Advanced Animation in Mathematica

Operating on the Animation Scene Graph in XML (X3D)

Stewart Dickson

The motivation for this article lies in 3D computer graphics animation systems that were developed twenty to thirty years ago. The author particularly sees the connection between *Mathematica* and the animation system that ran in Genera Lisp on the Symbolics platform. This article presents a method for programmatically operating on an animation scene graph in *Mathematica* represented in the X3D extension of the Extensible Markup Language (XML). Below is presented historical and conceptual background on the animation scene graph and a contemporary example. Further motivation came from comparing the user interface design of a current, advanced animation system to an animation scene graph implemented in X3D and uploaded to Wolfram|Alpha Pro. Finally, this article presents an instance where an X3D scene graph was used in the "Fingerspelling Sign Language" Demonstration [1].

Mathematica organizes text, computable mathematical typesetting, and graphics as a notebook document. 3D computer graphics simulation, animation, and rendering systems organize graphics primitives so as to give the user a handle on the complexity required to model reality—a formidable task. An animation system, such as Autodesk Maya [2], presents complexity in a hierarchical way such that it can be selectively hidden, but subparts of it can be easily navigated to in order to manipulate the graphics primitives that make up the simulated 3D graphic scene.

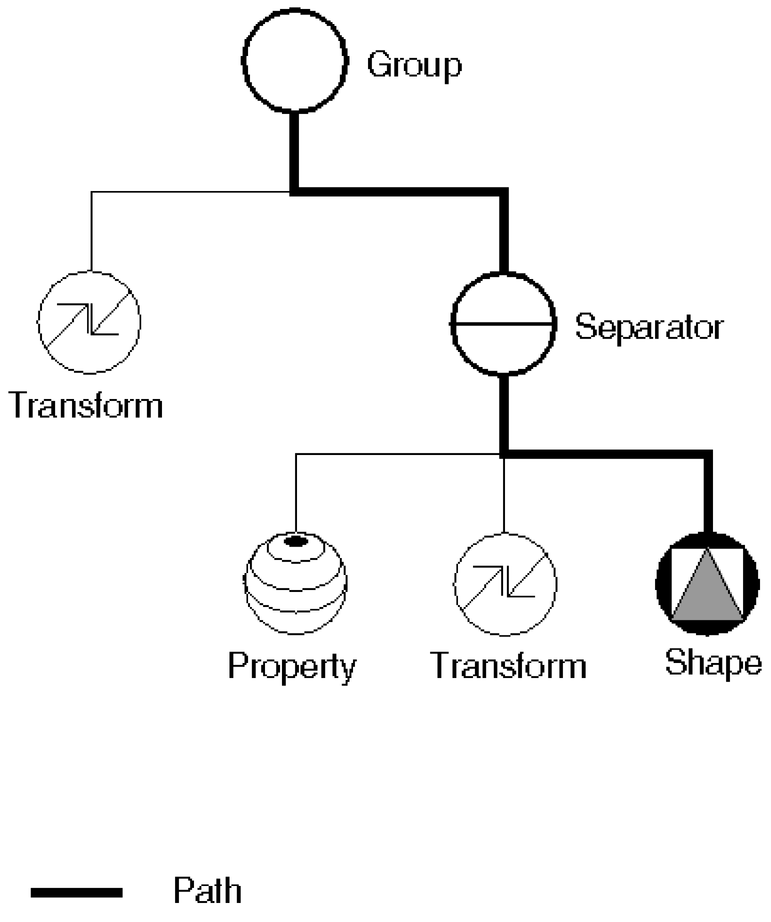
▲ **Figure 1.** Pt. Reyes [3, 4].

In *Mathematica* and in an advanced 3D graphics animation or simulation system, the task of representing 3D graphics primitives for manipulation and then passing them off to graphics hardware or to an external system for high-quality rendering is the same, but a high-performance animation system approaches the task differently from *Mathematica*. In the early 1990s, Silicon Graphics, Inc. (SGI) devised the scene graph as a hierarchical mechanism for organizing complex 3D graphics simulations. The scene graph model has been adopted by most powerful animation systems. But, expressions in Autodesk Maya—the language it provides to extend its capabilities to model real-world processes—are very limited. The high-performance animation system is optimized to manipulate complexity, but not to generate it.

