Chapter 12: Anchors Aweigh!

That Web Thing

By now you've all but forgotten that VRML is a Web technology – that's right, it not only gets served up by the Web, but it can serve it up. We've been busy building our foundation, but it's time to turn eyes outward, to the vast sea of documents that exist on the Web – some hundred million at last count. VRML doesn't stand outside that galaxy of words, images and sounds – rather, it represents perhaps the most useful way to get a handle on all of it.

In order to get at this riot of information, we need to be able to connect our virtual objects to other Web documents. This connection is commonly known as a link, but goes by the technical name of an *anchor*. Anchors are the foundation of the functionality of the Web, connecting documents seamlessly, across computers, across networks, across continents. The anchor creates a whole – a mosaic – from an infinity of separate parts.

How do we attach anchors to VRML objects? With the Anchor node. Here's how it looks:

```
Anchor {                # definition of Anchor node
      children [] # MFNode, mult. values
      description # SFString
      parameter []   # MFString, mult. values
      url []            # MFString, mult. values
      bboxCenter  # SFVec3f
      bboxSize          # SFVec3f
}
```

The url field of the Anchor node contains the URL of the document which the Anchor links to, while the children field contains the set of nodes which are linked by the Anchor to the URL. For our first example, let's create a purple Sphere and link it to the *VRML Repository*, a major resource for VRML on the Web:

```
#VRML V2.0 utf8
# This is the first example on anchors
Anchor {
      children [        # linked objects
            # Make a purple Sphere
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 1
                        }
                  }
                  geometry Sphere { }
            }
      ] # end list of linked objects
url [ "http://www.sdsc.edu/vrml" ] # Repository
```

```
}
```

When we load it into the browser, we'll find we get some unusual behavior when we put the mouse over the object.

If you're using Netscape Communicator and Cosmo, you'll see the URL of the object come up in bright yellow letters.

Or, if you're using Internet Explorer and WorldView, you'll note that the status bar of the browser shows the linked to the URL for the VRML Repository – and that the cursor turns into a pointing finger, just as it does in an HTML browser.

Now click (double-click for WorldView) on the Sphere, and you'll see that the browser loads the page for the Repository:

Notice that the VRML world is gone, replaced by the Web page.

Teleporting and Linking

The first example links a VRML world to a more-or-less normal Web page; but it's also possible to link VRML to any other Web media type, including other VRML worlds. The browser dumps the first world and loads the linked world; this is known as *teleporting* – because you move directly from one world to another without passing through any intervening space – just like the transporters on Star Trek.

In our second example, we'll create a yellow Cylinder that links back to our first example:

```
#VRML V2.0 utf8
# This is the second example on anchors
Anchor {
      children [        # linked objects
            # Make a yellow Cylinder
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 1 0
                        }
                  }
                  geometry Cylinder { radius 0.5 }
            }
      ] # end list of linked objects
url [ "13_1.wrl" ] # Previous example
}
```

When we load this example into a browser, and put our mouse pointer over the Cylinder, we see that it has linked correctly linked to the URL of the previous example.

Clicking on it returns us to the first example.

If we clicked on that green Sphere, we'd return to the Repository home page.

If you want to anchor multiple objects to the same URL, each object needs to be defined within the children field of the Anchor node. Here's the VRML triplet of red Box, green Sphere and blue Cone, each linked to the VRML Repository:

```
      #VRML V2.0 utf8
      # This is the third example on anchors
      Anchor {
            children [        # linked objects
                  # First object; Make a red Box
                  Shape {           #  Create a visible shape
                        appearance Appearance {
                              material Material {
                                    diffuseColor 1 0 0
                              }
                        }
                        geometry Box { }
                  }
                  # Transform with Sphere & Cone
                  Transform {
                        children [
                              # Second object, green Sphere
                  Shape {     #  visible
                        appearance Appearance {
                              material Material {
                                    diffuseColor 0 1 0
                              }
                        }
                        geometry Sphere { }
                  }
                  # Transform with Cone
                  Transform {
                        children [
                              # Third object, blue Cone
                  Shape {     #  visible
                        appearance Appearance {
                              material Material {
                                    diffuseColor 0 0 1
                              }
                        }
                        geometry Cone { }
                  }
            ]
            translation 3 0 0 # away from Sphere
      }
]
translation 3 0 0 # away from Box
                  }
            ] # end list of linked objects
      url [ "http://www.sdsc.edu/vrml/" ] # Repository
      }
```

In this example we used the nested Transform nodes inside the children field of the Anchor node to provide a separation between the objects. As you can see as you put the mouse over all of the objects, they're all linked to the same site.

