Graphics on Web
https://www.svgopen.org/2012/
Sandy Ressler, National Institute of Standards and Technology
Sept. 2012

Web-based Declarative 3D Graphics for Anthropometry Visualization and Education

Abstract

A system to educate users of human anthropometry was developed using a declarative 3D graphics system that is integrated seamlessly with web browsers. Users can select anthropometric landmarks and learn their names, view the body from many points of view, and highlight specific landmarks. The 3D graphics capabilities are implemented via a system called X3DOM[x3dom] and is integrated with the web browser utilizing standard jQuery UI interface elements. X3DOM is "...an experimental open source framework and runtime to support the ongoing discussion in the Web3D and W3C communities on how an integration of HTML5 and declarative 3D content could look like." In addition standard DOM click events are used to react to user actions resulting in information displays presented using standard jQuery dialogs.

In addition to using an idealized computer generated model, bodies taken from real humans via laser scans produced by the CAESAR project[caesar], can be displayed. The CAESAR project was the first large scale anthropometric survey produced using whole body laser scanners in the mid 1990s. We have the ability to take the original 3D scans and display them via a typical production chain, with their anthropometric landmarks, on an interactive web page. The scans for each subject were taken in three poses, two of which are standing. We can interactively flip between the two standing poses, another illustrative technique for examining the bodies.

Introduction

One of the problems this system seeks to address is the education of people in 3D human measurement concepts. In particular 3D human anthropometry where measurements occur in space and are not simply univariate measurements between two spots on the body. Traditional univariate measures don't take into account the wide variation in human shape, which depending on the anthropometric application, matters a great deal. It is a common experience to try on clothing of the correct size and have it fit poorly. This is often due to variation in the cut of the garment which corresponds poorly to the shape of the body.

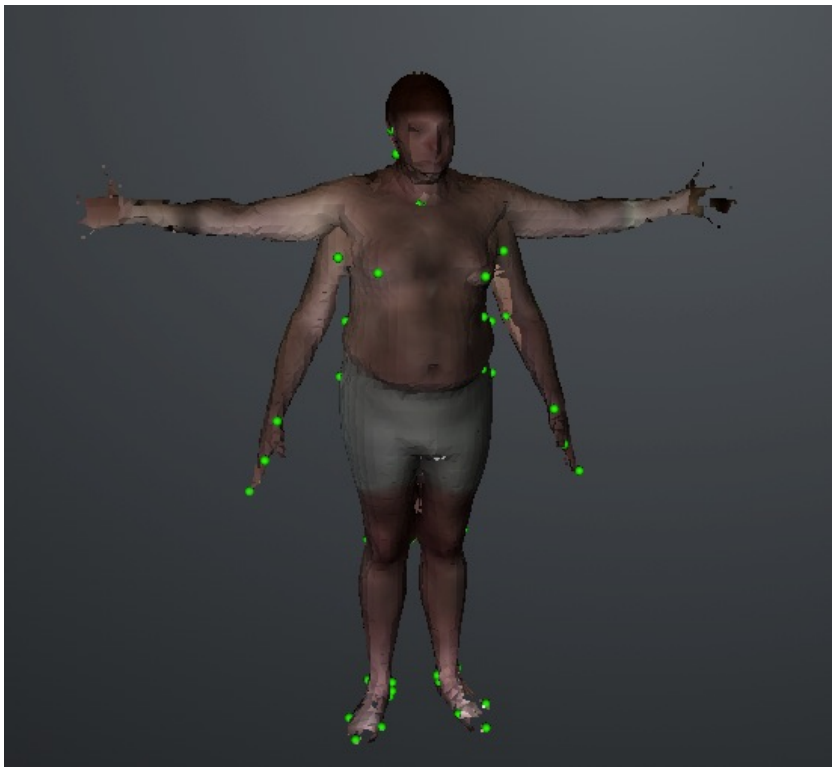
Humans have remarkably varied shapes. Products with which humans interact, automobiles,

[1] Any mention of commercial products is for information only; it does not imply recommendation or endorsement by NIST.

airplanes, and clothing all make use of the science of anthropometry.