The *Mathematica* language's ability to computationally model reality is absolutely unlimited. Circa 1993, the Symbolics S-Graphics package was widely thought to contain the best geometrical modeling system available for animation [5]. Symbolics spawned the Open Genera Lisp system and the Macsyma computer algebra system. The bottom-level string-rewriting system of *Mathematica* is like Lisp. *Mathematica*'s programming paradigm promotes use of (lambda) functions. I think that these parallels between *Mathematica* and a very advanced, albeit historically early, computer animation system should not be dismissed.

Advanced 3D graphics animation and simulation systems can benefit from *Mathematica*'s programming language and *Mathematica* graphics can benefit from the hierarchical organization of the animation scene graph. It is easy to get started using a scene graph in *Mathematica*. I used this technique in the animated hand model of the "Fingerspelling Sign Language" Wolfram Demonstration [1].

The scene graph is a basic structure of the SGI Open Inventor system [6]. (See Figure 2 and also the Coin3D project) [7]. These systems render graphic primitives interactively by traversing the scene graph (it is a tree structure) and emitting OpenGL code (a language understood by graphics hardware) [8] for the graph Shape node entities as they are traversed [9]. Graphics Shape primitives are animated by placing 3D coordinate Transform nodes ahead of them in the scene graph [10]. A transformation stack is accumulated as the traversal descends tree branches, and the stack of transforms is applied to the graphics shape primitives and emitted as OpenGL (or other RenderAction) directives as they are encountered [11].



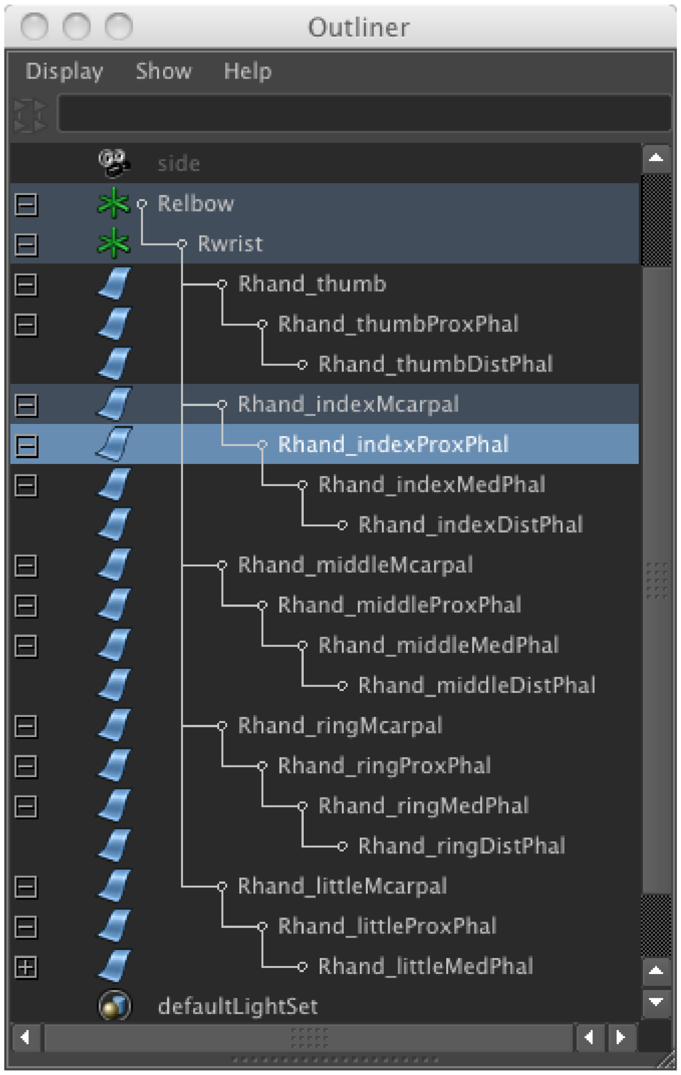
▲ **Figure 2.** "Example of a Scene Graph" (Chapter 1, Figure 1-2 , [6]).

