ARMY RESEARCH LABORATORY

Exploratory Visual Analytics of a Dynamically Built Network of Nodes in a WebGL-Enabled Browser

by Andrew M. Neiderer

ARL-MR-860 **January 2014**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-MR-860 **January 2014**

Exploratory Visual Analytics of a Dynamically Built Network of Nodes in a WebGL-Enabled Browser

Andrew M. Neiderer
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
January 2014	Final	May 2013–November 2013

4. TITLE AND SUBTITLE
Exploratory Visual Analytics of a Dynamically Built Network of Nodes in a WebGL-Enabled Browser

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Andrew M. Neiderer

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-CII-C
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-MR-860

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report describes an extensible hypertext markup language (XHTML) document that dynamically builds a network of nodes for a WebGL-enabled browser. It uses x3dom-full.js, a JavaScript library of functions that call WebGL functions for the low-level manipulation of extensible three-dimensional (3-D) scene content that is embedded. One can update the XHTML tree data structure defined by the browser for an immersive experience. In this report, exploratory visual analytics of a spatially-distributed network of nodes in 3-D space is computed. The geometric branch of the scene graph is discussed and the code is provided.

15. SUBJECT TERMS
dimensionality reduction, feature extraction, high-dimensional data, t-distributed stochastic neighbor embedding, neighbor retrieval visualizer, visual analytics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Andrew M. Neiderer
Unclassified	Unclassified	Unclassified	UU	36	19b. TELEPHONE NUMBER (*Include area code*) (410) 278-3203

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction

The U.S. Army Research Laboratory (ARL) has developed an exploratory visual analytics (EVA) application called netVA, which dynamically builds a network structure in Euclidean space. Vertices (V) (or nodes) and edges (E) (or links) between them create a graph data structure (G), a function of V and E: G = f(V,E) in two-dimensional (2-D) or three-dimensional (3-D) space. An affine transformation of G followed by an orthographic projection onto an arbitrary plane may reveal something informative about the spatial distribution of G that may otherwise go unnoticed in a textual display. The visual analytics capability has been done in a WebGL-enabled browser.

Gaming technology has been further strengthened by update of the hypertext markup language (HTML), or extensible HTML (XHTML), <canvas>.[*,1] Drawing is done in a 3-D WebGL rendering context defined for this 2-D rectangular array of pixels. The context provides object representation and methods for drawing and manipulating graphics on the canvas. Work on the definition of WebGL started in 2009. The specification WebGL 1.3 was released in March 2011, and now many computer graphics programmers/gamers believe WebGL is here to stay.

WebGL-enabled rendering is supported natively by browsers such as the latest Mozilla Firefox, Google Chrome, and Microsoft Internet Explorer 11. At the core of WebGL is OpenGL, which has withstood repeated competitive threats to "emerge as the undisputed standard for programming 3-D graphics."[2] An illustration of convergence and collaboration of various working groups in the Web3D consortium, including WebGL and OpenGL, can be found at the "What is X3D" link of http://www.web3d.org/realtime-3d/about. But 3-D drawing is not done declaratively in WebGL, i.e., within the markup, but rather procedurally using the JavaScript language from a HTML/XHTML <script>.

The WebGL application programming interface (API) is low level. Several libraries of JavaScript functions exist to ease the use of WebGL. We chose the one called Three.js, written by Ricardo Cabello Miguel,[†] which is widely used and intuitive for 3-D graphics programmers. Still, not everyone is a computer programmer, and WebGL development is typically out of reach for the casual web developer.

[*]Technically, this is the fifth-generation HTML specification, or HTML5. But there is often confusion when using this term (see page 8 of "Tracing the History of HTML5" in Jacob Seidelin's book *HTML5 Games*). Here we simply use HTML to refer to this latest version.

[1]Seidelin, J. *HTML5 Games*; John Wiley & Sons, Inc.: Hoboken, NJ, 2012.

[2]Parisi, T. *WebGL Up and Running*; O'Reilly Media, Inc.: North Sebastopol, CA, 2012.

[†]Three.js can be downloaded from https://githhub.com/mrdoob/three.js.

The following XHTML application was done declaratively using the X3DOM API: (1) embedded extensible 3-D (X3D) scene content and (2) document object model (DOM) methods for building the scene graph (SG). The function getElementById() accesses the root of the SG, an X3D <Group> node. Note that the distinction between procedural and declarative programming, e.g., WebGL versus X3DOM, becomes "fuzzy" when using the <script> node.

X3D is an International Standards Organization specification (http://www.web3d.org/x3d /specifications) for describing scene content. The SG is a directed acyclic graph of X3D nodes arranged in a hierarchical parent-child relationship. X3D is component-based; that is, X3D nodes are logically grouped to define a component, and components are arranged by profiles for a specific domain. The profile used here is "Immersive" for an EVA capability.

In 2010, X3D nodes were coupled with HTML nodes for a tree description of a document in a web browser. In our application, 2-D/3-D position vectors for nodes in the network are added (appendChild()) to the root of the scene graph, which is accessed by getElementById(). The result is EVA for a network of nodes.

