NISTIR 7827

Integrating jQuery with the 3D Descriptive Markup of X3DOM

Sandy Ressler

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 7827

Integrating jQuery with the 3D Descriptive Markup of X3DOM

Sandy Ressler
Applied and Computational Mathematics Division
Information Technology Laboratory

October, 2011

U.S. Department of Commerce
John E. Bryson, Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Under Secretary of Commerce for Standards and Technology and Director

Integrating jQuery with the 3D Descriptive Markup of X3DOM

Sandy Ressler*
National Institute of Standards and Technology

Abstract

This paper describes a number of techniques using jQuery to take advantage of the 3D descriptive markup implemented by X3DOM [Behr et al. 2009]. X3DOM is a project of the Web3D Consortium, primarily being implemented by staff at Fraunhoffer IGD to provide 3D descriptive markup. Using this markup embeds 3D directly into HTML such that a web browser can render 3D scenes without resorting to plug-ins. Given 3D descriptive markup this paper demonstrates a number of techniques that take advantage of having the 3D geometry and scene graph represented directly in a web browser. Placing a 3D scene graph into the browsers DOM (Document Object Model) offers the opportunity to leverage frameworks such as jQuery in powerful ways.

CR Categories: I.3.7 [Computer Graphics]: Web3D—X3D; I.3.6 [Methodology and Techniques]: Standards—Languages

Keywords: X3D, X3DOM, jQuery, javascript, DOM, HTML5

1 Background on X3D, X3DOM, jQuery, HTML, the DOM and all those Acronyms

X3D, the ISO standard for representing 3D virtual environments and is the successor to VRML (Virtual Reality Modeling Language). X3DOM [Behr et al. 2009] is a project to take X3D, 3D expressed with HTML or XML, and place the X3D graphics directly into the HTML document. The in-memory representation of the HTML document is called the DOM (Document Object Model) and is analogous to a typical scene-graph representation of 3D graphic geometry. X3DOM merges the 3D scene-graph represented by X3D with the DOM. OK, so what? Well given that there is a huge and growing collection of infrastructure for dealing with the Web and Web applications, one might attempt to take advantage of some of that infrastructure. It should also be noted that another effort towards defining 3D descriptive markup called XML3D [Sons et al. 2010] is also underway and there is no reason that the techniques described in this paper shouldn't work with it, we however did not test this assumption due to time constraints.

jQuery [Reisig 2010] is a Javascript framework that is very popular and more importantly makes Javascript into a robust language suitable for non-trivial applications. jQuerys power derives from its ability to let you succinctly select portions of the DOM, and then operate on them. Similarly we can use jQuery to select portions of the DOM representing 3D objects and perform operations on them. Related to jQuery is a project called jQuery UI (User Interface) which is a collection of jQuery plugins that provides a collection of robust widgets enabling the creation of highly interactive web pages. Throughout the rest of this paper we will refer both to jQuery and jQuery UI as simply jQuery. We also demonstrate the integration of some of these widgets with X3DOM to illustrate additional leveraging of the rich internet development infrastructure applied to 3D environments expressed as 3D descriptive markup. Figure 1 illustrates an integrated 3D scene with jQuery buttons, sliders, dialogs and an accordion interface.

*e-mail: sressler@nist.gov

2 Automatic Buttons

Let's examine how to leverage jQuery's button widget to automatically create one button for each of the viewpoints in our scene. In the AnthroGloss [Ressler 2011] X3DOM world, there is a collection of 74 body landmarks and a few other general overview viewpoints. We want to create a button for each of the viewpoint elements, such that clicking on the button moves us to the new viewpoint. The X3DOM description of a viewpoint looks like:

```
<viewpoint orientation='x, y, z' position='x, y, z'>
</viewpoint>
```

The usual mantra for jQuery coding is to select something then do something. The jQuery selectors are a straightforward derivation of CSS (Cascading Style Sheet) selectors. To select all of the viewpoint elements in the document we simply code:

```
$('viewpoint')
```

One technique to dynamically create a collection of buttons is to clone them from a single button. [Griefer 2009]

We iterate over all of the viewpoint elements, and create a new button via the .clone() function and place that new button in the appropriate place in the DOM. (See cloneViewpointButtons code in Appendix) The result is a collection of buttons, one for each of the viewpoints that enables the user to navigate the scene via buttons. In this case the buttons also sit inside of an accordion frame, utilizing the accordion UI elements provided by jQuery. The buttons in Figure 1 were generated automatically.