Open Inventor 2.0 became the basis for the first version of the Virtual Reality Modeling Language (VRML 1.0) [12]; X3D [13] is the latest successor to VRML Version 2.0 [14] (1997) and it is implemented in XML. In Open Inventor, Separator [15] nodes are objects that inherit from the Group class (e.g., a Separator node is a kind of Group object) [16]. In VRML 2.0, Transform nodes also act implicitly as a Group mechanism in the hierarchy as well. Property nodes include Materials, which specify object surface illumination and appearance properties to be applied to subsequent Shape nodes in the scene graph tree.

If the Computable Document Format (CDF) is *Mathematica*'s answer to the Portable Document Format (PDF), then CDF organizes `Graphics3D` objects in it as a two-dimensional document. The X3D Document Object Model (DOM) [17] could be *Mathematica*'s answer to organizing `Graphics3D` objects in the animation scene graph to represent a virtual world.

The key to this observation was the work I did on the Wolfram|Alpha Pro XML file upload scanner [18]. As of the time of this writing, Wolfram|Alpha and the Wolfram Demonstrations Project are compatible with *Mathematica* 8. In this version, *Mathematica* does not yet have an X3D importer, but I have discovered that it is actually much more powerful to import an animation scene graph implemented in X3D using *Mathematica*'s quite excellent XML importer. All you have to do to use it is to rename the X3D file to use the .xml extension instead. The XML importer preserves the tree hierarchy of the XML Document Object Model (DOM) and allows you to traverse the X3D animation scene graph in *Mathematica* after the style of an advanced animation system such as Autodesk Maya.

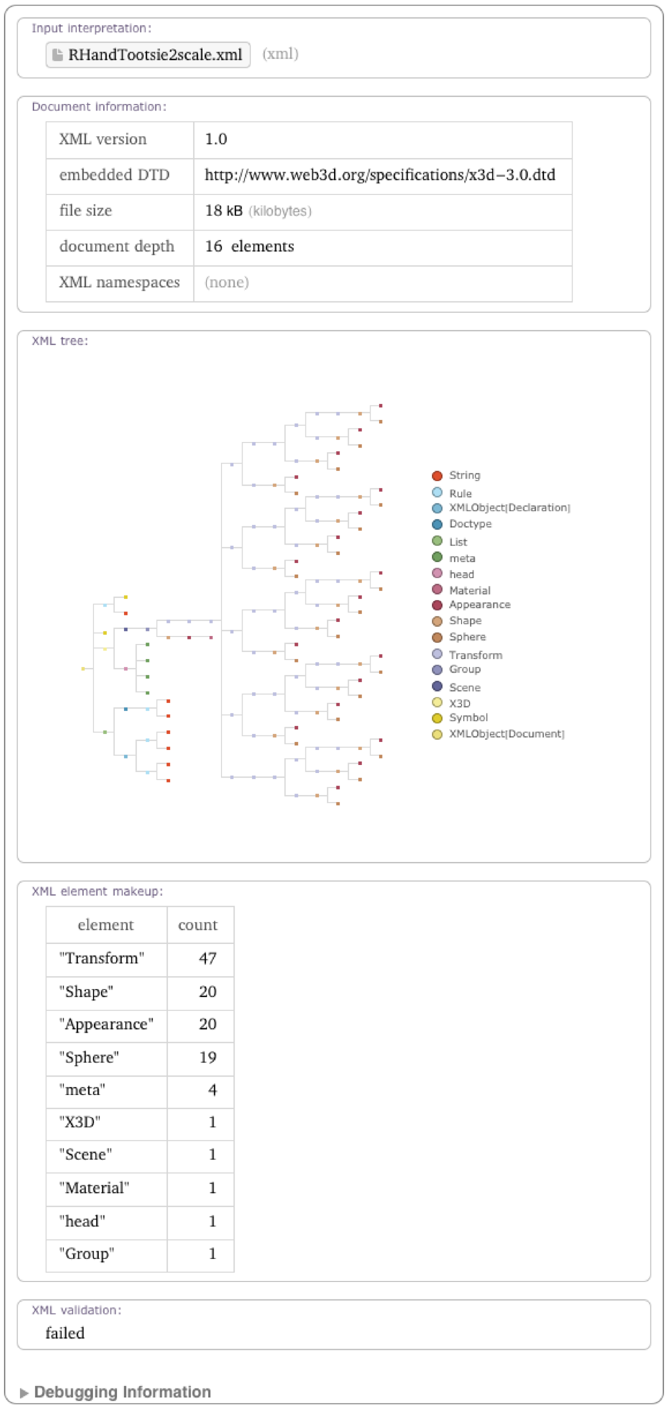
Figure 3 shows the Maya outliner user interface view of an animation rig to articulate a "sausage" hand model constructed of Non-Uniform Rational B-Spline (NURBS) sphere primitives. The tree structure represents the parent-child hierarchy of sphere geometry nodes and their 3D coordinate transforms. The nodes in the graph are selectable and, when selected, provide access to the Maya animation channel editor for the transform values (translation, rotation, scale) for each node.