What if we wanted to link these three objects to different sites, instead of linking them all to the same site? Let's say we wanted to link the Box to the Repository, but we also wanted to link the Sphere to the SGI site and the Cone to the Intervista site. We'd have to turn things inside out a bit, and put the Anchor nodes inside the Transform nodes, but outside the Shape nodes. Here's how that might look:

```
#VRML V2.0 utf8
# This is the fourth example on anchors
Anchor {    # Only one linked object inside of it
      children [        # linked objects
            # First object; Make a red Box
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 0
                        }
                  }
                  geometry Box { }
            }
      ] # end list of linked objects
url [ "http://www.sdsc.edu/vrml/" ] # Repository
}
# Transform with Sphere & Cone
Transform {
      children [
            Anchor {  # only one link, Sphere
                  children [
                        # Second object, green Sphere
            Shape {     #  visible
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 1 0
                        }
                  }
                  geometry Sphere { }
            }
      ]
            url [ "http://vrml.sgi.com/" ] # SGI
      }
      # Transform with Cone
Transform {
      children [
            Anchor {  # only one link, Cone
                  children [
                        # Third object, blue Cone
            Shape {     #  visible
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 0 1
                        }
                  }
                  geometry Cone { }
            }
      ]
                        url [ "http://www.intervista.com/" ]  #I
                  }
```

```
]  # end list of Transform children
translation 3 0 0 # away from Sphere
}
] # end list of Transform children
translation 3 0 0 # away from Box
      }
```

It's as if we turned the previous example inside out. We begin with an Anchor node for the red Box, then we have a Transform node – to move the Sphere and Cone away from the Box. Inside that, we have an Anchor node, and within that, the Sphere. Beneath the Anchor, we have another Transform, in that, we have another Anchor, with the Cone inside of that. That's what we need to do if we want to have separate anchors for every object.

As you move the mouse from object to object, you'll see that the target URL changes.

How and Why

One of the biggest differences between the cyberspace of VRML and the word-space of Web documents is the frightening lack of context in 3D environments. You might see an object in the VRML world, but have no idea why it happens to be there. It may be conveying some information – or it might just be decorative. But on a Web page, the text denotes context; this page, for example, is about VRML anchors; if you found an anchor on this page heading out to some Web document, you'd figure that it had something to do with anchors, and you'd probably be correct. But what if you're in a 3D environment, and you come across a virtual book that's got an anchor in it. You'd figure that the anchor linked to the book's content, but that's all you'd know – which ain't a whole lot.

It would be more useful – by far – if you could attach some descriptive information to the object – some *meta-data*, as it's called – so that you could know the why of the link, and not just the how. The Anchor node's description field allows you to supply a text string of any length along with the URL information; the browser then displays this information to the user. Let's go back to our first example, but add some descriptive information:

```
#VRML V2.0 utf8
# This is the fifth example on anchors
Anchor {
      children [        # linked objects
            # Make a purple Sphere
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 1
                        }
                  }
                  geometry Sphere { }
            }
      ] # end list of linked objects
      description "The VRML Repository" # explains!
url [ "http://www.sdsc.edu/vrml" ] # Repository
}
```

Now we see the descriptive text when we hold the mouse over the Sphere.

This text is applied to all objects within the children field of the Anchor node – so if you do have multiply-linked objects, make sure you choose your words carefully. Here's example four again, with descriptions that make some sense:

```
#VRML V2.0 utf8
# This is the sixth example on anchors
Anchor {    # Only one linked object inside of it
      children [        # linked objects
            # First object; Make a red Box
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 0
                        }
                  }
                  geometry Box { }
            }
      ] # end list of linked objects
      description "The VRML Repository" # explains!
url [ "http://www.sdsc.edu/vrml/" ] # Repository
}
# Transform with Sphere & Cone
Transform {
      children [
            Anchor {  # only one link, Sphere
                  children [
                        # Second object, green Sphere
            Shape {     #  visible
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 1 0
                        }
                  }
                  geometry Sphere { }
            }
      ]
      description "Silicon Graphics' VRML Site"
            url [ "http://vrml.sgi.com/" ] # SGI
      }
      # Transform with Cone
Transform {
      children [
            Anchor {  # only one link, Cone
                  children [
                        # Third object, blue Cone
            Shape {     #  visible
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 0 1
                        }
                  }
                  geometry Cone { }
            }
      ]
```

```
            description "Intervista's Home Page"
                                url [ "http://www.intervista.com/" ]  #IV
                            }
]  # end list of Transform children
translation 3 0 0 # away from Sphere
}
] # end list of Transform children
translation 3 0 0 # away from Box
      }
```

Now each item has its own link and its own descriptive text.

Getting Framed

As the Web has evolved, it's grown from a set of words on a page into an explosion of page layouts, tables, forms and – most lately – frames. Frames provide a way to break a single page into separate components, each with their own document for their content. That means it's now possible to mix media – particularly HTML and VRML – in ways we've never seen before.

Just for a change of pace, here's a bit of HTML which will generate a page with two blank frames:

```
<html>
<head>
<title>Learning VRML Chapter 13: HTML Frames</title>
</head>
<frameset rows="*,150">
<frame src="" name="top">
<frame src="" name="bottom">
</frameset>
```

Here's the blank page with two frames, with target names "top" and "bottom" – you can find it on the CD-ROM as blank_frames.html.

We can load VRML into either of these frames – or even both – by putting adding a URL of a VRML world to one of the frame src= tags. In this case, we'll take example six, and put it in the lower frame:

```
<html>
<head>
<title>Learning VRML Chapter 13: HTML Frames</title>
</head>
<frameset rows="*,150">
<frame src="" name="top">
<frame src="13_6.wrl" name="bottom">
</frameset>
```

Which gives us an blank frame above, and a VRML world below – on the CD-ROM as lower_vrml.html.

The world pops up in the lower frame – though a bit far away; you may need to move a bit closer to it before you can make out the three objects. These objects are linked to the Web; if we click on the blue Cone, we should find ourselves going to Intervista's home page.

That works well enough, but it'd a lot more useful if we could get Intervista's home page to load into the upper portion of the page, in the frame named "top". To do this, we need to pass some information along to the browser to tell it where we want our pages to show up; that information is provided in the parameter field of the Anchor node. The message to send to the browser – along with the link – is target=top. When we send along with the link, it places the document into the correct frame. Here's the VRML, adapted from example six:

```
#VRML V2.0 utf8
# This is the seventh example on anchors
Anchor {    # Only one linked object inside of it
      children [        # linked objects
            # First object; Make a red Box
            Shape {           #  Create a visible shape
                  appearance Appearance {
                        material Material {
                              diffuseColor 1 0 0
                        }
                  }
                  geometry Box { }
            }
      ] # end list of linked objects
      description "The VRML Repository" # explains!
      parameter [ "target=top" ] # sends to top frame
url [ "http://www.sdsc.edu/vrml/" ] # Repository
}
# Transform with Sphere & Cone
Transform {
      children [
            Anchor {  # only one link, Sphere
                  children [
                        # Second object, green Sphere
            Shape {     #  visible
                  appearance Appearance {
                        material Material {
                              diffuseColor 0 1 0
                        }
                  }
                  geometry Sphere { }
            }
      ]
      description "Silicon Graphics' VRML Site"
      parameter [ "target=top" ] # sends to top frame
            url [ "http://vrml.sgi.com/" ] # SGI
      }
      # Transform with Cone
Transform {
      children [
            Anchor {  # only one link, Cone
```

```
                        children [
                              # Third object, blue Cone
                  Shape {     #  visible
                              appearance Appearance {
                                    material Material {
                                          diffuseColor 0 0 1
                                    }
                              }
                              geometry Cone { }
                        }
            ]
            description "Intervista's Home Page"
                                    parameter [ "target=top" ]
                              url [ "http://www.intervista.com/" ]  #IV
                        }
]  # end list of Transform children
translation 3 0 0 # away from Sphere
}
] # end list of Transform children
translation 3 0 0 # away from Box
      }
```

We'll also need a slightly different frame definition, on the CD-ROM as working_frames.html:

```
<html>
<head>
<title>Learning VRML Chapter 13: HTML/VRML Frames</title>
</head>
<frameset rows="*,150">
<frame src="" name="top">
<frame src="13_7.wrl" name="bottom">
</frameset>
```

When we load this HTML document, we'll see the world correctly loaded into the bottom frame of the browser, but when we click on the green Sphere, we see that Silicon Graphics' VRML home page does load into the top frame.

This is really just scratching the surface of how to use VRML in conjunction with HTML, frames and the Web; you can put many other values into the parameter field of the Anchor node; it's an MFString, so you can have as many parameters as you'd like. This means you can use VRML as a navigation mechanism; Microsoft does this at their own VRML site.

The *VRML Toolbar* allows you to navigate through the frames of content at Web site in a way that's interesting and almost entirely intuitive. It should give you more than a few good ideas on how you can use VRML controls to drive users through your own Web site in a way that's both sensible and fun!