determine which Interpolator**

* X3D for Web Authors, Table 7.5, p. 199.

Notice that some types are not on the list! That is because there is no direct way to animate them. Usually a Script node is needed to do this.

- BackgroundColorArrayAnimation.x3d example in Script chapter 9
- TODO: also consider ColorArrayInterpolator node in Prototypes chapter 14
Animation chain for this example

HelloX3dAuthorsAnimationChain.x3d is our detailed animation-chain example

TouchSensor          TimeSensor          OrientationInterpolator  Transform
Trigger node         Clock node          Interpolator or Sequencer  Target node

SFTime event                  SFFloat event                   SFRotation event
              touchTime => startTime                               value_changed => rotation

fraction_changed => set_fraction

http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/HelloX3dAuthorsAnimationChain.x3d

Interestingly our example scene includes the TouchSensor trigger node, but as a default ignores that trigger by setting <TimeSensor loop="true"/> which starts the rotation automatically.
Hello X3D Authors showing ROUTEs

10-step process for constructing animation chains, applied to animated HelloWorld example
Hello X3D Authors 10-step process

10-step process for constructing animation chains, applied to animated HelloWorld example

http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/HelloX3dAuthorsAnimationChain.x3d
Hello X3D Authors 10-step process

1. **Pick target.** The target node is a Transform, and the target field is `set_rotation`.
2. **Name target.** The Transform is named `DEF='EarthCoordinateSystem'`.
3. **Check accessType and data type.** As shown by the Transform node field-definition table in Chapter 3 and the X3D-Edit tooltip, the `set_rotation` field has type `SFRotation`.
4. **Determine whether Sequencer or Script.** These special node types are not applicable to this example, because the data type for `set_rotation` is `SFRotation` which is a floating-point type.
5. **Determine which Interpolator.** The animating OrientationInterpolator is named `DEF='SpinThoseThings'` and placed just before the Transform.
6. **Triggering sensor.** A triggering TouchSensor is added next to the geometry to be clicked, and then named `DEF='ClickTriggerTouchSensor'`.
7. **TimeSensor clock.** The TimeSensor is added at the beginning of the chain, named `DEF='OrbitalTimeInterval'` and has both the `cycleInterval` and `loop` fields set.
8. **Connect trigger.** Add ROUTE to connect the triggering TouchSensor node's `touchTime` output field to the clock node’s `startTime` input field.
9. **Connect clock.** Add ROUTE to connect the clock node’s `fraction_changed` output field to the interpolator node’s `set_fraction` input field.
10. **Connect animation output.** Add ROUTE to connect the interpolator node’s `value_changed` output field to the original target input field, `set_rotation`.

---

**X3D for Web Authors**, Figure 7.5, pp. 193-195.

http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/HelloX3dAuthorsAnimationChain.x3d

10-step process for constructing animation chains, applied to animated HelloWorld example

I strongly recommend you print this out (or keep it handy) and check off each step as you proceed. After a few times you will find that you are doing this without needing the checklist. Keeping a consistent pattern lets you avoid thinking that the various animation nodes are “really different” when they are not. It also helps you avoid skipping steps and making mistakes that are hard to debug afterwards.
ROUTE editor examples

```xml
<ROUTE
    fromNode='OrbitalTimeInterval'
    fromField='fraction_changed'
    toNode='SpinThoseThings'
    toField='set_fraction' />

<ROUTE
    fromNode='ClickTriggerTouchSensor'
    fromField='touchTime'
    toNode='OrbitalTimeInterval'
    toField='startTime' />
```

http://www.web3d.org/x3d/content/examples/Basic/course/HelloX3dAuthorsAnimationChain.x3d
Interpolation

Interpolation is the estimation of intermediate values from other values.

Computing averages is computationally efficient and highly optimizable.

Linear approximation is thus well suited for high-performance graphics animation.

X3D provides interpolation nodes for each of the floating-point data types.

- including multiple-value types: Color, Vec3f, etc.
Interpolation node type

X3DInterpolationNode is the formal name for the interpolation node type.

Each interpolation node includes the following common fields and naming conventions:

- SF, MF <type> definition must be consistent for node in order to properly define response function.

<table>
<thead>
<tr>
<th>Type</th>
<th>accessType</th>
<th>Name</th>
<th>Default</th>
<th>Range</th>
<th>Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFfloat</td>
<td>inputOutput</td>
<td>key</td>
<td>[]</td>
<td>$(-\infty, \infty)$</td>
<td>Interchange</td>
</tr>
<tr>
<td>MF&lt;type&gt;</td>
<td>inputOutput</td>
<td>keyValue</td>
<td>[]</td>
<td>(type dependent)</td>
<td>Interchange</td>
</tr>
<tr>
<td>SFFloat</td>
<td>inputOnly</td>
<td>set_fraction</td>
<td></td>
<td></td>
<td>Interchange</td>
</tr>
<tr>
<td>[SF MF]&lt;type&gt;</td>
<td>outputOnly</td>
<td>value_changed</td>
<td></td>
<td></td>
<td>Interchange</td>
</tr>
<tr>
<td>SFNode</td>
<td>inputOutput</td>
<td>metadata</td>
<td>NULL</td>
<td>[X3DMetadataObject]</td>
<td>Core</td>
</tr>
</tbody>
</table>

*X3D for Web Authors*, Table 7.4, p. 197.
Common interpolator fields

- *key, keyValue* hold the point values defining the characteristic function
- *key* array always has type MFFloat
- *keyValue* array data type matches the named type of the parent Interpolator node
  - final value must equal first value in *keyValue* array if smooth looping is desired
- Lengths of *key, keyValue* arrays must be equal
- Note that *keyValue* array can hold values which are themselves MF (multi-field) array type
- Function output *value_changed* always has same name, but data type matches the Interpolator node
Linear interpolation

Piecewise-linear curve fitting can approximate any curve with arbitrary accuracy

Multi-field (MF) values are individually interpolated proportionately

`key='0 0.3333 0.666 1'`  
`keyValue='1 0 0, 0 1 0, 0 0 1, 1 0 0'`

First figure:  *X3D for Web Authors*, Figure 7.2, p. 191.