Variation in Human Shape and Size (from CAESAR project)

A key element of 3D anthropometry is the measurement of 3D landmarks. These landmarks are locations on the body, based on anatomy, which are used to characterize the shape of the body. Each landmark has a name and educating users to these names is a first step in introducing users to 3D measurement concepts. Bringing this application to the Web and doing so in an unobtrusive manner simply makes the education process easier.



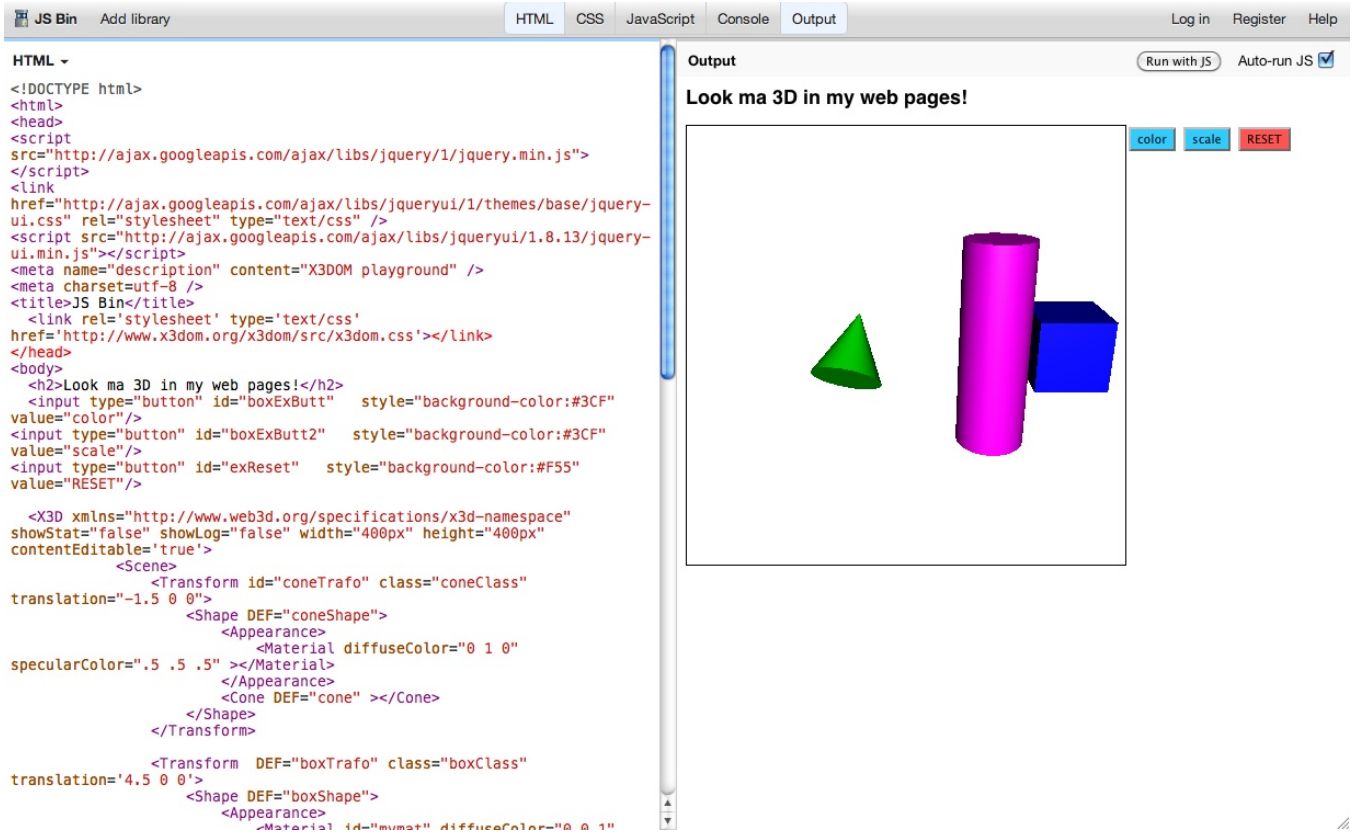
Two Poses of Whole Body Scan with Landmarks