▲ **Figure 3.** Outliner user interface in Autodesk Maya 2013.

Figure 4 shows the `ExpressionTreePlot` seen when you upload to Wolfram|Alpha Pro the X3D scene graph for the hand model of the "Fingerspelling Sign Language" Demonstration as an XML file. This file is supplied in the additional electronic files accompanying this article.

The Wolfram|Alpha XML Pro upload response shows an `ExpressionTreePlot`, restyled for Wolfram|Alpha, of a simplification of the XML DOM tree.



Input interpretation:

RHandTootsie2scale.xml (xml)

Document information:

XML version	1.0
embedded DTD	http://www.web3d.org/specifications/x3d−3.0.dtd
file size	18 kB (kilobytes)
document depth	16 elements
XML namespaces	(none)

XML tree:

XML element makeup:

element	count
"Transform"	47
"Shape"	20
"Appearance"	20
"Sphere"	19
"meta"	4
"X3D"	1
"Scene"	1
"Material"	1
"head"	1
"Group"	1

XML validation:

failed

▲ **Figure 4.** The response to uploading an X3D hand animation rig as XML to Wolfram|Alpha Pro.

The XML file itself is of the following form.

```
<?xml version=”1.0” encoding=”UTF-8”?>
…
<!DOCTYPE X3D PUBLIC “ISO//Web3D//DTD X3D 3.0//EN” …
…
<head>
…
</head>
<Scene>
 <Group>
 …
  <Transform DEF=’Relbow’ translation=’0.0 0.0 0.0’
                rotation=’0.0 0.0 1.0 0.0’>
   …
      <Shape DEF=’Rhand_thumbMcarpalGeom’>
       <Sphere DEF=’Rhand_thumbMcarpalSphere’/>
      </Shape>
   …
   </Transform>
  …
  </Group>
 </Scene>
</X3D>
```