`<ScalarInterpolator key='0 0.2 0.4 0.6 0.8 1' keyValue='0 5 8 9 4 0'/>`

Second figure:  *X3D for Web Authors*, Figure 7.4, p. 192.

`<ColorInterpolator key='0, 0.3333, 0.6666, 1' keyValue='1 0 0, 0 1 0, 0 0 1, 1 0 0'/>`

http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/ColorInterpolatorExample.x3d
Step-wise linear interpolation

Step functions are created by repeating time values and corresponding output.

key='0 0.25 0.25 0.5 0.5 1'
keyValue='1 1 2 2 3 4'

Note that time-fraction key array must always be monotonically (steadily) increasing.

X3D for Web Authors, Figure 7.3, p. 191.

<ScalarInterpolator key="0 0.25, 0.25 0.5, 0.5 1" keyValue="1 1, 2 2, 3 4"/>
Double linear-interpolation averaging

Matched *key, keyValue* arrays define the points for a linear-interpolator approximation function. Two-way weighted averaging is used to compute interpolated-input, interpolated-output results.

![Diagram showing linear interpolation](image)

*X3D for Web Authors*, Figure 7.8, p. 198.

First the entry-value $t$ is compared to the *key* array until the prior and following values of *key* are found that are less-than and greater-than $t$.

Then a percentage is computed that accounts for the proportion of $t$ between the bracketing values of *key*[i] and *key*[i+1].

Then this same percentage is applied to compute a new *result* value which equals the same percentage between corresponding output-array values of *keyValue*[i] and *keyValue*[i+1].

Interpolation equations are found on the TimeSensor Output slide and notes.
X3D Nodes and Examples
TimeSensor

TimeSensor is the heartbeat of an animation

- provides pulse that triggers event cascades
- initiates computations for drawing next frame
- Outputs values as fraction_changed, from 0 to 1

TimeSensor samples elapsed time based on the computer clock, rather than screen update rate

- Ensures that animations are smooth and realistic
- Fixed (constant) frame rate is typically not feasible since computation varies for screen-image updates
**TimeSensor output**

Output time is an SFTime ramp function ranging \([0,1]\) that repeats every `cycleInterval` seconds

- Sometimes called a 'sawtooth' function
- SFFloat output field `fraction_changed` used as input to other interpolators, sequencers

**Sawtooth function**

*X3D for Web Authors*, Figure 7.9, p. 201. TimeSensor `fraction_changed` varies over the range \([0,1]\) for each `cycleInterval` repetition.

*X3D for Web Authors*, Figure 7.10, p. 202. TimeSensor `fraction_changed` output algorithm, expressed in pseudocode.

```plaintext
time = now; // output field value
numberOfLoops = (now - startTime) / cycleInterval; // floating-point calculation
f = fractionalPart(numberOfLoops);
if (now <= startTime)
    fraction_changed = 0.0; // output field value
else if ((loop == 'false') && (now == (startTime + cycleInterval)))
    fraction_changed = 1.0; // output field value
else fraction_changed = f; // output field value
```
TimeSensor fields

- *enabled* controls whether node enabled or disabled
- *loop* is an SFBool indicating whether to continue looping indefinitely after first cycle is complete
- *cycleInterval* defines total loop duration in seconds, either for single-shot animation or looped repetition
- *cycleTime* field is sent an SFTIm output value upon completion of each loop

Two ways to stop an animation: set *enabled*='false' or send an SFTIm event to *stopTime*

Similarly, must reset enabled or send an SFTIm event to *startTime* to begin again.

Can alternatively send SFTIm values to *set_stopTime, set_startTime*
**TimeSensor fields 2**

- *startTime, stopTime* are provided (or contain) SFTime values for when to start, stop respectively
  - ROUTE an SFTime value to *startTime* or *stopTime*
  - *isActive, isPaused* are output SFBool true/false events sent whenever the TimeSensor is set to run or paused
- *pauseTime, resumeTime* are SFTime values for current clock time whenever paused or resumed
  - Corresponding boolean *isPaused* event is also sent, with value of true when paused and false when resuming
- *elapsedTime* output provides cumulative number of seconds since TimeSensor was activated and began running, without including paused time

Can alternatively send SFTime values to *set_pauseTime, set_resumeTime*
http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/ColorInterpolatorExample.x3d
### TimeSensor

<table>
<thead>
<tr>
<th>TimeSensor continuously generates events as time passes. Typical use: ROUTE TimeSensor_fraction_changed TO _set_fraction.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEF</strong></td>
</tr>
<tr>
<td>[DEF ID = &quot;IMPLIED&quot;]</td>
</tr>
<tr>
<td>DEF defines a unique ID name for this node, re-entrantable by other nodes.</td>
</tr>
<tr>
<td>[Hint: descriptive DEF names improve clarity and help document a model.]</td>
</tr>
<tr>
<td><strong>USE</strong></td>
</tr>
<tr>
<td>[USING IDREF = &quot;IMPLIED&quot;]</td>
</tr>
<tr>
<td>USE means return an already DEF ed node ID, ignoring <em>all</em> other attributes and children.</td>
</tr>
<tr>
<td>[Hint: USING other geometry (instead of duplicating nodes) can improve performance.]</td>
</tr>
<tr>
<td><strong>warning</strong></td>
</tr>
<tr>
<td>[enabled: accessType inputOutput, type SBOOL (true/false) &quot;true&quot;]</td>
</tr>
<tr>
<td>Enables/disables node operation.</td>
</tr>
</tbody>
</table>