2. Scene Graph for a Network of Nodes

The complete XHTML document for the following example, which builds a 26-node network, is given in appendix A. It results from a load mutation event for the document buildNetwork _X3DOM.xhtml. The property onload for the <body> tag is assigned buildNetwork(), a JavaScript function that is called when the document initializes/changes.

After initializing for network node data, the DOM function getElementById() is used to access the root of the scene; this is an X3D <Group> node. Then X3D nodes and node components are created for each position vector the user gives and then added to the scene. A position vector is defined as an X3D <Sphere>. This is all that is required to build the scene: just x, y, and z components for a network node. The XHTML document builds the rest of the SG.

The geometry branch includes X3D nodes for each network node added to the SG in the following order: an X3D <Transform>, <Shape>, <Sphere>, <Appearance>, and <Material>. Also, a <TouchSensor> and <PlaneSensor> are defined and added to the SG for user interactivity. The result is shown in figure 1. A mutator function, setAttribute(), is used in the document to assign values to appropriate names. The resultant 26-node network is displayed in a Mozilla Firefox browser in figure 2 (also see appendix B).

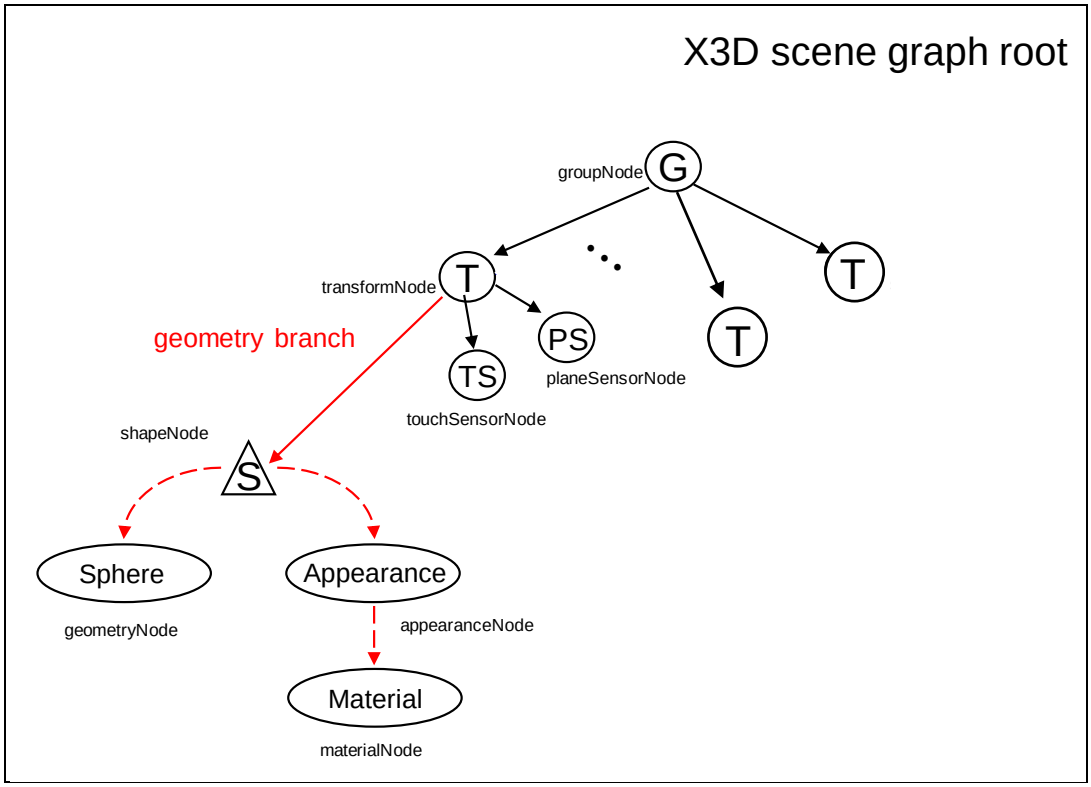

Figure 1. The directed acyclic graph of X3D scene graph objects for the geometry branch of a network node. There is such a description for each of the 26 nodes in the example.

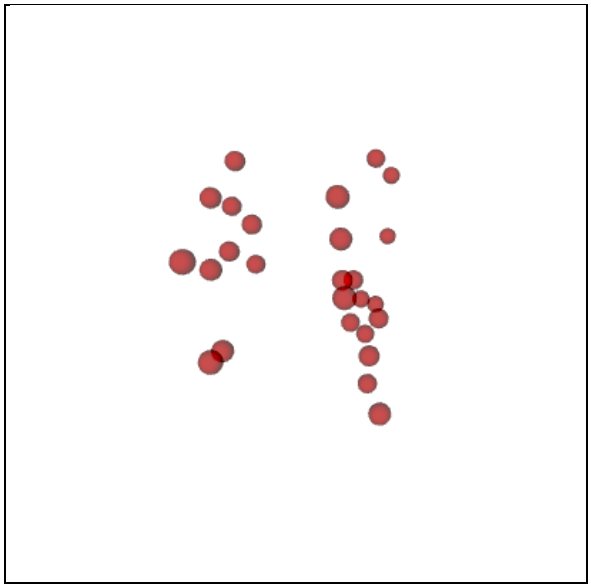

Figure 2. Web browser display of the 26-node network built using X3DOM, a JavaScript library of functions for WebGL graphics.