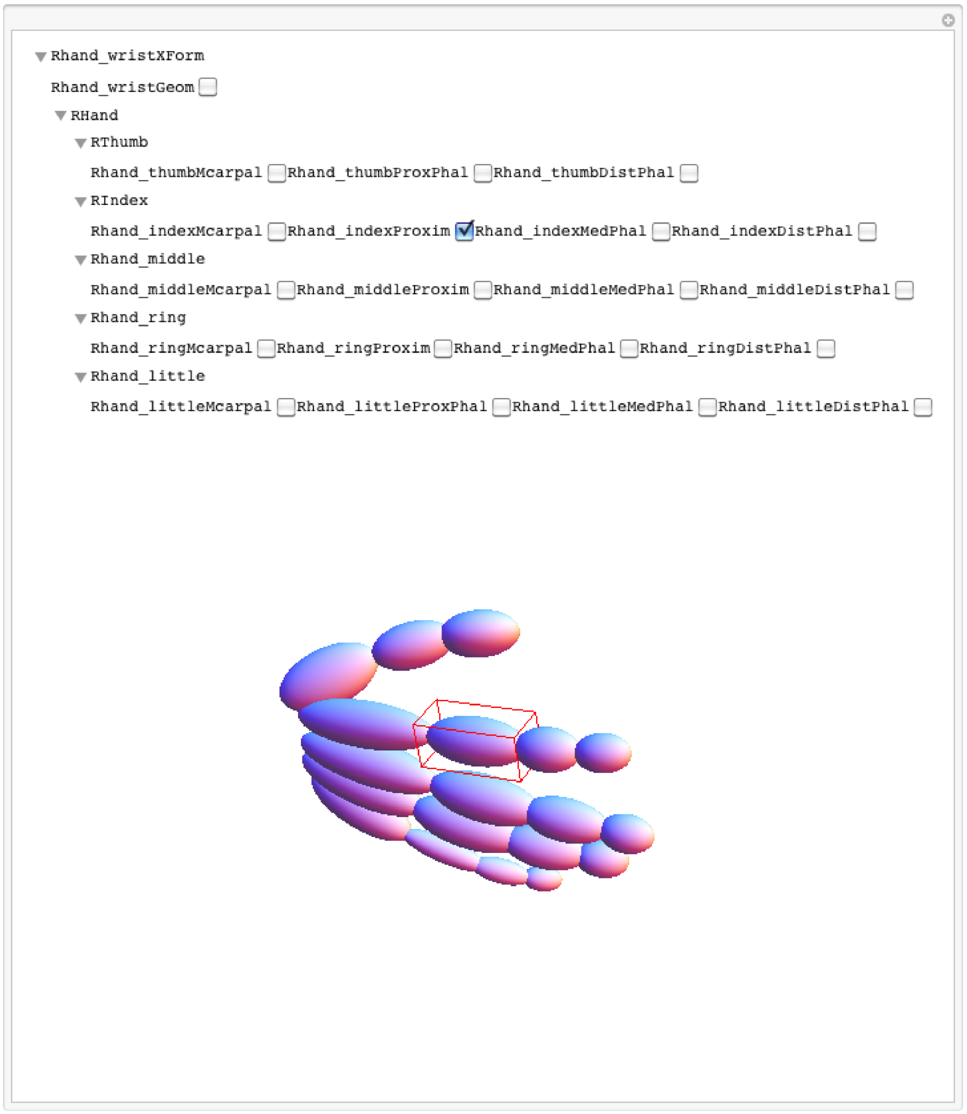
The XML should be imported into the *Mathematica* “Fingerspelling Sign Language” Demonstration notebook source as follows.

```
handXML = XMLObject["Document"][
    {
     XMLObject["Declaration"]["Version" -> "1.0",
      "Encoding" -> "UTF-8"],
     XMLObject["Doctype"]["X3D",
      "Public" -> "ISO//Web3D//DTD X3D 3.0//EN",
      …,
      XMLElement["X3D", {"profile" -> "Interchange",
        "version" -> "3.0",
        …,
        {XMLElement["head", {},
          …
           {}]}, …},
       …],
      XMLElement["Scene", {},
       {XMLElement["Group", {},
         …
          XMLElement["Transform", {"DEF" -> "Relbow",
            "translation" -> {0.0, 0.0, 0.0},
            "rotation" -> {0.0, 0.0, 1.0, 0.0}},
           …
            {XMLElement["Shape",
              {"DEF" -> "Rhand_thumbDistPhalGeom"},
              {XMLElement["Sphere", {}, {}], …, …}]}
          ]
        ]}
      ]
     ]
    }, {}, "Valid" -> True];
```

The `handXML` expression is paraphrased for readability. Please see the “Fingerspelling Sign Language” Demonstration notebook source for the complete expression. Where hierarchy in the XML file is denoted by <tag>…</tag>, hierarchy in the *Mathematica* `XMLObject[{…XMLElement[{…}] …}]` structure is denoted by bracketed lists. The complete structure of the `handXML` expression is depicted in the `Expres⋱ sionTreePlot` in the Wolfram|Alpha XML file upload result.

In order to generate `Graphics` from the XML object hierarchy, functions must be written to translate X3D Transform and Shape nodes into the *Mathematica* `Graphics3D` equivalents. I provide some of these functions in the downloadable source code for the “Fingerspelling” Demonstration. In the X3D file, the Transform and Shape nodes are all named using the "DEF" rule. In an advanced animation system like Maya, names carry a huge amount of information and are absolutely essential.