| cycleInterval: accessType inputOutput, type SFLOAT CDATA "L0"] |
| cycleInterval is loop duration in seconds. |
| [Hint: TimeSensor may be ignored if cycleInterval < 0.01 second.] |
| **loop** |
| [loop: accessType inputOutput, type SBOOL (true/false) "false"] |
| Repeat indefinitely when loop=true, repeat only once when loop=false. |
| **startTime** |
| [startTime: accessType inputOutput, type SFLOAT CDATA "0"] |
| When time now > startTime, isActive becomes true and TimeSensor becomes active. Absolute time number of seconds since Jan 1, 1970, 00:00:00 GMT. |
| [Hint: usually receives a ROUTED time value.] |
| **stopTime** |
| [stopTime: accessType inputOutput, type SFLOAT CDATA "0"] |
| When time now <= stopTime becomes <= time now, isActive becomes false and TimeSensor becomes inactive. Absolute time number of seconds since Jan 1, 1970, 00:00:00 GMT. |
| [Hint: usually receives a ROUTED time value.] |
| **cycleTime** |
| [cycleTime: accessType inputOutput, type SFLOAT CDATA "#FIXED"] |
| cycleTime sends a time outputOnly at startTime, and also at the beginning of each new cycle (useful for synchronization with other time-based objects). |
| **isActive** |
| [isActive: accessType outputOnly, type SBOOL (true/false) "#FIXED"] |
| isTrue=true false events are sent when TimeSensor starts/stops running. |
| **isPaused** |
| [isPaused: accessType outputOnly, type SBOOL (true/false) "#FIXED"] |
| isPaused=true false events are sent when TimeSensor is paused/resumed. |
| [Warning: not supported in VRML97] |
| **isRunning** |
| [isRunning: accessType outputOnly, type SBOOL (true/false) "0"] |
| When time now > pauseTime, isRunning becomes true and TimeSensor becomes paused. Absolute time number of seconds since Jan 1, 1970, 00:00:00 GMT. |
| [Hint: usually receives a ROUTEd time value.] |
| [Warning: not supported in VRML97] |
| **isEnded** |
| [isEnded: accessType outputOnly, type SFLOAT CDATA "#FIXED"] |
| Current clipped time since TimeSensor activated/running, cumulative in seconds, and not counting any paused time. |
| [Warning: not supported in VRML97] |

---

http://www.web3d.org/x3d/content/X3dTooltips.html#TimeSensor
TimeSensor tooltips

http://www.web3d.org/x3d/content/X3dTooltips.html#TimeSensor
ScalarInterpolator node

Generates a scalar (single-valued) SFFloat for value_changed output
key and keyValue arrays contain SFFloat values

set_fraction determines input value to piece-wise linear function
• Percentage between bracketing key[i], key[i+1] values used to compute corresponding output value_changed as weighted average between keyValue[i], keyValue[i+1]
• Which is same algorithm for all interpolators
The ScalarInterpolator output values are used to modify the Material transparency value of the Sphere.
ScalarInterpolator generates piecewise-linear values that can be ROUTEd to other Float attributes. Typical input: ROUTe someTime/linear_fraction_changed TO someScalarInterpolator.set_fraction Typical output: ROUTe someInterpolator.value_changed TO destinationNode.value_set_attributes.

| DEF | [DEF ID "SIMPLIFIED"] DEF defines a unique ID name for this node, referencable by other nodes. Hint: descriptive DEF names improve clarity and help document a model. |
| USE | [USE ID REF "SIMPLIFIED"] USE names reuse an already DEF-ed node ID, ignoring all other attributes and children. Hint: Using other geometry (instead of duplicating nodes) can improve performance. Warning: do NOT include DEF (or any other attribute values) when using a USE attribute! |
| Key | [key] access: Type input/Output, type MMFloat CDATA "SIMPLIFIED" Defines parameters for linear interpolation function time intervals, in increasing order and corresponding to keyValues. Hint: number of keys must match number of keyValues |
| KeyValue | [keyValue] access: Type input/Output, type MMFloat CDATA "SIMPLIFIED" Output values for linear interpolation, each corresponding to time-distribution keys. Hint: number of keys must match number of keyValues |
| set_fraction | [set_fraction] inputOnly type MMFloat CDATA "FIXED ***" set_fraction selects input key for corresponding keyValue output. |
| value_changed | [value_changed] access: Type outputOnly, type MMFloat CDATA "FIXED ***" Linearly interpolated output value determined by current key time and corresponding keyValue pair. |
| containerField | [containerField: NMTOKEN "children"] containerField is the field-label prefix indicating relationship to parent node. Examples: geometry Box, childrenGroup, proxy Shape. containerField attribute is only supported in XML encoding of X3D scenes. |
| class | [class] CDATA "SIMPLIFIED" class is a space-separated list of classes, reserved for use by XML stylesheet. class attribute is only supported in XML encoding of X3D scenes. |

http://www.web3d.org/x3d/content/X3dTooltips.html#ScalarInterpolator
ColorInterpolator node

Generates a 3-tuple (triple-valued) SFColor for continuous value_changed output

- key array contains SFFloat values
- keyValue array contains SFColor values

Linear interpolation of red, green, blue (RGB) values is respectively performed for each bracketing keyValue pair

- key='0 0.3333 0.666 1'
- keyValue='1 0 0, 0 1 0, 0 0 1, 1 0 0'
ColorInterpolator animation chain

Each node's output field matches data type of next node's input field
accessType outputOnly to inputOnly, initializeOnly also match
ColorInterpolator example output

Using the pointing device to select the text triggers the ColorInterpolator animation

- Colors vary gradually, by linear interpolation of each of the component red-green-blue RGB values
ColorInterpolator scene graph illustration

http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/ColorInterpolatorExample.x3d
ColorInterpolator scene graph with ROUTEs

Suggested exercise:
label each link with field names, types, and accessType.

http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/ColorInterpolatorExample.x3d
Chapter 7 - Event Animation and Interpolation

Pretty-printing a scene in HTML, printing it in landscape mode and then annotating it with ROUTE arrows is an excellent way to debug animation chains in a large scene.

