Title: VRML as a superset of HTML -- An Approach To Consolidation

Bruce Donald Campbell
Rensselaer Polytechnic Institute, camp5716@rpi.edu

Abstract—*This paper looks at the current methods of coexisting HTML and VRML on a Web page. Then, it suggests a way to improve the relationship between the two by considering VRML a superset of HTML. Only a few additional fields need to be added to VRML 2's Appearance node to handle HTML documents in 3-dimensional space. Current methods reviewed include Netscape's plug-in technology, HTML's frame technology, and the VRML Consortium's VRML 2 Java API technology. The advantages and disadvantages of each approach are critiqued. A framework for extending the VRML 2 Appearance node is proposed for extending VRML to handle HTML documents more naturally. Then, example uses of the improved Appearance node are presented and contrasted to the existing methodologies already available as reviewed within this paper. The author concludes with a suggestion that we begin to develop cyberspace along these lines.*

1. Introduction

The HyperText Markup Language standardizes the creation of two dimensional (2D) Web pages out of text, two dimensional images, and simple user controls such as buttons and text boxes. HTML was created without considering any three dimensional (3D) location for the information produced. HTML syntax formats informational objects using short text strings called tags.

On the other hand, the Virtual Reality Modeling Language (VRML) was created specifically to standardize 3D content for the Web. Any object created using VRML has a location in 3D space. A user of a VRML viewer walks into a VRML scene using a mouse or keyboard and navigates as she does in the real world. VRML syntax formats informational objects using short text strings called node identifiers.

VRML includes nodes for managing text strings in three dimensional space. These VRML nodes are very primitive compared with the richness and diversity of the HTML tags created specifically for advanced text formatting and presentation. It does not make sense to reinvent ASCII string formatting in the VRML standard. Instead, I suggest extending the Appearance node of the VRML 2 standard to define how to open an HTML document at a three dimensional location within a VRML scene. This paper reviews the most popular techniques Web authors use to coexist HTML and VRML on a Web site. Then, it suggests a way to incorporate HTML into the VRML standard as a natural extension of VRML's three dimensions. Consolidating HTML and VRML would simplify Web browsers and provide a framework for immersive Web navigation where the computer screen no longer exists.

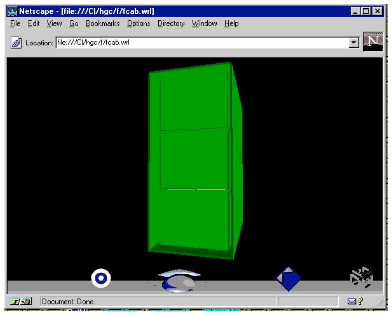
2. Current Methods of Coexisting HTML and VRML within a Web browser

In order to provide a working example, throughout this paper an example VRML file cabinet is developed from which HTML documents are accessed. The simplicity of the VRML cabinet is strictly for learning purposes. A VRML artist could create a very sophisticated file cabinet using VRML. A file cabinet is a quintessential example of a hierarchical, 3D model. Hierarchical models are especially efficient as VRML objects. Each file cabinet has multiple drawers, each of which houses multiple folders, each of which can contain multiple HTML documents. VRML 2 syntax for a simple file cabinet is provided in Listing 1.1 while a picture of the cabinet as rendered by the CosmoPlayer VRML viewer is shown in Figure 1.1. This paper refers to this example often.

```
#VRML V2.0 utf8

DEF in_front Viewpoint {
   position 0 -500 0
   orientation        1 0 0 1.57
   fieldOfView        0.785398
   description        "IN_FRONT"
}
NavigationInfo {
   type "EXAMINE"
}
DEF DRAWER0 Transform {
  children [
    DEF TS1 TouchSensor {}
    DEF DRAWER Transform {
      children [
       DEF BOTTOM Transform {
        children [
          DEF PLANK Shape {
            appearance Appearance {
               material Material {diffuseColor 0 .2 0}
            }
            geometry Box {size 1 1 1}
          }
         ]
         scale 100 200 2
         translation 0 0 0
       },
     DEF BACK Transform {
        children USE PLANK
        scale 100 2 80
        translation 0 100 40
     },
     DEF FRONT Transform {
        children USE PLANK
        scale 100 2 80
        translation 0 -100 40
     },
     DEF RIGHT Transform {
        children USE PLANK
        scale 2 200 80
        translation 50 0 40
     },
     DEF LEFT Transform {
            children USE PLANK
            scale 2 200 80
            translation  -50 0 40
         },
        ]
     }
    DEF FOLDER Transform {
       children [
          DEF TS3 TouchSensor {}
          DEF PAGE0 Transform {
             children [
               DEF PAGE Shape {
                  appearance Appearance {
                    material Material {diffuseColor 1 1 .85}
                  }
                  geometry Box {size 1 1 1}
               }
             ]
             scale 90 2 70
             translation 0 80 32
           },
          DEF PAGE1 Transform {
             children [
                USE PAGE
             ]
             scale 90 2 60
             translation 0 83 27
          }
       ]
     }
   ]
},
DEF DRAWER1 Transform {
  children [
    DEF TS0 TouchSensor {}
    USE DRAWER
  ]
  translation  0 0 -85
},
DEF DRAWER2 Transform {
  children [
    DEF TS2 TouchSensor {}
    USE DRAWER
  ]
  translation  0 0 85
},
Transform {
    children USE DRAWER
    scale  1.15 1.25 2.8
    translation  0 -100 40
    rotation  1 0 0 -1.57
}
```