A node selection user interface for the Animated Hand Model of the “Fingerspelling” Demonstration using a hierarchy of `OpenerView` instead of a static `Grid` of `Checkbox` can be constructed automatically in *Mathematica* by recursively traversing the X3D DOM tree (scene graph) using `XMLObject`. The structural hierarchy required to articulate the hand bones through a kinematic transform stack is reflected in the structure of the node selection user interface. Compare these graphics to the Outliner node selection user interface from Autodesk Maya, which was built precisely for the task of managing kinematic animation on a complex rig.



```
▼ Rhand_wristXForm
  Rhand_wristGeom ☐
  ▼ RHand
    ▼ RThumb
      Rhand_thumbMcarpal ☐Rhand_thumbProxPhal ☐Rhand_thumbDistPhal ☐
    ▼ RIndex
      Rhand_indexMcarpal ☐Rhand_indexProxim ☑Rhand_indexMedPhal ☐Rhand_indexDistPhal ☐
    ▼ Rhand_middle
      Rhand_middleMcarpal ☐Rhand_middleProxim ☐Rhand_middleMedPhal ☐Rhand_middleDistPhal ☐
    ▼ Rhand_ring
      Rhand_ringMcarpal ☐Rhand_ringProxim ☐Rhand_ringMedPhal ☐Rhand_ringDistPhal ☐
    ▼ Rhand_little
      Rhand_littleMcarpal ☐Rhand_littleProxPhal ☐Rhand_littleMedPhal ☐Rhand_littleDistPhal ☐
```

▲ **Figure 5.** A node selection user interface for the Animated Hand Model of the “Fingerspelling” Demonstration using a hierarchy of `OpenerView`.

See also the Demonstrations “3D Skeletal Anatomy of the Arm” and “3D Skeletal Anatomy of the Torso” [19, 20], which use a similar `Graphics3D` node selection user interface.

In the “Fingerspelling” Demonstration, I stored animation channels for keyframes as lists of node name and channel value rules.

```
base = {};
anim = {“Rwrist_PITCH” → 0, "Rwrist_ROLL" → 0,
   "Rhand_thumbPalm" → -0.46705,
                     "Rhand_thumbIndex" → -0.696228,
   …, “Rhand_littleNear” → 0.2,
                     "Rhand_littleMedial" → 0.2,
   "Rhand_littleJoint3" → 0.21};

(* Joint rotation angles *)
```

Similarly, the node selection state is keyed by the Shape node name. The `staticSelection` list contains twenty `Name` → `value` rules in the “Fingerspelling” Demonstration source, paraphrased here for readability.

```
staticSelection = {
   "Rhand_wristGeom" → False,
   "Rhand_thumbMcarpalSphere" → False,
   "Rhand_thumbProxPhalSphere" → False,
   "Rhand_thumbDistPhalSphere" → False,
   "Rhand_indexMcarpalSphere" → False,
   "Rhand_indexProximSphere" → True,
   …,
   "Rhand_littleMedPhalSphere" → False,
   "Rhand_littleDistPhalSphere" → False
  };
```