3 Sliding through the Viewpoints

Another useful UI widget is the slider. It is typically a horizontal widget with a control handle that enables the user to smoothly select from a minimum value to a maximum. Lets examine how we can use the slider to select from the first viewpoint to the last and update the display.

The slider widget itself must appear somewhere in the DOM so we must make an entry in the HTML something like:

```
<div id='viewpointSlider'></div>
```

The details of setting up a slider can be found at http://jqueryui.com/demos/slider/. The only X3DOM specific aspect of it that you must set is the 'set_bind' attribute of the viewpoint element. Set it to true for the scene to actually move to the viewpoint. We again take advantage of jQuerys selection mechanism to select all the viewpoints, via the dollar viewpoint code and iterate through all the viewpoints, this time via the slider widget (see slider code in Appendix) rather then making individual buttons as described earlier.

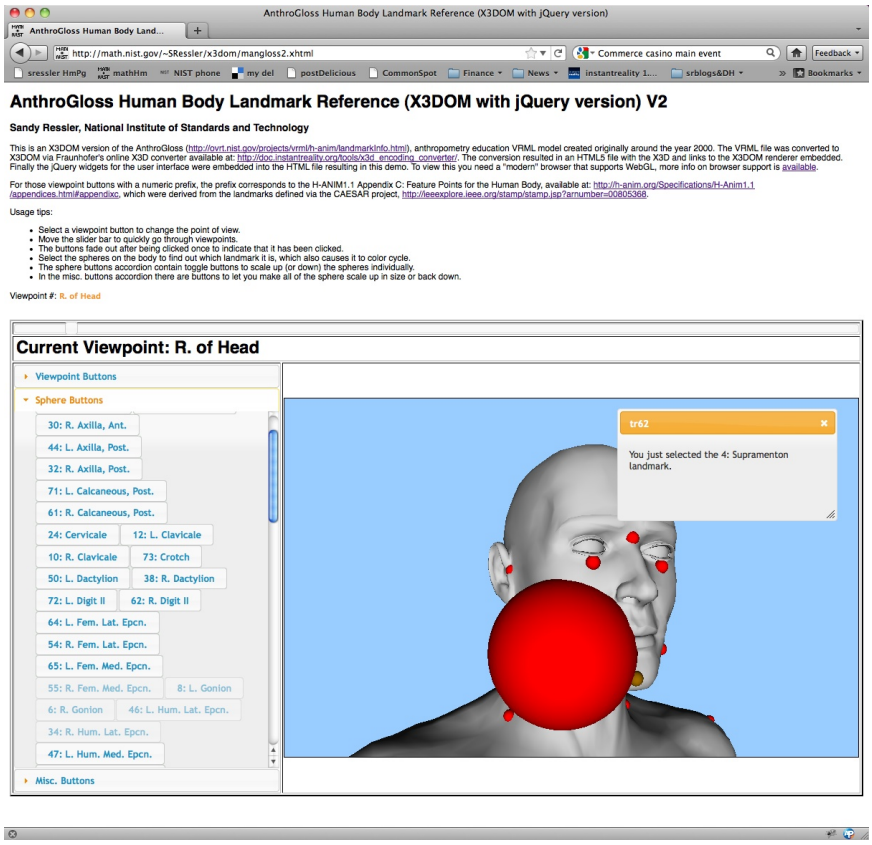

Figure 1: *Web page with X3DOM including jQuery UI elements*

Viewpoints are a type of element in X3DOM that are 'bindable' and only one viewpoint can be active at any one point in time. This makes perfect sense as the viewpoint represents the virtual camera, the point of view of the user and there can only be one active at any onc instant in time. To make a viewpoint active set its set_bind attribute to true, which also places it on top of the Viewpoint stack. This is however handled automatically by the X3DOM software, as a developer you simply need to set the set_bind attribute to TRUE.