Listing 1.1 - A VRML File Cabinet


Netscape - [file:///C|/hgc/f/fcab.wrl]
File Edit View Go Bookmarks Options Directory Window Help
Location: file:///C|/hgc/f/fcab.wrl
Document: Done
Figure 1.1 -- A VRML File Cabinet

The VRML 2 file in Listing 1.1 creates the geometry and material of each cabinet, drawer, and folder and places them in three dimensional space. Through the TouchSensor node, the VRML also creates an awareness of the users' pointing device in relation to each object that contains the TouchSensor. For example, for those users using a mouse to interact with the VRML scene, clicking on the first drawer activates the TouchSensor node defined as TS1. The VRML 2 file itself can also contain an event handling routine for each TouchSensor, but through an Application Programming Interface (API) into another programming language, the event can be handled by a separate routine written in programming source code. In the case of this simple file cabinet, an active TouchSensor for a drawer opens the drawer if the drawer is currently closed. Conversely, it will close the drawer if the drawer is open. TouchSensors for folders work the same, simply opening each folder in a location where the contents can easily be viewed and then closing them again. The file cabinet contains HTML documents. The manner in which a selected HTML document is presented to the audience varies based on the method chosen by the VRML author. Currently, an author can use one of three technologies to interface to HTML information from a VRML scene: use a Web browser plug-in architecture, use the HTML frame technology, or use a VRML 2 API to another programming language. The next three sections cover how the technology works for each option Then, the rest of the paper discusses a better option for handling HTML in three dimensions based on considering VRML a superset of HTML.

3. Web Browser Plug-in Technology

The simplest way for a VRML author to present an HTML document from within a VRML world is to require the audience to view the VRML scene using a VRML viewer designed and implemented as a Web browser plug-in. In this case, activation of a non-VRML object takes the form of a hyperlink to a Web based document which the browser will open based on the hyperlinked file's data type [7]. The document is identified to the browser by its URL and the Web browser loads the file upon looking up its MIME data type and using the application registered with the browser for that data type. In effect, the current VRML scene is ignored (though stored in a local cache), and the new file is opened and given full presentation by the browser in the browser window. When a user is done using the latest file, he closes the file to returns to the VRML scene for continued perusal or activation of a different hyperlink.

In the VRML file cabinet example, a hyperlink can be associated with a VRML object by embedding the object's Transform node inside an Anchor node. For example, to activate a link in a file folder, the folder could be embedded as follows (note that in this case, a simple relative URL is passed to the browser for retrieval upon activation of the Anchor):

```
Anchor {
   children [
      url ["example.htm"]
      DEF FOLDER Transform {
         -- see Listing 1.1 for details --
      }
   ]
}
```

When the user interacts with a child object in the Anchor node, the plug-in makes a request to the browser to load the file associated with its url field. In the case of a URL specifying an HTML document, the browser then loads the HTML document in the same browser window, but on top of the VRML scene.

Plug-in technology takes advantage of a Web browser's ability to request and receive files from Web servers around the world. The plug-in need only know how to make the request to the browser and then the browser does the rest. This provides the benefit of minimizing computer resources as the code to perform Web navigation is put in RAM or hard drive memory once by the Web browser. The navigation and load routines can be optimized within the browser and the plug-in can take advantage of the latest improvements on an ongoing basis. Browsers use many sophisticated routines to optimize system resource management. For example, under the guidance of the Web browser, a file can be loaded in the background or integrated into the current browser window.