Declarative Graphics

Before exploring the actual application let's examine the two types of graphic representations, declarative and imperative, both of which comprise graphics in web pages. Declarative graphics are best represented by X3DOM, which enables the inclusion of X3D, the successor to VRML,

directly in web pages. Imperative graphics are represented by WebGL[webgl], the web equivalent of OpenGL[opengl]. Each method is good for different types of applications. Imperative graphics offers a much higher degree of control of the rendering process and finer grain control in general. Declarative graphics is a higher level form of graphic representation. More importantly, in this case, where we are integrating with web pages, the declarative graphics approach enables graphic elements to become part of the internal representation of the web page, the DOM. Once graphic elements are part of the DOM, the rich web development infrastructure can be used to operate directly upon those graphical objects. JavaScript libraries such as jQuery and debugging tools such as Chrome's web debugger can be directly applied to graphical objects, not an insignificant capability.

For example we can use the jQuery[jquery] selection mechanism to select all elements of the type <sphere> and change the scale of the sphere. A rich WYSIWYG debugging tool called JS Bin [jsbin] can be used to directly modify graphical elements even though the tool knows nothing in particular about graphics.



JS Bin with an X3DOM scene

The overall system does not require any "plug-ins" and given the inclusion of the appropriate JavaScript libraries, the 3D world is seamlessly integrated with the rest of the 2D web page. Rendering is accomplished via WebGL, however it is not a requirement as rendering can be performed via Flash, for those browsers that do not support WebGL. The architecture of X3DOM explicitly separates the rendering functionality, an additional benefit of the declarative

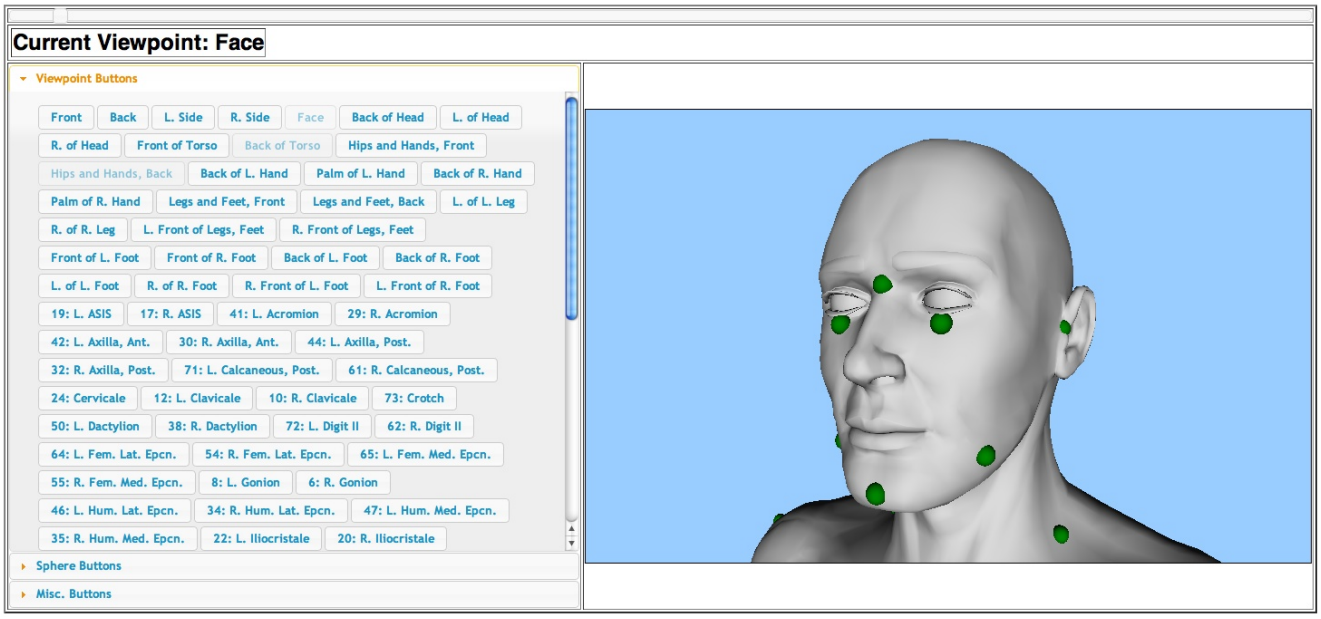
markup approach.