http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07/EventAnimationInterpolation/ColorInterpolatorExample.x3d
Pretty-printing a scene in HTML, printing it in landscape mode and then annotating it with ROUTE arrows is an excellent way to debug animation chains in a large scene.
http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/ColorInterpolatorExample.x3d
<table>
<thead>
<tr>
<th>ColorInterpolator</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEF</strong></td>
</tr>
<tr>
<td><strong>DEF ID</strong></td>
</tr>
<tr>
<td><strong>USE IDREF</strong></td>
</tr>
<tr>
<td><strong>Key</strong></td>
</tr>
<tr>
<td><strong>Key Values</strong></td>
</tr>
<tr>
<td><strong>Set fraction</strong></td>
</tr>
<tr>
<td><strong>Value changed</strong></td>
</tr>
<tr>
<td><strong>ContainerField</strong></td>
</tr>
<tr>
<td><strong>Class</strong></td>
</tr>
</tbody>
</table>

---

http://www.web3d.org/x3d/content/X3dToolTips.html#ColorInterpolator
OrientationInterpolator node

Generates a 4-tuple (four-valued orientation) SFRotation for \textit{value\_changed} output

\textit{key} array contains SFFloat fraction values
\textit{keyValue} array contains SFRotation output values

• As always: same number of \textit{key}, \textit{keyValue} entries

OrientationInterpolator animates along shortest path between the two normal vectors, also computes linear average between two corresponding angles, in \textit{keyValue} array
OrientationInterpolator example

This animation-chain example can be added to any scene (via cut and paste) to create a look-around Viewpoint. This bound camera view rotates about a fixed position.

```xml
<Viewpoint DEF='DizzyViewpoint' description='Rotating viewpoint' position='[somewhere you want it]' orientation='0 1 0 0'/>
<OrientationInterpolator DEF='Spinner' key='0 0.25 0.5 0.75 1'
  keyValue='0 1 0 0, 0 1 0 1.57, 0 1 0 3.14, 0 1 0 4.71, 0 1 0 6.28'/>
<TimeSensor DEF='SpinClock' cycleInterval='12' loop='true'/>
<ROUTE fromField='fraction_changed' fromNode='SpinClock'
  toField='set_fraction' toNode='Spinner'/>
<ROUTE fromField='value_changed' fromNode='Spinner'
  toField='orientation' toNode='DizzyViewpoint'/>
```
OrientationInterpolator node X3D-Edit

http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/PositionOrientationInterpolatorsExample.x3d
OrientationInterpolator generates a series of rotation values. Results can be ROUTEd to a <Transform> node's 'rotation' attribute or another RotationInterpolator attribute. Typical input: ROUTE someTimeSource::fraction_changed TO someInterpolator::set_fraction Typical output: ROUTE someInterpolator::value_changed TO destinationNode::set_attribute.

DEF
DEF ID=**implies**
DEF defines a unique ID name for this node, referenceable by other nodes.
Host: descriptive DEF names improve clarity and help document a model.

USE
USE ID=**implies**
USE means reuse an already DEF-ed node ID, ignoring all other attributes and children.
Host: USING other geometry (instead of duplicating nodes) can improve performance.
Warning: do NOT include DEF (or any other attribute) values when using a USE attribute!

key:
Key: accessType=inputOutput, type=MFfloat CDATA #IMPLIED
Defines parameters for linear interpolation of fractional time intervals, in increasing order and corresponding to keyValues.
Host: number of keys must match number of keyValues!

keyValues:
KeyValues: accessType=inputOutput, type=MFRotation CDATA #IMPLIED
Output values for linear interpolation, each corresponding to time-fraction keys.
Host: number of keys must match number of keyValues!

set_fraction:
set_fraction: inputOnly type=SFfloat CDATA #FIXED ""
set_fraction selects input key for corresponding keyValue output

value_changed:
value_changed: accessType=outputOnly, type=SFRotation CDATA #FIXED ""
Linearly interpolated output value determined by current key time and corresponding keyValue pair.

containerField:
containerField: XML_TOKEN "children"
containerField is the field-label prefix indicating relationship to parent node. Examples: geometryBox, childrenGroup, proxyShape. containerField attribute is only supported in XML encoding of X3D scenes.

class:
class CDATA #IMPLIED
class is a space-separated list of classes, reserved for use by XML stylesheets. class attribute is only supported in XML encoding of X3D scenes.

http://www.web3d.org/x3d/content/X3dToolips.html#OrientationInterpolator
PositionInterpolator node

Generates a 3-tuple (three-valued floating point) SFVec3f for value_changed output

- key array contains SFFloat fraction values
- keyValue array contains SFVec3f output values
  - As always: same number of key, keyValue entries

PositionInterpolator computes weighted average between corresponding x, y and z pairs in the keyValue array

- ROUTE to Transform, either translation or scale
http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/PositionOrientationInterpolatorsExample.x3d
Event tracing

X3D-Edit author-assist feature provides support for tracing output events

- “Trace” checkbox on editing pane adds some extra X3D source to your scene
- Captures all output events, routes them to a Script
- Script outputs event values to X3D browser console at run time so that you can trace execution logic

Can be helpful for selective debugging when animation chains are not behaving as expected

- Available for ROUTE, most event-producing nodes
- Simply remove when done troubleshooting

A long-standing challenge for many X3D authors is debugging event chains. It is sometimes hard to detect when an event is not being passed as expected. Problems can include missing or incorrect ROUTE connections, mismatched types or accessTypes, or other problems.

Sometimes (but only just sometimes) a browser might also be at fault.

X3D-Edit now includes a new capability when editing ROUTE connections or event-producing nodes. If you select the "Trace" checkbox, then a block of X3D code is inserted that connects all of the output events to a Script which reports whenever an event is passed.

Although a bit verbose, it is a cool capability and can be very helpful whenever you are troubleshooting or tracking progress.