To create a VRML plug-in, the plug-in developer utilizes two object-oriented classes provided to the plug-in developer by the browser developer. One class sets up communications from the plug-in to the browser and the other the browser to the plug-in. Then, each time the plug-in wants the browser's services, it runs the appropriate routines such as: UserAgent, Version, GetURL, GetURLNotify, and RequestRead. The browser then can communicate with the plug-in through routines such as: Initialize, New, SetWindow, NewStream, WriteReady, Write, Destroy, Shutdown [4].

Disadvantages associated with just using the plug-in technology are the lack of a centralized source of control and the lack of security over accessed files [11]. The browser continually loads files on top of files in the form of a simple stack. There is little control over this behavior from the standpoint of the user.

4. The HTML Frame Technology

The HTML Frame Technology is a way for a Web author to divide up the Web browser window into different subwindows, called frames, and independently present Web-based information in each separate frame. Frame technology has had much support for inclusion in the HTML 3.2 standard [13]. When using frames, Web based files are still loaded via the plug-in architecture, but each frame can run a different plug-in application concurrently [1]. In the file cabinet example, frames can be used to show a VRML file cabinet in one frame and display HTML documents that are opened by the user through interaction with the VRML scene in another frame. In effect, the frames overcome one of the biggest shortcomings of the plug-in methodology: the fact that files are loaded in a simple stack methodology. With frames, each frame has its own stack. This has tremendous potential for mixing graphical information (including 3D worlds) with text-based information in intuitive and innovative ways.

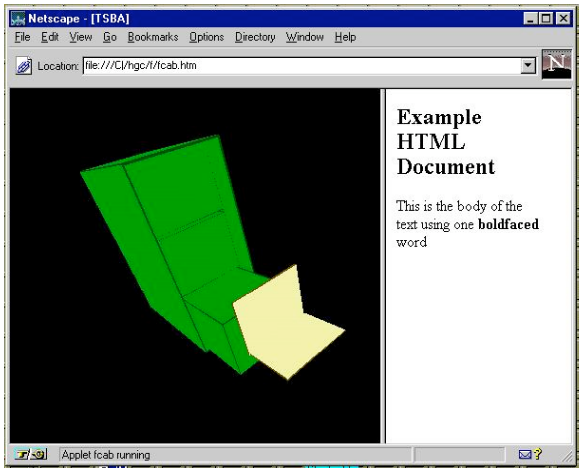
For the VRML file cabinet example using frames, the Anchor node is enhanced to load its url field in a different frame as follows:

```
Anchor {
   children [
      url ["example.htm"]
      parameter ["target=HTML"]
      DEF FOLDER Transform {
         -- see Listing 1.1 for details --
      }
   ]
}
```

The Anchor node has a parameter field that can be set to a string value of "target=HTML" to identify the HTML frame as its presentation destination. The frames are created through use of a HTML Frame Definition Document [3], the source of which is provided in Listing 1.2. Figure 1.2 shows the frame technology example for a VRML file cabinet. The VRML frame is presented on the left side of the browser window and the HTML frame is presented on the right side. Interactions with folders in the left frame change the contents of the right frame.

```
<HTML>
<HEAD>
    <TITLE>File Cabinet Frames Example</TITLE>
    <FRAMESET COLS=*, 290>
            <FRAME SRC=fcab.vrml NAME=VRML
            SCROLLING=NO>
            <FRAME SRC=example.html NAME=HTML>
    </FRAMESET>
</HEAD>
</HTML>
```

Listing 1.2 HTML Frame Definition Document


Netscape - [TSBA]
File Edit View Go Bookmarks Options Directory Window Help
Location: file:///C|/hgc/t/fcab.htm
Applet fcab running

Example HTML Document

This is the body of the text using one **boldfaced** word

Figure 1.2 -- VRML File Cabinet Using Frames

The frame technology provides more flexibility than the vanilla plug-in service. Yet, HTML document presentation is still limited to all or nothing decisions. Either the HTML document fills the frame, or it is not loaded at all [5]. And, the HTML document is always at a perpendicular angle to the reader. Although this is usually the preferred angle for ease of reading in desktop virtual reality applications, it is not flexible for other creative presentation formats nor natural for a fully immersed, virtual reality experience.. Frames continue to evolve as Web browser developers create APIs for programmers to use for inter-frame communications [6].