“Normal” Web-based User Interfaces

Traditionally the user interfaces for 3D graphics systems have been ad-hoc. The interface elements often exist in the 3D world and we have to trick the system into treating the 3D elements as though they were only 2D elements. While a compelling user interface can be created it is often difficult to maintain and more importantly artifacts such as disappearing UI elements often become problematic. In many existing VRML/X3D system the UIs are often the first thing to break, leaving users and developers frustrated.

The move to declarative 3D opens up the possibility of “normal” Web-based user interfaces. Interfaces using robust libraries of widgets, such as jQuery UI, that have traditional interface widgets such as accordions, buttons and tabs which can be created and appear as just another “normal” web application.

The 3D graphics appearing in a portion of the web page is not an isolated area. Buttons or sliders can cause actions in the 3D world. Likewise selection of 3D objects in the 3D world can cause native HTML dialogs to appear and trigger DOM events. The actions of HTML buttons can affect 3D viewpoints, and the scale of certain graphic elements in the 3D world. All of this is made easier because the graphic elements themselves appear in the DOM of the web page.



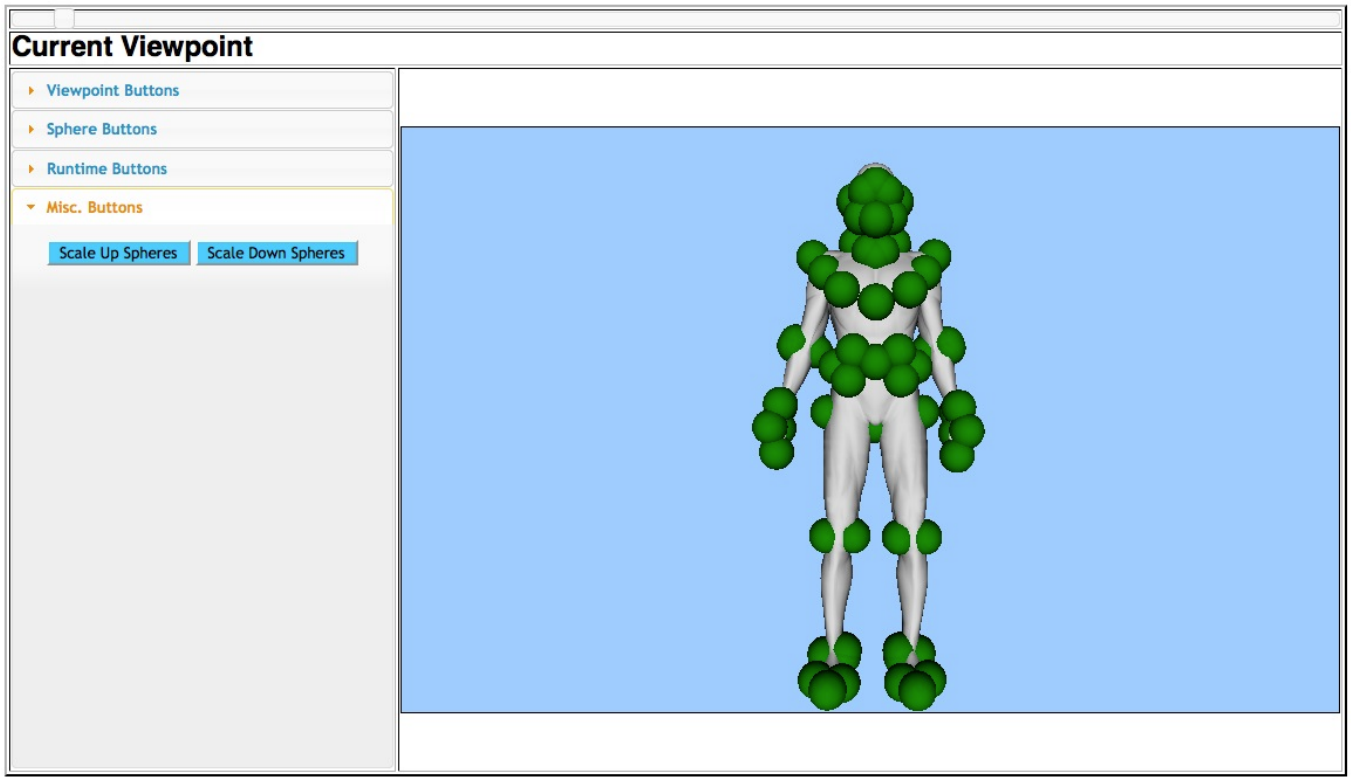
Accordion Interface on Left with Dynamically Generated Buttons