4 Operating on Geometry

Just as we can operate on viewpoints, a sort of environmental characteristic of an X3D scene, we can operate on the geometric elements as well. Again using the AnthroGloss world as an example, lets imagine that we want to modify the size of the spheres for the body landmarks. If we change the radius or anything else that is part of the <sphere >element we are unfortunately out of luck. In its current version the X3DOM software creates the scene graph and the actual structure and attributes of the geometry is fixed until another reload of the page. However we are not totally out of luck as the software does let you change attributes of a surrounding 'transform' or a sibling 'material' node which effectively lets you modify quite a bit in the world.

Imagine again that we want to create a set of new buttons, this time however we want one for each of the <sphere >elements in the scene. Not surprisingly this functionality can be achieved in exactly the same way that we created viewpoint buttons. Simply select all the sphere elements then iterate through and clone a set of sphere buttons. Figure 2 below illustrates the result of a global operation to modify the scale of all spheres via clicking on a single button which executed the code:

```
$('#sphScaleUp').click(function() {
    $('sphere').parent().parent().attr('scale','10 10 10');
});
```

This code says that when we click on the element with an id sphScaleUp (which happens to be a button) execute the code defined in the click function. The click function says, for each sphere element $(sphere), travel up two DOM levels (the parent().parent() portion and change the scale attribute to 10, 10, 10. This code depends on the spheres being structured in a specific way (see jQuery plugin section description for details).

5 Making Geometry Interactive

We can introduce interactivity into the scene using two fundamentally different methods. First we can associate HTML onclick events into the X3D scene graph. Second we can utilize the native X3D event model to control animations and other behaviors of the 3D scene.

5.1 Using the HTML onclick Event

Web browsers are really event driven systems. Code such as Javascript contains an event model in which different user, or computational actions result in an event. When the event fires we execute a callback function to respond to the event. X3D also contains an event model and the integration of the X3D events with HTML events is still a work in progress for X3DOM, however we can begin to use both quite effectively. First let's examine the use of the onclick event, which is fired when a mouse clicks on something of interest. In a typical HTML button we would see code such as:

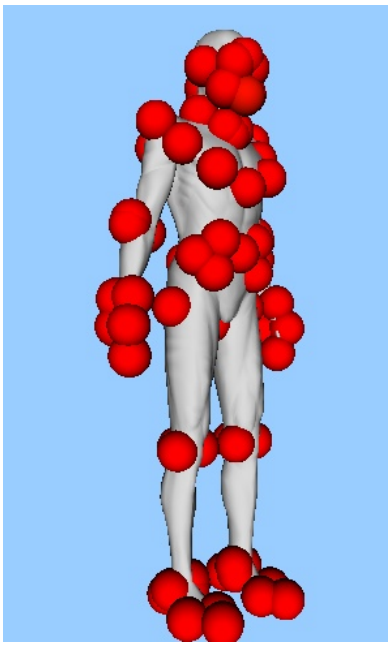

Figure 2: *Globally Modifying Geometry (spheres)*

```
<button type="button">Click Me!</button>
```

this will render as a button which does nothing when clicked. To cause it to take an action we could do something like:

```
<button type= button  onclick= alert( ouch! ) >Click Me</button>
```

which would cause an alert dialog to appear with the word ouch! in the dialog. Again given our descriptive markup we could create markup such as:

```
<sphere onclick='you clicked a sphere'></sphere>
```

Associating popup dialogs with complicated scenes is useful for interaction. This popup dialog is illustrated in Figure 1, informing the user about name of the landmark selected.

In addition we can use jQuery again to select all of the sphere elements and define a behavior for clicking all of the spheres. In this case we modify the message of the dialog based on other (viewpoint) descriptions in the attributes of the 3D descriptive markup as follows:

```
onclick=
 "$('#dialog').html('&lt;p&gt; You just selected the' +
 $('#vp" + x + "').attr('description') +
 ' landmark. &lt;/p&gt;');
 $('#dialog').dialog('open');
 $('#dialog').dialog( {title:
 $(this).parent().parent().attr('id') });"
```

5.2 Using X3D Events

X3D can represent not just geometry but can represent interactive elements and animation. It contains its own event model and the ability to specify interaction as well as the timing of events in a scene. In fact an active area of development for the X3DOM developers is exactly how to map and integrate the two event models, a still to be completed task. However we can use an event in the DOM to trigger an event in X3D. For example we can click on a button or a 3D object, use the onclick event to trigger an X3D TimeSensor which will send events to geometry via the X3D ROUTE mechanism.