A text branch for the SG could also be defined. We have done this in a console window of the browser, but it was used only for debugging purposes.

Note that for those familiar with X3D, animation and user interactivity within the scene is done using the <ROUTE> mechanism. An abstract connection between X3D nodes sending/receiving events is assigned. But X3DOM uses the HTML event model (EM). This is the topic of the next section.

3. The Event Model for netVA

The netVA application uses an EM for user interaction (and animation) with scene content. An EM usually includes consideration of a (1) property, (2) event type, (3) event handler, and/or (4) event listener. Currently we are using only the mutation event load, i.e., property onload for the <body> tag in buildNetwork_X3DOM.xhtml. This property is assigned to the function buildNetwork() that is in an internal <script> node of type "text/javascript". In this function, the network of nodes are created as X3D <Sphere>s, which are then attached to the SG for an "Immersive" profile.

There are many other events that exist for both the keyboard and mouse. For example, mouse events include click, dblclick, mousedown, mouseup, mouseover, mouseout, and mousemove. An HTML tag with corresponding "on" property, e.g., onclick, would be assigned to an event handler. There are also many keyboard events. A complete discussion of event modeling can be found in a book by Andreas Anyuru.[3]

[3]Anyuru, A. *Professional WebGL Programming, Developing 3-D Graphics for the Web*; John Wiley & Sons, Inc.: Hoboken, NJ, 2012.

4. Future Work

The application buildNetwork_X3DOM.xhtml dynamically builds an X3D SG of a network by only requiring position vectors for the nodes. It uses the DOM createElement() method for X3D nodes of the SG, and sets attribute values when necessary (see appendix A).

With the recent addition of an <Extrusion> node in X3DOM (August 2013), dynamic links between nodes will be added when appropriate. The links in netVA will also be done in buildNetwork(), in a manner similar to the addition of position vectors.

Also, the mutation event for page reloads will be changed to a jQuery ready() method. This is advertised to increase performance 1.5–7 times.

INTENTIONALLY LEFT BLANK.

Appendix A. An X3DOM Dynamic Build of a Network of Nodes

The following extensible hypertext markup language (XHTML) example builds a 26-node network. The code is thoroughly documented to assist in understanding. Only the JavaScript var's (1) npvs (number of position vectors) and (2) pv (position vector) need to be changed or added for a different application. This example is for three-dimensional position vectors of nodes; a two-dimensional situation is done by eliminating one of the components. JavaScript variable names are camel-cased: begin with a lower-case letter with successive words capitalized, and no spacing.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml11/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
  <head>
    <!-- FileName:         buildNetwork_X3DOM.xhtml                                  -->
    <!--                                                                            -->
    <!-- Description:      X3DOM XHTML application to dynamically build a            -->
    <!--                    network of nodes -                                      -->
    <!--                    (1) X3D for scene content                               -->
    <!--                    (2) HTML event model (EM) which uses DOM API to navigate -->
    <!--                        and manipulate the document, including X3D nodes    -->
    <!--                        and attributes.                                     -->
    <!--                                                                            -->
    <!-- By:               Andrew M. Neiderer, US Army Research Laboratory.        -->
    <!--                                                                            -->
    <!-- Date:             22 August 2013.                                         -->

    <meta http-equiv="X-UA-Compatible"
          content="chrome=1"/>
    <meta http-equiv="Content-Type"
          content="text/html; charset=UTF-8"/>

    <!-- qTip tooltips -->

    div#qTip {
    padding: 3px;
    border: 1px solid #666;
    display: none;
    background: #999;
    color: #FFF;
    font: bold 9px Verdana, Arial, sans-serif;
    position: absolute;
    z-index: 1000;
    }
```

```
    <!-- X3DOM cascading style sheet -->
    <link rel="stylesheet" type="text/css"
          href="x3dom.css"/>

    <!-- JavaScript lib X3DOM by Dr.Ing Johannes Behr (Fraunhofer IGD) -->
    <script type="text/ecmascript"
            src="x3dom-full.js"/>

    <!-- JavaScript lib for tooltips (qrayg.com/learn/code/qtip) -->
    <script lang="JavaScript" type="text/javascript"
            src="qtip.js"/>

    <title>
      X3DOM dynamic build of a network of nodes
    </title>
  </head>

  <body onload="buildNetwork()">
    <h1>
      X3DOM dynamic build of a network of nodes
    </h1>

    <!-- X3D scene content -->

    <X3D id="X3D_ID"
         profile="Immersive"
         xmlns="http://www.web3d.org/specifications/x3d-namespace"
         showStat="false" showLog="false"
         x="0px" y="0px" width="600px" height="460px">

      <Scene>
        <Viewpoint description="dynamic build of a network"
                   orientation="0.0 1.0 0.0 1.57"
                   position="12.0 0.0 0.0"/>
        <NavigationInfo type='"EXAMINE" "ANY"'/>

