Embedding Interactive Haptic Objects in HTML

by

Trudy Liu

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degrees of Bachelor of Science in Electrical Engineering and Computer Science and Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

May 23, 1997

© 1997 Trudy Liu. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and to distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author ___________________________ Department of Electrical Engineering and Computer Science

Certified by ___________ J. Kenneth Salisbury

Thesis Supervisor

Accepted by ____________________________ Arthur C. Smith

Chairman, Department Committee on Graduate Theses
Embedding Interactive Haptic Objects in HTML

by

Trudy Liu

Submitted to the
Department of Electrical Engineering and Computer Science

May 23, 1997

In Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This paper describes the design and implementation of a Web browser plug-in which allows HTML documents to contain touchable interactive objects. The plug-in is loaded by the browser whenever such a document is encountered. Since the plug-in renders the embedded objects both graphically and haptically, the user is able to see the objects in the document and physically manipulate them, achieving a new level of interaction with the World Wide Web.

Thesis Supervisor: Dr. J. Kenneth Salisbury
Title: Principal Research Scientist
## Table of Contents

1 *Introduction* ................................................................................................................. 6

2 *Background* ...................................................................................................................... 7
   2.1 Haptics ......................................................................................................................... 7
   2.2 Plugins .......................................................................................................................... 8

3 *Design* .............................................................................................................................. 9
   3.1 Loading and Unloading ................................................................................................. 9
   3.2 Object Layout ............................................................................................................... 9
      3.2.1 Graphic Representation ......................................................................................... 10
      3.2.2 Haptic Representation .......................................................................................... 10
   3.3 EMBED Tag Parameters ............................................................................................. 11
      3.3.1 Plug-in Type ........................................................................................................... 11
      3.3.2 Files ....................................................................................................................... 11
      3.3.3 Dimensions ........................................................................................................... 12
      3.3.4 Camera Distance ................................................................................................... 12
      3.3.5 Color ...................................................................................................................... 12
      3.3.6 Java Applets .......................................................................................................... 13
      3.3.7 Plug-in Information ............................................................................................... 13
   3.4 Data Files ..................................................................................................................... 14
   3.5 Dynamic Interaction .................................................................................................... 14
      3.5.1 Behavior Types ....................................................................................................... 15
      3.5.2 URL Links .............................................................................................................. 15
      3.5.3 Java Interaction ..................................................................................................... 16
   3.6 Other Features ............................................................................................................ 17
      3.6.1 Error Detection ...................................................................................................... 17
      3.6.2 Full-page Mode ..................................................................................................... 17
      3.6.3 Viewing without Haptic Device ............................................................................ 18

4 *Implementation* .............................................................................................................. 19
   4.1 Overview ...................................................................................................................... 19
      4.1.1 Netscape Interface ................................................................................................. 19
      4.1.2 Vibe Class .............................................................................................................. 20
      4.1.3 Object Class .......................................................................................................... 21
      4.1.4 Callback Functions ............................................................................................... 22
      4.1.5 Java Object ........................................................................................................... 23
   4.2 Plug-in Initialization ..................................................................................................... 23
      4.2.1 Inventor Libraries ................................................................................................. 24
      4.2.2 Vibe Instance ....................................................................................................... 24
   4.3 Object Creation ........................................................................................................... 24
      4.3.1 Object Instance ..................................................................................................... 25
      4.3.2 Parameters ............................................................................................................ 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.3 File Processing</td>
<td>27</td>
</tr>
<tr>
<td>4.3.4 Graphic to Haptic Conversion</td>
<td>29</td>
</tr>
<tr>
<td>4.3.5 Adding Object to Scene</td>
<td>32</td>
</tr>
<tr>
<td>4.3.6 Window Management</td>
<td>32</td>
</tr>
<tr>
<td>4.4 Scene Graphs</td>
<td>34</td>
</tr>
<tr>
<td>4.4.1 Sub-object</td>
<td>35</td>
</tr>
<tr>
<td>4.4.2 Object</td>
<td>36</td>
</tr>
<tr>
<td>4.4.3 Vibe</td>
<td>37</td>
</tr>
<tr>
<td>4.5 Events</td>
<td>38</td>
</tr>
<tr>
<td>4.5.1 URL</td>
<td>39</td>
</tr>
<tr>
<td>4.5.2 Java</td>
<td>39</td>
</tr>
<tr>
<td>4.6 Plug-in Shutdown</td>
<td>39</td>
</tr>
<tr>
<td>5 Application</td>
<td>41</td>
</tr>
<tr>
<td>5.1 Installation and Setup</td>
<td>41</td>
</tr>
<tr>
<td>5.1.1 Netscape and Open Inventor</td>
<td>41</td>
</tr>
<tr>
<td>5.1.2 Vibe Files</td>
<td>41</td>
</tr>
<tr>
<td>5.1.3 PHANToM</td>
<td>42</td>
</tr>
<tr>
<td>5.2 EMBED Tag Specifications</td>
<td>42</td>
</tr>
<tr>
<td>5.2.1 TYPE</td>
<td>43</td>
</tr>
<tr>
<td>5.2.2 FILE</td>
<td>43</td>
</tr>
<tr>
<td>5.2.3 WIDTH</td>
<td>43</td>
</tr>
<tr>
<td>5.2.4 HEIGHT</td>
<td>43</td>
</tr>
<tr>
<td>5.2.5 CAMERADIST</td>
<td>44</td>
</tr>
<tr>
<td>5.2.6 BGCOLOR</td>
<td>44</td>
</tr>
<tr>
<td>5.2.7 CURSORCOLOR</td>
<td>45</td>
</tr>
<tr>
<td>5.2.8 APPLET</td>
<td>45</td>
</tr>
<tr>
<td>5.2.9 PLUGINSPAGE</td>
<td>45</td>
</tr>
<tr>
<td>5.3 Data File Specifications</td>
<td>46</td>
</tr>
<tr>
<td>5.3.1 DYNAMICFRICTION</td>
<td>46</td>
</tr>
<tr>
<td>5.3.2 STATICFRICTION</td>
<td>47</td>
</tr>
<tr>
<td>5.3.3 DAMPING</td>
<td>47</td>
</tr>
<tr>
<td>5.3.4 SPRING</td>
<td>47</td>
</tr>
<tr>
<td>5.3.5 DYNAMIC</td>
<td>48</td>
</tr>
<tr>
<td>5.3.6 TRANSFORM</td>
<td>48</td>
</tr>
<tr>
<td>5.3.7 ID</td>
<td>48</td>
</tr>
<tr>
<td>5.3.8 URL</td>
<td>49</td>
</tr>
<tr>
<td>5.3.9 MASS</td>
<td>49</td>
</tr>
<tr>
<td>5.3.10 DYNAMICDAMPING</td>
<td>49</td>
</tr>
<tr>
<td>5.3.11 K</td>
<td>50</td>
</tr>
<tr>
<td>5.3.12 DEADBAND</td>
<td>50</td>
</tr>
<tr>
<td>5.3.13 RESTORINGFORCE</td>
<td>50</td>
</tr>
<tr>
<td>5.3.14 THROWDISTANCE</td>
<td>50</td>
</tr>
<tr>
<td>5.3.15 NUMBERNOTCHES</td>
<td>51</td>
</tr>
<tr>
<td>5.3.16 INITIALNOTCH</td>
<td>51</td>
</tr>
<tr>
<td>5.3.17 DISTANCE</td>
<td>51</td>
</tr>
<tr>
<td>5.4 Java Applet Specifications</td>
<td>52</td>
</tr>
</tbody>
</table>
6 Future Work ................................................................................................. 53
  6.1 Browser Interface .................................................................................. 53
  6.2 Event Types .......................................................................................... 54
  6.3 Java-controlled Elements ..................................................................... 54
7 Conclusion .................................................................................................. 55
Appendix: Source Code .................................................................................. 56
Acknowledgments ......................................................................................... 88
References ...................................................................................................... 88
1 Introduction

The purpose of this thesis project is to enable the embedding of interactive haptic objects in HyperText Markup Language (HTML). This involves the design and implementation of a browser plug-in capable of reading data from haptic object files and rendering those objects both graphically in the browser window and haptically on the haptic device.

The motivation for this project stems from the fact that current HTML documents can only provide information through visual and audio components — text, images, and sounds. By combining these elements with touchable, moveable objects, a completely new type of document can be published on the Web, one that is capable of conveying information more effectively and intuitively.
2 Background

This section presents brief background information on haptics and plug-ins.

2.1 Haptics

The definition of the word “haptic”, according to Webster’s New World Dictionary, is “of or having to do with the sense of touch; tactile” [1]. Humans have always relied on the sense of touch to interact with objects around them, and haptics makes it possible to create that type of interaction with a computer. Haptic interfaces are devices that can simulate the sensation of touching a physical object [2]. These devices can be used for many applications, including simulations, training, remote control, visualization, and object manipulation [1].

The haptic interface used in this project is the PHANToM, a device developed in 1993 by Massie and Salisbury at the Massachusetts Institute of Technology Artificial Intelligence Laboratory [2]. The PHANToM has a moveable arm with a finger thimble at the end, and it tracks the position of the user’s fingertip as the user’s finger moves within the workspace. By exerting a force vector on the fingertip, the device creates the illusion that the user is touching a physical object [3]. Various force-displacement profiles can be applied to haptic objects to make them behave in ways similar to objects found in the real world [4].
2.2 Plugins

A Netscape plug-in is a dynamic code module that extends the capabilities of the browser by handling a special data type. When the browser encounters a page containing embedded objects that are of the Multimedia Internet Message Extensions (MIME) type handled by the plug-in, it loads the plug-in and gives it part of the browser window. Plug-in objects are embedded with the <EMBED> tag, which can contain special parameters specific to the plug-in in addition to the standard plug-in parameters. Plug-ins are seamlessly integrated into the browser and are capable of receiving input events, as well as launching and posting to Uniform Resource Locators (URLs) [5]. Also, plug-ins can call Java methods, and by associating a Java class with the plug-in, native plug-in methods can be called by Java [6].
3 Design

The plug-in developed for this project, named Vibe (virtual interaction browser extension), enables haptic objects to be embedded in HTML documents. Objects are embedded via a special tag in the HTML file, which contains parameters specifying various attributes for an object and indicates which data files are to be used for the object.

3.1 Loading and Unloading

The Vibe plug-in is a dynamic-link library (DLL) which is loaded whenever the browser loads a page containing at least one embedded haptic object. When the browser leaves the page, the plug-in is unloaded. The haptic device is initialized each time the plug-in is loaded, and is disabled upon the unloading of the plug-in.

3.2 Object Layout

Embedded objects can be inserted anywhere in an HTML document, and usual formatting options for text and images, such as centering, also apply to embedded haptic objects. Haptic objects can also be placed within tables. All haptic objects on the page are automatically repositioned when the user resizes the window, just like all other elements on the page.
3.2.1 Graphic Representation

The graphic representation of a haptic object is placed within a rectangular subpart of the browser window, and appears like an inline image. A haptic object is depicted graphically by the Open Inventor rendering of the nodes comprising the object.

The scene camera is a perspective one, and is placed on the z-axis pointing toward the negative z direction. The far clipping plane of the camera is set so that anything beyond the x-y plane (z = 0) in the negative z direction is not seen.

The haptic interaction point is depicted by a small sphere which moves along with the user's finger and acts as a cursor to graphically indicate the position of the finger. The sphere is visible only within plug-in areas, and moves from one area to another as the user touches different objects on the page.

3.2.2 Haptic Representation

Each haptic object is positioned within virtual space so that the x-y plane in the local coordinates of the objects coincides with the x-y plane in world coordinates. An impenetrable haptic "wall" is placed at the x-y plane to represent the plane of the document page. Each object may protrude out of this "wall" to any extent, as specified by its parameters. While some objects may be partially buried within the "wall", others may lie completely outside it.

Within the document plane, objects are positioned so that their haptic positions relative to each other are the same as their graphic positions on the page layout, with the center (x = 0, y = 0) in world coordinates corresponding to the center of the browser
window. Whenever the graphic locations of the objects change due to a window resize, the haptic objects change their locations accordingly.

3.3 EMBED Tag Parameters

Each haptic object in an HTML document is inserted by using an EMBED tag. There are several parameter types which can be included in this tag. Some of these are common to all plug-in types, while others are special tags specific to the Vibe plug-in.

3.3.1 Plug-in Type

The MIME type of the embedded haptic object is specified by the TYPE parameter, which should have the value “application/x-haptic-object”. This parameter is required for the Vibe plugin, since the haptic object MIME type is not officially registered with the Internet Assigned Numbers Authority (IANA). By including the TYPE parameter in the EMBED tag, the browser will correctly load the Vibe plug-in when it encounters a haptic object.