Of interest has been the critical response to frames by Web traditionalists. Tim Berners-Lee's original design of the Web was based on the page as the atomic unit of information delivery. The URL addressing scheme provides a one to one correspondence of address to page. With frames, a single URL points to a browser window of information that is comprised of multiple pages. Since there is no saved state information associated with a Web bookmark, bookmarks become less exact when pointing to a frame definition document and inexperienced Web navigators are more likely to get lost or encounter unexpected results when using a Web browser to navigate a Web site using frames [14].

5. VRML 2's API Technology

The VRML community realizes that the added ability of a Web author to manipulate VRML objects outside of a VRML file increases acceptance of the VRML standard. VRML is already being widely used to save 3D models in a VRML file format [8]. Any computer process can output VRML syntax in a text based file. Now, with VRML 2, computer processes can continue to alter a VRML file after the file has been presented by the browser. The VRML community has been quick to develop a Java API that can be used by VRML 2 developers to maintain a Java representation of a 3D scene in a Java program, yet pass variable state changes to the VRML plug-in in order to render changes using the VRML viewer technology [9]. The VRML 2 standard identifies a Billboard node that can nest any geometry and texture node sets within its node [12]. Once nested, the Billboard maintains the same angle of view of the geometry to a world visitor as she moves in the scene. In this manner, a Billboard node can assure that an image map is always read perpendicular to the current viewpoint in the scene.

The Java API allows Java variables to be initialized with pointers to VRML nodes. Java is a powerful object-oriented programming language with a comprehensive API development kit [10]. Using Java, a variable within a computer program can be initialized to point to a Billboard node in a VRML file. Within a computer program, the Java variable can change the location of the Billboard in 3D space on the fly, and the VRML viewer will continually update the changes in the 3D world. In this manner, an HTML document's contents can be inserted on a Billboard (using a flat rectangular polygon as a backdrop) and experienced by the user following the behavior coded in the Java program [2]. This method has the flexibility to move the HTML document around in 3D space while maintaining a perpendicular angle to the user. Yet, if the perpendicular attribute is not desired, the Billboard node need not be used. In that case, the HTML document's contents can remain in a fixed 3D-space location without the Billboard and the user can move to view the document at any angle desired. In the VRML file cabinet example, the VRML file can be changed as follows to take advantage of a billboarded HTML document:

```
DEF FOLDER Transform {
  children [
      DEF TS3 TouchSensor {}
      DEF PAGE0 Transform {
        -- see Listing 1.1 for details--
      },
      DEF PAGE1 Transform {
        -- see Listing 1.1 for details --
      },
      DEF HTML_DOC Transform {
        Billboard {
            children [
              Shape {
                appearance Appearance {
                  texture ImageTexture {
                    url "html_bitmap.jpg"
                  }
                }
                geometry IndexedFaceSet {
                  coord Coordinate {
                    point [ -1 -1 0, 1 -1 0, 1 1 0, -1 1 0 ]
                  }
                  coordIndex [ 0, 1, 2, 3 ]
                }
              }
            ]
        }
      }
  ]
}
```

Currently, to use the API technology described above, each HTML document has to be converted to a bitmap representation (called an image map in VRML vernacular) in order to be displayed in a VRML scene. Current bitmap scaling algorithms used in VRML viewers are not the greatest and legibility is lost for the sake of presentation flexibility and rendering speed. The user can not control the HTML document as she could using the plug-in or frame technologies. The document is no longer loaded as an HTML MIME data type in the Web browser, but a 3D-space presentation is gained.

6. VRML 2 Anchor and Billboard nodes

The Anchor and Billboard nodes within VRML provide powerful capabilities. When compared to life in the real world, an Anchor node in a virtual world is like a transporter as envisioned on the Star Trek television series. Using the Anchor node, a VRML author can instantaneously transport the audience to any place in the world with a simple mouse click. Advertisers would love to have the Billboard behavior available in the real world. They would use it often to keep advertising billboards readable for a longer period of time from the consumer's viewpoint while driving down the highway.

Still, HTML document perusal is afforded some handy features by the Web browser. A reader can scroll, turn images on or off, or click on hyperlinks to go elsewhere. These features just are not available to a reader of an HTML-based image map referenced in a VRML file. Why not? I propose to demonstrate how compatible HTML would be as a subset of the VRML standard. Considered as a subset of VRML, HTML documents could still retain their popular scrolling, linking, and imaging features, yet still be included in three dimensional space for a more natural organization and use. The rest of this paper will build that vision and provide examples of how it could work.