        <Group id="sphereGroupID"/>;
```

```
  </Scene>
</X3D>

<!-- build network of nodes for X3D scene graph -->

<script type="text/javascript">
  function handleClick(i)
  {
    alert('click event');
  }

  function buildNetwork()
  {
    // 3-component position vectors for network of nodes

    var npvs = 26;
    var ncmps = 3;
    var X = 0, Y = 1, Z = 2;

    var pv = new Array(npvs);
    for ( var i = 0; i &lt; npvs; i++ )
      pv[i] = new Array(ncmps);

    pv[0][X]  =  -2.01240; pv[0][Y]  =   1.71360; pv[0][Z]  =   0.6;
    pv[1][X]  =  -2.71140; pv[1][Y]  =   0.67860; pv[1][Z]  =   0.7;
    pv[2][X]  =  -0.91680; pv[2][Y]  =   1.85220; pv[2][Z]  =   0.8;
    pv[3][X]  =  -2.36220; pv[3][Y]  =  -0.56100; pv[3][Z]  =   0.9;
    pv[4][X]  =  -1.26480; pv[4][Y]  =  -1.01280; pv[4][Z]  =   1.0;
    pv[5][X]  =  -0.22200; pv[5][Y]  =  -0.10740; pv[5][Z]  =   1.1;
    pv[6][X]  =  -0.80580; pv[6][Y]  =  -0.79560; pv[6][Z]  =   1.2;
    pv[7][X]  =  -1.63560; pv[7][Y]  =  -0.44460; pv[7][Z]  =   1.1;
    pv[8][X]  =  -0.42720; pv[8][Y]  =   1.03740; pv[8][Z]  =   3.0;
    pv[9][X]  =   0.13140; pv[9][Y]  =   0.31920; pv[9][Z]  =   2.9;
    pv[10][X] =   0.20040; pv[10][Y] =   1.65120; pv[10][Z] =   2.8;
    pv[11][X] =   1.29960; pv[11][Y] =  -1.04640; pv[11][Z] =   2.7;
    pv[12][X] =   2.25300; pv[12][Y] =  -1.08660; pv[12][Z] =   2.6;
    pv[13][X] =  -0.01920; pv[13][Y] =   0.72840; pv[13][Z] =   2.6;
```

```
pv[14][X] =  -0.76680; pv[14][Y] =   0.13980; pv[14][Z] =   2.7;
pv[15][X] =   2.71140; pv[15][Y] =   0.12660; pv[15][Z] =   2.8;
pv[16][X] =   1.13700; pv[16][Y] =   0.04260; pv[16][Z] =   2.9;
pv[17][X] =   0.78060; pv[17][Y] =   1.05240; pv[17][Z] =   3.0;
pv[18][X] =   1.93860; pv[18][Y] =   0.95520; pv[18][Z] =   1.1;
pv[19][X] =   0.43320; pv[19][Y] =  -0.10620; pv[19][Z] =   1.2;
pv[20][X] =   1.55640; pv[20][Y] =   0.44220; pv[20][Z] =   1.1;
pv[21][X] =   2.07480; pv[21][Y] =  -0.31260; pv[21][Z] =   1.0;
pv[22][X] =  -0.39420; pv[22][Y] =  -1.71900; pv[22][Z] =   0.9;
pv[23][X] =   0.63240; pv[23][Y] =  -1.17960; pv[23][Z] =   0.8;
pv[24][X] =   0.06060; pv[24][Y] =  -0.67980; pv[24][Z] =   0.7;
pv[25][X] =   1.48680; pv[25][Y] =  -1.85220; pv[25][Z] =   0.6;

// X3D scene graph description

var group = document.getElementById("sphereGroupID");
var transform;
var touchSensor;
var planeSensor;
var shape;
var geometry;
var appearance;
var material;
                        // attributes for Transform node
var TRANSFORM_translation;
var translation;
var TRANSFORM_SCALE = "0.15 0.15 0.15"
var scale;
                        // attribute for geometry node component,
                        //  which is a Sphere
var SPHERE_RADIUS = 0.25;
                        // attributes for Material node component
                        //  color from 0.0 to 1.0
var MATERIAL_DIFFUSE_R = 1.0,
    MATERIAL_DIFFUSE_G = 0.0,
    MATERIAL_DIFFUSE_B = 0.0;
var diffuse;
```

```
                        //  alpha from 0.0 to 1.0
var MATERIAL_TRANSPARENCY = 0.3
var transparency;
                        // attribute for PlaneSensor node
var PLANE_SENSOR_offset;
var offset;

var route;

// add position vectors as X3D spheres to the scene graph

for ( var i = 0; i &lt; npvs; i++ ) {
  // for Mozilla Ff Firebug console
  console.debug("node id=DEF=",i);

  // Transform node added to Group node

  transform = document.createElement("Transform");

  transform.setAttribute("id",i);
  translation = pv[i][X] + " " +
                pv[i][Y] + " " +
                pv[i][Z];
  transform.setAttribute("translation",translation);
  transform.setAttribute("scale",TRANSFORM_SCALE);

  group.appendChild(transform);

  // Shape node added to Transform node

  shape = document.createElement("Shape");

  shape.setAttribute("id",i);

  transform.appendChild(shape);