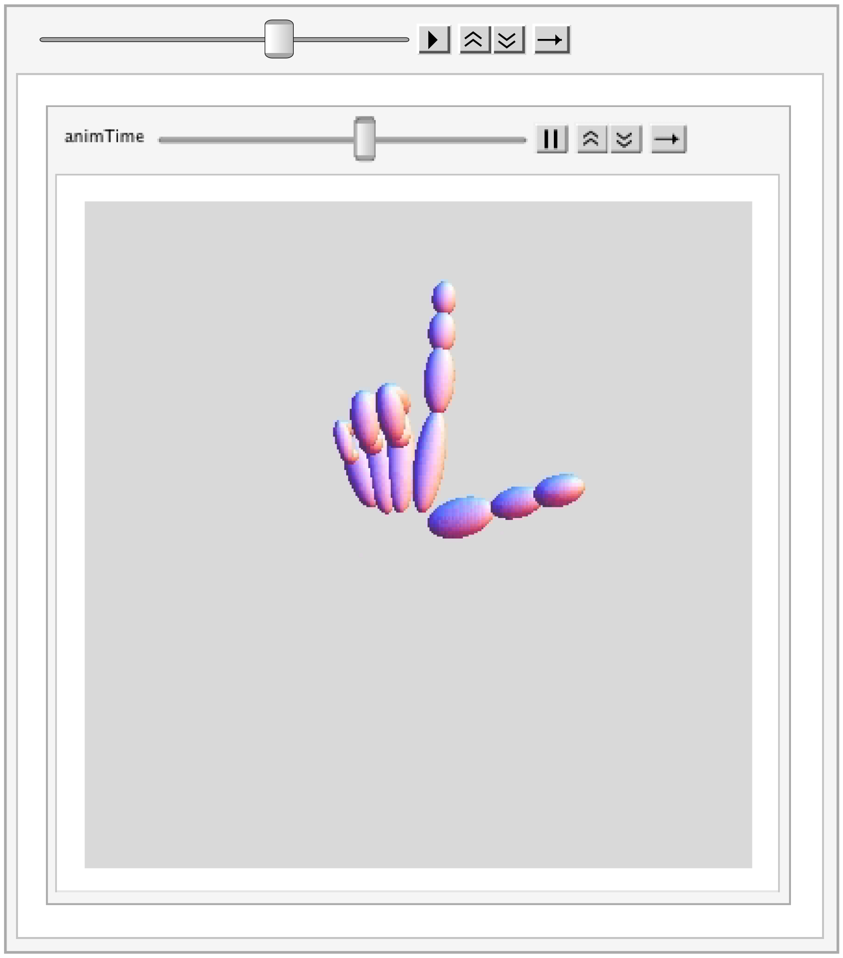
I apply the animated transform values to the X3D scene graph as imported into *Mathematica* using `ApplyXFormStack[element_XMLElement, xForm_List, anim_List, anim_List, selection_List]`, a recursive function. Then I use `collectTransforms` to collect a stack of geometric coordinate transforms from the XML transform elements descending a branch. Then I apply `Sow` to a function that applies *Mathematica* geometric transforms to every XML Shape node encountered. To show the `Graphics3D`, I apply `Reap` to the results of `ApplyXFormStack` as follows.

```
handGraphic = Graphics3D[
              Flatten[DeleteCases[Flatten[Reap[

      ApplyXFormStack[#, base, anim, staticSelection] & /@
                     Cases[handXML, XMLElement[___], 1]], 1],
    Null], 1],
              Boxed -> False]
```

Again, `ApplyXFormStack` calls `collectTransforms` descending the branches of the scene graph tree—analogous to glPushMatrix of OpenGL—and applies that transform stack to each Shape node it traverses. In the “Fingerspelling” Demonstration, we do ease-in/ease-out interpolation between `anim` lists containing keyframe poses corresponding to letters of the alphabet, then apply the interpolated rotation angle value to the joint transform of the same name.

Reading X3D as XML fully exposes all of the security vulnerabilities inherent in importing XML files into *Mathematica*. The *Mathematica* XML importer initially reads all of the Graphics Coordinate, Coordinate Index, and Transform fields as strings, which then must be converted to lists of numerical values using either `ToExpression` or `Read`. It is easy to encode expressions into an XML file that can do damage to a computer execution environment or file system when *Mathematica* runs `Read` or `ToExpression` on them. For similar reasons, `Read` and `ToExpression` are prohibited from Wolfram Demonstration notebooks.



▲ **Figure 6.** An animated clip from the “Fingerspelling” Demonstration.

In "Fingerspelling Sign Language Using an Animated Hand Model," keyframed animation curves are simple, sinusoidal ease-in/ease-out moves, whereas Maya uses cubic spline curves to interpolate animation channels. But the key point here is what is required to organize and animate multiple `GraphicsComplex` subparts of a complex `Graphics3D` object in *Mathematica*. In my opinion, the X3D incarnation of VRML represented in the XML DOM tree is an excellent model for doing this.

■ Acknowledgments

Thanks to George Fann, who first suggested that I look at X3D.

■ References

[1] S. Dickson and S. Martell. "Fingerspelling Sign Language Using an Animated Hand Model" from the Wolfram Demonstrations Project—A Wolfram Web Resource. www.demonstrations.wolfram.com/FingerspellingSignLanguageUsingAHandModel.

[2] Autodesk. "Maya." (Feb 1, 2013) usa.autodesk.com/maya.