7. A Different Syntax

HTML tags and VRML nodes have much in common. They both have a natural hierarchy in which tags or nodes are considered children of other tags or nodes. The syntax of one language is easily converted to the syntax of the other. For example, the following HTML tags:

could easily be translated to look like the following VRML syntax structure:

```
HTML {
  Head {
    title    [Example HTML Document]
  }
  Body {
    children [
      h2     [Example HTML Document]
      Text {
 This is the body of the text using one B{ bold [boldfaced]} word
       }
    ]
  }
}
```

The VRML parser could be extended to handle a whole new set of nodes created specifically to retain current HTML document functionality. I will present 8 new fields that could improve the ability to coexist HTML and VRML in a VRML scene.

New HTML Nodes and Fields
Once the VRML parser was extended to include current HTML functionality (whether parsed using VRML node syntax or HTML tag syntax), the VRML language could be extended to improve existing VRML nodes' ability to intuitively and innovatively present HTML documents within a VRML scene. I will present 8 possible nodes and fields: The HTML node, the HTMLurl node, the HTMLstring field, the HTMLstringType field, the HTMLstringSize field, the HTMLactiveType filed, the HTMLactiveSize field, and the HTMLparameters field.

The HTML Node
The HTML node would be used similarly to a textureImage node to place an HTML document's contents over a geometry nested within the same Shape node. Currently, a textureImage node contains a filename that provides a texture map to be used for the appearance of a 2D polygon oriented in 3D space. The HTML node could be used similarly to the textureImage node to refer to an HTML document that should be placed over a 2D polygon in 3D space. For example, the following VRML syntax:

```
Shape {
    appearance Appearance {
        html HTML {
            Head {
              title    [Example HTML Document]
             }
            Body {
              children [
                 h2      [Example HTML Document]
                 Text {
                        This is the body of the text using one B{ bold
[boldfaced]} word
                  }
               ]
            }
        }
    }
    geometry DEF IFS IndexedFaceSet {
          coord Coordinate {
               point [ -1 -1 0, 1 -1 0, 1 1 0, -1 1 0 ]
          }
          coordIndex [ 0, 1, 2, 3 ]
    }
}
```

puts the example HTML document on a square polygon defined in the geometry field included in the same Shape node.

8. The HTMLurl node

The HTMLurl node is similar to the HTML node except that it points to an HTML node in a separate file. Such organization is consistent throughout VRML in order to keep a structured hierarchy from filling one long and cumbersome, single file. By using the HTMLurl node, the HTML document can be contained in a separate file and then used within many different VRML worlds. The following example shows the use of the HTMLurl node as it would be used to access an HTML node contained in a separate file named example.htm:

```
Shape {
    appearance Appearance {
        html HTMLurl {
              url ["example.htm"]
        }
    }
    geometry DEF IFS IndexedFaceSet {
          coord Coordinate {
               point [ -1 -1 0, 1 -1 0, 1 1 0, -1 1 0 ]
          }
          coordIndex [ 0, 1, 2, 3 ]
    }
}
```

This example is the same as the HTML node example, except with the added benefit of manageability of component objects.

9. The HTMLstring field

If the HTML and HTMLurl nodes were the only extensions added to the VRML specification, an HTML document would have to be scaled down to a very small size when seen from a distance. At that distance, any text would not be legible and the scaling work of the browser would be wasted. By adding another field to the HTML and HTMLurl nodes, the browser could use alternative text once the viewpoint moved far away from the HTML document. The HTMLstring field could fill that purpose. An example of its use is the following:

```
Shape {
    appearance Appearance {
        html HTMLurl {
              url          ["example.htm"]
              HTMLstring    "HTML Example"
        }
    }
    geometry DEF IFS IndexedFaceSet {
          coord Coordinate {
               point [ -1 -1 0, 1 -1 0, 1 1 0, -1 1 0 ]
          }
          coordIndex [ 0, 1, 2, 3 ]
    }
}
```

In this example, at a certain distance from the current viewpoint, the string "HTML example" would be displayed by the viewer instead of the full HTML document. More sophisticated rules could be assigned to the HTML and HTMLurl fields such that a string might be taken from the title of the HTML document by default. In that case, the HTMLstring field would be an opportunity to override the default behavior.

10. The HTMLstringType field

Taking the HTMLstring concept further, VRML could identify multiple ways to present the HTMLstring to the audience. The string might appear in place of the HTML document at large distances, but still be confined to the geometry. Or, perhaps, the string might appear on the browser status bar when a mouse passed over the Shape that contained the HTML or HTMLurl nodes. If that were the case, what would appear on the geometry: the HTML document, the HTMLstring string, or something else?

