X3D Animation
- Dynamic Scenes -

Felix G. Hamza-Lup, Ph.D
Associate Professor / Director NEWS Lab
Computer Science and Information Technology
Armstrong State University
Savannah, Georgia, USA
Outline

• X3D Animation Basics
  • Definitions: Behaviors, Routes, Events, Event Chains
  • Examples

• Animation Process
  • 10-Steps for Animation
  • Interpolation: Step-wise and Double Linear Interpolation
  • Time and Interpolator Nodes Examples

• Examples
Behaviors
Behavior Traversal of Scene Graph

• Once frame is swapped to update screen image need to update values in the scene

• **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

• Event **updates** modify the scene graph
  • Changing rendering properties, or
  • Generating further event outputs
Example of a Behavior Event Chain

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

Dr. Felix Hamza-Lup - Fulbright Lecture, Bangkok University 2015
Definitions (1)

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

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

- **ROUTE** statements connect *output of a field* of one node, to *an input of a field* of another node.
<Scene>
  <NavigationInfo type=""EXAMINE" "WALK" "FLY" "ANY"'' transitionType=""ANIMATE"''/>
  <Viewpoint description='Front View' orientation='0 1 0 0'/>

  <TimeSensor DEF='TIME' cycleInterval='5'/>

  <TouchSensor DEF='TOUCH' />

  <PositionInterpolator DEF='INTERP' key='0 0.25 0.5 0.75 1' keyValue='0 0 0 3 0 0 0 0 0 -3 0 0 0 0'/>

  <Transform DEF='BALL'>
    <Shape>
      <Appearance>
        <Material diffuseColor='1 0 0'/>
      </Appearance>
    </Shape>
  </Transform>

  <ROUTE fromNode='TOUCH' fromField='touchTime' toNode='TIME' toField='startTime'/>

  <ROUTE fromNode='TIME' fromField='fraction_changed' toNode='INTERP' toField='set_fraction'/>

  <ROUTE fromNode='INTERP' fromField='value_changed' toNode='BALL' toField='translation'/>
</Scene>
Definitions (2)

• **Event** is 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
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 `dataType` and `accessType` must both match between node/field of source and target
X3D 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
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
Field Data Types (1)  

<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 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>MFINTE32</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>
# Field Data Types (2)

<table>
<thead>
<tr>
<th>Field-type names</th>
<th>Description</th>
<th>Example values</th>
</tr>
</thead>
<tbody>
<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>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>“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>
</tbody>
</table>
**Field Data Types (3)**

<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;`**
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
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)

• Understanding naming conventions helps authors understand ROUTE definitions and results
Outline

• X3D Animation Basics
  • Definitions: Behaviors, Routes, Events, Event Chains
  • Examples

• Animation Process
  • 10-Steps for Animation
  • Interpolation: Step-wise and Double Linear Interpolation
  • Time and Interpolator Nodes Examples

• Examples
Animation Process - 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
Interpolating Animation Chains

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

• 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
Example

<Scene>
  <NavigationInfo type=""EXAMINE" "WALK" "FLY" "ANY"" transitionType=""ANIMATE""/>
  <Viewpoint description='Front View' orientation='0 1 0 0'/>
  <TimeSensor DEF='TIME' cycleInterval='5'/>
  <TouchSensor DEF='TOUCH' />
  <PositionInterpolator DEF='INTERP' key='0 0.25 0.5 0.75 1' keyValue='0 0 0 3 0 0 0 0 -3 0 0 0 0'/>
  <Transform DEF='BALL'>
    <Shape>
      <Appearance>
        <Material diffuseColor='1 0 0'/>
      </Appearance>
      <Sphere/>
    </Shape>
  </Transform>
  <ROUTE fromNode='TOUCH' fromField='touchTime' toNode='TIME' toField='startTime'/>
  <ROUTE fromNode='TIME' fromField='fraction_changed' toNode='INTERP' toField='set_fraction'/>
  <ROUTE fromNode='INTERP' fromField='value_changed' toNode='BALL' toField='translation'/>
</Scene>
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
Interpolating Animation Chains (5-6)

• 5. *Determine which Interpolator*. 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
Example

<Scene>
  <NavigationInfo type=""EXAMINE" "WALK" "FLY" "ANY" transitionType=""ANIMATE""/>
  <Viewpoint description='Front View' orientation='0 1 0 0'/>
  <TimeSensor DEF='TIME' cycleInterval='5'/>
  <Transform DEF='BALL'>
    <Shape>
      <Appearance>
        <Material diffuseColor='1 0 0'/>
      </Appearance>
      <Sphere/>
    </Shape>
  </Transform>
  <ROUTE fromNode='TOUCH' fromField='touchTime' toNode='TIME' toField='startTime'/>
  <ROUTE fromNode='TIME' fromField='fraction_changed' toNode='INTERP' toField='set_fraction'/>
  <ROUTE fromNode='INTERP' fromField='value_changed' toNode='BALL' toField='translation'/>
</Scene>