[3] Wikipedia. "Reyes Rendering." (Feb 1, 2013) en.wikipedia.org/wiki/Reyes_rendering.

[4] R. Cook, L. Carpenter, T. Porter, B. Reeves, D. Salesin, and A. R. Smith. "Pt. Reyes." (Feb 14, 2013) alvyray.com/Art/PtReyes.htm.

[5] Wikipedia. "Symbolics." (Feb 1, 2013) en.wikipedia.org/wiki/Symbolics.

[6] J. Wernecke, "The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor™, Release 2," Redwood City, CA: Addison-Wesley, 1994. techpubs.sgi.com/library/dynaweb_docs/0630/SGI_Developer/books/Inv_Mentor/sgi_html/index.html.

[7] Kongsberg Oil & Gas Technologies. "Coin3D." (Feb 1, 2013) bitbucket.org/Coin3D/coin/wiki/Home.

[8] Silicon Graphics, Inc. "OpenGL: The Industry's Foundation for High Performance Graphics." (Feb 1, 2013) www.opengl.org.

[9] Kongsberg Oil & Gas Technologies. "SoShape Class Reference [Node Classes]." (Feb 1, 2013) coin3d.bitbucket.org/Coin/classSoShape.html.

[10] Kongsberg Oil & Gas Technologies. "SoTransform Class Reference [Node Classes]." (Feb 1, 2013) coin3d.bitbucket.org/Coin/classSoTransform.html.

[11] Kongsberg Oil & Gas Technologies. "SoGLRenderAction Class Reference [Action Classes]." (Feb 1, 2013) coin3d.bitbucket.org/Coin/classSoGLRenderAction.html.

[12] VRML.org. "Virtual Reality Modeling Language." (Feb 18, 2013) web.archive.org/web/20121017095254/http://www.vrml.org.

[13] Web3D Consortium. "X3D Developers." (Feb 4, 2013) www.web3d.org/x3d.

[14] A. Ames, D. Nadeau, and J. Moreland, *VRML 2.0 Sourcebook*, New York: John Wiley & Sons, 1996. www.web3d.org/x3d/content/examples/Vrml2.0Sourcebook/index.html.

[15] Kongsberg Oil & Gas Technologies. "SoSeparator Class Reference [Node Classes]." (Feb 4, 2013) coin3d.bitbucket.org/Coin/classSoSeparator.html.

[16] Kongsberg Oil & Gas Technologies. “SoGroup Class Reference [Node Classes].” (Feb 4, 2013) coin3d.bitbucket.org/Coin/classSoGroup.html.

[17] w3schools.com. “XML DOM Tutorial.” (Feb 5, 2013) www.w3schools.com/dom/default.asp.

[18] Wolfram Research. “WolframlAlpha Pro XML File Upload Example.” (Feb 4, 2013) www.wolframalpha.com/input/?i=+&examplefile=1&lk=3&fileinput=FileUpload%2FAnimated Methane.xml.

[19] S. Dickson. “3D Skeletal Anatomy of the Arm” from the Wolfram Demonstrations Project—A Wolfram Web Resource. demonstrations.wolfram.com/3DSkeletalAnatomyOfTheArm.

[20] S. Dickson. “3D Skeletal Anatomy of the Torso” from the Wolfram Demonstrations Project—A Wolfram Web Resource. demonstrations.wolfram.com/3DSkeletalAnatomyOfTheTorso.

S. Dickson, “Advanced Animation in *Mathematica*,” *The Mathematica Journal*, 2013.
dx.doi.org/doi:10.3888/tmj.15-2.

List of Additional Material

Additional electronic files:

1. www.mathematica-journal.com/data/uploads/2013/02/RHandTootsie2scale.xml

About the Author

Stewart Dickson was a programmer of 3D computer graphics and animation for broadcast video, theatrical film, digital cinema, and location-based entertainment from 1984 to 2002. He was a pioneer in digital sculpture from 1989. He has worked at Walt Disney Feature Animation, Oak Ridge National Laboratory, and the NOAA National Climatic Data Center. He has worked at Wolfram Research since 2011.

Stewart Dickson
Wolfram Research, Inc.
100 Trade Center Drive
Champaign, IL 61820-7237
sdickson@wolfram.com