An HTMLstringType field could enumerate different behaviors for tying the user's actions to the browser's presentation. An example of using the HTMLstringType would be the following:

```
Shape {
    appearance Appearance {
        html HTMLurl {
            url             ["example.htm"]
            HTMLstring      "HTML Example"
            HTMLstringType  "STATUS"
        }
    }
    geometry DEF IFS IndexedFaceSet {
        coord Coordinate {
            point [ -1 -1 0, 1 -1 0, 1 1 0, -1 1 0 ]
        }
        coordIndex [ 0, 1, 2, 3 ]
    }
}
```

The "STATUS" keyword would tell the browser to show the HTMLstring string on the status bar at great distances. Common values for the HTMLstringTypes could be identified as part of the VRML specification.

11. The HTMLstringSize field

One last example of a helpful field for the HTML or HTMLurl nodes would be the HTMLstringSize field. The HTMLstringSize field would identify the size a HTMLstring would appear for those HTMLstringTypes that presented the HTMLstring within the world but outside of the Shape node's geometry. The HTMLstringSize field would control how intrusive the text would be to a visitor. The scale could be represented in many ways, but the example will consider the portion of the horizontal screen as its measure. An example of using HTMLstringSize follows:

Shape { appearance Appearance { html HTMLurl { url ["example.htm"] HTMLstring "HTML Example" HTMLstringType "ABOVE" HTMLstringSize .2 } } geometry DEF IFS IndexedFaceSet { coord Coordinate { point [-1 -1 0, 1 -1 0, 1 1 0, -1 1 0] } coordIndex [0, 1, 2, 3] } }

In this example, the HTMLstring string would cover 20% of the horizontal width of the field of view. Of course, other HTMLstring related fields could be derived to determine string color, orientation, duration, etc. The HTMLstringSize is just an example of one of those fields.

12. The HTMLactiveType field

Once the HTML document was effectively oriented and scaled to fit a 2D polygon geometry in 3D space, other fields would dictate how the user interacted with the HTML document while visiting a VRML world. An HTMLactiveType field would enumerate how the user could activate the HTML document to make it full screen. This would save the user the need to use the VRML viewer controls to get the document perpendicular and within full view. An example of the HTMLactiveType field follows:

```
Shape {
    appearance Appearance {
        html HTMLurl {
            url             ["example.htm"]
            HTMLstring      "HTML Example"
            HTMLactiveType  "TOUCH"
        }
    }
    geometry DEF IFS IndexedFaceSet {
        coord Coordinate {
            point [ -1 -1 0, 1 -1 0, 1 1 0, -1 1 0 ]
        }
        coordIndex [ 0, 1, 2, 3 ]
    }
}
```

In this example, the HTMLactiveType is set to "TOUCH" which would allow the user to make it active as she would any TouchSensor in the world. For desktop VR systems of today, TouchSensors are usually activated through a mouse click. Immersive systems use the touch of a virtual hand. Other example enumerated possibilities for the HTMLactiveType field would be "DRAG", "OVER", and "PROXIMITY". The "DRAG" value would require the user to move the geometry with the mouse, the "OVER" value would make it active upon the mouse passing over the geometry, and the "PROXIMITY" value would activate the HTML full screen when the user was within a certain distance. Note that a similar field could identify how the HTML document would be de-activated and put back within its Shape node's geometry.

13. The HTMLactiveSize field

Although activating an HTML document could make it full screen, an HTMLactiveSize field would give the ability to change that default behavior. The field could take a value that represented the percentage of the screen to be covered by the document when activated. For example in the following:

```
Shape {
    appearance Appearance {
        html HTMLurl {
            url             ["example.htm"]
            HTMLstring      "HTML Example"
            HTMLactiveType  "TOUCH"
            HTMLactiveSize  .4
        }
    }
    geometry DEF IFS IndexedFaceSet {
            coord Coordinate {
                  point [ -1 -1 0, 1 -1 0, 1 1 0, -1 1 0 ]
            }
            coordIndex [ 0, 1, 2, 3 ]
      }
}
```

only 40% of the screen would be covered by an active HTML document.