Dr. Felix Hamza-Lup - Fulbright Lecture, Bangkok University
2015
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
Example

<Scene>
  <NavigationInfo type="EXAMINE" "WALK" "FLY" "ANY"" transitionType="ANIMATE"/>
  <Viewpoint description='Front View' orientation='0 1 0 0'/>
  <TouchSensor DEF='TOUCH' />
  <PositionInterpolator DEF='INTERP' key='0 0.25 0.5 0.75 1' keyValue='0 0 3 0 0 0 0 -3 0 0 0 0'/>
  <Transform DEF='BALL'>
    <Shape>
      <Appearance>
        <Material diffuseColor='1 0 0'/>
      </Appearance>
      <Sphere/>
    </Shape>
  </Transform>
  <ROUTE fromNode='TIME' fromField='fraction_changed' toNode='INTERP' toField='set_fraction'/>
  <ROUTE fromNode='INTERP' fromField='value_changed' toNode='BALL' toField='translation'/>
</Scene>
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
Example

<Scene>
  <NavigationInfo type=""EXAMINE" "WALK" "FLY" "ANY"" transitionType=""ANIMATE""/>
  <Viewpoint description='Front View' orientation='0 1 0 0'/>
  <TimeSensor DEF='TIME' cycleInterval='5'/>
  <TouchSensor DEF='TOUCH' />
  <PositionInterpolator DEF='INTERP' key='0 0.25 0.5 0.75 1' keyValue='0 0 3 0 0 0 0 -3 0 0 0 0'/>
  <Transform DEF='BALL'>
    <Shape>
      <Appearance>
        <Material diffuseColor='1 0 0'/>
      </Appearance>
      <Sphere/>
    </Shape>
  </Transform>
  <ROUTE fromNode='TOUCH' fromField='touchTime' toNode='TIME' toField='startTime'/>
</Scene>

Dr. Felix Hamza-Lup - Fulbright Lecture, Bangkok University
2015
Animation chain for our example

\[ \text{TouchSensor} \rightarrow \text{TimeSensor} \rightarrow \text{PositionInterpolator} \rightarrow \text{Transform} \]

- \text{touchTime} \Rightarrow \text{startTime}
- \text{value\_changed} \Rightarrow \text{translation}
- \text{fraction\_changed} \Rightarrow \text{set\_fraction}

Dr. Felix Hamza-Lup - Fulbright Lecture, Bangkok University
2015
Demo

Animation.x3d
Example Animation Chains

- 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

Dr. Felix Hamza-Lup - Fulbright Lecture, Bangkok University
2015
# 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>SBool</td>
<td>Single-field boolean value</td>
<td>BooleanSequencer</td>
</tr>
<tr>
<td>SColor</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*
Interpolation is a method of constructing new data points within the range of a discrete set of known data points.
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>

X3D for Web Authors, Table 7.4, p. 197

Dr. Felix Hamza-Lup - Fulbright Lecture, Bangkok University 2015
Common Interpolator Fields

- \textit{key}, \textit{keyValue} hold the point values defining the characteristic function
- \textit{key} array always has type MFFloat
- \textit{keyValue} array data type matches the named type of the parent Interpolator node
- Final value must equal first value in \textit{keyValue} array if smooth looping is desired
- Lengths of \textit{key}, \textit{keyValue} arrays must be equal
- Note that \textit{keyValue} array can hold values which are themselves MF (multi-field) array type
- Function output \textit{value\_changed} always has the 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 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
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 tracks 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

```python
    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
```

Dr. Felix Hamza-Lup - Fulbright Lecture, Bangkok University
2015
TimeSensor fields 1

- *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

- `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
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.

See Example
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
Demo

Color.x3d
OrientationInterpolator node

- Generates a 4-tuple (four-valued orientation) SFRotation for `value_changed` output
  - `key` array contains SFFloat values
  - `keyValue` array contains SFRotation 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' orientation='0 0 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'/>
```
PositionInterpolator node

• Generates a 3-tuple (three-valued floating point) SFVec3f for `value_changed` output
  • `key` array contains SFFloat values
  • `keyValue` array contains SFVec3f values

• PositionInterpolator computes weighted average between corresponding x, y and z pairs in the `keyValue` array
  • ROUTE to Transform, either `translation` or `scale`
PositionInterpolator2D node

• Generates a 2-tuple (two-valued floating point) SFVec2f for value_changed output
  • key array contains SFFloat values
  • keyValue array contains SFVec2f values

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

Computer Graphics Effects
- Soft Shadows -
Normal Interpolator & Soft Shadows
NormalInterpolator node

- Generates a 3-tuple (three-valued floating point) SFVec3f for
  value_changed output
  - Key array contains SFFloat values

- 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
Demo

Normal.x3d