If we define an onclick of a <sphere >to be:

```
onclick= $( #clock5 ).attr( cycleInterval , 2.0)
```

When the sphere is clicked, the cycleInterval attribute of the element with the id clock5 will be fired. This assumes of course that clock5 is the id of a TimeSensor node. We then link up the changing values of the TimeSensor (via the fraction_changed attribute) via two ROUTES to a ColorInterpolator in order to create a color cycling sphere. Simply put, you click the sphere and it starts cycling through colors.

Again taking advantage of jQuery we now have a method to easily duplicate the functionality of a color cycling sphere. We clone the elements of a ColorInterpolater the same way we cloned buttons for viewpoints and can have jQuery effectively create a great many TimeSensors and ColorInterpolators by executing a javascript function rather then us explicitly authoring X3D. It is a technique much more efficient and less prone to error. We created a template for the color cycling code that is as follows:

```
<group id="colorInterpTemplate">
   <ColorInterpolator id='colorTemplate'
      keyValue='1 0 0, 0 1 0, 0 0 1, 1 0 0'
      key='0.0 0.333 0.666 1.0'/>
   <TimeSensor class='timer' id='clockTemplate'
      cycleInterval='0' loop='true'/>
    <ROUTE id="clockTemplateR1" fromNode='clockTemplate'
       fromField='fraction_changed' toNode='colorTemplate'
       toField='set_fraction'/>
    <ROUTE id="colorTemplateR2" fromNode='colorTemplate'
       fromField='value_changed' toNode='matTemplate'
       toField='diffuseColor'/>
</group>
```

Next we create a javascript function which operates on all spheres since they were all made part of the class ball.

```
function cloneColorInterp() {
// for each transform of the class ball
//create new color interps and place in group
//as child of transform
$(".ball").each(function(index) {

  // clone the group
newElem = $('#colorInterpTemplate').clone()
 .attr('id','colorInterp'+index);
$(newElem).children("colorinterpolator").
 .attr('id','color'+index);
$(newElem).children("timesensor")
 .attr('id','clock'+index);
$(newElem).children("ROUTE:first")
 .attr('id','clockR1_'+index);
$(newElem).children("ROUTE:first")
 .attr('fromNode','clock'+index);
$(newElem).children("ROUTE:first")
 .attr('toNode','color'+index);
$(newElem).children("ROUTE:eq(1)")
 .attr('id','clockR2_'+index);
$(newElem).children("ROUTE:eq(1)")
 .attr('fromNode','color'+index);
$(newElem).children("ROUTE:eq(1)")
 .attr('toNode','mat'+index);
$(this).after(newElem); // place the new group
   });
};
```

6 Developing a jQuery Plugin to Modify Geometry and Interaction

Last, but not least, let's conclude our jQuery exploration by creating a jQuery plugin that lets the user perform more powerful operations on the X3D geometry. jQuery plugins are a technique to extend the capabilities of jQuery in a seamless manner. It would be useful to create a plugin that enables us to turn small pieces of geometry into interactive 'buttons', and modify some of the geometric characteristics such as position or color. We can of course simply edit the X3D description itself adding 'onclick' and 'diffuseColor' information as follows:

```
<transform id='tr0' class="ball"
translation='5 0 0' scale= 5 1 5
onclick= alert( hey you hit me! ) >
   <shape>
    <appearance>
     <material id='mat0' diffuseColor='0.9 0 0.4'>
     </material>
     </appearance>
    <sphere radius='0.26' ></sphere>
   </shape>
</transform>
```