14. The HTMLparameters field

As a catch all field, an HTMLparameters field could be included in the specification to catch any additional options that might become part of the standard in the future. For example, if "PROXIMITY" were used for the HTMLactiveType, the HTMLparameters filed might define the distance at which the HTML document would be activated. In fact, even the HTMLactiveSize field could be implemented as an HTMLparameters field list item. If that were the case, the example would look like this instead:\

```
Shape {
     appearance Appearance {
           html HTMLurl {
                 url              ["example.htm"]
                 HTMLstring       "HTML Example"
                 HTMLactiveType   "TOUCH"
                 HTMLparameters   ["size=.4"]
           }
      }
      geometry DEF IFS IndexedFaceSet {
            coord Coordinate {
                  point [ -1 -1 0, 1 -1 0, 1 1 0, -1 1 0 ]
            }
            coordIndex [ 0, 1, 2, 3 ]
      }
}
```

15. A Return to the VRML File Cabinet

Having defined all the fields, this section walks through an example of using them in conjunction with the VRML file cabinet example. In Figure 1.3, the bottom drawer has been opened by activation of the drawer's TouchSensor and the folder has been opened by activation of the folder's TouchSensor. Inside the folder sits an HTML document which has been scaled to fit the inside of the folder cover. Adding an HTMLurl field to the Appearance node in the VRML file, the HTML document is as fully functional as an HTML document viewed in a Web browser as an HTML MIME data type. The user has two choices to read the document. The user can change her viewpoint through the controls available in the VRML viewer. Moving forward increases the size of the HTML document and other controls can change the angle of view. Alternatively, the user can activate the HTML document by whatever mode the world author has made available. Activating the HTML document places the HTML document in a convenient place in the world and the user can read, scroll or click on the document to interact with it as if made full screen in a Web browser loading a native HTML data type document. The result is shown in Figure 1.4.

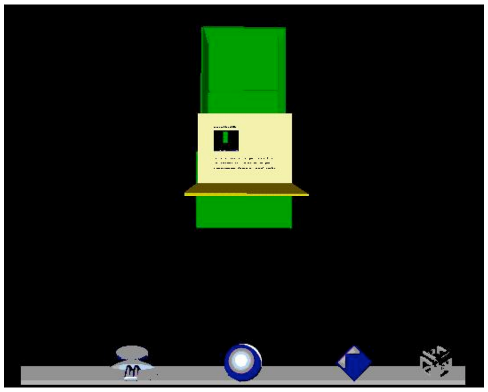
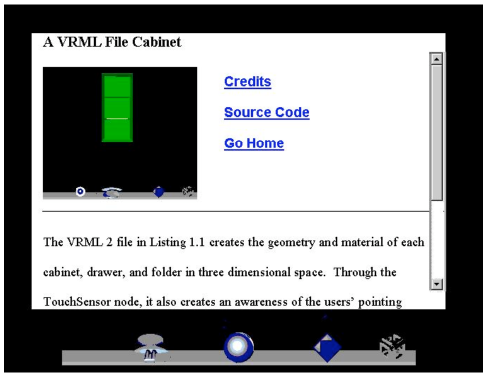


Figure 1.3 VRML Folder Containing an HTML Document


A VRML File Cabinet

Credits

Source Code

Go Home

The VRML 2 file in Listing 1.1 creates the geometry and material of each cabinet, drawer, and folder in three dimensional space. Through the TouchSensor node, it also creates an awareness of the users' pointing

Figure 1.4 An Active HTML Document

16. Consistency in Immersed Virtual Reality

For VRML to stand the test of time, any VRML nodes or fields must be flexible enough to make sense to an immersed world visitor. Immersion relates to the completeness of the sensory experience. In traditional virtual reality systems, a Head Mounted Device (HMD) is worn by the user to totally immerse the user visually. A user immersed in a virtual world has no sense of the barrier provided by a traditional computer screen. HTML document presentation is not as obvious in a fully immersed environment as it is on a computer screen. The HTML and HTMLurl nodes and their related fields presented in this paper will help define HTML document presentation when the screen is no longer present. In a fully immersed virtual experience, HTML documents should be able to be presented as naturally as in the real world. Using a VRML Shape node and an HTML node as proposed in this paper, an HTML document can be placed on top of a virtual desk or table and read naturally while the user continues to look around the rest of the virtual scene comfortably.

17. Conclusion

This paper demonstrates an opportunity to organize two-dimensional HTML documents in three dimensional spaces within the VRML standard itself. With a different text structure, HTML can be considered a subset of the official VRML standard and used appropriately within an HTML or HTMLurl field of the Appearance node. This adds to the flexibility of VRML by helping information gatherers find text documents within a natural 3D space.