Script nodes and Javascript are covered in Chapter 9, Event Utilities and Scripting.
Event tracing example

PositionOrientationInterpolatorsExampleTraced.x3d
http://www.web3d.org/x3d/content/X3dTooltips.html#PositionInterpolator
PositionInterpolator2D node

Generates a 2-tuple (two-valued floating point) SFVec2f for value_changed output

- key array contains SFFloat fraction values
- keyValue array contains SFVec2f output values
  - As always: same number of key, keyValue entries

PositionInterpolator2D computes weighted average between corresponding (x, y) pairs in the keyValue array

Note that PositionInterpolator2D provides single 2D values, while CoordinateInterpolator2D node produces arrays of 2D vectors.
http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/PositionInterpolator2dExample.x3d
Selecting the texture with the mouse pointer starts the TextureTransform \textit{scale} animation, deselecting the texture stops the animation.

http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/PositionInterpolator2dExample.x3d

fun: replace destination toField \textit{set\_scale} with \textit{set\_translation} in the ROUTE
Chapter 7 - Event Animation and Interpolation

<table>
<thead>
<tr>
<th><strong>PositionInterpolator2D</strong> generates a series of Vector2f values that can be ROUTED to a Vector2f attribute. Typical input: ROUTE some/TimeSensor/fraction_changed TO com.example.SomeInterpolator2D.set_fraction. Typical output: ROUTE com.example.SomeInterpolator2D.value_changed TO destinationNode.set_attribute.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEF</strong></td>
</tr>
<tr>
<td><strong>USE</strong></td>
</tr>
<tr>
<td><strong>key</strong></td>
</tr>
<tr>
<td><strong>keyValue</strong></td>
</tr>
<tr>
<td><strong>value</strong></td>
</tr>
</tbody>
</table>

---

http://www.web3d.org/x3d/content/X3dTooltips.html#PositionInterpolator2D
NormalInterpolator node

Generates a 3-tuple (three-valued floating point) SFVec3f for value_changed output

- key array contains SFFloat values
- keyValue array contains SFVec3f values
  - As always: same number of key, keyValue entries
  - SFVec3f outputs: unit-normal vectors, magnitude=1

NormalInterpolator animates along shortest path between the pair of normal vectors currently being referenced in keyValue array

Normal vectors used for special shading effects
  - see Chapter13 - Geometry Triangles Quadrilaterals

Why is NormalInterpolator different than PositionInterpolator?
  - Normal vectors are 3-tuple values that begin at the local origin
  - Normal vectors have unit length (or can be normalized to unit length)
  - Interpolation between normals thus travels along surface of unit sphere
  - Position values are points in space
  - Position values may also be used as 3D dimensions, e.g. Transform scale
  - PositionInterpolator performs linear averaging between position values
For alternate examples, see

VRML 2.0 Sourcebook, Chapter 19 - Normals Shading

http://www.web3d.org/x3d/content/examples/Vrml2.0Sourcebook/Chapter19-NormalsShading
NormalInterpolator editor

Rows correspond to each key
Correct results have same number of MFVec3f values in each row
Normal-to-normal interpolation occurs on each column, allowing easier comparison
**NormalInterpolator**

NormalInterpolator generates a series of normal (perpendicular) vector sets along the surface of a unit sphere. ROUTE values to vector attribute of a `NormalNode` node or another `Vector3fArray` node. Typical input: ROUTE someTimeSensor.fraction_changed TO someNormalInterpolator.set_fraction

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF ID</td>
<td>#IMPLIED&lt;br&gt;DEF defines a unique ID name for this node, referencable by other nodes. If not, descriptive ID names improve clarity and help document a model.</td>
</tr>
<tr>
<td>USE IDREF</td>
<td>#IMPLIED&lt;br&gt;USE means reuse an already DEF-ed node ID, ignoring <em>all</em> other attributes and children. Using USEing other geometry (instead of duplicating nodes) can improve performance. <strong>Warning:</strong> do NOT include DEF (or any other attribute values) when using a USE attribute!</td>
</tr>
<tr>
<td>key</td>
<td>#IMPLIED&lt;br&gt;accessType inputOutput, type MFString CDATA #IMPLIED&lt;br&gt;Definition parameters for linear-interpolation function time intervals, in increasing order and corresponding to keyValues. &lt;br&gt;hint: number of keys must match number of keyValues?</td>
</tr>
<tr>
<td>keyValue</td>
<td>#IMPLIED&lt;br&gt;accessType inputOutput, type MFVec3f CDATA #IMPLIED&lt;br&gt;Output values for linear interpolation, each corresponding to time-fraction keys. &lt;br&gt;hint: number of keys must match number of keyValues?</td>
</tr>
<tr>
<td>set_fraction</td>
<td>#IMPLIED&lt;br&gt;inputOnly, type MFFloat CDATA #IMPLIED&lt;br&gt;set_fraction selects input key for corresponding keyValue output.</td>
</tr>
<tr>
<td>value_changed</td>
<td>#IMPLIED&lt;br&gt;outputOnly, type MFVec3f CDATA #IMPLIED&lt;br&gt;Linearly interpolated output value determined by current key time and corresponding keyValue pair.</td>
</tr>
</tbody>
</table>

**normalField**

containerField: NXTOKEN `"children"`<br>containerField is the field-label prefix indicating relationship to parent node. Examples: geometryBox, childrenGroup, proxyShape. containerField attribute is only supported in XML encoding of X3D scenes.

**class**
<br>class is a space-separated list of classes, reserved for use by X3D styleSheets. class attribute is only supported in XML encoding of X3D scenes.

[http://www.web3d.org/x3d/content/X3dTooltips.html#NormalInterpolator](http://www.web3d.org/x3d/content/X3dTooltips.html#NormalInterpolator)
CoordinateInterpolator2D node

Generates 2-tuple (two-valued floating point) array, MFVec2f for `value_changed` output

- `key` array contains SFFloat values
- `keyValue` array contains MFVec2f values
  - As always: same number of `key, keyValue` entries
  - Counting is very important for arrays of arrays!