Imperative type graphics can of course perform similar functionality however the author must use various “tricks” to inject content into the DOM and interact with web page elements. Using a declarative approach, graphic elements are simply another type of page content and existing development tools work quite naturally. As a simple example examine the JavaScript/jQuery

code:

```
$('sphere').attr('scale','10 10 10');
```

In pseudo-code this means: select all of the “sphere” type elements, then set the “scale” attribute for those elements to values ‘10 10 10’, the x,y,z scaling factors. This is a very simple and straightforward method to program web pages with 3D graphic content. (Note in actual reality the code is more complex as we must “walk” up the DOM tree a little bit to locate the <transform> node which is used to actually modify the scale attribute, but the code above simplifies the explanation and the technique is accurate.)



Result of Scaling up All <sphere> Elements

X3DOM uses “declarative” markup which has several advantages over “imperative” graphics. Chief among these is 1) ease of authoring by using standard HTML tools, 2) ease of debugging using standard debugging tools, and 3) ability to insert text for use by screen readers and other web accessibility tools.

Dynamically Generating Geometry and UI Elements

We can generate geometry dynamically using the declarative nature of the graphics. Of course we could just as easily generate geometry if this was an imperative collection of graphics, however we can associate declarative actions with the graphics in a straightforward manner. In this case the body has a collection of spheres associated with the landmarks of the body. For the version of this system that uses the CAESAR (real human whole body scans) bodies, a dynamic generation capability that reads the landmark locations from a separate file and