It is this author's belief that a 3D organization of text material is more natural than the current HTML search engine methods afforded by Digital's Alta Vista or Yahoo's Yahoo search engines. These search engines produce one dimensional lists of links to HTML documents related to a requested topic. Currently, Web navigators access HTML documents through URLs (Universal Resource Locators) which are not translatable into any 3D location. Web travelers using URLs to locate documents feel more like discontinuous time travelers than geographical travelers. Yet, outside of the Web, human beings travel on foot, by car, or by plane over a continuous landscape. This continuity helps people remember where things are in the universe. It helps everyone navigate daily in a logical manner. A Web traveler visiting a 3D virtual library can walk among the book stacks as in the real world. She can navigate within a specific aisle to find a book of interest. She may find an interesting book she was not aware of but stumbled upon because of its neighbors in three dimensional space.

More research can be done as to which metaphor most easily helps information searchers find the information they desire. Once a flexible and scaleable metaphor is identified, all Web page authors can work to insert their documents into a global, three dimensional space. The tools that are being created to assist in authoring VRML worlds should help Web authors insert HTML documents into an organized cyberspace of information combining both text and three dimensional information should our planet ever get together to create an organized cyberspace.

References

[1] Arpajian, Scott and Mullen, Robert, How To Use HTML 3.2, Ziff Davis Press, Emeryville, CA (1996) [Overview of the HTML 3.2 standard as promoted within the World Wide Web Consortium. Excellent reference for detailing HTML tags for Frames with examples as well as a high level overview of related technologies.]

[2] Cornell, Gary and Horstmann, Cay S., Core Java, SunSoft Press, Mountain View, CA (1997) [Highly technical overview of popular Java APIs available for use by Java developers. Overview of the design specifications of the Java language as well as implications on Object Oriented Programming styles.]

[3] Lemay, Laura, Teach Yourself Web Publishing With HTML 3.2 in 14 Days Professional Reference Edition, Sams.net Publishing, Indianapolis, IN (1996) [The bible of HTML technical references. Excellent overview of Frames, JavaScript and Java implications on Web publishing. Lots of examples of use of HTML syntax.]

[4] Morgan, Mike, Netscape Plug-Ins Developer's Kit, MacMillan Computer Publishing, Indianapolis, IN (1996) [Technical reference on creating plug-in applications within Netscape Navigators' class hierarchy and application programming interface. Overview of the benefits of the plug-in architecture with examples of creating various types of plug-in applications.]

[5] Netscape Corporation, Frames Page (Accessed 4 December 1996) [Overview of using frames within HTML. Description of the process along with the benefits of using frames.]

[6] Netscape Corporation, Java API For Frames Overview Page (Accessed 4 December 1996) [Overview of using the Java API within frames on a Web page. Description of the API functionality along with examples of the messaging structure between Java, JavaScript and the contents of a frame.]

[7] Netscape Corporation, Plug-Ins Home Page (Accessed 4 December 1996) [Overview of the plug-in environment with examples of the communication between plug-ins and Netscape Navigator. Examples of plug-ins available for download.]

[8] San Diego Supercomputer Center, VRML Repository Home Page (Accessed 12 September 1996) [Web site detailing the latest information available on VRML. Links to Web sites incorporating VRML, tutorials about VRML, and opinions on the proper use of VRML.]

[9] Silicon Graphics Inc., Java/VRML API Page (Accessed 2 December 1996) [Technical specification for a standard VRML 2.0 Java API implemented in version 2 of the CosmoPlayer VRML viewer available from SGI. Overview of the function and syntax of the function calls available.]

[10] Sun Microsystems Inc., Java Developers Kit Index Page (Accessed 12 September 1996) [Technical overview of the Java Developers Kit application programming interface (API). Discussion of available classes, methods, and appropriate usage including syntax and parameter lists.]

[11] Turlington, Shannon, Official Netscape Plug-in Book, Netscape Press, Research Triangle Park, NC (1996) [Overview of Netscape Navigator plug-ins including disadvantages of the plug-in architecture and lack of central control point. Examples of good plug-in design concepts and available plug-in products.]

[12] VRML Architecture Group, VRML 2 Final Specification (Accessed 2 December 1996) [The final specification of VRML 2.0 from the organization responsible for its maintenance. Includes design considerations, overview of the purpose of VRML, complete node and field syntax and examples of appropriate use.]

[13] World Wide Web Consortium, HTML Guidelines Page (Accessed 21 December 1996) [The final specification of HTML 2.0 with comment as to the status of HTML 3.2 from the organization responsible for its maintenance. Links to many articles and papers defining and challenging the HTML standard.]

[14] Jakob Nielsen, Why Frames Suck (Most of the Time) (Accessed 6 March 1997)