CoordinateInterpolator2D computes weighted average between corresponding x and y pairs for each subarray in the `keyValue` array

Note that CoordinateInterpolator2D node produces arrays of 2D vectors, while PositionInterpolator2D provides single 2D values.
http://X3dGraphics.com/examples/X3dForWebAuthors/Chapter07-EventAnimationInterpolation/CoordinateInterpolator2dExample.x3d
CoordinateInterpolator2D generates a series of Vector2FloatArray values that can be ROUTEd to a Vector2FloatArray attribute. Typical input: ROUTEvent time source, fraction changed TO someInterpolator.set_fraction. Typical output: ROUTEvent someInterpolator.value_changed TO destinationNode.node_set_attribute.

<table>
<thead>
<tr>
<th><strong>DEF</strong></th>
<th>[DEF ID #IMPLIED]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF</td>
<td>defines a unique ID name for this node, referring by other nodes.</td>
</tr>
<tr>
<td>Host</td>
<td>describes DEF names improve clarity and help document a model.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>USE IDREF #IMPLIED</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>URE uses or returns an already DEF-ed node ID, ignoring all other attributes and children.</td>
</tr>
<tr>
<td>Host: UNDef other geometry (instead of duplicating nodes) can improve performance.</td>
</tr>
<tr>
<td>Warning: DO NOT include DEF (or any other attribute values) when using a USE attribute!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>key</strong></th>
<th>accessType inputOutput, type MFFloat CDATA #IMPLIED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>determines parameters for linear interpolation function time intervals, in increasing order and corresponding to keyValues.</td>
</tr>
<tr>
<td>Host</td>
<td>number of keyValues must be an integer multiple of the number of keys!</td>
</tr>
<tr>
<td>Host</td>
<td>private keyValue integer multiple defines how many coordinates are sent in value_changed outputOnly.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>keyValue</strong></th>
<th>accessType inputOutput, type MFVec3F CDATA #IMPLIED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output values for linear interpolation, each corresponding to time fraction of interpolation.</td>
<td></td>
</tr>
<tr>
<td>Host: number of keyValues must be an integer multiple of the number of keys!</td>
<td></td>
</tr>
<tr>
<td>Host: keyValue key integer multiple defines how many coordinates are sent in value_changed outputOnly.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>set_fraction</strong></th>
<th>inputOnly type SFFloat CDATA #FIXED <strong>””</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>set_fraction sets input key for corresponding keyValue output.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>value_changed</strong></th>
<th>accessType outputOnly, type MFVec2F CDATA #FIXED <strong>””</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>LInely interpolated output value determined by current key time and corresponding keyValue pair.</td>
<td></td>
</tr>
<tr>
<td>Host: keyValue key integer multiple defines how many coordinates are sent in value_changed outputOnly.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>containerField</strong></th>
<th>XMTOKEN (&quot;children&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>containerField is the field-label prefix indicating relationship to parent node. Examples: geometry Box, children Group, proxy Shape. containerField attribute is only supported in XML encoding of X3D scenes.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>class</strong></th>
<th>CDATA #IMPLIED</th>
</tr>
</thead>
<tbody>
<tr>
<td>class is a space-separated list of classes, reserved for use by XML attributes; class attribute is only supported in XML encoding of X3D scenes.</td>
<td></td>
</tr>
</tbody>
</table>

http://www.web3d.org/x3d/content/X3dTooltips.html#CoordinateInterpolator2D
CoordinateInterpolator node

Generates $n$-tuple (multiple-valued floating point) array, MFFloat for `value_changed` output

*key* array contains $n$ SFFloat values

*keyValue* array contains $n$ MFFloat values

- As always: same number of *key, keyValue* entries
- Counting is very important for arrays of arrays!

CoordinateInterpolator computes weighted average between corresponding element pairs for each subarray in the *keyValue* array
Morphing images in 2D

Morphing 2D is the smooth transformation of one image into another using digital in-betweening

- Not a feature built into X3D

Typically this is done by identifying control points and gradually changing individual pixel colors from one 2D image into another

- Initial and final 2D images are already defined
- Corresponding control points are carefully chosen
- Special algorithms are applied
- Now a common photographic editing technique
Morphing models in X3D

Morphing 3D is the smooth transformation of one model into another using digital in-betweening

- Supported in X3D by CoordinateInterpolator node

Typically this is done by identifying control points and gradually changing coordinates from one set of values to another

- Linear interpolation value-by-value for each point
- Also usable for colors, normals, texture coordinates

We will use this technique to morph between 3D coordinate sets using CoordinateInterpolator

SIGGRAPH 1992. Image Morphing History. Morphing is the process of turning one image into another through a seamless transition. The following slideset by Thaddeus Beier and Shawn Neeley includes case study examination of Michael Jackson's “Black or White” which first applied this technique to music video,

http://www.cs.unc.edu/~lazebnik/research/fall08/qi_mo.ppt
Creating a morphable model

1. **Create baseline model geometry**
   - typically an IndexedFaceSet node

2. **Create alternate poses in key positions**
   - Ensure that same geometric layout is maintained
   - These are referred to as “key frames”
   - Can observe proper sequencing via a Switch node

3. **Use each set of point position values as part of a CoordinateInterpolator keyValue array**
   - Corresponding key array holds fraction durations, similar to any other interpolator node
CoordinateInterpolator node X3D-Edit

Case-study example: morphing a dolphin model
- Chris Lang, author
- Monterey High School class of 2008
- Models online at

https://savage.nps.edu/Savage/Biologics/Dolphin
Creating a morphable dolphin  

1. **Create baseline model geometry**
   - typically an IndexedFaceSet node

Chris Lang first built a dolphin model using Maya
   - advanced 3D modeling tool, commercial product
   - [http://autodesk.com > Products > Maya](http://autodesk.com)

Such models tend to be complex, perhaps saved in proprietary or perhaps-confusing formats
   - But can nevertheless be saved as X3D or VRML, making this approach reasonably repeatable
   - Can also use Aaron Bergstrom’s Rawkee tool