The result is perfectly good X3D code however editing a large quantity of geometry is not a pleasant task, and is prone to error. A plugin allows us to modify many pieces of geometry simultaneously, is easier to maintain, and is clearly more useful. The plugin we developed, X3DOMgeomButton, in fact lets us create these geometric buttons along with modifications to a few geometric parameters. An example usage of the plugin is as follows:

```
// associates an onclick event to the
//selected element (in this case a single id)
$( #tr0 ).X3DOMgeomButton({ onclick:
  alert(Ouch you hit me! )  });
// associates an onclick and diffuse
//color with all items of the class  ball
$( .ball ).X3DOMgeomButton({ onclick:
 alert(Ouch you hit me! ) ,
diffuseColor: 1 0 0  })
```

The code for the plugin is as follows:

```
// X3DOMgeomButton plugin
(function($) {
  $.fn.X3DOMgeomButton = function(options) {
      // Extend our default options with those provided.
      var opts =
         $.extend({},$.fn.X3DOMgeomButton.defaults, options);
      // iterate  each matched element
      return this.each(function() {
      var base = $(this);
      //var attribute = base.attr(opts);
      var scaleOpt = opts.scale;
      var onclickOpt = opts.onclick;
      var diffuseColorOpt = opts.diffuseColor;
      var translationOpt = opts.translation;

      if (scaleOpt) base.attr('scale',scaleOpt);
      if (onclickOpt) base.attr('onclick',onclickOpt);
      if (translationOpt)
         base.attr('translation',translationOpt);
      // base is the transform then -> shape -> app -> mat
      if (diffuseColorOpt) base
          .children().children().children()
          .attr('diffuseColor',diffuseColorOpt);
       });
     };
  $.fn.X3DOMgeomButton.defaults = {
     scale: '10 10 10',
     onclick: 'alert("I am hit");',
     diffuseColor: '0 1 0'
   };
 })(jQuery);
```

We can use the selection methods of jQuery to carefully and explicitly add functionality to the geometry of a scene, by using jQuery plugins designed to operate on 3D descriptive markup. X3DOMgeomButton is a toy example but demonstrates yet another advantage of using descriptive 3D markup. See Appendix for full listing of X3DOMgeomButton code.

7 Summary and Conclusions

In this paper we demonstrate a number of techniques to take descriptive 3D markup, instantiated via the X3DOM software and illustrate how this markup can be manipulated in powerful ways using the jQuery Javascript framework. User interface elements such as buttons can be automatically created by associating them with selected element types. We made buttons for each viewpoint in a scene as well as buttons for each sphere in a scene. In addition we demonstrate how to link up the interactive capabilities of modern HTML widgets via jQuery to the 3D geometry of a scene. Finally a jQuery plugin was developeded to illustrate how to seamlessly extend jQuery for operations on the X3D scene. Note that full listings of the code can be found online at http://math.nist.gov/ SRessler/x3dom/, including demonstrations of the X3DOM worlds and plugin's discussed.

3D descriptive markup is not yet another 3D file format. It leverages the infrastructure of the web in such as way as to finally offer the hope of ubiquitous portable interoperable 3D. Recent developments, in particular WebGL based browsers, that can render 3D scenes, taking advantage of powerful GPU hardware coupled with a standard format such as X3D is a realistic set of technologies to bring 3D to everyone.

DISCLAIMER: Please note that mention of any commercial products, companies and technologies does not constitute an endorsement by NIST.

References

BEHR, J., ESCHLER, P., JUNG, Y., AND ZOLLNER, M. 2009. X3dom: a dom-based html5/x3d integration model. In *Web3D 2009*, Fraunhoffer, 127–135.

GRIEFER, C. 2009. jquery dynamically adding form elements. *http://charlie.griefer.com/blog/index.cfm/2009/9/17/jQuery–Dynamically-Adding-Form-Elements*.

REISIG, J. 2010. The jquery project. *http://jquery.org*.

RESSLER, S. 2011. Anthrogloss human body landmark reference (x3dom with jquery version) v2. *http://math.nist.gov/ SRessler/x3dom/mangloss2.xhtml*.

SONS, K., KLEIN, F., RUBINSTEIN, D., BYELOZYOROV, S., AND SLUSALLEK, P. 2010. Xml3d: interactive 3d graphics for the web. In *Web3D '10: Proceedings of the 15th International Conference on Web 3D Technology*, ACM, New York, NY, USA, DFKI and Sarrland University, 175–184.