  // geometry node component added to Shape node
```

```
geometry = document.createElement("Sphere");

geometry.setAttribute("id",i);
geometry.setAttribute("DEF",i);
geometry.setAttribute("onclick",
                      "handleClick(" +
                      i +
                      ");");

shape.appendChild(geometry);

// Appearance node component added to Shape node

appearance = document.createElement("Appearance");

appearance.setAttribute("id",i);

shape.appendChild(appearance);

// material node component added to Appearance node component

material = document.createElement("Material");

material.setAttribute("id",i);

if ( i == 0 )
  diffuse = MATERIAL_DIFFUSE_R + " " +
            MATERIAL_DIFFUSE_G + " " +
            MATERIAL_DIFFUSE_B;
else if ( i &gt; 0  &amp;&amp;  i &lt; 8 )
  diffuse = 0.0 + " " +
            0.0 + " " +
            0.9;
else if ( i &gt; 8  &amp;&amp;  i &lt; 21 )
  diffuse = 1.0 + " " +
            1.0 + " " +
            1.0;
```

```
else
  diffuse = 1.0 + " " +
            0.6 + " " +
            0.0;
material.setAttribute("diffuseColor",diffuse);
material.setAttribute("transparency",MATERIAL_TRANSPARENCY);

appearance.appendChild(material);

// TouchSensor node

touchSensor = document.createElement("TouchSensor");

touchSensor.setAttribute("id",i);
touchSensor.addEventListener("touchTime",function ()
                                         {
                                           alert("clicked " + i);
                                         },
                                 false);

transform.appendChild(touchSensor);

// PlaneSensor node is necessary for animation, and
//  added to Transform node.

planeSensor = document.createElement("PlaneSensor");

planeSensor.setAttribute("id",i);
offset = pv[i][X] + " " +
         pv[i][Y] + " " +
         pv[i][Z];
planeSensor.setAttribute("offset",offset);

transform.appendChild(planeSensor);

// animation from PlaneSensor node to Transform node
```

```
          route = document.createElement("ROUTE");

          route.setAttribute("fromNode","PLANE_SENSOR");
//          route.setAttribute("fromField",translation_changed);
          route.setAttribute("toNode","TRANSFORM");
//          route.setAttribute("toField",translation);
        }
      }

      document.onload = function()
                        {
                          alert("tooltips here?");
                        }
    </script>
  </body>
</html>
```

Appendix B. A WebGL d^2 Network of Nodes

The following extensible hypertext markup language (XHTML) application, called inetVA, dynamically builds a network of dynamic nodes procedurally using the WebGL application programming interface (API). This in contrast to appendix A, where the network was built declaratively using X3DOM (document object model) libraries. Position vectors of nodes in the network are assigned values in the JavaScript function buildNetwork() as before. But now the user is responsible for providing much more, as can be seen in buildInteractiveNetwork _WebGL.xhtml. Display of output in a Mozilla Firefox browser is also included (see figure B-1).

Currently we are using X3DOM libraries for adding dynamic nodes and links to the network. This should be easier now that an X3D extrusion node has been implemented.

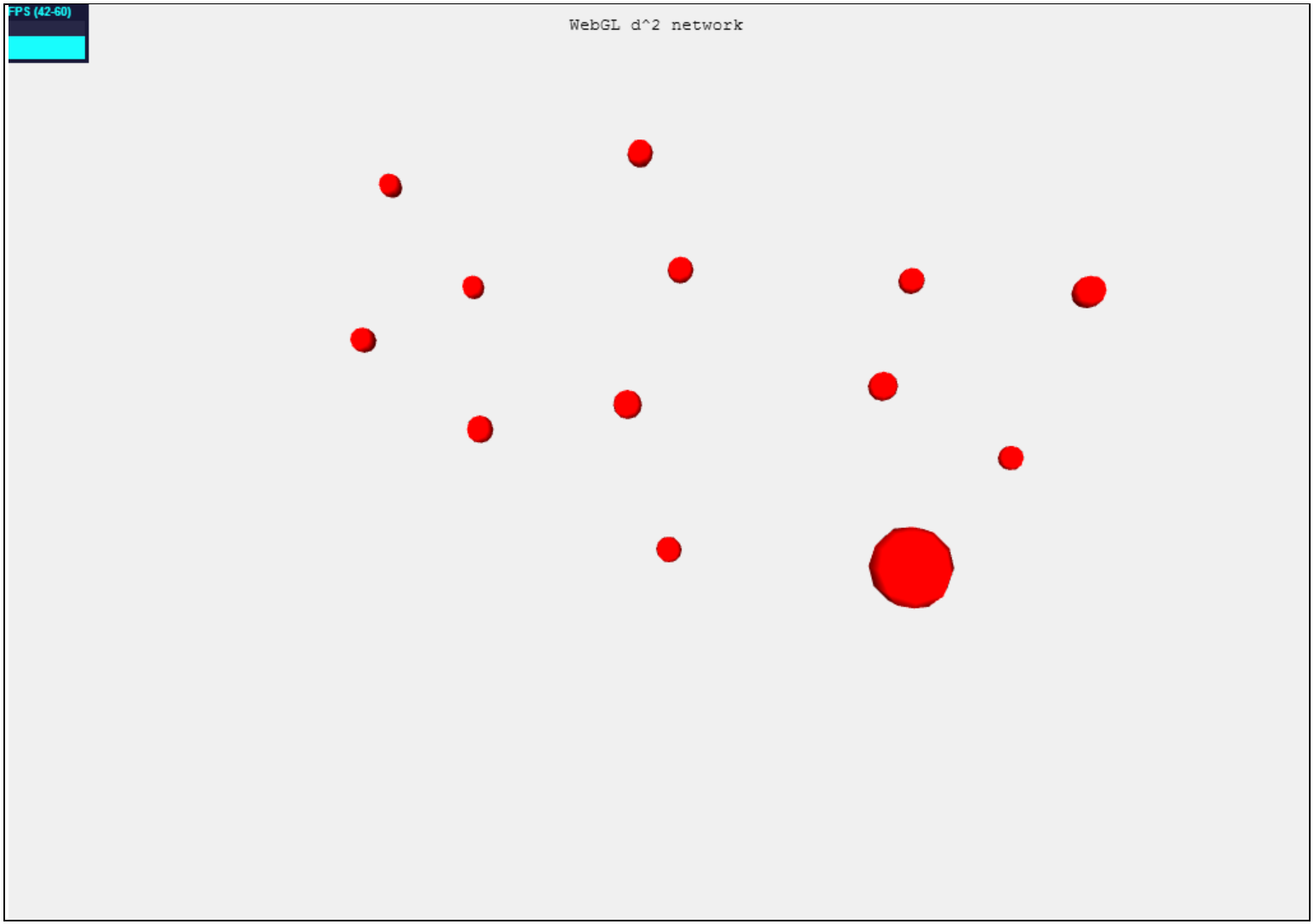

Figure B-1. A dynamically built 26-node network, where each node is dynamic. The network was built using the WebGL API, which is procedure-based (JavaScript).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml11/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xml:lang="en" lang="en">
  <head>
    <!-- Description:      WebGL XHTML application for dynamically building a          -->
    <!--                    dynamic network of nodes using JavaScript lib calls       -->
    <!--                    to WebGL, which calls OpenGL to control graphics          -->
    <!--                    processor unit.                                           -->
    <!--                                                                              -->
    <!-- By:               Andrew M. Neiderer, US ARL.                               -->
    <!--                                                                              -->
    <!-- Reference:        Ricardo Cabello Miguel (aka Mr.doob)                      -->
                                 -->
    <!--                    WebGL interactive draggable cubes.                        -->
    <!--                                                                              -->
    <!-- Date:             27 January 2013.                                          -->

    <meta charset="utf-8"/>
    <meta name="viewport"
          content="width=device-width, user-scalable=no,
                   minimum-scale=1.0, maximum-scale=1.0"/>

    <!-- WebGL cascading style sheet -->

    <link rel="stylesheet" type="text/css"
          href="webglbook.css"/>

    <style>
      body {
          font-family: Monospace;
          background-color: #f0f0f0;
          margin: 0px;
          overflow: hidden;
         }
    </style>
```

```
    <!-- minimal JavaScript libs by Mr.doob; original libs by  -->
    <!--  Tony Parisi, WebGL Up and Running.                    -->

    <script src="three.min.js"/>
    <script src="TrackballControls.js"/>
    <script src="stats.min.js"/>

    <title>
      WebGL dynamic build of a dynamic network of nodes
    </title>
  </head>

  <body>
    <script>
      // 3-component position vectors for network of nodes

      var npvs = 26;
      var ncmps = 3;
      var X = 0, Y = 1, Z = 2;

      var pv = new Array(npvs);

      for ( var i = 0; i &lt; npvs; i++ )
        pv[i] = new Array(ncmps);

      buildNetwork(pv,npvs);

      // for debugging in Mozilla Ff,
      //  Tools->Web Developer->Web Console

      console.debug(" ");
      console.debug("main()");
      console.debug(" ");

      for ( var i = 0; i &lt; npvs; i++ )
        console.debug("node " + i + "  [x,y,z] = [" +
                        pv[i][X] + "," +
```

```
                    pv[i][Y] + "," +
                    pv[i][Z] + "]");

// 3D position vectors for location of geometry nodes in network

function buildNetwork(pv,npvs)
{
  var X = 0, Y = 1, Z = 2;

  // for debugging

  console.debug(" ");
  console.debug("buildNetwork()");
  console.debug(" ");

  pv[0][X]  =  -2.01240; pv[0][Y]  =   1.71360; pv[0][Z]  =   0.6;
  pv[1][X]  =  -2.71140; pv[1][Y]  =   0.67860; pv[1][Z]  =   0.7;
  pv[2][X]  =  -0.91680; pv[2][Y]  =   1.85220; pv[2][Z]  =   0.8;
  pv[3][X]  =  -2.36220; pv[3][Y]  =  -0.56100; pv[3][Z]  =   0.9;
  pv[4][X]  =  -1.26480; pv[4][Y]  =  -1.01280; pv[4][Z]  =   1.0;
  pv[5][X]  =  -0.22200; pv[5][Y]  =  -0.10740; pv[5][Z]  =   1.1;
  pv[6][X]  =  -0.80580; pv[6][Y]  =  -0.79560; pv[6][Z]  =   1.2;
  pv[7][X]  =  -1.63560; pv[7][Y]  =  -0.44460; pv[7][Z]  =   1.1;
  pv[8][X]  =  -0.42720; pv[8][Y]  =   1.03740; pv[8][Z]  =   3.0;
  pv[9][X]  =   0.13140; pv[9][Y]  =   0.31920; pv[9][Z]  =   2.9;
  pv[10][X] =   0.20040; pv[10][Y] =   1.65120; pv[10][Z] =   2.8;
  pv[11][X] =   1.29960; pv[11][Y] =  -1.04640; pv[11][Z] =   2.7;
  pv[12][X] =   2.25300; pv[12][Y] =  -1.08660; pv[12][Z] =   2.6;
  pv[13][X] =  -0.01920; pv[13][Y] =   0.72840; pv[13][Z] =   2.6;
  pv[14][X] =  -0.76680; pv[14][Y] =   0.13980; pv[14][Z] =   2.7;
  pv[15][X] =   2.71140; pv[15][Y] =   0.12660; pv[15][Z] =   2.8;
  pv[16][X] =   1.13700; pv[16][Y] =   0.04260; pv[16][Z] =   2.9;
  pv[17][X] =   0.78060; pv[17][Y] =   1.05240; pv[17][Z] =   3.0;
  pv[18][X] =   1.93860; pv[18][Y] =   0.95520; pv[18][Z] =   1.1;
  pv[19][X] =   0.43320; pv[19][Y] =  -0.10620; pv[19][Z] =   1.2;
  pv[20][X] =   1.55640; pv[20][Y] =   0.44220; pv[20][Z] =   1.1;
  pv[21][X] =   2.07480; pv[21][Y] =  -0.31260; pv[21][Z] =   1.0;
```

```
  pv[22][X] =  -0.39420; pv[22][Y] =  -1.71900; pv[22][Z] =   0.9;
  pv[23][X] =   0.63240; pv[23][Y] =  -1.17960; pv[23][Z] =   0.8;
  pv[24][X] =   0.06060; pv[24][Y] =  -0.67980; pv[24][Z] =   0.7;
  pv[25][X] =   1.48680; pv[25][Y] =  -1.85220; pv[25][Z] =   0.6;
}

// geometry branch of scene graph for network of nodes

var container;

var scene;
var renderer;
var camera;

var projector;
var light;
var geometry;
var object, objects = [];
var plane;

var controls;
var stats;

var INTERSECTED;
var SELECTED;

var mouse = new THREE.Vector2();

var offset = new THREE.Vector3();

geometryBranchNetwork(pv,npvs);

animate();

// use JavaScript lib calls to WebGL to build
//  geometry branch of scene graph for network of nodes
```

```
function geometryBranchNetwork(pv,npvs)
{
  var X = 0, Y = 1, Z = 2;

  // for debugging

  console.debug(" ");
  console.debug("geometryBranchNetwork()");
  console.debug(" ");

  for ( var i = 0; i &lt; npvs; i++ )
    console.debug("node " + i + "  [x,y,z] = [" +
                  pv[i][X] + "," +
                  pv[i][Y] + "," +
                  pv[i][Z] + "]");

  container = document.createElement('div');

document.body.appendChild(container);

camera = new THREE.PerspectiveCamera(70.0,
                                     window.innerWidth / window.innerHeight,
                                     1.0,10000.0);

camera.position.z = 1000.0;

controls = new THREE.TrackballControls(camera);

controls.rotateSpeed = 1.0;
controls.zoomSpeed = 1.2;
controls.panSpeed = 0.8;
controls.noZoom = false;
controls.noPan = false;
controls.staticMoving = true;
controls.dynamicDampingFactor = 0.3;

scene = new THREE.Scene();
```

```
scene.add(new THREE.AmbientLight(0x505050));

light = new THREE.SpotLight(0xffffff,1.5);

light.position.set(0.0,500.0,2000.0);
light.castShadow = true;
light.shadowCameraNear = 200.0;
light.shadowCameraFar = camera.far;
light.shadowCameraFov = 50.0;
light.shadowBias = -0.00022;
light.shadowDarkness = 0.5;
light.shadowMapWidth = 2048.0;
light.shadowMapHeight = 2048.0;

scene.add(light);

geometry = new THREE.SphereGeometry(0.5);

for ( var i = 0; i &lt; npvs; i ++ ) {
  var object = new THREE.Mesh(geometry,
                                  new THREE.MeshLambertMaterial(
                                   {color: 0xff0000}));

  object.material.ambient = object.material.color;

  object.position.x = pv[i][X] * 10.0 + 100.0;
  object.position.y = pv[i][Y] *  1.0 + 100.0;
  object.position.z = pv[i][Z] * 10.0 + 600.0;
     console.debug("  object.position.x = " + object.position.x +
                   "  object.position.y = " + object.position.y +
                   "  object.position.z = " + object.position.z);

  scene.add(object);

  objects.push(object);
   }
```

```
      plane = new THREE.Mesh(new THREE.PlaneGeometry(2000.0,2000.0,8.0,8.0),
                             new THREE.MeshBasicMaterial({color: 0x000000,opacity: 0.25,
                                                          transparent: true,wireframe: true}));

      plane.visible = false;

      scene.add(plane);

      projector = new THREE.Projector();

      renderer = new THREE.WebGLRenderer({antialias: true});

      renderer.sortObjects = false;
      renderer.setSize(window.innerWidth,window.innerHeight);
      renderer.shadowMapEnabled = true;
      renderer.shadowMapSoft = true;

      container.appendChild(renderer.domElement);

      var info = document.createElement('div');

      info.style.position = 'absolute';
      info.style.top = '10px';
      info.style.width = '100%';
      info.style.textAlign = 'center';
      info.innerHTML = '<a href="http://threejs.org" target="_blank">three.js</a> WebGL d^2
network';

      container.appendChild(info);

      stats = new Stats();

      stats.domElement.style.position = 'absolute';
      stats.domElement.style.top = '0px';

      container.appendChild(stats.domElement);
```

```
renderer.domElement.addEventListener('mousemove',onDocumentMouseMove,false);
renderer.domElement.addEventListener('mousedown',onDocumentMouseDown,false);
renderer.domElement.addEventListener('mouseup',onDocumentMouseUp,false);

window.addEventListener('resize',onWindowResize,false);
 }

 // resize event

 function onWindowResize()
 {
camera.aspect = window.innerWidth / window.innerHeight;
camera.updateProjectionMatrix();

renderer.setSize(window.innerWidth,window.innerHeight);

 }

 // mousemove event

 function onDocumentMouseMove(event)
 {
event.preventDefault();

mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;

var vector = new THREE.Vector3(mouse.x,mouse.y,0.5);

projector.unprojectVector(vector,camera);

var ray = new THREE.Ray(camera.position,
                            vector.subSelf(camera.position).normalize());

if ( SELECTED ) {
  var intersects = ray.intersectObject(plane);
```

```
    SELECTED.position.copy(intersects[0].point.subSelf(offset));

    return;
  }

  var intersects = ray.intersectObjects(objects);

  if ( intersects.length > 0 ) {
    if ( INTERSECTED != intersects[0].object ) {
      if ( INTERSECTED )
           INTERSECTED.material.color.setHex(INTERSECTED.currentHex);

      INTERSECTED = intersects[0].object;
      INTERSECTED.currentHex = INTERSECTED.material.color.getHex();

      plane.position.copy(INTERSECTED.position);
      plane.lookAt(camera.position);
    }

    container.style.cursor = 'pointer';
  }
     else {
    if ( INTERSECTED )
         INTERSECTED.material.color.setHex(INTERSECTED.currentHex);

    INTERSECTED = null;

    container.style.cursor = 'auto';
     }
  }

  // mousedown event

  function onDocumentMouseDown(event)
  {
  event.preventDefault();
```

```
var vector = new THREE.Vector3(mouse.x,mouse.y,0.5);

projector.unprojectVector(vector,camera);

var ray = new THREE.Ray(camera.position,
                            vector.subSelf(camera.position).normalize());

var intersects = ray.intersectObjects(objects);

if ( intersects.length > 0 ) {
  controls.enabled = false;

  SELECTED = intersects[0].object;

  var intersects = ray.intersectObject(plane);

  offset.copy(intersects[0].point).subSelf(plane.position);

  container.style.cursor = 'move';
}
 }

 // mouseup event

 function onDocumentMouseUp(event)
 {
event.preventDefault();

controls.enabled = true;

if ( INTERSECTED ) {
  plane.position.copy(INTERSECTED.position);

  SELECTED = null;
}

container.style.cursor = 'auto';
```

```
      }

      // (see p. 28 of WebGL Up and Running by Tony Parisi, and/or
      //   p. 222 of WebGL Programming by Andreas Anyuru)

      function animate() {
        // ask for another frame before you start doing the current frame
      requestAnimationFrame(animate);

      render();

      stats.update();
      }

      // render the scene

      function render()
      {
      controls.update();

      renderer.render(scene,camera);
      }
    </script>
  </body>
</html>
```

List of Symbols, Abbreviations, and Acronyms

2-D	two-dimensional space
3-D	three-dimensional space
API	application programming interface
ARL	U.S. Army Research Laboratory
DOM	document object model
E	edge (or link)
EM	event model
EVA	exploratory visual analytics
HTML	hypertext markup language
IE	Microsoft Internet Explorer
npvs	number of position vectors
PV	position vector
SG	scene graph
V	vertex (or node)
VA	visual analytics
X3D	extensible 3-D Graphics specification
XHTML	extensible hypertext markup language

NO. OF
COPIES ORGANIZATION

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

1 (PDF) DIRECTOR US ARMY RESEARCH LAB IMAL HRA

1 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL

ABERDEEN PROVING GROUND

1 (PDF) DIR USARL RDRL CII C A NEIDERER