Rawkee is for Maya version 7.
   - Maya is a commercial authoring tool by Autodesk
   - [http://autodesk.com](http://autodesk.com)  [http://en.wikipedia.org/wiki/Maya_software](http://en.wikipedia.org/wiki/Maya_software)
   - Project Rawkee: Open-Source X3D Plugin for Maya by the Archaeology Technologies Laboratory (ATL) of North Dakota State University (NDSU).
   - [http://rawkee.sourceforge.net](http://rawkee.sourceforge.net)

Currently we simply save Maya models as VRML or X3D.
Creating a morphable dolphin

2. Create alternate poses in key positions
   • Ensure that same geometric layout is maintained
   • These are referred to as “key frames”
   • Can observe proper sequencing via a Switch node

The original model was then modified to match other poses, making sure each time that no new coordinate points were added or deleted
   • Each individually saved as X3D or VRML scenes
   • Switch cycler allows direct comparison of each
https://savage.nps.edu/Savage/Biologics/Dolphin/DolphinSwitcher.x3d
Chapter 7 - Event Animation and Interpolation

DolphinPose02.x3d  DolphinPose01.x3d  DolphinPose03.x3d

https://savage.nps.edu/Savage/Biologics/Dolphin/DolphinPose02.x3d
https://savage.nps.edu/Savage/Biologics/Dolphin/DolphinPose01.x3d
https://savage.nps.edu/Savage/Biologics/Dolphin/DolphinPose03.x3d

X3jD viewer wireframe mode is toggled with key Alt-w
Creating a morphable dolphin

3. Use each set of point position values as part of a CoordinateInterpolator keyValue array
   Corresponding key array holds fraction durations, similar to any other interpolator node

Define pose sequence for CoordinateInterpolator
   · 02 CurvedUpwardPose
   · 01 NeutralPose
   · 03 CurvedDownwardPose
   · 01 NeutralPose
   · 02 CurvedUpwardPose completes loop, then repeat
Creating a morphable dolphin

CoordinateInterpolator key array lists all five at equal time-fraction intervals:
• key='0 0.25 0.5 0.75 1'

Now need to build keyValue array
• Brute-force approach is then to copy set of array values from each <Coordinate point='...'/> pose

But note that each point sequence is quite long
• 508 points, meaning 1524 x-y-z values for each!!
• 5 arrays hold 7620 points total, 40% duplication
• Prefer 3 arrays, 4572 points, no array duplication
CoordinateInterpolator editor

Rows correspond to each key
Correct results have same number of MFVec3f values in each row
Point-by-point interpolation occurs on each column, allowing easier comparison
Modifying TimeSensor outputs

Two ways to achieve the same pose sequence

- Five poses: forward 02, 01, 03, 01, 02, repeat
- Three poses: forward 02, 01, 03, backwards, repeat

Build the second approach by modifying the TimeSensor fraction_changed output via a ScalarInterpolator

- TimeSensor output ramps from 0..1 linearly
- Five-pose CoordinateInterpolator uses 0..1 directly
- ScalarInterpolator output ramps 0 to 1 then back to 0
- Three-pose CoordinateInterpolator uses 0..1..0 instead
Modifying TimeSensor outputs

Can modify TimeSensor output by ROUTE connections through a ScalarInterpolator

- Change timing characteristic from 0..1 to 0..1..0

Resulting sequence of five poses:
- forward 02, 01, 03, backwards 03, 01, 02, repeat

Try it yourself: you might even reduce the CoordinateInterpolator further by removing the middle neutral pose. Is the resulting animation still satisfactory, or is it too jerky? You decide. (Thanks to Terence Tan for this suggestion.)
https://savage.nps.edu/Savage/Biologics/Dolphin/DolphinMorpher.x3d
**Chapter 7 - Event Animation and Interpolation**

<table>
<thead>
<tr>
<th>CoordinateInterpolator generates a series of Coordinate values that can be ROUTED to a &lt;Coordinate&gt; node’s <code>point</code> attribute or another Vector3fArray attribute. Typical input: ROUTE someTimeSource: fraction_changed TO someInterpolator.set_fraction. Typical output: ROUTE someInterpolator.value_changed TO destinationNode.set_attribute.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEF</strong></td>
</tr>
<tr>
<td>[DEF ID=&quot;#IMPLIED&quot;]</td>
</tr>
<tr>
<td>DEF defines a unique ID name for this node, referencable by other nodes.</td>
</tr>
<tr>
<td>Hint: descriptive DEF names improve clarity and help document a model.</td>
</tr>
<tr>
<td><strong>USE IDREF=&quot;#IMPLIED&quot;</strong></td>
</tr>
<tr>
<td>USE means reuse an already DEF-ed node ID, ignoring all other attributes and children.</td>
</tr>
<tr>
<td>Hint: USEing other geometry (instead of duplicating nodes) can improve performance.</td>
</tr>
<tr>
<td>Warning: do NOT include DEF (or any other attribute values) when using a USE attribute!</td>
</tr>
<tr>
<td><strong>key</strong></td>
</tr>
<tr>
<td>[key: accessType: input/output, type: MFFloat CDATA=&quot;#IMPLIED&quot;]</td>
</tr>
<tr>
<td>Definition parameters for linear interpolation function time intervals, in increasing order and corresponding to keyValues.</td>
</tr>
<tr>
<td>Hint: number of keyValues must be an integer multiple of the number of keys!</td>
</tr>
<tr>
<td>Hint: keyValues/key integer multiple defines how many coordinates are sent in value_changed outputOnly.</td>
</tr>
<tr>
<td><strong>keyValue</strong></td>
</tr>
<tr>
<td>[keyValue: accessType: input/output, type: MFVec3f CDATA=&quot;#IMPLIED&quot;]</td>
</tr>
<tr>
<td>Output values for linear interpolation, each corresponding to time-fraction keys.</td>
</tr>
<tr>
<td>Hint: number of keyValues must be an integer multiple of the number of keys!</td>
</tr>
<tr>
<td>Hint: keyValues/key integer multiple defines how many coordinates are sent in value_changed outputOnly.</td>
</tr>
<tr>
<td><strong>set_fraction</strong></td>
</tr>
<tr>
<td>[set_fraction: inputOnly, type: MFFloat CDATA=&quot;#IMPLIED&quot;]</td>
</tr>
<tr>
<td>set_fraction selects input key for corresponding keyValue output.</td>
</tr>
<tr>
<td><strong>value_changed</strong></td>
</tr>
<tr>
<td>[value_changed: accessType: outputOnly, type: MFVec3f CDATA=&quot;#IMPLIED&quot;]</td>
</tr>
<tr>
<td>Linearly interpolated output value determined by current key time and corresponding keyValue pair.</td>
</tr>
<tr>
<td>Hint: keyValue/key integer multiple defines how many coordinates are sent in value_changed outputOnly.</td>
</tr>
<tr>
<td><strong>containerField</strong></td>
</tr>
<tr>
<td>[containerField: NMTOKEN=&quot;#&quot; &quot;containerField&quot;]</td>
</tr>
<tr>
<td>containerField is the field-label prefix indicating relationship to parent node. Examples: geometryBox, childrenGroup, proxyShape. containerField attribute is only supported in XML encoding of X3D scenes.</td>
</tr>
<tr>
<td><strong>box</strong></td>
</tr>
<tr>
<td>[box CDATA=&quot;#IMPLIED&quot;]</td>
</tr>
<tr>
<td>box is a space-separated list of classes, reserved for use by XML eventSchema. class attribute is only supported in XML encoding of X3D scenes.</td>
</tr>
</tbody>
</table>