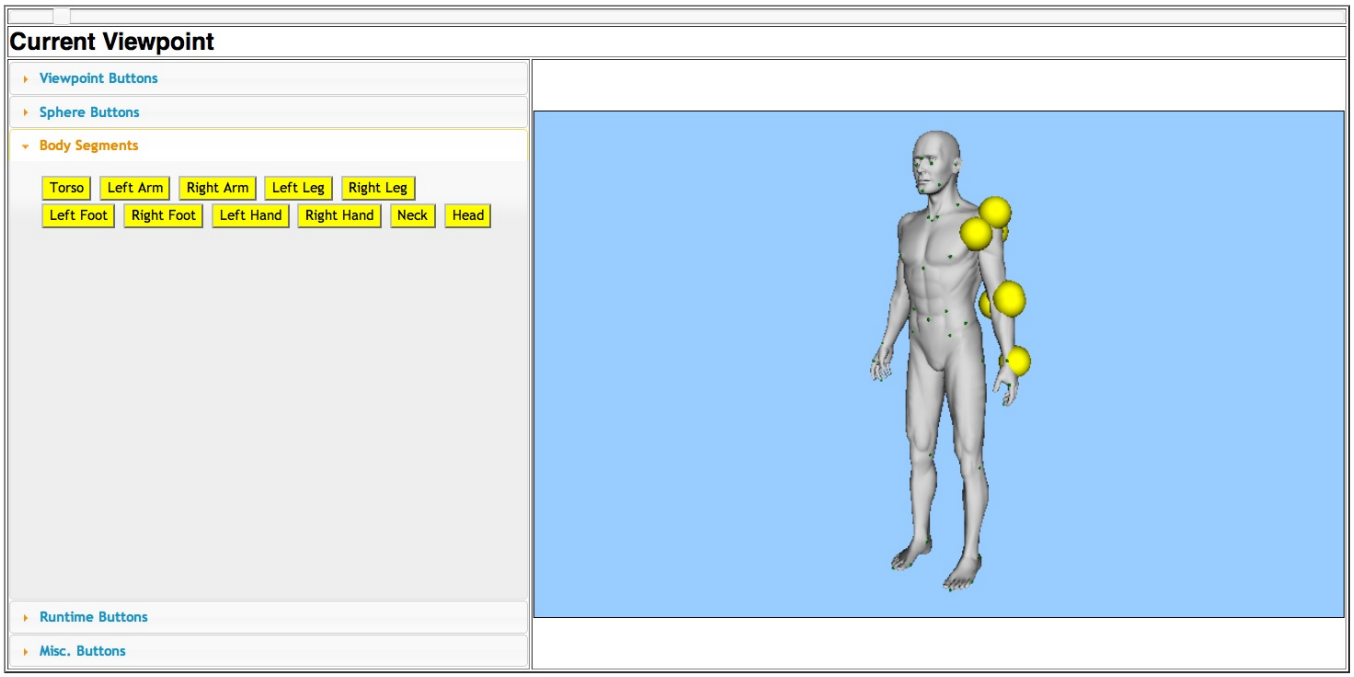
generates the landmark spheres, was implemented. During the process of generation we also associate the text and a dialog action that occurs when the sphere is clicked.

More interesting than generating the geometry of graphics, which can be accomplished in many ways, the dynamic generation of UI elements takes full advantage of the declarative nature of the graphics system. We simply use one of jQuery's selection mechanisms to pick the elements with which we wish to interact. For example there are approximately 20 different viewpoints. These viewpoints appear in the DOM as <viewpoint> elements. We iterate through the viewpoint elements and create a button for each. Pressing the button moves the viewpoint of the 3D scene to that viewpoint.

Declarative Segmentation

Given the declarative nature of the graphics we can add additional semantics to the geometry for functionality that has traditionally been the domain of pure graphics. For example these particular human bodies, are what is colloquially known as a "bag of polygons". There is nothing inherent to identify that one particular area is an arm or another a leg. If we want to perform more complicated graphical actions, such as animating the body, we must go through a process of segmentation, to identify the body parts.

Similarly, the spheres can also be associated with segments. However given the declarative nature of the graphics we can simply associate an HTML class with each sphere, and thus segment the spheres. Finally, given a collection of segment classes we can create some meaningful semantic user interface actions for these segments.



Body Segmentation Using Declarative Semantics

In this case 11 body segments were identified (by hand) and the semantics were added as classes to the markup. Pressing the buttons, executes functions which use these classes to scale and color the spheres associated with those segments.

Summary

Anthropometry is but one example of the types of applications that can use declarative 3D graphics. The clean integration of 3D graphics with web pages offers the potential for wide ubiquitous dissemination of 3D applications via web pages and reduces the complexity of prior integration techniques. The inclusion of graphic content to the DOM enables authors and content providers the ability to leverage the rich environment of web development tools for their own 3D applications. This application demonstrates the potential for rich 3D applications using a variety of techniques that take advantage of the declarative approach to graphics.

Live web examples

http://math.nist.gov/~SRessler/x3dom/mangloss2.xhtml synthetic body example
http://math.nist.gov/~SRessler/x3dom/x3domCaesar/csr0084aIncDyn.xhtml CAESAR example
http://math.nist.gov/~SRessler/x3dom/x3domCaesar/csr2Poses.xhtml multiple poses
http://math.nist.gov/~SRessler/x3dom/mangloss5.xhtml illustrating body segment functionality

References

[caesar] CAESAR: Civilian American and European Surface Anthropometry Resource web site: http://www.hec.afrl.af.mil/cardlab/CAESAR/index.html, Also at http://store.sae.org/caesar/, 2006.

[jsbin] Sharp, Remy, https://github.com/remy/jsbin, 2012.

[jquery] Reisig, J. The jquery project. http://jquery.org, 2010.

[opengl] OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition), 01 August 2005) by D. Shreiner, M. Woo, J. Neider, T. Davis.

[webgl] Khronos, WebGL Specification https://www.khronos.org/registry/webgl/specs/1.0/.

[x3dom] BEHR, J., ESCHLER, P., JUNG, Y., AND ZOLLNER, M. 2009.
X3dom: a dom-based html5/x3d integration model. In proceedings of Web3D 2009, Fraunhoffer, pg. 127–135.

Appendix

Code to Generate <sphere> Elements with Dialogs

```
//Defining a sphere template
<sphere radius="0.0125"
 onclick="$("#dialog").dialog("open");
      $(".timer").attr("cycleInterval","0");
      $("#clock1").attr("cycleInterval","2.0");
      $("#dialog").html("<p> You just selected the "+
$(this).parent().parent().attr("description") + " landmark. </p>");
      $("#dialog").dialog( {title:
$(this).parent().parent().attr("description") });" >
 </sphere>

//Cloning the landmark sphere
function cloneLandmarks(idstr, matstr) {
for (var index = 0; index < OrigLandmarks[0].length; index++) {
    $(".ball").each(function(index) {
         newNum = new Number(index + 1);
         newElem = $("#"+idstr+"0").clone().attr("id", "idstr" + newNum);
         newElem.attr("description",
            OrigLandmarks[0][index].description);
         newElem.attr("translation",
            OrigLandmarks[0][index].translation);
         //2nd child of shape is sphere
         //replace clock1 with correct clock id

         tmpObj = newElem.children().children(":nth-child(2)");
         tmpClickStr = tmpObj.attr("onclick");
         ntmpClickStr = tmpClickStr.replace(/clock1/i,"clock"+newNum);
newElem.children().children(":nth-child(2)")
.attr("onclick",ntmpClickStr);
         //replace mat1 with mat id
         newElem.children()
            .children().children()
            .attr("id",matstr+newNum);
        // place the new transform
        $("#"+idstr+"0").after(newElem);
       }
};

//Usage
cloneLandmarks('lm','mat');
```

Code to iterate through the viewpoints cloning from a template button

```
function cloneViewpointButtons() {
  $('#nameClone').show(); // show the button to clone
  //select all viewpoint elements and place into the array Viewpoints
```

```
  var Viewpoints = $("viewpoint");
  // original source of clone button code
//http://charlie.griefer.com/blog/index.cfm/2009/9/17/
    jQuery--Dynamically-Adding-Form-Elements

  //iterate through all of the viewpoints
  $("viewpoint").each( function(index) {
    newNum = new Number(index + 1);
    //clones only the button
    newElem = $('#nameClone').clone().attr('id', 'name' + newNum);
    //assign a classname to allow deletion of all but first button
    newElem.attr('class', 'viewbutts');     newElem.attr('value',
$(this).attr('description'));
    //use the animate visual effect and
    //create an anonymous function to execute when button is clicked
    newElem.button().click(function() {
      $(this).animate({opacity: 0.5}, 1500);
      // these must be a actual viewpoints
      $(Viewpoints[index]).attr('set_bind','true');
      var descrip = $(Viewpoints[index]).attr('description');
      var htmlStr = "Current Viewpoint: " + descrip + "";
      $('#currViewpoint').html(htmlStr);
    });;
    // place the new button after the old div
    $('#nameClone').before(newElem);     });
    $('#nameClone').hide(); // hide the button to clone
};
```