3.3.2 Files

The FILE parameter specifies the name of a haptic object data file to be used for the object. When this parameter is included multiple times, the contents of all the files specified are superimposed on each other in virtual space in such a way that the local coordinate origin of all the files coincide with each other. Since each file can only have one associated dynamic behavior, this allows an object to contain multiple parts, each with a different dynamic behavior.
In this document, the term “object” will refer to one haptic instance of the `EMBED` tag, whether it is composed of just one data file or multiple data files. Each object has its own rectangular area in the browser window. The term “sub-object” will refer to the haptic element corresponding to one data file, whether it by itself is an entire object or is combined with other sub-objects to form an object. Sub-objects from the same object will occupy the same rectangular area in the browser window.

### 3.3.3 Dimensions

The `WIDTH` and `HEIGHT` parameters specify the width and height, respectively, of the rectangular area occupied by the plug-in in the browser window. These parameters, defined in pixels, specify only the perceived graphic size of the object and have no effect on the actual haptic size of the object.

### 3.3.4 Camera Distance

The perceived graphic size of a haptic object is also affected by the distance of the camera from the object. This distance can be specified by the `CAMERADIST` parameter, which indicates, in millimeters, how far the camera is placed from the origin in the object’s local coordinates. If this parameter is omitted, the plug-in will by default attempt to place the camera at a distance which allows it to view the entire object.

### 3.3.5 Color

The `BGCOLOR` parameter can be used to specify the background color of the plug-in area. This parameter is specified in the format `#rrggbb`, and when identical to the...
BGDCOLOR parameter used in the BODY tag of the HTML document, the plug-in will blend in perfectly with the rest of the document, giving the object the appearance of floating in the document.

Since the plug-in allows for different background colors, and the objects themselves can have any color, the color of the graphical cursor sphere can be specified so that it has high contrast and is more easily visible to the user. This is done via the CURSORCOLOR parameter, which is specified using the same format as BGDCOLOR.

3.3.6 Java Applets

Each embedded haptic object can communicate with one Java applet located in the same document. The APPLET parameter specifies the name of the applet that the object is to communicate with. Different objects on the same page can communicate with different applets. If this parameter is omitted for an object, then the object does not communicate with a Java applet.

3.3.7 Plug-in Information

The PLUGINSPAGE parameter can be used to specify a URL location where instructions on downloading and installing the plug-in can be obtained. This optional parameter, when supplied, provides a first-time user with the information necessary to find and setup the plug-in.
3.4 Data Files

A file containing specifications for a haptic sub-object is of the MIME type “application/x-haptic-object” and has the .hap suffix, which indicates that it is a haptic object file. As mentioned previously, files can be used alone or in combination with other files. Thus, each file provides the specifications for one sub-object, which can be an object by itself or be combined with other sub-objects to form an object.

The files used by the Vibe plug-in are in the standard Open Inventor format, using a subset of the Open Inventor node types. Thus, haptic sub-objects can have attributes such as visual material properties and texture maps. Each file defines a scene graph, which is read in by the plug-in and converted into a haptic element.

Special haptic attributes are specified by inserting information nodes in the scene graph, which affect graph traversal state in the same way transformations do. The plug-in supports the following surface attributes: dynamic friction, static friction, damping, and spring coefficients. Each information node can define a value for one of these attributes. If these values are not specified, default values are used.

3.5 Dynamic Interaction

Each part of a haptic object can be either static or dynamic. Static sub-objects do not move, and are simply felt by the user, while dynamic sub-objects can be pushed and moved around by the user. Dynamic sub-objects are also capable of generating events.
3.5.1 Behavior Types

There are three dynamic behavior types available in the Vibe plug-in: button, dial, and slider. A button can be pushed along one axis a specified distance, then released, causing it to bounce back to its original position. It generates an event whenever it is pressed or released. A dial is rotated about one axis, snapping into predefined notches. It generates an event whenever it lands in a notch. A slider can be moved back and forth a specified distance along one axis, and like the dial, it snaps into predefined notches and generates an event whenever it lands in a notch. Each sub-object data file can be associated with one dynamic behavior. Objects which require more than one dynamic part can include multiple behaviors by using multiple sub-objects. The dynamic type of a sub-object is defined by an information node inserted in the data file.

3.5.2 URL Links

Each dynamic sub-object can generate an event which will load a specified URL in the browser window. Sub-objects which utilize this feature must insert an information node in the data file which specifies the URL. For buttons, the URL is launched whenever the button is pressed. For dials and sliders, the URL is launched whenever a notch other than the initial notch is landed in. This feature provides a easy way to make haptic URL links in an HTML document.
3.5.3 Java Interaction

For more complex event behavior, each dynamic sub-object can send event notifications to a Java applet residing in the HTML document. Different objects can communicate with different applets, while sub-objects from the same object will communicate with the same applet.

The name of the applet which receives events from an object is specified in the APPLET parameter of the EMBED tag, as mentioned previously. The applet itself must specify its own name by including the NAME parameter in the APPLET tag. Furthermore, the applet class must implement the HapticApplet interface, which ensures that it has the functions necessary for handling object events.

When an applet receives notification of an event, it is supplied with the sub-object ID number, which is specified by an information node in the sub-object data file. The applet also receives an integer, which represents the current state of a button or the current notch of a slider or dial, depending on the behavior type of the sub-object.

Once an applet receives an event notification, it can do whatever it wants, including playing a sound clip, drawing objects in its window, launching a URL, or changing the state of its variables. It can also ignore events that it is uninterested in, or pass the events on to another applet on the page. The plug-in to applet communication provides a lot of versatility in the kind of HTML documents that can be created.
3.6 Other Features

The Vibe plug-in also supports other features geared toward user-friendliness and convenience.

3.6.1 Error Detection

There is a good error-detection mechanism built into the plug-in. Since the plug-in relies heavily on data files, it will warn the user whenever there is anything wrong with the format of a file. The types of errors that can occur include an invalid format caused by mistyped characters, graphic nodes that are unsupported haptically, and information nodes that are misnamed or misplaced. Also, sometimes a specified file may not exist. In each case, the plug-in will display a dialog which states the file name and a brief message describing the type of problem encountered. In the case of unsupported nodes, it will list exactly which nodes are unsupported.

3.6.2 Full-page Mode

While the main use of the Vibe plug-in is to embed haptic objects in HTML documents, a user may also open a .hap file that is located on the local machine in the browser window by itself. This will result in the plug-in filling the entire window, with the sub-object from the data file centered in the window. The haptic cursor is enabled just like in the embedded plug-in, and dynamic objects are also active and can be moved. This mode of operation is useful for viewing a sub-object by itself and tweaking its attributes to ready it for use in an HTML document.
3.6.3 Viewing without Haptic Device

The Vibe plug-in is implemented so that haptic HTML documents can still be viewed without a haptic device. Users without a haptic device will see all the haptic objects in their initial state, but will not be able to move them or generate events with them. Documents that use Java applets can provide alternative ways of generating the same events in the applets without requiring the haptic objects to be used.
4 Implementation

The Vibe plug-in was implemented on a machine running Windows NT 4.0, using Microsoft Developer Studio. For graphics and haptics rendering, the plug-in uses the Open Inventor and GHOST toolkits, respectively. The source code for the project can be found in the appendix.

4.1 Overview

The implementation of the plug-in can be divided into different parts: functions that comprise the interface to Netscape, a class that represents the collection of objects, a class that represents one object, callback functions for events, updates, and errors, and a Java class associated with each instance of the plug-in.

4.1.1 Netscape Interface

The functions which are called by Netscape are implemented in the file npshell.cpp. These function have names prefixed by NPP_ and serve as the interface between the plug-in and the browser. Functions implemented by the Vibe plug-in:

- **NPP_Initilize** and **NPP_Shutdown**: These functions are called when the browser loads and unloads the plug-in. Resources used by all instances of the plug-in on the page are allocated and de-allocated by these functions.

- **NPP_GetJavaClass**: This function associates a Java class with the plug-in. An object of this class is created for each instance of the plug-in.
• **NPP New and NPP Destroy**: These functions are called for each embedded object on the page. Code for the creation and destruction of an object are placed here.

• **NPP SetWindow**: This function is called for each object when a window area is initially assigned to it, and called again every time the browser window is resized.

• **NPP NewStream and NPP StreamAsFile**: These functions are called whenever an object reads from one of its data files. Calling these functions invoke code that sends the data to the plug-in in a file, which is then processed.

### 4.1.2 Vibe Class

The **Vibe class** oversees all the objects embedded on the page, and represents the haptic scene in which the objects reside. It keeps track of the handle to the browser window, the haptic process and the haptic root node, the timer which invokes scene update, and the graphic PHANToM cursor, which is shared by all the objects on the page.

The public interface to the **Vibe class**:

• **Vibe and ~Vibe**: The constructor and destructor for the class, which perform all the necessary initialization and shutdown procedures.

• **setWindow and getWindow**: These methods set and get the handle of the browser window, which is the parent window of all the object windows.

• **addObject and removeObject**: These methods add or remove a haptic object from the virtual "world", and are called for each object that is created or destroyed.

• **handleError**: This method handles any error generated by the haptic process, and displays a dialog which indicates the source of the problem.
4.1.3 Object Class

The Object class represents one instance of an embedded haptic object. It keeps track of the plug-in instance handle, the graphic and haptic scene graph for the object, the window area occupied by the object, the scene camera, and object parameters. The public interface to the Object class:

- Object and ~Object: The constructor and destructor of the class, which take care of creating and destroying a haptic object.

- setParameter: This method reads in a parameter specified in the EMBED tag and processes it accordingly.

- createWindow, getWindow, and setWindowPosition: These methods take care of creating a window area to be used by the object, getting the window handle when it is needed, and setting the new window position whenever the browser window is resized.

- addGraphicPHANToM and getHapticObject: These methods are used by the Vibe class when an object is added to the overall scene. They allow all the objects to share the same PHANToM cursor and to have their haptic counterparts added to the same haptic scene.

- processFile: This method is called for each data file used by the object. It reads the file and converts the contents into the appropriate graphic and haptic elements.
• dynamicEvent: This method handles an event generated by one of the object’s parts, and calls the appropriate Java method and/or launches the specified URL, depending on which parameters have been set.

4.1.4 Callback Functions

The Vibe plug-in uses callback functions to update the graphics, handle dynamic events, and handle errors. These are global functions that are first registered with the appropriate elements and then called by the process when needed.

• sceneUpdateCallback: This function is called by the plug-in 10 times per second, and notifies the haptic process to call any registered graphic and event callback functions.

• positionUpdateCallback and dynamicGraphicCallback: These functions are responsible for updating the graphic state so that it matches the haptic state. One function updates the PHANToM cursor, while the other updates the dynamic sub-objects.

• buttonEventCallback, dialEventCallback, and sliderEventCallback: These functions are called whenever a haptic sub-object generates an event. They take care of extracting the relevant event information and passing that to the object method for processing.

• errorCallback: When the haptic process encounters a problem, this function is called so that the Vibe object is notified of the error and can handle it accordingly.
4.1.5 Java Object

Each instance of the plug-in, that is, each embedded haptic object, has a Java object associated with it which acts as the link between the plug-in and the applet that receives events from the plug-in. The Java object class, VibePlugin, is derived from the netscape.plugin.Plugin class. Each instance of VibePlugin keeps track of the plug-in instance that it is associated with, as well as the name of the Java applet that it is to communicate with.

Java applets which want to receive haptic events must implement the HapticApplet interface, which requires the implementation of three methods: buttonEvent, dialEvent, and sliderEvent. When an event is generated by a haptic sub-object, the object to which the sub-object belongs passes the event information to the associated VibePlugin object, which then invokes the appropriate method in the specified applet.

4.2 Plug-in Initialization

The Vibe plug-in is loaded when a browser launches a document containing one or more embedded haptic objects. At this time, NPP_Initiate is called, and initialization is performed for resources shared by all the objects on the page. The Java class associated with the plug-in is also loaded.
4.2.1 Inventor Libraries

The first step in the initialization process is the loading of the Open Inventor libraries via the function LoadLibrary. This is done for both of the Open Inventor DLLs needed by the plug-in.

4.2.2 Vibe Instance

Once the necessary DLLs have been loaded, an instance of the Vibe class is created. This instance is global to the plug-in and will affect all the haptic objects on the page. When it is created, the Vibe constructor does the following:

- Initialize the Open Inventor database.
- Create a new scene object which will represent the overall haptic scene and manage the haptic process.
- Create a gstSeparator node named hapticRoot and set it to be the root node of the haptic scene.
- Create a SoSeparator node named graphicPHANToM which will represent the graphic PHANToM cursor. The sub-tree rooted at this node will be shared by all objects on the page.

4.3 Object Creation

Once the global initialization for the plug-in has been performed, the object creation process is invoked for each embedded object on the page via the NPP_New function.
4.3.1 Object Instance

For each embedded object, an instance of the Object class is created. The constructor does the following:

- Save the pointer to the instance of the plug-in for this object, which is passed in as an argument to the constructor.

- Initialize fields which will have their values set later by other methods.

- Create a SoSeparator node named graphicRoot which will be the graphic root node for the object.

- Add to graphicRoot a SoTranslation node named graphicObjectTranslation which determines the object’s position.

- Add to graphicRoot a SoPerspectiveCamera named camera for viewing the scene. The near clipping plane of camera is set immediately in front of its lens, and camera is placed at \((0, 0, 1)\) pointing toward the negative z direction.

- Add to graphicRoot a SoDirectionalLight light source, pointed in the negative z direction.

- Add to graphicRoot a SoSeparator node named graphicObject which will represent the graphic version of the object.

- Add to graphicRoot another SoTranslation node, named worldSpaceTranslation, which exactly offsets graphicObjectTranslation and brings the transformation state back to world coordinates.
• Create a \texttt{gstSeparator} node named \texttt{hapticObject} which represents the haptic object. This node is the haptic version of \texttt{graphicObject}.

4.3.2 Parameters

Once the \texttt{Object} instance is created, the plug-in will take the parameters specified in the \texttt{EMBED} tag and pass them to the object one by one for processing by calling the \texttt{setParameter} method.

• For the \texttt{FILE} parameter: The object takes the file name and requests a new data stream which will pass the file to the plug-in for processing.

• For the \texttt{CAMERADIST} parameter: The object applies the distance specified to camera, positioning it at \((0, 0, distance)\) in the object's local coordinate space. The object also notes that the camera distance was specified in the tag.

• For the \texttt{BGCOLOR} parameter: The object first verifies that the value specified is in the format \texttt{#rrggbb}. If the value has the correct format, then it is converted into a color variable that is saved by the object.

• For the \texttt{CURSORCOLOR} parameter: The value is verified and then converted to a color variable. A \texttt{SoMaterial} node is created, and the color specified is set as the diffuse color. The node is then added to \texttt{graphicRoot}.

• For the \texttt{APPLET} parameter: The object retrieves a pointer to the \texttt{VibePlugin} Java object associated with it, then passes the specified applet name to the \texttt{VibePlugin} object, which stores it for future use. The object also notes that an applet name was specified.
• The parameters TYPE, WIDTH, HEIGHT, and PLUGINSPAGE are handled directly by Netscape and are not used by the object. The object also ignores any other parameter included in the EMBED tag not listed above.

4.3.3 File Processing

For each data file specified in the parameters of the EMBED tag, a new stream is created which will send the file to the plug-in object via the function processFile. When the object receives the file name, it will read the file and convert the contents to a graphic scene graph.

First, the object will search the scene graph for a SoInfo node named DYNAMIC, which specifies the dynamic behavior type of the sub-object in the file. If no such a node is found, then the sub-object is static, and the graphic scene graph is added to graphicRoot. The graphic scene graph is then converted into a haptic scene graph, which is added to hapticRoot.

If, however, the file contains a dynamic behavior type specification, then a dynamic sub-object of the appropriate type is created:

• First, an SoSeparator node representing the graphic dynamic sub-object is created and added to graphicRoot, and a gstSeparator node representing the haptic dynamic sub-object is created and added to hapticRoot.

• The graphic scene graph from the file is searched for an SoTransform node named TRANSFORM which specifies the position of the dynamic sub-object. If such a node is found, then it is removed from the graphic scene graph and added to the graphic
dynamic node. The transform which it represents is then applied to the haptic dynamic node. If no such transform node is found, then the dynamic sub-object is assumed to be positioned at (0, 0, 0) in the object’s local coordinate space.

- A new SoTransform node is created and added to the graphic dynamic node. This node represents the position of the moving part and is modified whenever the sub-object is moved, so that the graphic representation matches the haptic representation.

- The graphic scene graph from the file is added to the dynamic node.

- A new gstButton, gstDial, or gstSlider is created and added to the haptic dynamic node. The graphic scene graph from the file is converted to a haptic scene graph, and added to this dynamic element.

- The graphic scene graph is searched for dynamic sub-object parameters. Parameters common to all three dynamic types are MASS, DYNAMICDAMPING, K, ID, and URL. MASS, DYNAMICDAMPING, and K specify the dynamic attributes of the sub-object. ID is an integer identifying the sub-object for event processing purposes, and URL specifies the URL to be launched when the sub-object is moved. Each dynamic type also has its own specialized parameters: buttons have DEADBAND, RESTORINGFORCE, and THROWDISTANCE; dials and sliders have NUMBERNOTCHES and INITIALNOTCH; and sliders have DISTANCE. For each SoInfo node specifying a parameter that is found in the graphic scene graph, the value is applied to the sub-object.
• An event callback function is registered for this dynamic sub-object, with the user-defined parameter containing a pointer to the object, the sub-object ID, and the URL.

• A graphics callback function is registered for this dynamic sub-object, with the user-defined parameter containing a pointer to the dynamic object transform node.

Once the file has been processed, and the appropriate static or dynamic sub-object has been added to the object, the camera is repositioned so that it can see the entire object if the CAMERADIST parameter has not been set.

4.3.4 Graphic to Haptic Conversion

A graphic scene graph is converted into an equivalent haptic scene graph by traversing the graphic tree depth first and visiting each node in the graph, generating in the process a haptic tree with the appropriate nodes. During the traversal, four things need to be kept track of:

• Current values of haptic surface properties. This includes dynamic friction, static friction, damping coefficient, and spring constant. These parameters affect the feel of a haptic element. The current values are the values specified by the most recently traversed \texttt{SoInfo} node for each property.

• Current transformation matrix. The transform matrix represents the cumulative effect of all the transformation nodes that have been visited. This includes translation, rotation, and uniform scaling, all of which affect how a haptic element is positioned and sized in virtual space.
• Current value of non-uniform scaling. Non-uniform scaling is only performed on a leaf node corresponding to a shape. Thus, transformation nodes which specify non-uniform scaling must be positioned immediately before a leaf node in tree traversal. The values specified are kept track of so that they can be applied accordingly.

• Current SoCoordinate3 node. This is simply the most recently traversed SoCoordinate3 node. This node is used by SoIndexedFaceSet and SoIndexedTriangleStripSet nodes.

The tree traversal begins with the root node of the graphic scene graph. Traversal is performed depth first, and upon encountering each node:

• If the node is a SoSeparator or SoGroup node: A corresponding gstSeparator node is created in the haptic scene graph. For a SoSeparator node, a copy of all the current state values is made, and further state changes under that node are made on the new copy, so that the state can be restored to the original values when traversal leaves that SoSeparator node. For a SoGroup node, any modifications to the state are simply made on the original copy, so that state changes under the node are preserved when traversal leaves that SoGroup node.

• If the node is a SoInfo node: The name of the node to checked to see if the node specifies a haptic surface property. If so, the current value of that property is updated. Four property types are possible: DYNAMICFRICTION, STATICFRICTION, DAMPING, and SPRING.
- If the node is derived from SoTransformation: When a transformation node is encountered, the transformation matrix that it represents is multiplied to the current transformation matrix, so that the current matrix represents the cumulative effect of every transformation node visited so far. Non-uniform scaling is multiplied to the current values. The supported transformation node types are SoTransform, SoTranslation, SoRotation, SoRotationXYZ, and SoScale.

- If the node is a SoCoordinate3 node: The node becomes the current SoCoordinate3 node, to be used by subsequent shape nodes that need it.

- If the node is derived from SoShape: The corresponding node derived from gstShape is created in the haptic scene graph. The current transformation matrix and current non-uniform scale are applied to the node, as well as the current surface properties. The node is then added as a child to the most recently created gstSeparator node. Supported SoShape nodes include SoCone, SoCube, SoCylinder, and SoSphere, which correspond to gstCone, gstCube, gstCylinder, and gstSphere, respectively. SoIndexedFaceSet and SoIndexedTriangleStripSet nodes are converted to gstPolyMesh, and make use of the current SoCoordinate3 node.

- If the node is derived from SoCamera: Any camera node encountered during tree traversal is removed from the scene graph, since the plug-in relies on its own camera node to view the object.
• Nodes which affect purely visual elements, such as SoMaterial, SoTexture2, and nodes derived from SoLight, are simply ignored since they do not affect the haptic scene graph in any way. Nodes which affect haptic elements but are not supported are listed in a dialog box warning when tree traversal is finished.

4.3.5 Adding Object to Scene

After the object has been created, it is added to the Vibe instance via a call to addObject. This adds the hapticObject node from the object to hapticRoot in the Vibe instance, so that the haptic object is added to the scene. Also, the shared graphicPHANToM node is added to graphicRoot in the object so that the PHANToM cursor can be viewed in the object’s window area.

4.3.6 Window Management

For each embedded object on the page, Netscape allocates a child window in the browser and passes the handle of that window to the plug-in, via a call to setWindow. This function is also called every time the browser window is resized, so that information regarding new window positions is passed to the plug-in. When window information changes, setWindow is called for each object instance.

The very first time that setWindow is called with a non-NULL window handle, the parent of that window is set as the window for the Vibe instance. Thus, the Vibe window is the parent window of all the Object windows. The Vibe instance then proceeds to display a dialog box for initializing the haptic device. Once the haptic device
has been initialized, a graphic cursor which is to be shared by all objects is created, and
the haptic servo loop is started. A haptic “wall” placed at \( z = 0 \) is also created.

When \text{setWindow} \ is called to initialize a window for each object, the object
creates a render area in that window, and sets graphicRoot to be the root of that render
area. If the \text{BGColor} parameter was set, then the render area’s background color is set to
the specified value (default is \#000000). The aspect ratio of the object camera is set to
equal the ratio of the width to the height of the window. The object also keeps track of the
dimensions of the window, which are used to position the objects in virtual space.

During initialization, and for each subsequent browser window resize, the position
of the object window is passed to the object as the coordinates of the upper left corner
with respect to the upper left corner of the browser window (window units are in pixels).
For an object window with dimensions width \( \times \) height, and position \((x, y)\):

\[ (0, 0) \]

\[ \text{browser window} \]

\[ \text{object window} \]

\[ \text{browser height} \]

\[ \text{browser width} \]

\[ (\text{browser width}, \text{browser height}) \]
Given the dimensions and position of the object window, as above, the object would be placed in virtual space so that \( z = 0 \) coincides with the plane of the document, and \( x \) and \( y \) correlate with the relative position of the object window. The center of the browser window corresponds to the coordinate origin in virtual space. Virtual units are in millimeters, and a conversion factor of three pixels per millimeter, which would work with most displays, is used. The object is positioned as follows:

\[
X = \frac{x + \frac{\text{width}}{2} - \frac{\text{browserwidth}}{2}}{\text{pixelspermm}}
\]

and

\[
Y = \frac{\frac{\text{browserheight}}{2} - y - \frac{\text{height}}{2}}{\text{pixelspermm}}
\]

4.4 Scene Graphs

The Vibe plug-in makes use of Open Inventor and GHOST nodes to represent the graphic and haptic portions of embedded objects. The following are scene graph structures used by the plug-in. Solid lines indicate parent-child relationships between nodes. A dashed line between two nodes indicates an interdependence between those nodes, and a node enclosed in a dashed box does not originate from the indicated class but is shared from another class.
4.4.1 Sub-object

A sub-object represents data from one file. Objects can be composed of any number of sub-objects, static and dynamic. Graphic sub-objects are added to the graphic object structure, and haptic sub-objects are added to the haptic object structure.

A static sub-object is simply the graphic scene graph read from the data file, and the haptic scene graph that is its counterpart:

A dynamic sub-object requires a more complicated node structure to handle the placement of the sub-object as well as the changing dynamic position:
4.4.2 Object

The Object class keeps track of the graphic and haptic nodes that comprise an object. The graphic portion is rooted at graphicRoot, which is set as the root node of the graphic render area. The graphicPHANToM node which represents the graphic cursor, although part of the scene graph for the Object class, is managed by the Vibe class.

The haptic portion of the Object class is rooted at hapticObject. This portion is generated by the Object class and used by the Vibe class.
Since there is one instance of the Object class for each embedded object, there are multiple instances of the Object scene graphs, one for each object on the page.

4.4.3 Vibe

The Vibe class keeps track of the overall haptic scene and the nodes that are shared by all objects on the page. The Vibe haptic scene graph is rooted at hapticRoot, which is set as the root node of the haptic scene. The hapticObject nodes which are added to hapticRoot are not generated by the Vibe class but are part of the Object class. There is one hapticObject node for each object in the scene. The graphicPHANToM node representing the graphic cursor is created by the Vibe class, and shared by all instances of the Object class.
4.5 Events

When an event is generated by a dynamic sub-object, the appropriate event callback function is called. This function will first extract the relevant event information from its arguments, then pass the information to the object to which the sub-object belongs. There are three pieces of information that are passed to the object:

- **Sub-object ID**: This is an integer which identifies the sub-object that generated the event. The value is specified in the data file in the IDSoInfo node, and defaults to 0 if not specified. The IDs do not have to be unique; sub-objects which are to be treated the same for the purposes of event processing may use the same ID.

- **Event number**: This is an integer which specifies what event occurred. For a button, the event number is 0 for a release and 1 for a press. For dials and sliders, the event number is the current notch number.

- **URL pointer**: This is a pointer to a string that defines the URL registered with this sub-object. If no URL is registered, or if the event generated does not launch an URL, the pointer equals NULL.

When the object receives the event information, it determines what actions to perform, then executes the appropriate ones. There are two types of actions which can be triggered by a dynamic event. One is the launching of a URL, and the other is a Java action defined by an applet on the page. When an event is generated by an embedded object, either one, both, or none of the two action types may be triggered, depending on the object parameters.
4.5.1 URL

If the URL pointer passed to the object is non-NULL, then the sub-object has a registered URL. Also, the event generated is one which causes a registered URL to be launched: for buttons, this happens when the button is pressed, and for dials and sliders, this happens when the current notch is different from the initial notch. When the object receives the pointer, it reads the string representing the URL and launches it in the browser window.

4.5.2 Java

If the APPLET parameter was specified in the object tag, then the object notifies the named Java applet whenever an event occurs. The applet can then process the event any way it wants. The object passes event information to the applet indirectly, via the VibePlugin Java object that is associated with the object. The object does this by first obtaining a pointer to the associated Java object, then calling the Java object’s dynamicEvent method. This method takes as arguments an integer identifying the dynamic type (button, dial, or slider), the sub-object ID, and the event number. It then calls either the buttonEvent, dialEvent, or sliderEvent method in the Java applet, passing to it the sub-object ID and the event number.

4.6 Plug-in Shutdown

Whenever the browser leaves a page that uses the Vibe plug-in, the plug-in performs its shutdown process and is unloaded from memory. First, each object is
destroyed by a call to NPP_Destroy. This removes the object from the Vibe haptic scene via the function removeObject, which removes the associated haptic object from the scene. The destructor of the object is then called, which deletes the graphic object, haptic object, and render area from memory, and frees up any other memory that the object used.

The Java class that is associated with the plugin, VibePlugin, is unloaded at this time. Finally, the Vibe instance is destroyed. The Vibe destructor first stops the haptic servo loop, then deletes the haptic root, the haptic scene, and the graphic PHANToM cursor, de-allocating any memory that was used.
5 Application

This section contains a detailed user manual for the Vibe plug-in. The information presented here allows a user to setup and use the plug-in, create documents that contain embedded haptic objects, and write Java applets that communicate with haptic objects.

5.1 Installation and Setup

The Vibe installation and setup procedure involves obtaining the necessary files, placing them in the appropriate directory, and setting an environment variable.

5.1.1 Netscape and Open Inventor

The Vibe plug-in is a dynamic-link library for the Windows NT 4.0 platform. It requires Netscape Navigator version 3.0 and Open Inventor runtime libraries version 2.1.2. Both should be installed on the system before installing and setting up the plug-in. Information about Netscape Navigator is available at http://home.netscape.com, and information about Open Inventor is available at http://www.tgs.com.

5.1.2 Vibe Files

There are three files in the Vibe plug-in package. The first is NPVibe.d11, which is the plug-in dynamic-link library. The other two files are Java class files used by the plug-in, VibePlugin.class and HapticApplet.class. Once these files have been downloaded, all of these files should be placed in the following directory:
[Netscape directory]\Program\plugins\n
For example, if Netscape was installed in C:\Netscape, then the three files should be placed in the C:\Netscape\Program\plugins\ directory.

To make sure that the plug-in is installed, launch the Netscape browser and select “Help | About Plug-ins” from the menu. The Vibe plug-in should be listed, with the MIME type “application/x-haptic-object” and the .hap suffix.

5.1.3 PHANToM

To feel the Vibe objects, the user must have a PHANToM haptic device. Without it, the objects can only be viewed. Information about the PHANToM is available at http://www.sensible.com. For haptics to be enabled in the plug-in, the PHANToM should be properly configured, and there should be an initialization file for the device. The path to that file must be set in the PHANTOM environment variable so that it can be found by the plug-in. To set the variable, the following line should be placed in AUTOEXEC.BAT:

```
SET PHANTOM = [path of initialization file]
```

For example, the line might be “SET PHANTOM = C:\phantom.ini”. The system must be restarted for this modification to take effect.

5.2 EMBED Tag Specifications

A haptic object is embedded in an HTML document by inserting an EMBED tag. The tag has the following syntax:
Parameters supported by the Vibe plug-in are listed here.

5.2.1 TYPE

Syntax: TYPE = "application/x-haptic-object"

This parameter is required by the plug-in, and should appear exactly as shown.
The parameter informs the browser that the embedded object is of the haptic object
MIME type, so that the Vibe plug-in is used for the object.

5.2.2 FILE

Syntax: FILE = "filename"

This parameter specifies a data file to use for the object, and may be included
multiple times in the tag. Each file that is used by the object represents a sub-object. All
sub-objects are superimposed on each other and use the same local coordinates.

5.2.3 WIDTH

Syntax: WIDTH = value

This parameter is required and specifies the width, in pixels, of the window area
occupied by the embedded object. Although it affects the graphic size of the object, it has
no effect on the haptic size of the object.

5.2.4 HEIGHT

Syntax: HEIGHT = value
This parameter is required and specifies the height, in pixels, of the window area occupied by the embedded object. Although it affects the graphic size of the object, it has no effect on the haptic size of the object.

5.2.5 CAMERADIST

Syntax: CAMERADIST = value

This parameter specifies the distance, in millimeters, from the camera to the origin of the object's local coordinate space. The camera is placed on the z-axis facing the negative z direction. By default, the camera is placed at a distance that allows it to view the entire object. The distance specified by this parameter affects the graphic size of the object, and should be used in conjunction with the WIDTH and HEIGHT values to achieve a graphic size that corresponds with the haptic size of the object.

5.2.6 BGCOLOR

Syntax: BGCOLOR = #rrggbb

This parameter specifies the background color of the object's window area. The value is a "#" symbol followed by the red, green and blue components of the color in hexadecimal notation. The default value is #000000. To make the background color of the object window area match exactly with the background color of the document, the value specified here should be the same as the one used for the BGCOLOR parameter in the BODY tag of the document.
5.2.7 CURSORCOLOR

Syntax: CURSORCOLOR = #rrggb

This parameter specifies the color of the graphic PHANToM cursor. The value is a “#” symbol followed by the red, green, and blue components of the color in hexadecimal notation. The default value is #CCCCCC. This parameter is generally used to make the cursor more easily visible against a specific background or object color.

5.2.8 APPLET

Syntax: APPLET = "name"

This parameter specifies the name of the Java applet that the object is to send events notifications to. The name specified should be the same as the one specified in the NAME parameter of the applet’s APPLET tag. The applet must also implement the HapticApplet interface.

5.2.9 PLUGINSPAGE

Syntax: PLUGINSPAGE = "URL"

This parameter specifies a URL where information on downloading and installing the Vibe plug-in can be found. When the browser encounters a page containing embedded haptic objects and the Vibe plug-in is not installed, the assisted installation process will launch the URL.
5.3 Data File Specifications

Data files used by the Vibe plug-in should have the .hap suffix, and begin with the following line:

    #Inventor V2.1 ascii

The basic format of the file is the standard Open Inventor format, with the exception that only a subset of the nodes are supported. The following is a list of the supported nodes:

- **Grouping nodes**: SoSeparator, SoGroup
- **Transformation nodes**: SoTransform, SoTranslation, SoRotation, SoRotationXYZ, SoScale
- **Coordinate node**: SoCoordinate3
- **Shape nodes**: SoCone, SoCube, SoCylinder, SoSphere, SoIndexedFaceSet, SoIndexedTriangleStripSet
- All nodes that specify a purely visual element, or do not have any effect on haptic attributes, are also supported.

The plug-in will display a warning dialog if it encounters a node that it cannot handle. In addition to the nodes listed above, the plug-in also uses a set of special nodes that specify attributes for haptic objects. These nodes are described below.

5.3.1 DYNAMICFRICTION

Syntax: DEF DYNAMICFRICTION Info { string "value" }
This node specifies a value for the dynamic friction coefficient surface property. The default value is 0.2. When encountered in a scene graph, this node affects the surface property state in the same way that transformation nodes affect the transformation state.

5.3.2 STATICFRICITION

Syntax: DEF STATICFRICITION Info { string "value" }

This node specifies a value for the static friction coefficient surface property. The default value is 0.35. When encountered in a scene graph, this node affects the surface property state in the same way that transformation nodes affect the transformation state.

5.3.3 DAMPING

Syntax: DEF DAMPING Info { string "value" }

This node specifies a value for the damping coefficient surface property. The default value is 0.0. When encountered in a scene graph, this node affects the surface property state in the same way that transformation nodes affect the transformation state.

5.3.4 SPRING

Syntax: DEF SPRING Info { string "value" }

This node specifies a value for the spring constant surface property. The default value is 0.4. When encountered in a scene graph, this node affects the surface property state in the same way that transformation nodes affect the transformation state.
5.3.5 DYNAMIC

Syntax: DEF DYNAMIC Info { string "type" }  

This node specifies the dynamic behavior type of the sub-object contained in the file. Types permitted are BUTTON, DIAL, and SLIDER. If this node is included multiple times, the first one will be used. If this node is not included in the file, the sub-object is assumed to be static.

5.3.6 TRANSFORM

Syntax: DEF TRANSFORM Transform {  
    translation x y z  
    rotation x y z angle  }

This node specifies transformation values which determine the position of a dynamic sub-object, and is not used for static sub-objects. The node has the same syntax as a standard SoTransform node, but only the translation and rotation fields are used. This node must be placed as a top-level node in the file; if it is not, or if there is no such node in the file, then the dynamic sub-object is placed at the origin of the object’s local coordinates.

5.3.7 ID

Syntax: DEF ID Info { string "value" }  

This node specifies an integer value which identifies the sub-object, and is used only by dynamic sub-objects for event processing. The ID value for a sub-object does not
have to be unique; different sub-objects can use the same ID if they are to be treated the
same during event processing. The default value is 0.

5.3.8 URL

Syntax: DEF URL Info { string "URL" }

This node specifies a string representing the URL that is to be launched when the
sub-object in the file generates a certain event. For buttons, the URL is launched when the
button is pressed. For dials and sliders, the URL is launched when the current notch is
different from the initial notch. If this node is not included, no URL is launched.

5.3.9 MASS

Syntax: DEF MASS Info { string "value" }

This node specifies a value, in kilograms, for the mass of a dynamic sub-object,
and is ignored by static sub-objects. The default value is 0.5 for a button, 0.0018 for a
dial, and 0.5 for a slider. If this node is included multiple times in a file, the first one is
used.

5.3.10 DYNAMICDAMPING

Syntax: DEF DYNAMICDAMPING Info { string "value" }

This node specifies a value for the dynamic damping coefficient of a dynamic sub-
object, and is ignored by static sub-objects. The default value is 0.5 for a button, 0.02 for
a dial, and 2.0 for a slider. If this node is included multiple times in a file, the first one is
used.
5.3.11 K

Syntax: DEF K Info { string "value" }

This node specifies a value for the dynamic spring constant of a dynamic sub-object, and is ignored by static sub-objects. The default value is 0.00007 for a button, 0.06 for a dial, and 0.004 for a slider. If this node is included multiple times in a file, the first one is used.

5.3.12 DEADBAND

Syntax: DEF DEADBAND Info { string "value" }

This node specifies a value, in millimeters, for the deadband of a dynamic button sub-object, and is ignored by all other sub-objects. The default value is 5.0. If this node is included multiple times in a file, the first one is used.

5.3.13 RESTORINGFORCE

Syntax: DEF RESTORINGFORCE Info { string "value" }

This node specifies a value, in Newtons, for the restoring force of a dynamic button sub-object, and is ignored by all other sub-objects. The default value is 0.1. If this node is included multiple times in a file, the first one is used.

5.3.14 THROWDISTANCE

Syntax: DEF THROWDISTANCE Info { string "value" }
This node specifies a value, in millimeters, for the throw distance of a dynamic button sub-object, and is ignored by all other sub-objects. The default value is 20.0. If this node is included multiple times in a file, the first one is used.

5.3.15 NUMBERNOTCHES

Syntax: DEF NUMBERNOTCHES Info { string "value" }

This node specifies an integer value for the number of notches of a dynamic dial or slider sub-object, and is ignored by all other sub-objects. The default value is 6 for a dial and 4 for a slider. If this node is included multiple times in a file, the first one is used.

5.3.16 INITIALNOTCH

Syntax: DEF INITIALNOTCH Info { string "value" }

This node specifies an integer value for the initial notch of a dynamic dial or slider sub-object, and is ignored by all other sub-objects. The default value is 0. If this node is included multiple times in a file, the first one is used.

5.3.17 DISTANCE

Syntax: DEF DISTANCE Info { string "value" }

This node specifies a value, in millimeters, for the traversal distance of a dynamic slider sub-object, and is ignored by all other sub-objects. The default value is 100.0. If this node is included multiple times in a file, the first one is used.
5.4 Java Applet Specifications

A Java applet must implement the HapticApplet interface in order to receive event notifications from haptic objects. Below is a template for the applet code:

```java
import java.applet.Applet;
import HapticApplet;

public class TemplateApplet extends Applet
    implements HapticApplet {
    // implement applet methods here

    public void buttonEvent(int ID, int event) {
        // implement button event handling here
        //    ID = sub-object identifier
        //    event = 0 if button is released
        //    1 if button is pressed
    }

    public void dialEvent(int ID, int event) {
        // implement dial event handling here
        //    ID = sub-object identifier
        //    event = current notch
    }

    public void sliderEvent(int ID, int event) {
        // implement slider event handling here
        //    ID = sub-object identifier
        //    event = current notch
    }
}
```

When compiling the applet code, the path to the HapticApplet.class file must be included in the CLASSPATH environment variable. In the HTML document, the applet must have a name specified in the NAME parameter of the APPLET tag, and that name must match the name specified by the haptic object.
6 Future Work

Future work in this area involves adding features to the Vibe plug-in. Possible extensions include an improved browser interface, a wider range of event types, and elements controlled by Java, all of which enhance user interaction with the plug-in.

6.1 Browser Interface

One improvement which should be implemented is support for scrolling. This will allow the virtual PHANToM space to be scrolled around along with the HTML document, so that all embedded objects on the page can be accessed no matter where they are located. Another feature is support for frames. With frame support, objects in one frame can coexist in the virtual space with objects in another frame, and objects in one frame can communicate with applets with another frame.

Other interface enhancements include pop-up menus that allow a user to change the haptic environment variables on the fly, such as the location of the haptic "wall" or the pixels per millimeter conversion factor. Also, the plug-in should display the target URL in the status bar when the haptic cursor touches a dynamic URL element (this is similar to the browser displaying a URL when the mouse is moved over a link). Additional measures should be implemented to enable users without a haptic device to access all the features on the page.
6.2 Event Types

In the current implementation, only button, dial, and slider events are sent to Java applets. Other event types could be implemented, such as the cursor coming in contact with a sub-object. This could be implemented by sending the ID of the sub-object to the applet whenever it is touched. Also, the applet should be able to request the current position of the haptic cursor at any time, or ask that the cursor position is sent to it during every graphic update. Other pieces of information that the applet can request include the magnitude and direction of the current force, and any graphic or haptic properties of an object. This would give applets more types of information to respond to.

6.3 Java-controlled Elements

With the current Vibe plug-in, Java applets are passive, only receiving events and information from haptic objects and responding to them. The plug-in implementation can be extended so that Java applets can control the properties of embedded objects. This involves implementing methods in the VibePlugin class that are capable of making modifications such as translating, rotating, or scaling the associated object, or changing material properties, both visual and haptic. A Java applet that wishes to modify an object can call the appropriate method in the VibePlugin object. This feature would open up a lot of new possibilities for object-to-applet interaction.
7 Conclusion

The Vibe plug-in allows touchable, moveable objects to be placed alongside text, images, audio elements, and Java applets in HTML. The component of touch adds a new dimension to Web publishing, allowing users to download virtual objects from the Internet and interact with them. Embedded haptic objects can make a document more informative and educational, because they are capable of conveying information to the user through the sense of touch.
Appendix: Source Code

- npshell.cpp
- Vibe.h
- Vibe.cpp
- Object.h
- Object.cpp
- VibePlugin.java
- HapticApplet.java
// npshell.cpp

#include "npapi.h"
#include "VibePlugin.h"

#include "Object.h"
#include "Vibe.h"

Vibe *vibe;

NPError NPP_Initialize(void)
{
    LoadLibrary("inv212d.dll");
    LoadLibrary("invw212d.dll");

    vibe = new Vibe;

    return NPERR_NO_ERROR;
}

jref NPP_GetJavaClass(void)
{
    return VibePlugin::_use(NPN_GetJavaEnv());
}

void NPP_Shutdown(void)
{
    VibePlugin::_unuse(NPN_GetJavaEnv());
    delete vibe;
}

NPError NPP_New(NPMIMETYPE pluginType, NPP instance, uint16 mode, int16 argc, char* argn[], char* argv[], NPSavedData* saved)
{
    if (instance == NULL)
        return NPERR_INVALID_INSTANCE_ERROR;

    Object *object = new Object(instance);

    if (object == NULL)
        return NPERR_OUT_OF_MEMORY_ERROR;

    instance->pdata = object;

    for (int i = 0; i < argc; i++)
        object->setParameter(instance, argn[i], argv[i]);

    vibe->addObject(object);

    return NPERR_NO_ERROR;
}

NPError NPP_Destroy(NPP instance, NPSavedData** save)
{
    if (instance == NULL)
        return NPERR_INVALID_INSTANCE_ERROR;

    return NPERR_NO_ERROR;
}
Object *object = (Object *) instance->pdata;

vibe->removeObject(object);

delete object;
instance->pdata = NULL;

return NPERR_NO_ERROR;

}  

NPErrr NPP_SetWindow(NPP instance, NPWindow* window)
{
    if (instance == NULL)
        return NPERR_INVALID_INSTANCE_ERROR;

    if (window == NULL || window->window == NULL)
        return NPERR_GENERIC_ERROR;

    if (vibe->getWindow() == NULL)
        vibe->setWindow(GetParent((HWND) window->window));

    Object *object = (Object *) instance->pdata;

    if (object->getWindow() == NULL)
        object->createWindow((HWND) window->window, window->width, window->height);

   object->setWindowPosition(window->x, window->y, vibe->getWindow());

    return NPERR_NO_ERROR;

}

\( \text{NPErrr}\) NPP_NewStream(NPP instance, NPMIMETYPE type, NPStream *stream, NPBOOL seekable, uint16 *stype)
{
    if (instance == NULL)
        return NPERR_INVALID_INSTANCE_ERROR;

    *stype = NP_ASFFILE;

    return NPERR_NO_ERROR;
}

\text{void}\ NPP_StreamAsFile(NPP instance, NPStream *stream, const char* fname)
{
    if (instance == NULL) return;

    Object *object = (Object *) instance->pdata;

    if (fname != NULL)
        object->processFile(fname);
}

int32 NPP_WriteReady(NPP instance, NPStream *stream)
{
    int32 STREAMBUFSIZE = OX0FFFFFF;
}
return STREAMBUFSIZE;
}

int32 NPP_Write(NPP instance, NPStream *stream, int32 offset, int32 len, void *buffer)
{
    return len;
}

NPErr NPP_DestroyStream(NPP instance, NPStream *stream, NPErr reason)
{
    return NPErr_NO_ERROR;
}

void NPP_Print(NPP instance, NPPrint* printInfo)
{
}

void NPP_URLNotify(NPP instance, const char* url, NPreason reason, void* notifyData)
{
}

int16 NPP_HandleEvent(NPP instance, void* event)
{
    return 0;
}
// Vibe.h

#ifndef _VIBE
#define _VIBE

#include "Object.h"

#include <gstScene.h>
#include <gstSeparator.h>

#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/sensors/SoTimerSensor.h>

class Vibe
{
    public:

        Vibe();
    ~Vibe();

        HWND getWindow() { return window; }
    void setWindow(HWND windowHandle);

        void addObject(Object *object);
    void removeObject(Object *object);

        void handleError(int errorCode, char *errorMessage);

    protected:

        HWND window;
    SbBool hapticsEnabled;

        gstScene *scene;
    gstSeparator *hapticRoot;

        SoSeparator *graphicPHANToM;

        SoTimerSensor *sceneUpdateSensor;

    void initializePHANToM();
};

void positionUpdateCallback(gstTransform *PHANToMPointer, void *, void *positionPointer);
void errorCallback(int errorCode, char *errorMessage, void *vibePointer);
void sceneUpdateCallback(void *scenePointer, SoSensor *);

#endif
// Vibe.cpp

#include "Vibe.h"

#include <gstCube.h>
#include <gstPHANToM_SCP.h>

#include <Inventor/nodes/SoSphere.h>
#include <Inventor/Win/SoWin.h>

Vibe::Vibe()
{
    SoWin::init();

    window = NULL;

    scene = new gstScene;

    hapticRoot = new gstSeparator;
    scene->setRoot(hapticRoot);

    graphicPHANToM = new SoSeparator;
    graphicPHANToM->ref();
}

Vibe::~Vibe()
{
    if (hapticsEnabled)
    {
        scene->stopServoLoop();
        sceneUpdateSensor->unschedule();
        delete sceneUpdateSensor;
    }

    delete hapticRoot;
    delete scene;

    graphicPHANToM->unref();
}

void Vibe::setWindow(HWND windowHandle)
{
    window = windowHandle;
    initializePHANToM();

    gstCube *wall = new gstCube;
    wall->setWidth(1000.0);
    wall->setHeight(1000.0);
    wall->setLength(1000.0);
    wall->setTranslate(0.0, 0.0, -500.0);
    hapticRoot->addChild(wall);
}

void Vibe::initializePHANToM()
{
    setErrorCallback(errorCallback, this);
    hapticsEnabled = TRUE;
SoWin::createSimpleErrorDialog(window, "PHANToM Initialization",
   "Make sure PHANToM is centered, then click OK.");

char *initializationFile = getenv("PHANTOM");
gstPHANToM *PHANToM = new gstPHANToM(initializationFile);

if (!hapticsEnabled) return;

SoTranslation *position = new SoTranslation;
graphicPHANToM->addChild(position);

SoSphere *point = new SoSphere;
point->radius.setValue(2.0);
graphicPHANToM->addChild(point);

gstSeparator *hapticPHANToM = new gstSeparator;
hapticRoot->addChild(hapticPHANToM);

gstPHANToM_SC_PSCP = new gstPHANToM_SC_PSCP;
SCP->setGraphicsCallback(positionUpdateCallback, position);
hapticPHANToM->addChild(SCP);

PHANToM->setSCPNode(SCP);
hapticPHANToM->addChild(PHANToM);

sceneUpdateSensor = new SoTimerSensor(sceneUpdateCallback, scene);
sceneUpdateSensor->setInterval(1.0/10.0);
sceneUpdateSensor->schedule();
scene->startServoLoop();
}

void Vibe::addObject(Object *object) {
   object->addGraphicPHANToM(graphicPHANToM);
   hapticRoot->addChild(object->getHapticObject());
}

void Vibe::removeObject(Object *object) {
   hapticRoot->removeChild(object->getHapticObject());
}

void Vibe::handleError(int errorCode, char *errorMessage) {
   if (errorCode == GST_PHANTOM_INIT_ERROR) {
      SoWin::createSimpleErrorDialog(window, "PHANToM Error",
         "Initialization failed.\nPHANToM will not be enabled.");
      hapticsEnabled = FALSE;
   } else
   SoWin::createSimpleErrorDialog(window, "Error",
      errorMessage);
}
void positionUpdateCallback(gstTransform *PHANToMPointer, void *,
void *positionPointer)
{
    gstPHANToM *PHANToM = (gstPHANToM *) PHANToMPointer;
    SoTranslation *position = (SoTranslation *) positionPointer;

    position->translation.setValue(PHANToM->getPosition().x(),
                                   PHANToM->getPosition().y(), PHANToM->getPosition().z());
}

void errorCallback(int errorCode, char *errorMessage, void *vibePointer)
{
    Vibe *vibe = (Vibe *) vibePointer;
    vibe->handleError(errorCode, errorMessage);
}

void sceneUpdateCallback(void *scenePointer, SoSensor *)
{
    gstScene *scene = (gstScene *) scenePointer;
    scene->updateGraphics();
    scene->updateEvents();
}
// Object.h

#ifndef _OBJECT
#define _OBJECT

#include "npapi.h"
#include "VibePlugin.h"

#include <gstBasic.h>
#include <gstSeparator.h>

#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoShape.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/Win/SoWinRenderArea.h>

typedef struct __Properties {
    double dynamicFriction;
    double staticFriction;
    double damping;
    double spring;
} Properties;

class Object
{
    public:

    Object(NPP pluginInstance);
    ~Object();

    void setParameter(NPP instance, char *name, char *value);
    void processFile(const char *filename);

    void addGraphicPHANTOm(SoSeparator *graphicPHANTOm);
    gstSeparator *getHapticObject();

    void createWindow(HWND window, uint32 width, uint32 height);
    HWND getWindow();
    void setWindowPosition(uint32 x, uint32 y, HWND parentWindow);

    void dynamicEvent(int type, int ID, int event, SbString *URL);
    enum { BUTTON = 0, DIAL = 1, SLIDER = 2 }; 

    protected:

    NPP instance;

    SbBool appletSpecified;
    SbBool cameraDistanceSpecified;
    SbColor backgroundColor;

    uint32 windowX, windowY;
    float halfWidth, halfHeight;
SoSeparator *graphicRoot;
SoWinRenderArea *renderArea;
SoPerspectiveCamera *camera;
SoTranslation *graphicObjectTranslation;
SoTranslation *worldSpaceTranslation;
SoSeparator *graphicObject;
gstSeparator *hapticObject;
SbString unsupportedElements;

void setAppName(char *name);
void setColor(SbColor *color, char *colorString);

SoSeparator *readFile(const char *filename);
SoNode *getNode(const char *nodeName, SoType type, SoNode *root);
void addDynamic(int type, SoSeparator *graphicSceneGraph,
    const char *filename);

void createButton(gstSeparator *hapticDynamic,
    SoSeparator *graphicSceneGraph,
    SoTransform *buttonTransform);
void createDial(gstSeparator *hapticDynamic,
    SoSeparator *graphicSceneGraph,
    SoTransform *dialTransform);
void createSlider(gstSeparator *hapticDynamic,
    SoSeparator *graphicSceneGraph,
    SoTransform *sliderTransform);

gstSeparator *graphicsToHaptics(SoSeparator *graphicSceneGraph);
gstSeparator *processGroup(SoGroup *group, Properties *properties,
    gstTransformMatrix *currentTransform,
    gstPoint *currentNonuniformScale,
    SoCoordinate3 *currentCoordinates);

void setSurfaceProperties(gstShape *shape, Properties *properties);
void processTransformation(SoTransformation *transformation,
    gstTransformMatrix *transform, gstPoint *nonuniformScale);
void copyMatrix(gstTransformMatrix sourceMatrix,
    gstTransformMatrix *destinationMatrix);
void processShape(SoShape *shape, gstSeparator *hapticParent,
    Properties *properties, gstTransformMatrix *transform,
    gstPoint *nonuniformScale, SoCoordinate3 *coordinates);

typedef struct _DynamicData {
    Object *object;
    int ID;
    SbString *URL;
} DynamicData;

void dynamicGraphicCallback(gstTransform *dynamicPointer, void *,
    void *transformPointer);
void buttonEventCallback(gstTransform *button, void *eventData,
void *buttonData);
void dialEventListener(gstTransform *dial, void *eventData,
            void *dialData);
void sliderEventListener(gstTransform *slider, void *eventData,
            void *sliderData);
#endif
// Object.cpp

#include "Object.h"

#include <gstButton.h>
#include <gstCone.h>
#include <gstCube.h>
#include <gstCylinder.h>
#include <gstDial.h>
#include <gstPolyMesh.h>
#include <gstSlider.h>
#include <gstSphere.h>

#include <Inventor/actions/SoSearchAction.h>
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoCube.h>
#include <Inventor/nodes/SoCylinder.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoIndexedFaceSet.h>
#include <Inventor/nodes/SoIndexedTriangleStripSet.h>
#include <Inventor/nodes/SoInfo.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoRotation.h>
#include <Inventor/nodes/SoRotationXYZ.h>
#include <Inventor/nodes/SoScale.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/Win/SoWin.h>

Object::Object(NPP pluginInstance)
{
    instance = pluginInstance;

    appletSpecified = FALSE;
cameraDistanceSpecified = FALSE;

    unsupportedElements.makeEmpty();

    windowX = 0;
    windowY = 0;
    halfWidth = 0.0;
    halfHeight = 0.0;

    renderArea = NULL;

    graphicRoot = new SoSeparator();
    graphicRoot->ref();

    graphicObjectTranslation = new SoTranslation;
graphicRoot->addChild(graphicObjectTranslation);

    camera = new SoPerspectiveCamera;
camera->nearDistance.setValue(0.001);
graphicRoot->addChild(camera);

    graphicRoot->addChild(new SoDirectionalLight);

    graphicObject = new SoSeparator;
graphicRoot->addChild(graphicObject);

worldSpaceTranslation = new SoTranslation;
graphicRoot->addChild(worldSpaceTranslation);

hapticObject = new gstSeparator;

Object::~Object()
{
    graphicRoot->unref();
    delete renderArea;
    delete hapticObject;
}

void Object::setParameter(NPP instance, char *name, char *value)
{
    if (!strcmp(name, "bgcolor"))
        setColor(&backgroundColor, value);
    else if (!strcmp(name, "cursorcolor"))
    {
        SbColor cursorColor;
        setColor(&cursorColor, value);
        SoMaterial *cursorMaterial = new SoMaterial;
        cursorMaterial->diffuseColor.setValue(cursorColor);
        graphicRoot->addChild(cursorMaterial);
    }
    else if (!strcmp(name, "applet"))
        setAppName(value);
    else if (!strcmp(name, "cameradist"))
    {
        camera->position.setValue(0.0, 0.0, atof(value));
        camera->farDistance.setValue(atof(value) + 0.001);
        cameraDistanceSpecified = TRUE;
    }
    else if (!strcmp(name, "file"))
        NPN_GetURL(instance, value, NULL);
}

void Object::setAppName(char *name)
{
    java_lang_String *appletName =
        JRI_NewStringUTF(NPN_GetJavaEnv(), name, strlen(name));
    VibePlugin *plugin = (VibePlugin *) NPN_GetJavaPeek(instance);
    plugin->setAppName(NPN_GetJavaEnv(), appletName);
    appletSpecified = TRUE;
}

void Object::setColor(SbColor *color, char *colorString)
if (strlen(colorString) != 7 || colorString[0] != '\#')
    return;

    char newColorString[9] = { '0', 'x', colorString[5],
        colorString[6], colorString[3], colorString[4],
        colorString[1], colorString[2], '\0' );

    char *temp;
    uint32_t colorValue = strtoul(newColorString, &temp, 16);

    float transparency;
    color->setPackedValue(colorValue, transparency);
}

void Object::createWindow(HWND window, uint32 width, uint32 height)
{
    halfWidth = width/2.0;
    halfHeight = height/2.0;

    camera->aspectRatio.setValue((float) width / (float) height);

    renderArea = new SoWinRenderArea(window);
    renderArea->setBackgroundColor(backgroundColor);
    renderArea->setSceneGraph(graphicRoot);

    SoWin::show(window);
}

HWND Object::getWindow()
{
    if (renderArea != NULL)
        return renderArea->getWidget();
    else return NULL;
}

void Object::setWindowPosition(uint32 x, uint32 y, HWND parentWindow)
{
    if (windowX != x || windowY != y)
    {
        windowX = x;
        windowY = y;

        RECT rect;
        GetClientRect(parentWindow, &rect);

        double objectX = (x + halfWidth - (rect.right / 2.0)) * 0.33;
        double objectY = ((rect.bottom / 2.0) - y - halfHeight) * 0.33;

        graphicObjectTranslation->translation.setValue(objectX, objectY, 0);
        worldSpaceTranslation->translation.setValue(-objectX,
            -objectY, 0);
        hapticObject->setTranslate GSTPoint(objectX, objectY, 0));
    }
SoSeparator *Object::readFile(const char *filename)
{
    SoInput input;
    if (!input.openFile(filename))
    {
        SbString errorMessage = "File \\
; errorMessage += filename;
        errorMessage += \\
; cannot be opened."
        SoWin::createSimpleErrorDialog(getWindow(), "File Error", (char *) errorMessage.getString());
        return NULL;
    }

    SoSeparator *graphicSceneGraph = SoDB::readAll(&input);
    if (graphicSceneGraph == NULL)
    {
        SbString errorMessage = "File \\
; errorMessage += filename;
        errorMessage += \\
; invalid format."
        SoWin::createSimpleErrorDialog(getWindow(), "File Error", (char *) errorMessage.getString());
    }

    return graphicSceneGraph;
}

SoNode *Object::getNode(const char *nodeName, SoType type, SoNode *root)
{
    if (root == NULL) return NULL;

    SoSearchAction searchAction;

    searchAction.setInterest(SoSearchAction::FIRST);
    searchAction.setName(nodeName);

    searchAction.apply(root);

    if (searchAction.getPath() == NULL)
        return NULL;

    SoNode *node = searchAction.getPath()->getTail();

    if (node->isOfType(type))
        return node;

    return NULL;
}

void Object::addGraphicPHANToM(SoSeparator *graphicPHANToM)
{
    graphicRoot->addChild(graphicPHANToM);
}

gstSeparator *Object::getHapticObject()
{
}
return hapticObject;
}

void Object::processFile(const char *filename)
{
    SoSeparator *graphicSceneGraph = readFile(filename);
    if (graphicSceneGraph == NULL) return;

    graphicSceneGraph->ref();
    SoNode *dynamicNode = getNode("DYNAMIC", SoInfo::getClassTypeId(),
                                 graphicSceneGraph);
    if (dynamicNode != NULL)
    {
        SbString dynamicType((SoInfo *)
                             dynamicNode)->getString().getString();

        if (dynamicType == "BUTTON")
            addDynamic(BUTTON, graphicSceneGraph, filename);
        else if (dynamicType == "DIAL")
            addDynamic(DIAL, graphicSceneGraph, filename);
        else if (dynamicType == "SLIDER")
            addDynamic(SLIDER, graphicSceneGraph, filename);
        else
        {
            SbString errorMessage = "File ";
            errorMessage += filename;
            errorMessage += ":\ninvalid DYNAMIC type.";
            SoWin::createSimpleErrorDialog(getWindow(),
                                           "File Error",
                                           (char *) errorMessage.getString());
        }
    }
    else
    {
        graphicObject->addChild(graphicSceneGraph);
        gstSeparator *hapticSceneGraph =
            graphicsToHaptics(graphicSceneGraph);
        hapticObject->addChild(hapticSceneGraph);
    }
    graphicSceneGraph->unref();

    if (!cameraDistanceSpecified)
    {
        float tempX, tempY, cameraDistance;
        camera->viewAll(graphicObject, SbViewportRegion());
        camera->position.getValue().getValue(tempX, tempY,
                                             cameraDistance);
        camera->position.setValue(0.0, 0.0, cameraDistance);
        camera->farDistance.setValue(cameraDistance + 0.001);
    }
}

void Object::addDynamic(int type, SoSeparator *graphicSceneGraph,
                        const char *filename)
{
    SoSeparator *graphicDynamic = new SoSeparator;
graphicObject->addChild(graphicDynamic);

gstSeparator *hapticDynamic = new gstSeparator;
hapticObject->addChild(hapticDynamic);

SoNode *transformNode = getNode("TRANSFORM",
   SoTransform::getClassTypeId(), graphicSceneGraph);
if (transformNode != NULL)
{
   SoTransform *graphicDynamicTransform = (SoTransform *)
      transformNode;
   if (graphicSceneGraph->findChild(graphicDynamicTransform)
      == -1)
   {
      SbString errorMessage = "File \\
      ":nTRANSFORM for dynamic
      element must be at the top level.";
      SoWin::createSimpleErrorDialog(getWindow(),
         "File Warning",
         (char *) errorMessage.getString());
   }
   else
   {
      graphicDynamic->addChild(graphicDynamicTransform);
      graphicSceneGraph->
         removeChild(graphicDynamicTransform);

      SbVec3f translation, rotationAxis;
      float rotationAngle;
      translation =
         graphicDynamicTransform->translation.getValue();
      graphicDynamicTransform->rotation.getValue().
         getValue(rotationAxis, rotationAngle);
      hapticDynamic->setTranslate(gstPoint{translation[0],
         translation[1], translation[2]});
      hapticDynamic->setRotate(gstVector{rotationAxis[0],
         rotationAxis[1], rotationAxis[2]},
         rotationAngle);
   }
}

SoTransform *dynamicObjectTransform = new SoTransform;
graphicDynamic->addChild(dynamicObjectTransform);
graphicDynamic->addChild(graphicSceneGraph);

if (type == BUTTON)
   createButton(hapticDynamic, graphicSceneGraph,
      dynamicObjectTransform);
else if (type == DIAL)
   createDial(hapticDynamic, graphicSceneGraph,
      dynamicObjectTransform);
else if (type == SLIDER)
   createSlider(hapticDynamic, graphicSceneGraph,
      dynamicObjectTransform);
}
void Object::createButton(gstSeparator *hapticDynamic,
    SoSeparator *graphicSceneGraph, SoTransform *buttonTransform)
{
    gstButton *dynamicButton = new gstButton;
    hapticDynamic->addChild(dynamicButton);

    gstSeparator *hapticSceneGraph =
        graphicsToHaptics(graphicSceneGraph);
    dynamicButton->addChild(hapticSceneGraph);

    SoNode *tempNode;

    tempNode = getNode("MASS", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicButton->setMass(atof(((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("DYNAMICDAMPING", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicButton->setDamping(atof(((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("K", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicButton->setK(atof(((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("DEADBAND", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicButton->setDeadband(atof(((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("RESTORINGFORCE", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicButton->setRestoringForce(atof(((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("THROWDISTANCE", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicButton->setThrowDist(atof(((SoInfo *)
            tempNode)->string.getValue().getString()));

    DynamicData *buttonData = (DynamicData *)
        malloc(sizeof(DynamicData));
    buttonData->object = this;

    tempNode = getNode("ID", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        buttonData->ID = atoi(((SoInfo *)
            tempNode)->string.getValue().getString());
else buttonData->ID = 0;

tempNode = getNode("URL", SoInfo::getClassTypeId(),
    graphicSceneGraph);
if (tempNode != NULL)
    buttonData->URL = new SbString(((SoInfo *)
        tempNode)->string.getValue());
else buttonData->URL = NULL;

dynamicButton->setEventCallback(buttonEventCallback, buttonData);
dynamicButton->setGraphicsCallback(dynamicGraphicCallback,
    buttonTransform);
}

void Object::createDial(gstSeparator *hapticDynamic,
    SoSeparator *graphicSceneGraph, SoTransform *dialTransform)
{
    gstDial *dynamicDial = new gstDial;
    hapticDynamic->addChild(dynamicDial);

    gstSeparator *hapticSceneGraph =
        graphicsToHaptics(graphicSceneGraph);
    dynamicDial->addChild(hapticSceneGraph);

    SoNode *tempNode;

    tempNode = getNode("MASS", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicDial->setMass(atof(((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("DYNAMICDAMPING", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicDial->setDamping(atof(((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("K", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicDial->setK(atof(((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("NUMBERNOTCHES", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicDial->setNumberNotches(atoi(((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("INITIALNOTCH", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicDial->setInitialNotch(atoi(((SoInfo *)
            tempNode)->string.getValue().getString()));

    DynamicData *dialData = (DynamicData *)
malloc(sizeof(DynamicData));
dialData->object = this;

tempNode = getNode("ID", SoInfo::getClassTypeId(),
    graphicSceneGraph);
if (tempNode != NULL)
    dialData->ID = atoi((SoInfo *)
        tempNode)->string.getValue().getString());
else dialData->ID = 0;

tempNode = getNode("URL", SoInfo::getClassTypeId(),
    graphicSceneGraph);
if (tempNode != NULL)
    dialData->URL = new SbString((SoInfo *)
        tempNode)->string.getValue());
else dialData->URL = NULL;

dynamicDial->setEventCallback(dialEventCallback, dialData);
dynamicDial->setGraphicsCallback(dynamicGraphicCallback,
    dialTransform);
}

void Object::createSlider(gstSeparator *hapticDynamic,
    SoSeparator *graphicSceneGraph, SoTransform *sliderTransform)
{
    gstSlider *dynamicSlider = new gstSlider;
hapticDynamic->addChild(dynamicSlider);

    hapticSceneGraph =
        graphicsToHaptics(graphicSceneGraph);
    dynamicSlider->addChild(hapticSceneGraph);

    SoNode *tempNode;

    tempNode = getNode("MASS", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicSlider->setMass(afof((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("DYNAMICDAMPING", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicSlider->setDamping(afof((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("K", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicSlider->setK(afof((SoInfo *)
            tempNode)->string.getValue().getString()));

    tempNode = getNode("DISTANCE", SoInfo::getClassTypeId(),
        graphicSceneGraph);
    if (tempNode != NULL)
        dynamicSlider->setDistance(afof((SoInfo *)
            tempNode)->string.getValue().getString()));

    75
tempNode = getNode("NUMBERNOTCHES", SoInfo::getClassTypeId(),
    graphicSceneGraph);
if (tempNode != NULL)
    dynamicSlider->setNumberNotches(atoi(((SoInfo *)
        tempNode)->string.getValue().getString()));

tempNode = getNode("INITIALNOTCH", SoInfo::getClassTypeId(),
    graphicSceneGraph);
if (tempNode != NULL)
    dynamicSlider->setInitialNotch(atoi(((SoInfo *)
        tempNode)->string.getValue().getString()));

DynamicData *sliderData = (DynamicData *)
    malloc(sizeof(DynamicData));
sliderData->object = this;

tempNode = getNode("ID", SoInfo::getClassTypeId(),
    graphicSceneGraph);
if (tempNode != NULL)
    sliderData->ID = atoi(((SoInfo *)
        tempNode)->string.getValue().getString());
else sliderData->ID = 0;

tempNode = getNode("URL", SoInfo::getClassTypeId(),
    graphicSceneGraph);
if (tempNode != NULL)
    sliderData->URL = new SbString(((SoInfo *)
        tempNode)->string.getValue());
else sliderData->URL = NULL;

dynamicSlider->setEventCallback(sliderEventCallback, sliderData);
dynamicSlider->setGraphicsCallback(dynamicGraphicCallback,
    sliderTransform);
}

void Object::dynamicEvent(int type, int ID, int event, SbString *URL)
{
    if (appletSpecified)
    {
        VibePlugin *plugin = (VibePlugin *)
            NPN_GetJavaPeer(instance);
        plugin->dynamicEvent(NPN_GetJavaEnv(), (jint) type,
            (jint) ID, (jint) event);
    }
    if (URL != NULL)
        NPN_GetURL(instance, URL->getString(), "_self");
}

gstSeparator *Object::graphicsToHaptics(SeSeparator *graphicSceneGraph)
{
    Properties *properties = (Properties *)
        malloc(sizeof(Properties));
    properties->dynamicFriction = 0.2;
    properties->staticFriction = 0.35;
    properties->damping = 0.0;

    76
properties->spring = 0.4;

gstSeparator *hapticSceneGraph = processGroup(graphicSceneGraph,  
    properties, new gstTransformMatrix,  
    new gpoint(1.0, 1.0, 1.0), new SoCoordinate3);

if (unsupportedElements != "")
{
    SoWin::createSimpleErrorDialog(getWindow(),  
    "Unsupported Nodes",  
    (char *) unsupportedElements.getString());  
    unsupportedElements.makeEmpty();
}

return hapticSceneGraph;
}

gstSeparator *Object::processGroup(SoGroup *group,  
    Properties *currentProperties,  
    gstTransformMatrix *currentTransform,  
    gpoint *currentNonuniformScale,  
    SoCoordinate3 *currentCoordinates)
{
    gstSeparator *returnNode = new gstSeparator;
    Properties *properties;
    gstTransformMatrix *transform;
    gpoint *nonuniformScale;
    SoCoordinate3 *coordinates = currentCoordinates;

    if (group->getTypeId() == SoSeparator::getClassTypeId())
    {
        properties = (Properties *) malloc(sizeof(Properties));  
        properties->dynamicFriction =  
            currentProperties->dynamicFriction;  
        properties->staticFriction =  
            currentProperties->staticFriction;  
        properties->damping = currentProperties->damping;  
        properties->spring = currentProperties->spring;

        transform = new gstTransformMatrix;  
        copyMatrix(*currentTransform, transform);

        nonuniformScale = new gpoint(currentNonuniformScale->x(),  
            currentNonuniformScale->y(),  
            currentNonuniformScale->z());
    }
    else if (group->getTypeId() == SoGroup::getClassTypeId())
    {
        properties = currentProperties;
        transform = currentTransform;
        nonuniformScale = currentNonuniformScale;
    }
    else
    {
unsupportedElements += "\n";
unsupportedElements +=
    group->getTypeId().getName().getString();
return returnNode;
}

SoNode *child;
SbName name;

int numberChildren = group->getNumChildren();

for (int i = 0; i < numberChildren; i++)
{
    child = group->getChild(i);

    if (child->isOfType(SoGroup::getClassTypeId()))
        returnNode->addChild(processGroup((SoGroup *) child, properties, transform, nonuniformScale, coordinates));

    else if (child->isOfType(SoInfo::getClassTypeId()))
    {
        name = child->getName();

        if (name == "DYNAMICFRICTION")
            properties->dynamicFriction = atof(((SoInfo *) child)->string.getValue().getString());

        else if (name == "STATICFRICTION")
            properties->staticFriction = atof(((SoInfo *) child)->string.getValue().getString());

        else if (name == "DAMPING")
            properties->damping = atof(((SoInfo *) child)->string.getValue().getString());

        else if (name == "SPRING")
            properties->spring = atof(((SoInfo *) child)->string.getValue().getString());
    }

    else if (child->isOfType(SoShape::getClassTypeId()))
        processShape((SoShape *) child, returnNode, properties, transform, nonuniformScale, coordinates);

    else if (child->isOfType(SoTransformation::getClassTypeId()))
        processTransformation((SoTransformation *) child, transform, nonuniformScale);

    else if (child->isOfType(SoCoordinate3::getClassTypeId()))
    {
        coordinates = (SoCoordinate3 *) child;
        if (!group->isOfType(SoSeparator::getClassTypeId()))
            currentCoordinates = (SoCoordinate3 *) child;
    }

78
else if (child->isOfTyp e(SoCamera::getClassTypeId()))
    group->removeChild(child);
}

return (returnNode);
}

void Object::processShape(SoShape *shape, gstSeparator *hapticParent, Properties *properties, gstTransformMatrix *transform, gstPoint *nonuniformScale, SoCoordinate3 *coordinates)
{
    if (shape->isOfTyp e(SoCone::getClassName()))
    {
        SoCone *graphicCone = (SoCone *) shape;
        gstCone *hapticCone = new gstCone;
        hapticCone->setTransformMatrix(*transform);
        hapticCone->setScale(nonuniformScale->x(),
                            nonuniformScale->y(),
                            nonuniformScale->z(),
                            shape->getValue()/2.0);
        setSurfaceProperties(hapticCone, properties);
        hapticParent->addChild(hapticCone);
    }

    else if (shape->isOfTyp e(SoCube::getClassName()))
    {
        SoCube *graphicCube = (SoCube *) shape;
        gstCube *hapticCube = new gstCube;
        hapticCube->setTransformMatrix(*transform);
        hapticCube->setScale(nonuniformScale->x(),
                            nonuniformScale->y(),
                            nonuniformScale->z(),
                            shape->getValue()/2.0);
        setSurfaceProperties(hapticCube, properties);
        hapticParent->addChild(hapticCube);
    }

    else if (shape->isOfTyp e(SoCylinder::getClassName()))
    {
        SoCylinder *graphicCylinder = (SoCylinder *) shape;
        gstCylinder *hapticCylinder = new gstCylinder;
        hapticCylinder->setTransformMatrix(*transform);
        hapticCylinder->setScale(nonuniformScale->x(),
                                  nonuniformScale->y(),
                                  nonuniformScale->z(),
                                  shape->getValue()/2.0);
        setSurfaceProperties(hapticCylinder, properties);
        hapticParent->addChild(hapticCylinder);
    }
}
else if (shape->isOfAType(SoSphere::getClassTypeId()))
{
    SoSphere *graphicSphere = (SoSphere *) shape;
    gstSphere *hapticSphere = new gstSphere;
    hapticSphere->setTransformMatrix(*transform);
    hapticSphere->setScale(nonuniformScale->x(),
      nonuniformScale->y(),
      nonuniformScale->z(),
      graphicSphere->radius.getValue(),
      graphicSphere->radius.getValue(),
      graphicSphere->radius.getValue());
    setSurfaceProperties(hapticSphere, properties);
    hapticParent->addChild(hapticSphere);
}

else if (shape->isOfAType(SoIndexedFaceSet::getClassTypeId()) ||
  shape->isOfAType(SoIndexedTriangleStripSet::getClassTypeId()))
{
    SoIndexedFaceSet *graphicPolyMesh =
      (SoIndexedFaceSet *) shape;
    double array3Doubles[3]; int array3Ints[3]; int i;
    array3Doubles[0] = 0.0; array3Ints[0] = 0;

    int numV = coordinates->point.getNum();
    double (*v)[3];
    v = (double (*)(3))[3] malloc(numV * sizeof(array3Doubles));
    for (i = 0; i < numV; i++)
    {
        v[i][0] = coordinates->point[i][0];
        v[i][1] = coordinates->point[i][1];
        v[i][2] = coordinates->point[i][2];
    }

    int numCoordIndex = graphicPolyMesh->coordIndex.getNum();
    int numP = 0;
    int tempSides = 0;
    for (i = 0; i < numCoordIndex; i++)
    {
        if (graphicPolyMesh->coordIndex[i] == -1)
        {
            numP += tempSides - 2;
            tempSides = 0;
        }
        else tempSides++;
    }
    if (graphicPolyMesh->coordIndex[i-1] != -1)
    {
        numP += tempSides - 2;
    }

    int (*p)[3];
    p = (int (*)(3)) malloc(numP * sizeof(array3Ints));
    int tempCoordIndex = 0, firstCoord = -1, secondCoord = -1,
    newPolygon = TRUE;
    for (i = 0; i < numP; i++)
    {
if (newPolygon)
{
    firstCoord =
    graphicPolyMesh->coordIndex[tempCoordIndex++];
    secondCoord =
    graphicPolyMesh->coordIndex[tempCoordIndex++];
    newPolygon = FALSE;
}

p[i][0] = firstCoord;
p[i][1] = secondCoord;
p[i][2] =
    graphicPolyMesh->coordIndex[tempCoordIndex++];
secondCoord = p[i][2];

if (graphicPolyMesh->coordIndex[tempCoordIndex] == -1)
{
    tempCoordIndex++;
    newPolygon = TRUE;
}

}

gstPolyMesh *hapticPolyMesh =
    new gstPolyMesh(numV, v, numP, p);
    hapticPolyMesh->setTransformMatrix(*transform);
    hapticPolyMesh->setScale(nonuniformScale->x(),
        nonuniformScale->y(), nonuniformScale->z());
    setSurfaceProperties(hapticPolyMesh, properties);
    hapticParent->addChild(hapticPolyMesh);
}

else
{
    unsupportedElements += "\n";
    unsupportedElements +=
        shape->getTybeId().getName().getString();
}

}
scaleOrientationAxisZ = 1.0;
float centerX = 0.0, centerY = 0.0, centerZ = 0.0;
SbVec3f translation, rotationAxis, scaleFactor,
scaleOrientationAxis, center;
if (transformation->isOfType(SoTransform::getClassTypeId()))
{
    translation = (((SoTransform *) transformation)->translation.getValue());
    translation.getValue(translationX, translationY, translationZ);

    (((SoTransform *)
    transformation)->rotation.getValue()).getValue(rotationAxis,
    rotationAngle);
    rotationAxis.getValue(rotationAxisX, rotationAxisY,
    rotationAxisZ);

    scaleFactor = (((SoTransform *) transformation)->scaleFactor.getValue());
    scaleFactor.getValue(scaleFactorX, scaleFactorY,
    scaleFactorZ);

    (((SoTransform *)
    transformation)->scaleOrientation.getValue()).
    getValue(scaleOrientationAxis, scaleOrientationAngle);
    scaleOrientationAxis.getValue(scaleOrientationAxisX,
    scaleOrientationAxisY, scaleOrientationAxisZ);

    center = (((SoTransform *)
    transformation)->center.getValue());
    center.getValue(centerX, centerY, centerZ);
}
else if (transformation->isOfType(SoTranslation::getClassTypeId()))
{
    translation = (((SoTranslation *)
    transformation)->translation.getValue());
    translation.getValue(translationX, translationY, translationZ);
}
else if (transformation->isOfType(SoRotation::getClassTypeId()))
{
    (((SoRotation *)
    transformation)->rotation.getValue()).getValue(rotationAxis,
    rotationAngle);
    rotationAxis.getValue(rotationAxisX, rotationAxisY,
    rotationAxisZ);
}
else if
(transformation->isOfType(SoRotationXYZ::getClassTypeId()))
{
    (((SoRotationXYZ *

transformation->getRotation().getValue(rotationAxis, 
    rotationAngle);
    rotationAxis.getValue(rotationAxisX, rotationAxisY, 
    rotationAxisZ);
}

else if (transformation->isOfTpe(SoScale::getClassName()))
{
    scaleFactor = 
        ((SoScale *) transformation)->scaleFactor.getValue();
    scaleFactor.getValue(scaleFactorX, scaleFactorY, 
        scaleFactorZ);
}

else
{
    unsupportedElements += "\n";
    unsupportedElements += 
        transformation->getTypeId().getName().getString();
}

gstTransformMatrix tempMatrix;

if (!(translationX == 0.0 && translateY == 0.0 
    && translationZ == 0.0))
{
    gstTransformMatrix translationMatrix;
    translationMatrix.setTranslate(gstPoint(translationX, 
        translationY, translationZ));
    mulM(tempMatrix, translationMatrix, *transform);
    copyMatrix(tempMatrix, transform);
}

if (rotationAngle!= 0.0)
{
    gstTransformMatrix rotationMatrix;
    rotationMatrix.setRotate(gstVector(rotationAxisX, 
        rotationAxisY, rotationAxisZ), 
        rotationAngle);
    mulM(tempMatrix, rotationMatrix, *transform);
    copyMatrix(tempMatrix, transform);
}

if (!(scaleFactorX == 1.0 && scaleFactorY == 1.0 
    && scaleFactorZ == 1.0))
{
    if (scaleFactorX == scaleFactorY && scaleFactorY == 
        scaleFactorZ)
    {
        gstTransformMatrix scaleMatrix;
        scaleMatrix.setScale(gstPoint(scaleFactorX, 
            scaleFactorY, scaleFactorZ));
        mulM(tempMatrix, scaleMatrix, *transform);
        copyMatrix(tempMatrix, transform);
    }
    else

```c

} nonuniformScale->setx(nonuniformScale->x() * scaleFactorX);
nonuniformScale->sety(nonuniformScale->y() * scaleFactorY);
nonuniformScale->setz(nonuniformScale->z() * scaleFactorZ);

}

if (! (scaleOrientationAxisX == 0.0 && scaleOrientationAxisY == 0.0 && scaleOrientationAxisZ == 1.0))
UnsupportedElements += "\nscaleOrientation in SoTransform";

if (! (centerX == 0.0 && centerY == 0.0 && centerZ == 0.0))
UnsupportedElements += "\ncenter in SoTransform";

void Object::copyMatrix(gstTransformMatrix sourceMatrix,
gstTransformMatrix *destinationMatrix)
{
    destinationMatrix->set(0, 0, sourceMatrix.get(0, 0));
    destinationMatrix->set(0, 1, sourceMatrix.get(0, 1));
    destinationMatrix->set(0, 2, sourceMatrix.get(0, 2));
    destinationMatrix->set(0, 3, sourceMatrix.get(0, 3));
    destinationMatrix->set(1, 0, sourceMatrix.get(1, 0));
    destinationMatrix->set(1, 1, sourceMatrix.get(1, 1));
    destinationMatrix->set(1, 2, sourceMatrix.get(1, 2));
    destinationMatrix->set(1, 3, sourceMatrix.get(1, 3));
    destinationMatrix->set(2, 0, sourceMatrix.get(2, 0));
    destinationMatrix->set(2, 1, sourceMatrix.get(2, 1));
    destinationMatrix->set(2, 2, sourceMatrix.get(2, 2));
    destinationMatrix->set(2, 3, sourceMatrix.get(2, 3));
    destinationMatrix->set(3, 0, sourceMatrix.get(3, 0));
    destinationMatrix->set(3, 1, sourceMatrix.get(3, 1));
    destinationMatrix->set(3, 2, sourceMatrix.get(3, 2));
    destinationMatrix->set(3, 3, sourceMatrix.get(3, 3));
}

void dynamicGraphicCallback(gstTransform *, void *callbackData, 
void *transformPointer)
{
    gstDynamicGraphicsCBData *dynamicData = 
        (gstDynamicGraphicsCBData *) callbackData;
    gstTransformMatrix matrix = dynamicData->transform;
    SoTransform *transform = (SoTransform *) transformPointer;
    transform->setMatrix(SbMatrix(matrix.get(0, 0), matrix.get(0, 1),
                               matrix.get(0, 2), matrix.get(0, 3), matrix.get(1, 0),
                               matrix.get(1, 1), matrix.get(1, 2), matrix.get(1, 3),
                               matrix.get(2, 0), matrix.get(2, 1), matrix.get(2, 2),
                               matrix.get(2, 3), matrix.get(3, 0), matrix.get(3, 1),
                               matrix.get(3, 2), matrix.get(3, 3)));
}

void buttonEventCallback(gstTransform *button, void *eventData,
void *buttonData)

```
{
    int ID = ((DynamicData *) buttonData)->ID;
    int event = ((gstEvent *) eventData)->id;

    SbString *URL = NULL;
    if (event == gstButton::PRESSED)
        URL = ((DynamicData *) buttonData)->URL;

    ((DynamicData *) buttonData)->object->dynamicEvent(Object::BUTTON, ID, event, URL);
}

diaEventCallback(gstTransform *dial, void *eventData, void *dialData)
{
    int ID = ((DynamicData *) dialData)->ID;
    int event = ((gstEvent *) eventData)->id;

    SbString *URL = NULL;
    if (event != ((gstDial *) dial)->getInitialNotch())
        URL = ((DynamicData *) dialData)->URL;

    ((DynamicData *) dialData)->object->dynamicEvent(Object::DIAL, ID, event, URL);
}

sliderEventCallback(gstTransform *slider, void *eventData, void *sliderData)
{
    int ID = ((DynamicData *) sliderData)->ID;
    int event = ((gstEvent *) eventData)->id;

    SbString *URL = NULL;
    if (event != ((gstSlider *) slider)->getInitialNotch())
        URL = ((DynamicData *) sliderData)->URL;

    ((DynamicData *) sliderData)->object->dynamicEvent(Object::SLIDER, ID, event, URL);
}
// VibePlugin.java

import netscape.javascript.JSONObject;
import netscape.plugin.Plugin;
import HapticApplet;

public class VibePlugin extends Plugin {
    static final int BUTTON = 0;
    static final int DIAL = 1;
    static final int SLIDER = 2;

    JSONObject document;
    String appletName;

    public void init()
    {
        JSONObject window = (JSONObject) this.getWindow();
        document = (JSONObject) window.getMember("document");
    }

    public void setAppletName(String name)
    {
        appletName = name;
    }

    public void dynamicEvent(int type, int ID, int event)
    {
        HapticApplet applet =
            (HapticApplet) document.getMember(appletName);

        if (type == BUTTON)
            applet.buttonEvent(ID, event);

        else if (type == DIAL)
            applet.dialEvent(ID, event);

        else if (type == SLIDER)
            applet.sliderEvent(ID, event);
    }
}
/ HapticApplet.java

public interface HapticApplet
{
    public void buttonEvent(int ID, int event);
    public void dialEvent(int ID, int event);
    public void sliderEvent(int ID, int event);
}
Acknowledgments

Special thanks to: Thesis supervisor J. Kenneth Salisbury and the team at SensAble Technologies, Inc.

This work was supported in part by the Naval Air Warfare Center Training Systems Division (NAWCTSD) Contract N61339-96-K-0002.

References