**http://www.web3d.org/x3d/content/X3dToolTips.html#CoordinateInterpolator**
Chapter Summary
Chapter Summary: Event Animation

Behaviors, events, ROUTE connections, animation

Animation as scene-graph modification

Event-animation design pattern: 10-step process

Interpolation nodes

- TimeSensor and event timing
- ScalarInterpolator and ColorInterpolator
- OrientationInterpolator, PositionInterpolator, PositionInterpolator2D and NormalInterpolator
- CoordinateInterpolator, CoordinateInterpolator2D
Suggested exercises

Illustrate and annotate ROUTE connections in an animation scene graph (documenting 10 steps)

• Print out one of these scenes in landscape mode, either using the X3dToXhtml.xslt stylesheet version or Netbeans 'Save as HTML' option.
• Then draw all ROUTE connections, label beginning and end of each by name, type and accessType

Draw animation chain diagrams to document behaviors in your own example scenes

• Add use-case summaries about user intent

Someday we hope to automate the production of such diagrams.

X3dToXhtml.xslt is available via X3D-Edit menu File, Export from X3D, Export as Annotated XHTML...
References
References  1

X3D: Extensible 3D Graphics for Web Authors  
by Don Brutzman and Leonard Daly, Morgan Kaufmann Publishers, April 2007, 468 pages.  
• Chapter 7, Event Animation and Interpolation  
• http://x3dGraphics.com  
• http://x3dgraphics.com/examples/X3dForWebAuthors

X3D Resources  
• http://www.web3d.org/x3d/content/examples/X3dResources.html
References 2

X3D Scene Authoring Hints
- http://x3dgraphics.com/examples/X3dSceneAuthoringHints.html

X3D Graphics Specification
- http://www.web3d.org/x3d/specifications
- Also available as help pages within X3D-Edit
References

  • http://www.wiley.com/legacy/compbooks/vrml2sbk/cover/cover.htm
  • http://www.web3d.org/x3d/content/examples/Vrml2.0Sourcebook
  • Chapter 08 – Animating Position Orientation Scale
  • Chapter 19 – Normals Shading

References  4

Wikipedia

- Double buffering
- Interlacing
  http://en.wikipedia.org/wiki/Interlace
- Event-based computing
Contact

Don Brutzman
brutzman@nps.edu
http://faculty.nps.edu/brutzman

Code USW/Br, Naval Postgraduate School
Monterey California 93943-5000 USA
1.831.656.2149 voice
CGEMS, SIGGRAPH, Eurographics

The Computer Graphics Educational Materials Source (CGEMS) site is designed for educators

- to provide a source of refereed high-quality content
- as a service to the Computer Graphics community
- freely available, directly prepared for classroom use
- http://cgems.inesc.pt

_X3D for Web Authors_ recognized by CGEMS! 😊

- Book materials: X3D-Edit tool, examples, slidesets
- Received jury award for Best Submission 2008

CGEMS supported by SIGGRAPH, Eurographics

From the CGEMS home page:

- http://cgems.inesc.pt

Welcome to CGEMS - Computer Graphics Educational Materials Source. The CGEMS site is designed for educators to provide a source of refereed high-quality content as a service to the Computer Graphics community as a whole. Materials herein are freely available and directly prepared for your classroom.

List of all published modules:

- http://cgems.inesc.pt/authors/ListModules.aspx

CGEMS Editorial Policy:

Creative Commons open-source license

http://creativecommons.org/licenses/by-nc-sa/3.0

Attribution-Noncommercial-Share Alike 3.0 Unported
You are free:

* to Share — to copy, distribute and transmit the work
* to Remix — to adapt the work

Under the following conditions:

* Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Attribute this work: What does "Attribute this work" mean?

The page you came from contained embedded licensing metadata, including how the creator wishes to be attributed for re-use. You can use the HTML here to cite the work. Doing so will also include metadata on your page so that others can find the original work as well.

* Noncommercial. You may not use this work for commercial purposes.

* Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

* For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

* Any of the above conditions can be waived if you get permission from the copyright holder.

* Nothing in this license impairs or restricts the author's moral rights.
Open-source license
for X3D-Edit software and X3D example scenes

http://www.web3d.org/x3d/content/examples/license.html

Copyright (c) 1995-2013 held by the author(s). All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

• Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
• Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
• Neither the names of the Naval Postgraduate School (NPS) Modeling Virtual Environments and Simulation (MOVES) Institute nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

License available at
http://www.web3d.org/x3d/content/examples/license.txt
http://www.web3d.org/x3d/content/examples/license.html

Good references on open source:
