Chapter 7

Publishing Paradigms for X3D

Nicholas F. Polys

7.1 Introduction: Publishing Paradigms

As the demands of data and user tasks evolve and expand, the field of Information Visualization presents many challenges for designers and systems developers. Of primary concern is the mapping of data records and attributes to a visual presentation that enables the user to detect patterns and relationships within the data. The goal of this mapping is to minimize the user's cognitive requirements for understanding and insight into the nature of the data that may not be apparent from viewing it in its raw form. The mapping of data to a visualization must take into account the data's volume and types and this chapter will discuss some approaches to this display problem. However, static presentations are limiting in their power to inform because the data and mappings cannot be interactively explored or rearranged. Computer-based visualizations can address this problem because users can now have control over the selection of data records, the encoding of those records as visual markers and the presentation of those markers in a 2D screen or a 3D world. In this chapter, we will examine how data may be mapped to interactive 3D worlds that may be published and distributed over the World Wide Web (WWW).

In the early days of Web publishing, repurposing data content for multiple formats and platforms was expensive and, as a result, a majority of useful information was locked into technology "silos" for a particular delivery format, method and platform. International standards organizations serve the computing community by developing and specifying open platforms for digital data exchange. By adhering to industry standards, organizations can lower their software and data integration costs and maximize their data re-use while guaranteeing reliability and user access beyond market and political vagaries. Extensible Markup Language (XML) and Extensible 3D (X3D) are two examples of such standards and are covered in this volume. This chapter provides an overview of issues, strategies and technologies used for publishing information visualizations with XML and X3D.

7.1.1 File Formats and the Identity Paradigm

Initially, the majority of published information on the World Wide Web was in a format called HyperText Markup Language (HTML). HTML was revolutionary in

that it specified a declarative language for sharing documents (Web pages) across a network. The resulting boom to now multiple millions of Web pages is largely due to the simplicity and portability of this language. Information and images can be easily layed out, linked and accessed from all over the globe. If the author knows the HTML content header and tags, a basic document can be produced with a text editor and an image editing program. A document's headings, layouts, images, links, colours and fonts are all described with HTML tags. More complex or innovative layouts require the use of `<table>` tags, which are difficult to manage without authoring software.

One major drawback of HTML is that its tags are strictly specified and overloaded. Tags in an HTML document represent both the informational content *and* the presentation of that content, that is, the data and the display information are included in the same file, often in the same tags. This limitation makes HTML tags less attractive as a data storage medium since it is difficult to repurpose data to other formats and applications. For example, if a customer's name and order number are enclosed by separate header tags, such as `<h1>`, there is no way to distinguish which information is the name and which information is the number from the tags in the file alone. Cascading Stylesheets (CSS) attempts to separate content and presentation in HTML by allowing the author to specify classes of tags with defined display attributes such as font, colour, fill and border. CSS provides flexibility by allowing definitions to reside within document files or as remote resources. CSS is useful for presenting the same page with different styles. However, this flexibility is not really a qualitative improvement in the language because the tagset is mostly unchanged and still finite in its descriptive power for data.

Virtual Reality Modelling Language (VRML) is an international standard (ISO/IEC 14772-1:1997 and ISO/IEC 14772-2:2002), but was designed as a portable format for describing and delivering interactive 3D worlds. The VRML standard is similar to HTML in that it is declarative, strictly specified and carries both data and display information. In contrast to an HTML page, the VRML scene contains spatial viewpoint and navigation information, 3D geometry with colours, transparency and textures, text, fonts, links, backgrounds and temporal information such as object animations and behaviours (defined in Interpolators, Sensors and Scripts). Also, in contrast to HTML, VRML authors have the ability to define their own node types through the PROTO(type) node. The `PROTO` definitions can reside within the document file or as remote resources.

In VRML and X3D, nodes are analogous to element tags and fields are analogous to element attributes. Nodes are instantiated in a directed, acyclic graph called the scenegraph. A VRML file describes a scenegraph of interactive objects in space which the user can see and navigate through. Coloured and textured objects are manifested in the world (the scene), animated and visualized from a viewpoint or camera. When discussing Web3D media, the "viewpoint" will be referred to as the `Viewpoint` node itself and the "camera" as the rendered result of the `Viewpoint` via any superseding transformations. Similarly, "navigation" refers to the scale and nature of the user's control over their `Viewpoint` (by way of the values bound in the active `NavigationInfo` node).

Early ease of authoring was complicated by lack of browser compliance with the standards and scripting support for JavaScript (now officially "ECMAScript") varied widely. In some cases, Web publishers were forced to maintain multiple, browser-specific copies of their content in order to guarantee the widest possible

accessibility. This amount of redundancy is expensive, even to percolate a small change across multiple Web site versions. Yet as the standards, client software and server technologies have matured, HTML, VRML or ECMAScript compliance is less the reason for maintaining multiple Web sites. Now, the motivation for permutable content is founded on the goal of customizing information for an audience or partner with a range of capabilities and interests. Although HTML enabled the exponential growth of the Web, it also required organizations to grapple with content management and personalization issues. The result was the design and deployment of Web "application servers" and Web "portals". We shall examine how these architectures currently apply to Web publishing and then to Web3D content, specifically X3D and VRML.

Hypertext Markup Language was originally designed to describe and deliver hypertext documents over the Web. Virtual Reality Modelling Language was originally designed to describe and deliver interactive 3D worlds over the Web. They are consequently unable to describe much else. Each is really only suitable as a Web publishing format, not as the formats for content storage, archiving and exchange. If pages are authored and maintained in a specific format (such as HTML or VRML) and the content is also delivered in that format (HTML or VRML), we can characterize the architecture as conforming to the "Identity Paradigm" – the source is identical with the deliverable. As mentioned above, this presents some problems both with maintaining a large set of documents and with re-using the documents' information in other contexts. Due to the limitations and expenses of this methodology, there was an immediate demand for other solutions. XML and X3D were designed to meet this demand.

7.1.2 Server Technologies and the Composition Paradigm

In recent years, a number of alternatives have been provided by Web server technologies and scripting languages to address these issues of maintaining static Identity Paradigm archives. Some well-known technologies include Server Side Includes (SSI), Perl, Hypertext Preprocessor (PHP) and Java Server Pages (JSP). These technologies do have significant differences, but the common denominator is that they permit the composition and delivery of a document "on-the-fly" in response to a user request. For example, when a user requests a page through the Hypertext Transfer Protocol (HTTP), SSI can get the current date and time from the Web server and display it in the delivered page. SSI can also insert markup fragments into a document. This allows different documents to include consistent display objects [such as headers, menus, tables and footers in 2D HTML and Heads-Up-Displays (HUDs) in 3D], reducing redundant content across multiple documents.

Scripting languages add another level of capability since they can connect and query online databases to recover information for display. For example, the user requests an online data set and the server script queries a database and writes it into the delivered (result) document. These solutions can all be classified as supporting the "Composition Paradigm", where documents are dynamically generated from one or more data sources. The Composition Paradigm brings more flexibility to Web publishing as developers can define common elements in a single location, pull data from multiple sources and combine them according to a user's request. As a result, dynamic Web sites are now commonplace.

A crucial issue in Web publishing that relates to the Composition Paradigm is the notion of content-type headers, or MIME types. MIME stands for Multipart Internet Mail Extension and was originally designed to distinguish files in Email attachments. The MIME type tells the client what kind of data is contained in the file so that the client can decode and handle it appropriately. For files on a local machine, this delegation can be accomplished simply by the file extension. In a Web server context, however, the MIME type is sent first as a single line and does not appear in the document source. Each Web server is configured to associate a document MIME type with a file extension and deliver it to the client. Web browsers or client operating systems also maintain such a list, which determines what plug-in or application will display the content. Hence every file on the Web has a content header that declares what kind of file it is.

Table 7.1 shows the relevant content types treated in this chapter. X3D (VRML V3.0) is a new standard for creating real-time 3D content. The specification of X3D's Architecture and API follows ISO/IEC FCD 19775:200x. Using the VRML97 specification as its starting point, X3D is cross-platform and hardware independent. It adds a number of new features, such as XML integration, multi-texturing, NURBS and a new scripting API. The X3D "Classic" encoding is a brace-and-bracket utf8 file encoding that looks like VRML97. The X3D "XML" encoding uses XML tags and attributes. At the time of writing, the X3D "Binary" encoding is still under development, but suffice it to say that a given scenegraph may be equivalently expressed in any encoding. The encoding specification for X3D is ISO/IEC FCD 19776:200x.

In practice, composability is generally accomplished with three ingredients: a structured template(s) for the delivered document, an accessible data source and a server technology such as SSI, PHP, JSP or Perl to compose the template with the appropriate data. Document templates basically structure the delivered document. As we shall see in Section 7.3.2, they are the skeletal form of the file, either implicit or explicit. Accessible data sources include both databases (i.e., SQL) and/or documents or fragments of documents. Server-side scripts manage the data sources and populate the template before it is sent to the user. The composed content is then delivered to the user with the appropriate MIME type.

On an Apache Web server, you might want to compose an X3D or VRML scene with PHP, Perl or some other server-side scripting language, but still have user's 3D plug-ins recognize it when it is received. In the case of composing VRML

Table 7.1 Principle filename extensions and MIME content types discussed in this chapter

File format	Content type/filename extension
Text	text/plain
HTML	text/html
VRML V2.0	model/vrml
XML	text/xml
VRML V3.0:	
X3D (Classic encoding)	model/x3d + vrml .x3dv and .x3dvz
X3D (XML encoding)	model/x3d + xml .x3d and .x3dz
X3D (Binary encoding)	model/x3d + binary .x3db and .x3dbz

or X3D Classic files, you could specify MIME types for a given folder by adding the line

```
AddType application/x-httpd-php .php .wrl .x3dv
```

or analogous definition to the .htaccess file in the directory on the Web server. This line configures the Web server to treat .wrl and .x3dv files and .php file requests as PHP files. This way, the Hypertext Preprocessor (PHP) engine is invoked when it serves both types of files and downstream applications such as browser plug-ins will recognize VRML and X3D content composed from PHP scripts in that directory.

The Composition Paradigm introduced a new level of capability for publishing dynamic Web content. In enabled Web "Portals", which refer to a single site that links and includes relevant information for a particular audience or domain. Portals are usually dynamic and customizable per individual user. Users can specify what information is included in what part of the layout, and what look and feel they prefer. In most cases, this kind of personalization system requires the user either to log in to the site or grant permission to set a cookie on their machine. Once the user is identified by the system, personalized content can be dynamically generated and delivered. This includes delivering customized information content to a user who is logged in from a workstation, a VR system or a mobile device, such as a PDA.

7.1.3 XML and the Pipeline Paradigm

The World Wide Web Consortium's (W3C) meta-language codification of XML has opened new and powerful opportunities for information visualization, as a host of structured data can now be transformed and/or repurposed for multiple presentation formats and interaction venues. XML is a textual format for the interchange of structured data between applications (W3C, 1998–2002; Kay, 2001; White, 2002). The great advantage of XML is that it provides a structured data representation built for the purpose of separating content from presentation. This allows the advantage of manipulating and transforming content independently of its display. It also dramatically reduces development and maintenance costs by allowing easy integration of legacy data to a *single data representation which can be presented in multiple contexts or forms,* depending on the needs of the viewer (i.e., the client). Publishers reduce the ratio of maintained source files to presentation venues as source data tends toward semantic markup (Apache Foundation, 2002).

Data becomes portable as multiple formats may be generated downstream according to application or user needs. Another important aspect of XML is the tools that it provides: the DTD and the Schema. The Document Type Definitions (DTD) define "valid" or "legal" document structure according to the syntax and hierarchy of the language elements. The Schema specifies data types and allowable expressions for the language elements and their attributes – it is a primitive ontology describing the document language's semantics. Using any combination of these, "high-level markup tags" may be defined by application developers and integration managers. This allows customized and compliant content to be built by authors and domain specialists. These tags could describe prototyped user-interface elements, humanoid taxonomies or geospatial representations. Developers describe the valid datamodel for their application using the DTD and Schema, share it over the Web and standardize it amongst their community.

XML can be as strict or as open as needed. Content, or fragments of content, can be "well-formed" and still processed with most XML tools. Typically, data validation is at author time, but it can be done at serving, loading or runtime if needed. Publishing advances using XML technologies can be characterized as the "Pipeline Paradigm" – information is stored in an XML format and transformed into a document or parts of a document for delivery. From an XML-compliant source document (or fragment), logical transformations [Extensible Style Sheet Transformations (XSLT)] can be applied to convert the XML data and structure to another XML document or fragment. A series of such transformations may be applied ending with a presentation-layer transformation for a final delivery-target style, content-type integration and display.

Numerous developer resources exist for the W3C's XSLT specification (Kay, 2001; White, 2002). However, a review of the typical XSL Transformation process is in order:

1. An XSLT engine parses the source XML document into a tree structure of elements.
2. The XSLT engine transforms the XML document using pattern matching and template rules in the .xsl style-sheet.
3. Template elements and attribute values replace matched element/attribute patterns in the source document to the result document.

The Web3D Consortium's next-generation successor to VRML is X3D. Like XML, which moves beyond just specifying a file format or a language like VRML or HTML, it is a set of objects and interfaces for interactive 3D Virtual Environments with defined bindings for multiple profiles and encodings collected under a standard API (Walsh and Sévenier, 2001; Web3D, 2002). Like VRML, the X3D specification describes the abstract performance of a directed, acyclic scenegraph for interactive 3D worlds. In addition, it takes advantage of recent graphics advancements such as MultiTexturing and information technology advancements such as XML. X3D can be encoded with an XML binding using DTDs and Schema (Web3D, 2002). The X3D Task Group has provided a DTD, Schema, an interactive editor and a set of XSLT and conversion tools for working with X3D and VRML97. Using the XML encoding of X3D, authors can leverage all the benefits of XML and XML tools such as user-defined markup tags, XSLT, authoring environments and server systems.

Additionally, rather than defining a monolithic standard, the X3D specification is modularized into components which make up "Profiles". Profiles are specific sets of functionality designed to address different applications – from simple geometry interchange or interaction for mobile devices and thin clients to the more full-blown capabilities of graphical workstations and immersive computing platforms. The notion of X3D Profiles is important for publishing visualizations and we will examine them in more detail in subsequent sections. X3D may be presented in a native X3D browser such as Xj3D (Web3D, 2002), or transformed again and delivered to a VRML97 viewer.

7.1.4 Hybrid Paradigm

The last publishing paradigm we will describe is the "Hybrid Paradigm". The Hybrid Paradigm combines the Pipeline and Composition paradigms. Data from

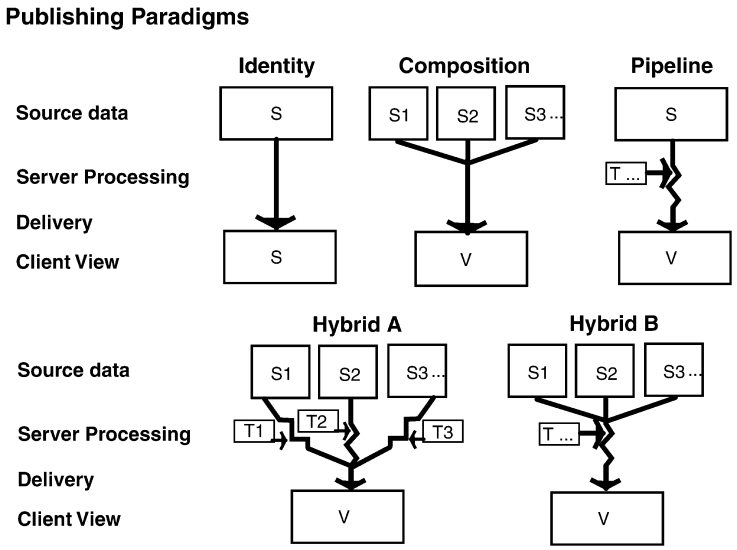

Figure 7.1 Publishing paradigms summarized: S = source; V = view; T = transformation.

various sources and transformational pipelines can be dynamically composed into a scene and delivered to the client machine. Apache Cocoon and Perl with the Gnome XML libraries are two well-known examples of technologies that permit such a flexible scheme. Figure 7.1 shows the principal differences between the paradigms described in this section.

7.2 Visualizing Information

Card et al. (1999) have defined Information Visualization as "The use of computer-supported, interactive, visual representations of abstract data to amplify cognition" (p. 7). This definition provides us with a clear starting point to describe visualization techniques for X3D as it distinguishes abstract data from other types of data that directly describe physical reality or are inherently spatial (e.g., anatomy or molecular structure). Abstract data includes things such as financial reports, collections of documents and Web traffic records. Abstract data does not have obvious spatial mappings or visible forms, hence the challenge is to determine effective visual representations and interaction schemes for human analysis, decision-making and discovery. Therefore, information visualization concerns presenting the user with the perceptual substrates for comprehension and insight. It must account for the nature, scope and amount of data and also human cognitive factors in the building of mental models.

7.2.1 Graphical Information Design

The nature of visual perception is obviously a crucial factor in the design of effective graphics. The challenge is to understand human perceptual dimensions and map

data to its display in order that dependent variables can be instantly perceived and processed preconsciously and in parallel (Friedhoff and Peercy, 2000). Such properties of the visual system have been described (i.e., sensitivity to texture, colour, motion, depth) and graphical presentation models have been formulated to exploit these properties, such as preattentive processing (Pickett et al., 1996) and visual cues and perception (Keller, 1993).

Primary factors in visualization design concern both the data (its dimensionality, type, scale, range and attributes of interest) and human factors (the user's purpose and expertise). Different data types and tasks require different representation and interaction techniques. How users construct knowledge about what a graphic "means" is also of inherent interest to visualization applications. For users to understand and interpret images, higher level cognitive processes are usually needed. A number of authors have enumerated design strategies and representation parameters for rendering signifieds in graphics (Bertin, 1981; Tufte, 1990) and there are effects from both the kind of data and the kind of task (Schneiderman, 1996). Card et al. (1999) examined a variety of graphical forms and critically compared visual cues in scatter-plots, cone-trees, hyperbolic trees, tree maps, points-of-interest and perspective wall renderings. As we shall see, their work is important since any of these 2D visualizations may be embedded inside, or manifested as, a virtual environment.

Interactive computer graphics present another level of complication for determining meaning as they are responsive, dynamic and may take diverse forms. There are challenges both on the input medium for the user and the action medium for the user. These are known as the Gulf of Evaluation and the Gulf of Execution, respectively (Norman, 1986). Typically in the literature, visualizations are described and categorized per user task such as exploring, finding, comparing, and recognizing (patterns). These tasks are also common in interactive 3D worlds. Information objects may need to be depicted with affordances for such actions. Here we shall examine how interactive visual markers can be designed and delivered with X3D.

7.2.2 Visual Markers

General types of data can be described as quantitative (numerical), ordinal and nominal (or categorical). Visualization design requires the mapping of data attributes to "visual markers" (the graphical representations of those attributes). Mappings must be computable (they must be able to be generated by a computer) and they must be comprehensible by the user (the user must understand the rules that govern the mapping in order to interpret the visualization). The employment of various visual markers can be defined by the visualization designer or defined by the user. Tools such as Spotfire (Ahlberg and Wistrand, 1995) and Snap (North and Schneiderman, 2000) are good examples of this interactive user control over the display process. This functionality can also be accomplished with X3D. In addition, a set of "modes" of interaction have been proposed for exploratory data visualizations which attempt to account for user feedback and control in a runtime display (Hibbard et al., 1995a). Table 7.2 summarizes the ordering of visual markers by accuracy for the general data types. These rankings lay a foundation for identifying parameters that increase the information bandwidth between visual stimuli and user.

Table 7.2 Accuracy rankings for visual markers by general data type

Data type	Quantitative	Ordinal	Nominal
Graphical representation	Position Length Angle/slope Area Volume Colour/density (Cleveland and McGill, 1984)	Position Density Colour Texture Connection Containment Length Angle Slope Area Volume (Mackinlay, 1986)	Position Colour Texture Connection Containment Density Shape Length Angle Slope Area Volume (Mackinlay, 1986)

Schneiderman (1996) outlined a task and data type taxonomy for information visualizations which is also useful for our description of techniques for X3D. Top-level tasks are enumerated as Overview, Zoom, Filter, Detail-on-demand, Relate, History and Extract. Overview refers to a top-level or global view of the information space. Zoom, Filter and Details-on-demand refer to the capability to "drill down" to items of interest and inspect more details (of their attributes). History refers to the "undo" capability (i.e., returning to a previous state or view) and Extract is visualizing sub-sets of the data. Enumerated data types are One-dimensional, Two-dimensional, Three-dimensional, Multidimensional, Temporal, Tree and Network. Since each of these can be implemented with X3D, these distinctions will be referred to throughout the remainder of the chapter.

Card et al. (1999) and Hibbard et al. (1995b) have described a reference model for mapping data to visual forms that we can apply to our discussion. Beginning with raw data, which may be highly dimensional and heterogeneous, data transformations are applied to produce a "Data Table" that encapsulates the records and attributes of interest. The data table also includes metadata which describes the respective axis of the data values. Visual mappings such as those shown in Table 7.2 are then applied to the data table to produce the visual structures of the visualization. The final transformation stage involves defining the user views and navigational mechanisms to these visual structures. As the user interactively explores the data, this process is repeated. Ideally, the user has interactive control (feedback) on any step in the process (the data transformation, visual mappings and view transformations).

7.3 Design Principles and Interactive Strategies

Many challenges exist in the design of interactive 3D worlds and interfaces when integrating symbolic and perceptual information (Bolter et al., 1995). Similarly to efforts for 2D Visualization, researchers have experimented with the mapping of attributes to various visualization metaphors including the cone-tree, the city and the building metaphor (Dos Santos et al., 2000). They have shown that accurate

characterization of the data is crucial to a successful 3D visualization, especially when the scenegraph is auto-generated. Bowman et al. (1998, 1999, 2003) have implemented and evaluated "Information-Rich Virtual Environments" (IRVEs) with a number of features that are common to most Web3D information spaces. Information-rich virtual environments "… consist not only of three-dimensional graphics and other spatial data, but also include information of an abstract or symbolic nature that is related to the space", and IRVEs "embed symbolic information within a realistic 3D environment" (Bowman et al., 1999). This symbolic information could be attributes such as text and numbers, images, audio clips and hyperlinks that are related to the space or the objects in the space. In this section, we will attempt to formalize an approach that is consistent with the capabilities of X3D.

Delivering arbitrary XML data to information visualizations with the Pipeline Paradigm requires both design and implementation considerations. As mentioned above, the generation of a data table is the first step in the delivery of a visualization. The transformation of raw data to the data table may be accomplished by XSLT or extracted by an XPath query or a query to a database. For the second phase of mapping – the data table to visual structures – we should remember from our definition that the abstract data in the table does not contain any inherently spatial information; have it requires that the author determine the visual markers that will be employed. We shall examine this step in more detail in Section 7.4 especially as it relates to XSLT and X3D.

When designing 3D scenes for any purpose, a crucial step is that of "Storyboarding", which helps authors specify what objects the scene contains and their appearance, from what points of view it can be perceived and what kinds of interaction are appropriate at various points in time and space. When designing a usable visualization, Schneiderman's (1996) mantra of information design should ring in your head: "Overview first, zoom and filter, then details-on-demand".

7.3.1 Scene Production Process

Beginning with user requirements, a typical scene production process will follow these steps:

1. Define environment and locations.
2. Define user interface and viewpoints.
3. Define interactions.
4. Organize declarative scenegraph.
5. Model objects.
6. Build prototypes.
7. Transform data and compose visual markers.
8. Deliver to user.

Steps 1–4 can be accomplished from the storyboard. Step 5 is typically done with a 3D modelling package that can export X3D or VRML. Steps 4 and 6 require at minimum a text editor and a developer familiar with the scenegraph capabilities of X3D or VRML. Steps 7 and 8 use server technologies and scripts to manifest the scene and deliver its final presentation form to the user.

7.3.2 Scene Structure

Structured design in the case of X3D means dividing a scene into blocks which account for the various functional parts of the world. Using a modular structure to build a scene means that it may be built (composed) and managed from any number of applications or databases to the final target presentation. The result of this approach should be an implicitly structured X3D document template describing scenes in the form of:

Served Content type

1. Header.
2. Scenegraph root:
   - Custom node declarations: `PROTO` definitions and/or `EXTERNPROTO` references.
   - Universe set (`Backgrounds`, global `ProximitySensors`).
   - HUD and User Interface.
   - Scripts.
   - World and Inhabitants set (lighting, geometry and objects).
   - ROUTEs.

The X3D specification defines a set of standard nodes that can be instantiated in the scenegraph, what kinds of events they can send and receive and where they can live in the scenegraph. The "transformation hierarchy" of a scenegraph describes the spatial relationship of rendering objects. The "behaviour graph" of a scenegraph describes the connections between fields and the flow of events through the system. Events in the X3D scenegraph are called `ROUTE`s and exist between nodes. If nodes are uniquely named (DEFed), data events can be programmatically addressed and routed to that node. Custom logic and behaviours can be built into a scene with `Script` nodes which use ECMAScipt and/or Java to execute data type conversion, computation and logic with events.

Designing scenes in modular blocks has additional benefits. For example, if the universe and HUD are kept consistent while the user navigates an information space, this helps to maintain the notion of presence when the world and its inhabitants change. Such runtime swapping of scenegraph branches (blocks) is possible with `Browser` API method calls in a Script node (see Section 7.4.4).

A primary consideration in mapping data to a visual form is the range of values in the data. For quantitative and ordinal data, designers should examine the highest and lowest values in order to scale coordinates properly. For categorical data, the number of categories will determine the colours that can be employed. Since visual mappings must be comprehensible, axes, labels and colour legends should be instantiated. Designers may choose to put axes and labels in the universe block or the world block, depending on the design and compositional resources of their visualization application.

7.3.2.1 Custom Nodes

Authors can aggregate nodes and field interfaces into "Prototype" nodes (`PROTO`s), which can be easily instantiated and reused in other scenegraphs and

scenegraph locations. Prototypes allow the efficient definition, encapsulation and re-use of interactive 3D objects. As we shall see, Prototypes are especially suited to designing visual markers and interactive widgets. In the interest of promoting the re-use of code without redundancy, Prototypes can also be defined in external files (`EXTERNPROTO`s). This prototype definition is a separate, singular resource that can be instantiated into multiple scenes.

One caveat to this abstract document structure is important: the ability to use Prototypes (e.g., `PROTO`s and `EXTERNPROTO`s) to create user-defined objects and to use Scripts to define special behaviours (e.g., world or interaction logic) exist only in the "Immersive Profile" (and higher) of X3D, which is analogous to (but not identical with) the functionality enabled by VRML97. As we mentioned above, Profiles are specific sets of functionality designed to address different application domains (Web3D, 2003). The "Interchange Profile" contains a node-set to describe simple geometries, materials and textures for sharing between applications such as modelling tools. The "Interactive Profile" adds interpolator nodes for animation, sensors and event utilities for interactive behaviours and a more capable lighting model. Additionally, on top of the Immersive Profile, other software components may be defined and implemented. Currently specified components include Humanoid Animation (H-Anim), Geospatial 3D graphics (Geo-VRML) and Distributed Interactive Simulation (DIS). The "Full Profile" refers to full support for all components currently defined in the X3D specification. Authors should design to Profiles as they define what capabilities the client has – what nodes it can read and render.

7.3.2.2 Viewpoints and Navigation

An X3D scene defines objects in Euclidean coordinates, and animation interpolators generally proceed along linear time (although programmatic generation and manipulation of time values are possible with the Script node). Virtual environment X3D scenes would not be visible or explorable without a way to describe user viewpoints and navigation. A key to understanding how this is accomplished with X3D (or VRML) is the idea of a runtime "binding stack". A binding stack is basically a list of "bindable" children nodes in the scene where the top node is active or "bound". The first `Viewpoint` and `NavigationInfo` nodes defined in a file are the first to be actively bound. Other `Viewpoint` and `NavigationInfo` nodes are made active by `ROUTE`ing a Boolean event of TRUE to their `set_bind` field. When this happens, the user's view and navigation function according to the field values of the newly bound node. Alternatively, events routed to the active node change the observed behaviour of that node. For example, the `Viewpoint` node has fields for `position`, `orientation`, `fieldOfView` and `jump`. The `fieldOfView` defines the user's viewing frustum and can therefore be modulated to create fish-eye or telescoping effects. It is recommended that a FALSE value be used for the `jump` field, as the user's view is then smoothly animated to that Viewpoint when it is bound, reducing disorientation (Bowman et al., 1997).

Similarly, the `NavigationInfo` node carries fields that have a direct impact on the user's perception, including `avatarSize`, `speed` and `type`. For example, as a user navigates into smaller and smaller scales the `avatarSize` and `speed` fields should also be proportionally scaled down. Specified X3D navigation types are "`WALK`", "`FLY`", "`EXAMINE`", "`LOOKAT`", "`ANY`", and "`NONE`". While the

first five types give the user different ways of controlling their movement within the scene, in some cases it may be preferrable to use "NONE" in order to constrain their movement. Such a value would be desirable in the case of a "guided tour". If developers have access to mouse or want data in their runtime engine, they can build their own navigation types using prototypes, scripts and other scenegraph nodes.

7.3.2.3 Example Scenegraph: a Heads-Up-Display

`ProximitySensor` nodes output events called `position_changed` and `orientation_changed`. By placing a `ProximitySensor` at the origin, we have access to constant updates of the user's location and direction in the 3D world. If appropriate, we can then place a Heads-Up-Display (HUD) in front of the user and within their field-of-view. `ROUTE`ing the output of the `ProximitySensor` to the HUD's parent transform allows the HUD to travel continually with the user. The following code fragments illustrated this basic design:

```
<Scene>
<ProtoDeclare name="markerP">
  <ProtoInterface> ...
  </ProtoInterface>
    <ProtoBody> ...
    </ProtoBody>
</ProtoDeclare>
  ...
<Group DEF="universe_context">
  <ProximitySensor DEF="universe_origin" centre=
    "0 0 0" size="1000 1000 1000"/>
<NavigationInfo type="EXAMINE ANY"/>
  <Background/>
  </Group>
  <Group DEF="HUD_UI">
    <Transform DEF="HUD">
      <Transform translation="-0.05 0.03 -0.2">
        <!-- some hud scenegraph translated by an offset
          to user's point of view -->
        <Shape DEF="hud_geometry">
          <Box size=".1 .1 .1"/>
          <Appearance>
            <Material diffuseColor="1 1 1"/>
          </Appearance>
        </Shape>
      </Transform>
    </Transform>
  </Group>
...
<Group DEF="worldGroup">...
</Group>
...
```

```
  <ROUTE fromField="position_changed" fromNode=
    "universe_origin" toField="set_translation"
    toNode="HUD"/>
  <ROUTE fromField="orientation_changed" fromNode=
    "universe_origin" toField="set_rotation"
    toNode="HUD"/>
</Scene>
```

7.4 X3D and XSLT Techniques

Kim and Fishwick (2002) demonstrated the power of the content/presentation distinction when they used XML, Schemas and XSLT to render their XML descriptions of dynamic, physical systems to different 3D visual and system metaphors that they call "rubes". Dachselt et al. (2002) have demonstrated an abstracted, declarative XML and Schema to model Web3D scene components and especially interfaces. More recently, Dachselt et al. (2003) leveraged object-oriented concepts and XML Schema to componentize scenegraph node sets in the definition of user interface "Behaviour Graphs", which can be applied to arbitrary geometries or widgets. Finally, XSLT data transformations for audience-specific interactive visualizations have been shown for the delivery of Chemical Markup Language (CML) using X3D and VRML (Polys, 2003).

Applying the power of XSLT to the delivery of interactive 3D scenes is relatively new, and much more research is required in this area. As mentioned in Section 7.1.3, the representation of an XML document is by a tree data model. The nodes of the source graph can be selected and their attributes operated on in XSLT by the definition of `<xsl:templates match=""/>` that use XPath expressions and the `<xsl:variable name=""/>` element. XPath provides 13 axes by which the data tree may be navigated: child, descendant, parent, ancestor, following-sibling, preceding-sibling, following, preceding, attribute, namespace, self, descendant-or-self and ancestor-or-self. The target X3D tree (scenegraph) can be composed with the DOCTYPE

```
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
  "http://www.web3d.org/specifications/x3d-3.0.dtd">
```

There is a content model in X3D (expressed in the DTD and Schema) that constrains the target output and lets tools validate scene. While more formal theories including graph transformation principles are still forthcoming, we can begin to describe techniques for mapping data to visual structures (X3D nodes) for information visualization.

Including the X3D and VRML specifications, a number of resources exist (Ames et al., 1997; Walsh and Sévenier, 2001) that describe the syntax and behaviour of nodes in the scenegraph. Therefore, we will not cover all nodes in detail in this chapter, but rather show how particular nodes may be used to manifest visual markers for information visualizations. We will consider the X3D Immersive Profile as the target platform, although position, orientation, size, colour and shape can be mapped to the Interchange and Interactive profiles. All that is required to deliver content to these platforms is an alternative set of XSLT stylesheets that map the data to the supported target nodes and fields (attributes).

7.4.1 Target Nodes – Geometry

The Transform node manifests its children in the scene and provides fields such as `translation`, `rotation` and `scale` that account for position, orientation and size respectively. The `Transform` node's `translation` field takes an SFVec3f (a 3 float tuple) to define coordinates in 3-space where the children are located. `Rotation` is an SFRotation field where the first three values define a vector which serves as the rotational axis and the last value is an angle in radians which is the amount of rotation around that axis. The `scale` field is also a SFVec3f which defines a scaling factor for the node's children between 0 and 1 along each dimension (x, y and z).

`Shape` is obviously a crucial X3DChildNode. The `Shape` node describes both `geometry` and its `appearance`, such as colour and texture. The X3D colour model is defined in RGB space and specified in the `Material` node. In X3D, colour is specified by RGB values. The `specularColor` and `emissiveColors` modulate the diffuse colour by lighting, shape and point-of-view. In the literature on information visualization, there is a distinction between hue and saturation as visual markers in display mappings (Mackinlay, 1986). When colours are interpolated, the VRML Sourcebook (Ames et al., 1997) notes that colours are converted to HSV space (which does have a saturation factor) and then converted back to RGB. For readers interested in specifying saturation factors or converting between these colour spaces, the book by Foley et al. (1995) can be recommended. When mapping data to colour as a visual marker, it is important to use distinctive or contrasting colour scales so that users can differentiate the rendered values.

Three-dimensional geometry in an X3D scene may be built with any number of nodes, including the geometric "primitives" (`Box`, `Cylinder`, `Sphere`) and others such as `PointSet`, `IndexedLineSet`, `IndexedFaceSet`, `Extrusion`, the `Triangle*` family and `Text`. Each of these nodes has its own field signature and, depending on the designer's goal or user's task, the same data may be mapped to these different markers. Some brief notes about these shapes are in order. The `PointSet` node may be used for a scatter-plot, for example, but as a point does not have any volume, their specific values may be difficult to perceive in the rendering. Owing to the way in which some primitives' dimensions (e.g., the `Cylinder`'s `height` and the `Box`'s `size`) are defined, they usually need to be `Transform`ed (offset) by half of this dimension. `IndexedFaceSet` and `IndexedLineSet` geometries require a `coordIndex` field to specify the order in which the `Coordinate` points are connected.

In addition, X3D has extended VRML geometries by adding the Geometry2D component. `Arc2D`, `ArcClose2D`, `Circle2D`, `Disk2D`, `Polyline2D`, `Polypoint2D` and `Triangleset2D` are defined with this component. Similar 2D primitives are defined in SVG (W3C, 2002; this volume). The shapes in this component are new to Web3D worlds, and we expect them to be very useful in future visualization and interface designs. Currently, the Geometry2D component is only supported in the Immersive Profile.

7.4.2 Target Nodes – Hyperlinks and Direct Manipulation

The `Anchor` node is a grouping node that provides the ability for the user to click on its `children` and load an external resource. This is analogous to the hyperlinking

<a> tag in HTML and the default behaviour is for the resource to replace totally the currently loaded scene. The `url` field is of MFString type that lists the location of one or more resources. The browser attempts to find the first resource and load it; if it is not accessible, it tries the next one. Similarly to the HTML hyperlink, the `Anchor`'s `parameter` field can specify a frame or window target where the resource is to be loaded. When X3D or VRML files are specified as the resource, the link may also include a `Viewpoint` which is to be bound. This is done simply by appending `#DEFedViewpointName` to the `url`. The specified resource may also be a CGI script on the server and variable values may be passed to it, for example:

```
    url
  http://www.somedomain.org/sample/vistransformer.pl?
  marker=markerP&data=autos
```

In this case, the CGI script is responsible for delivering the content header and composing the scene.

Direct manipulation (such as clicking on an object and dragging it) in X3D can be accomplished through the use of DragSensors such as the `PlaneSensor`, `CylinderSensor` and `SphereSensor`. These nodes are activated when the user clicks on any of its sibling nodes and the output values are typically `ROUTE`d to a `Transform` node to effect a `translation` or `rotation`. A `TouchSensor` generates events such as `isOver`, and `touchTime` events (among others) that can be `ROUTE`d to other nodes in the scenegraph such as `Script`s to process user actions. Again, depending on the application and interactivity requirements, these may also be included in a Prototype definition.

7.4.3 Examples

Using the knowledge we have outlined above, let us have a look at some examples (Figures 7.2–7.4) of using XSLT to transform some abstract data into X3D scenes. Here is some sample XML data :

```
  <Vehicles>
<Auto name="SUV2001" MPG="8" Cylinders="8"
  Price="40,000"/>
<Auto name="SUV2000" MPG="12" Cylinders="8"
  Price="35,000"/>
<Auto name="Van2000" MPG="16" Cylinders="6"
  Price="30,000"/>
<Auto name="Pickup1990" MPG="23" Cylinders="4"
  Price="21,000"/>
<Auto name="Sedan1999" MPG="30" Cylinders="4"
  Price="18,000"/>
<Auto name="Compact2002" MPG="38" Cylinders="4"
  Price="14,000"/>
    </Vehicles>
```

In order to transform this data to an X3D visualization with XSLT, we define a template (or set of templates) that extract the source elements and attribute values in which we are interested. The templates in an XSLT stylesheet provide a mapping

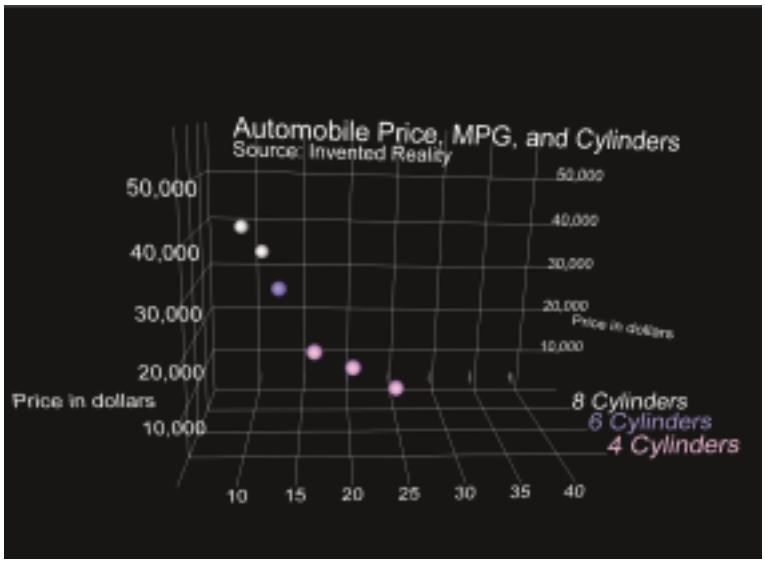

Figure 7.2 X3D scatter-plot geometry using positioned, colour-coded Spheres as the visual markers.

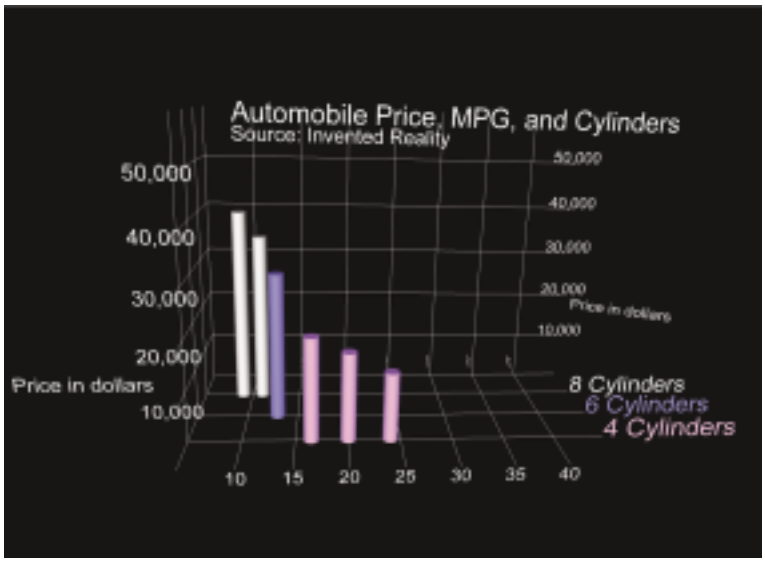

Figure 7.3 X3D bar graph (or histogram) geometry using positioned, colour-coded Cylinders and markers. Box primitives could also be used in this way.

from XML data to X3D informational objects. Common XSLT design patterns have been described, such as fill-in-the-blank, navigational, rule-based and computational (Kay, 2001). Based on this mapping, the XSL Transformation engine writes the data values into the template X3D tags and writes the result to the network or to

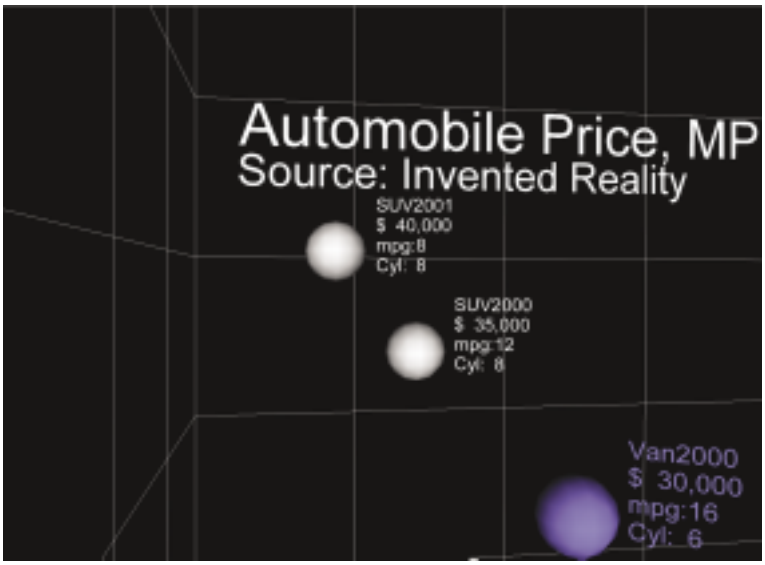

Figure 7.4 A zoomed-in view of Prototyped visual markers encapsulating perceptual and abstract information. The user has navigated into the higher price range.

a file (as in Section 7.5). For this example source data, we might write our XSLT as follows:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="UTF-8" media-type=
  "model/x3d+xml" indent="yes" cdata-section-elements=
  "Script" doctype-system="http://www.web3D.org/
  TaskGroups/x3d/translation/x3d-compact.dtd"/>

    <xsl:template match="/">
      <X3D profile="Immersive">&#10; <head>&#10;
          <meta content="translatedVehicleData.x3d"
            name="filename"/>
          <meta content="XSLT translation 1"
            name="description"/>
          <meta content="n_polys" name="author"/>
          </head>
          <Scene>
            <!-- Insert EXTERN / PROTO declarations,
              universe set, UI,

              and Scripts as needed -->
                <xsl:apply-templates/>
          </Scene>
      </X3D>
  </xsl:template>
  <xsl:template match="Vehicles">
```

```
    <Group DEF="worldGroup">
      <xsl:for-each select="Auto">
<xsl:variable name="name" select="@name"/>
<xsl:variable name="mpg" select="@MPG"/>
<xsl:variable name="cyl" select="@Cylinders"/>
<xsl:variable name="price" select="@Price"/>
                <Transform>
                  <!-- Manipulate the variables as
                    necessary and nstantiate X3D visual
                    markers (target geometry) -->
                </Transform>
      </xsl:for-each>
    </Group>
  </xsl:template>

</xsl:stylesheet>
```

Let us take a look at how some visual markers may be instantiated in an X3D scene. The following code fragment (shown in Figure 7.2) generates a scatter-plot view of the automobile dataset using an XSLT stylesheet to map quantitative data to a `Transform` node's `translation` field and categorical values to `Material`:

```
<Group DEF="worldGroup">
  <Transform translation=".8 4 1">
    <Shape DEF="marker1">
      <Appearance>
        <Material diffuseColor="1.0 1.0 1.0"/>
</Appearance>
       <Sphere radius=".15"/>
       </Shape>
  </Transform>
  <Transform translation="1.2 3.5 1">
  <Shape>
    <Appearance>
      <Material diffuseColor="1.0 1.0 1.0"/>
</Appearance>
       <Sphere radius=".15"/>
       </Shape>
  </Transform>
  <Transform translation="1.6 3 2">
    <Shape>
      <Appearance>
        <Material diffuseColor="0.31 0.3 0.61" />
</Appearance>
      <Sphere radius=".15"/>
  </Shape>
  </Transform>
  <Transform translation="2.3 2.1 3">
    <Shape>
      <Appearance>
        <Material diffuseColor="0.89 0.44 0.89" />
```

```
</Appearance>
    <Sphere radius=".15"/>
    </Shape>
  </Transform>
...
</Group>
```

The second example (Figure 7.3) implements quantitative values mapped to `Cylinder height` (which are `Transform`ed vertically by half their `height` value) and categorical values mapped to `Material`. The target X3D code for this example would be as follows:

```
<Group DEF="worldGroup">
  <Transform translation=".8 2 1">
    <Shape DEF="marker2">
      <Appearance>
        <Material diffuseColor="1.0 1.0 1.0"/>
</Appearance>
       <Cylinder height="4" radius=".15"/>
      </Shape>
  </Transform>
  <Transform translation="1.2 1.75 1">
    <Shape>
      <Appearance>
        <Material diffuseColor="1.0 1.0 1.0"/>
</Appearance>
       <Cylinder height="3.5" radius=".15"/>
      </Shape>
  </Transform>
  <Transform translation="1.6 1.5 2">
    <Shape>
      <Appearance>
      <Material diffuseColor="0.31 0.3 0.61" />
</Apearance>
      <Cylinder height="3" radius=".15"/>
    </Shape>
  </Transform>
  <Transform translation="2.3 1.05 3">
    <Shape>
      <Appearance>
        <Material diffuseColor="0.89 0.44 0.89" />
</Appearance>
       <Cylinder height="2.1" radius=".15"/>
      </Shape>
  </Transform>
...
</Group>
```

Prototypes' definitions can add another level of efficiency to the definition of data objects where multiple nodes can be encapsulated and re-used. In the first two examples, the initial overview `Viewpoint` gives us a rough idea about the

distribution of automobiles across the three variables. However, we would probably want to find out more detailed information about an automobile that met our criteria. To accomplish this without cluttering the visual space, we can define our visual markers with an LOD (Level-of-Detail) functionality, which renders different children based on the user's proximity. One such design would show the detailed view (a Text node reading the name, miles per gallon and price) when the user zooms in closer to an item of interest. In addition, Text could be placed on a Billboard node that rotates its children around their y-axis to always face the user.

Our third example populates a PrototypeInstance with values and has the high LOD containing Billboarded Text and the low level containing the geometry from the first example. The PrototypeDeclaration is named "markerP". The code for these visual markers using the automobile dataset is as follows:

```
<Group DEF="worldGroup">
  <ProtoInstance name="markerP">
    <fieldValue name="position" value=".8 4 1"/>
    <fieldValue name="cost" value="40,000"/>
    <fieldValue name="name" value="SUV2001"/>
    <fieldValue name="numcyl" value="8"/>
    <fieldValue name="miles" value="8"/>
    <fieldValue name="color" value="1.0 1.0 1.0"/>
  </ProtoInstance>
  <ProtoInstance name="markerP">
    <fieldValue name="position" value="1.2 3.5 1"/>
    <fieldValue name="cost" value="35,000"/>
    <fieldValue name="name" value="SUV2000"/>
    <fieldValue name="numcyl" value="8"/>
    <fieldValue name="miles" value="12"/>
    <fieldValue name="color" value="1.0 1.0 1.0"/>
  </ProtoInstance>
  <ProtoInstance name="markerP">
    <fieldValue name="position" value="1.6 3 2"/>
    <fieldValue name="cost" value="30,000"/>
    <fieldValue name="name" value="Van2000"/>
    <fieldValue name="numcyl" value="6"/>
    <fieldValue name="miles" value="16"/>
    <fieldValue name="color" value="0.31 0.3
      0.61"/>
  </ProtoInstance>
  <ProtoInstance name="markerP">
    <fieldValue name="position" value="2.3 2.1 3"/>
    <fieldValue name="cost" value="21,000"/>
    <fieldValue name="name" value="Pickup1990"/>
    <fieldValue name="numcyl" value="4"/>
    <fieldValue name="miles" value="23"/>
    <fieldValue name="color" value="0.89 0.44
      0.89"/>
  </ProtoInstance>
...
</Group>
```

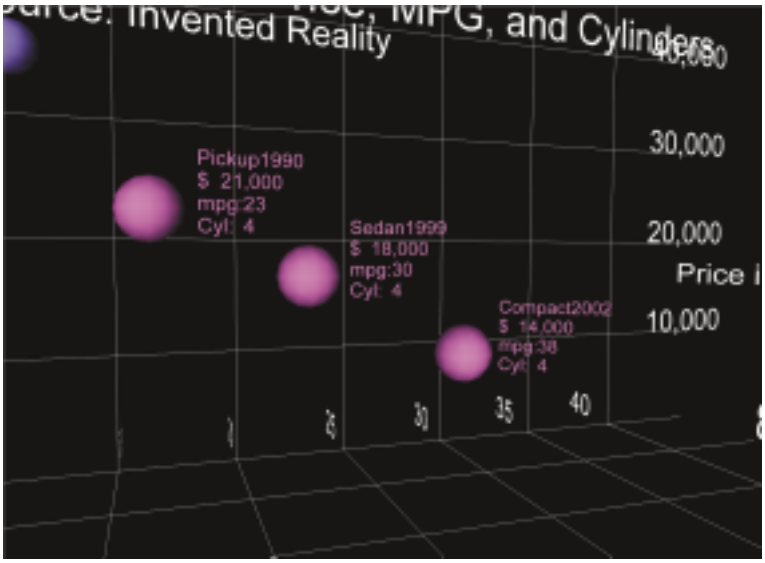

Figure 7.5 A zoomed-in view of Prototyped markers encapsulating perceptual and abstract information. The user has navigated into the lower price range.

Figures 7.4 and 7.5 show a sample visual marker `PROTO` that includes `LOD`, `Billboard` and `Text` features. From outside the detail `LOD` range, the scene would look exactly as in Figure 7.2.

XSLT can, of course, also be used to transform and compose X3D from data that has inherent spatial meaning such as locations, sizes and connectivity. For example, Figures 7.6 and 7.7 show the results of two different stylesheets that process a Chemical Markup Language (CML) file of the cholesterol molecule to X3D. The first version (Figure 7.6) builds geometry from atom and bond elements and text from abstract attributes and other meta-information.

The second transformed version (Figure 7.7) shows that the XSLT can add control widgets to the resulting X3D scene; in this case, a slider controls the `transparency` of every atom. In addition, the transformation in Figure 7.7 shows a new text style and also movable measuring axis instantiated in the "universe block" of the scene.

Figures 7.8 and 7.9 illustrate the XML to X3D transformation results of a finite-difference mesh of tissue used for in silico biological simulation.

7.4.4 Scene Management and Runtimes

Another important consideration in the composition and maintenance of world content is the use of the `Inline` node. In VRML, `Inlines` were opaque in that events could not be `ROUTE`d between the inlined and the inlining scenes. This event opacity is also a limitation of the `Browser.createX3DFromURL` method since nodes in the new world are not programmatically addressable. If authors wanted dynamically to replace a world block and connect it with event `ROUTE`s, the

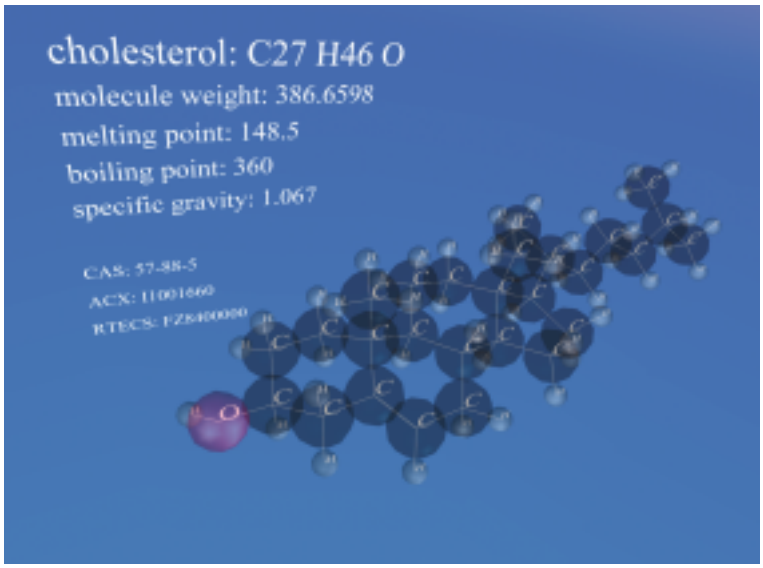

Figure 7.6 The results of an XSLT transformations of a CML file for cholesterol.

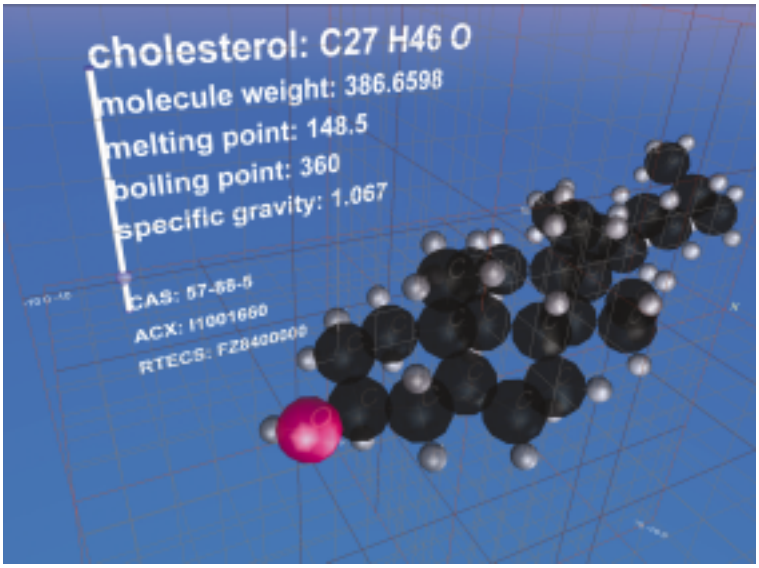

Figure 7.7 The results of an XSLT transformation of a CML file for cholesterol. A new FontStyle has been used and a slider widget has been added during the transformation and ROUTEd to visual markers in the scene.

not-so-obvious solution in VRML has been to define the entire replacement scene as Prototypes and then use the `Browser.createX3DfromString` method to add the new node and the `Browser.addRoute` method to connect events to it. The new X3D API is called the Scene Access Interface (SAI) and unifies the object

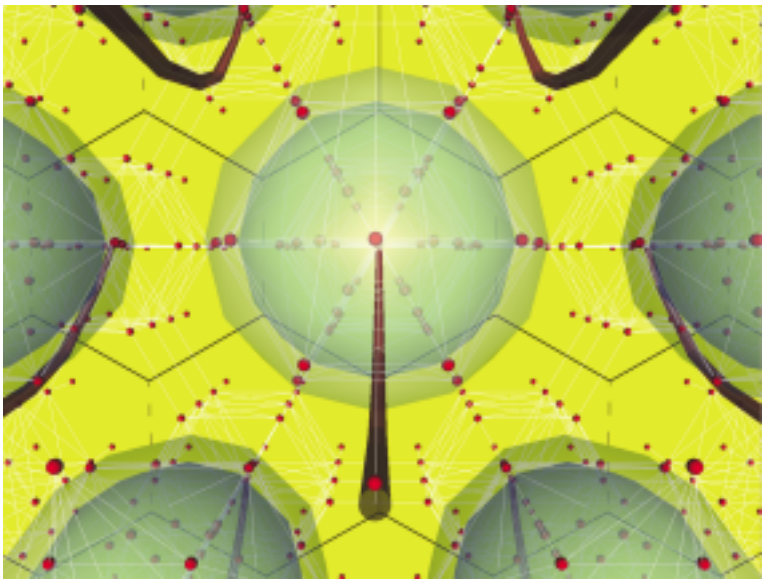

Figure 7.8 Underside view of an XML finite-difference mesh description generated via XSLT to X3D in order to visualize the spatial locations and connectivity of mesh points.

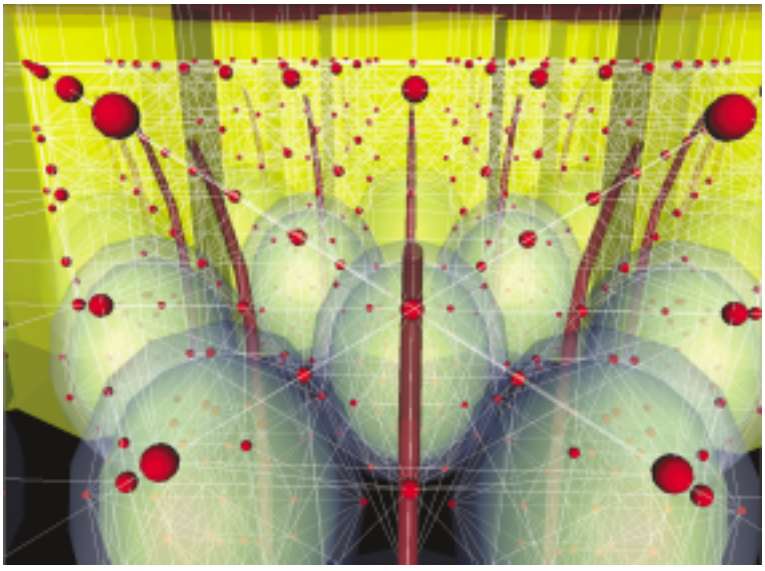

Figure 7.9 A front view of the XML finite-difference mesh.

definitions for both internal and external scripting. The SAI is a much more rich and rigorous programming specification than VRML supported and it introduces a number of new objects and functions. The bindings for the Java and ECMAScript languages are described in ISO/IEC FCD 19777 : 200x.

The `Browser` object interface, for example, has a number of useful methods for managing content dynamically, such the `Browser.createX3DFromURL` or `Browser.createX3DFromString` methods that can be invoked from a `Script`. These methods (whose analogues were specified in VRML97) allow scene content to be swapped during runtime. The content is added to a specific part of the scenegraph by specifying a DEFed node which the new content replaces. If the world has been designed in a modular way as we described above, this can be a very powerful technique.

Other important functionality newly introduced in X3D is the use of `IMPORT` and `EXPORT` keywords with `Inlines`. The `IMPORT` statement provides `ROUTE`ing access to all the fields of an externally defined node with a single statement and without a `PROTO` interface wrapper and `Scripts` building String objects. The `EXPORT` statement is used within an X3D file to specify nodes that may be imported into other scenes when inlining the file. Only names exported with an `EXPORT` statement are eligible to be imported into another file (Web3D, 2002). In this way, entire X3D files can declare event communication routes for embedding and embedded files. This is a significant improvement in the composability and re-use of X3D worlds themelves.

7.5 Publishing Technologies

We have examined some techniques for transforming XML data to X3D with the use of XSLT stylesheets. The X3D Task Group has provided a number of XSLT stylesheets for the transformation of X3D to VRML97 and also X3D to HTML. Also, courtesy of the National Institute of Standards and Technology, a translator application for VRML97 → X3D data migration has been made freely available and been integrated into a number of Web3D editing tools including the structured editor X3D Edit (Web3D, 2003) and others. Within the Pipeline and Hybrid paradigms, there are two general ways we shall consider in publishing XML content to X3D (or other): the back-end production of a file archive, and the serving of a transformed and presented source document in response to a "live" (networked) visualization request. Thus we distinguish between the auto-generation of content archives and the serving of dynamic content for on-the-fly service.

Given server overhead, bandwidth and delivery constraints, periodically auto-generating content archives may be appropriate. These approaches use X3D source files and directories with naming conventions with scripted XSLT to produce framed HTML, VRML and X3D document trees complete with linked with chapters, titles and embedded views of the source file. The generated document trees can be organized and hyperlinked for navigation with a Web browser, for example. The X3D Task Group's Web collection of X3D content examples is an ideal showcase of this technique (Web3D, 2002). The auto-generation can be done with straightforward batched XSLT Java (Kay, 2001; McLaughlin, 2001; White, 2002) or Perl (Brown, 2002; Polys, 2003) scripts. These content publications can then be served over the Web or distributed on CD or DVD as in the Identity Paradigm.

The second approach is to use XSL Transformations "on-the-fly" using common Web server software such as Apache Cocoon, Perl and the XML Gnome libraries, or PHP (Brown, 2002). This approach can provide custom presentations of the source data with a proportionate server and network overhead. Either of these delivery

approaches may be classified as conforming to the Pipeline, Composition or Hybrid paradigms depending on how the data is transformed and composed.

7.6 Summary

In this chapter, we have reviewed the literature on interactive 3D visualizations and enumerated criteria to design successful and comprehensible visualizations. We looked at modular approaches to X3D scene design and production and examined how XSLT can be used to transform and deliver XML data to X3D visualizations within current publishing paradigms. The separation of content from presentation in XML gives organizations a great deal of flexibility in how developers re-purpose and publish their data. The XML encoding of X3D allows developers to leverage the power of XML to transform the same data to multiple forms and interactive contexts. As XML databases and server technologies improve, we can expect further refinements to the techniques we have outlined.

The investigation of human computer interaction for information-rich 3D worlds and visualizations is still in its infancy. We expect that by enumerating effective data mappings, the combinations of coordinated information and media types and interaction strategies for information-rich virtual environments, we can work toward advantageous computational, compositional and convivial systems for real-time exploration, analysis and action. This work will have a direct impact on the usability and design of such heterogeneous 3D worlds. With such mappings, coordinations and strategies in hand, effective displays and user interfaces may be automatically generated or constructed by users depending on the expertise level and the task. The coming years hold great potential to amplify the bandwidth between interactive computer graphics technologies and human understanding.

7.7 Acknowledgements

Screenshots are VRML views of the result X3D scenes through the ParallelGraphics Cortona browser. X3D syntax was validated within X3DEdit 2.4. Thanks are due to Dr Doug Bowman, Dr Christopher North, Scott Preddy and the Virginia Tech Visualization and Animation Research (UVAG) Laboratory for their continued support and review of this chapter.

Most thanks are due to my dear friend, advocate and wife, Kat Mills.

References

Ahlberg C and Wistrand E (1995) IVEE: an Information Visualization and Exploration Environment. In *Proceedings of IEEE InfoVis*, 66–73, 142–143. Spotfire: www.spotfire.com

Ames AL, Nadeau DR and Moreland JL (1997) *VRML Sourcebook*, 2nd edn. New York: Wiley.

Apache Foundation (2002) Apache Web server: http://www.apache.org. Cocoon: http://xml.apache.org / cocoon/. Hypertext Preprocessor: http://www.php.net. Tomcat: http://jakarta.apache.org/tomcat/ index.html.

Bertin (1981) *Graphics and Graphic Information Processing* (transl. Berg W and Scott P). Berlin: Walter de Gruyter.

Bolter J, Hodges LF, Meyer T and Nichols A (1995) Integrating perceptual and symbolic information in VR. *IEEE*, July.

Bowman D, Koller D and Hodges L (1997) Travel in immersive virtual environments: an evaluation of viewpoint motion control techniques. In *Proceedings of the Virtual Reality Annual International Symposium (VRAIS)*, pp. 45–52.
Bowman D, Hodges L and Bolter J (1998) The virtual venue: user–computer interaction in information-rich virtual environments. *Presence: Teleoperators and Virtual Environments*, **7**(5)8, 478–493.
Bowman D, Wineman J, Hodges L and Allison D (1999) The educational value of an information-rich virtual environment. *Presence: Teleoperators and Virtual Environments*, **8**(3), 317–331.
Bowman D, North C, Chen J, Polys N, Pyla P and Yilmaz U (2003) Information-rich virtual environments: theory, tools, and research agenda. In *ACM Symposium on Virtual Reality Software and Technology (VRST)*.
Brown M (2002) *XML Processing with Perl, Python, and PHP*. San Francisco: Sybex.
Card S, Mackinlay J and Schneiderman B (1999) *Information Visualization: Using Vision to Think*. San Francisco: Morgan Kaufmann.
Cleveland W (1993) *Visualizing Data*. Summit, NJ: Hobart Press.
Cleveland WS and McGill R (1994) Graphical perception: theory, experimentation and application to the development of graphical methods. *Journal of the American Statistical Association* **79**, 387.
Dachselt R and Rukzio E (2003) Behavior3D: an XML-based framework for 3D graphics behavior. In *Proceeding of the Web3D 2003 Symposium*, ACM SIGGRAPH.
Dachselt R, Hinz M and Meissner K (2002) CONTIGRA: an XML-based architecture for component-oriented 3D applications. In *Proceedings of the Web3D 2002 Symposium*, ACM SIGGRAPH.
Dos Santos CR, Gros P, Abel P, Loisel D, Trichaud N and Paris JP (2000) Mapping information onto 3D virtual worlds. *In Proceedings of IEEE International Conference on Information Visualization*, London, 19–21 July 2000.
Foley JD, van Dam A, Feiner SK and Hughes JF (1995) *Computer Graphics: Principles and Practice in C*, 2nd edn. Boston: Addison-Wesley.
Friedhoff R and Peercy M (2000) *Visual Computing*. New York: Scientific American Library.
Gnome XML and XSLT Libraries for Perl. Available: http://www.gnome.org
Hibbard W, Levkowitz H, Haswell J, Rheingans P and Schoeder F (1995a) Interaction in perceptually-based visualization. In Grinstein G and Levkoitz H (eds), *Perceptual Issues in Visualization*. New York: Springer, pp. 23–32.
Hibbard W, Dyer CR, Paul BE (1995b) Interactivity and the dimensionality of data displays. In Grinstein G and Levkoitz H (eds), *Perceptual Issues in Visualization.* New York: Springer, pp. 75–82.
Kay M (2001) *XSLT*, 2nd edn. Birmingham: Wrox Press.
Keller PR (1993) *Visual Cues: Practical Data Visualization*. Piscataway, NJ: IEEE Computer Society Press.
Kim T and Fishwick P (2002) A 3D XML-based customized framework for dynamic models. In *Proceedings of the Web3D 2002 Symposium*, ACM SIGGRAPH.
Mackinlay J (1986) Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, **5**, 111–141.
McLaughlin B (2001) *Java and XML*, 2nd edn. Cambridge: O'Reilly.
Norman DA (1986) Cognitive engineering. In Norman DA and Draper SD (eds), *User Centered System Design.* Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 31–61.
North C and Schneiderman B (2000) Snap-together visualization: can users construct and operate coordinated views? *International Journal of Human – Computer Studies*, **53**, 715–739.
Perl Mongers: http://www.perl.org
Pickett RM, Grinstein G, Levkowitz H, Smith S (1995) Harnessing preattentive perceptual processes in visualization. In Grinstein G and Levkoitz H (eds), *Perceptual Issues in Visualization***.** New York: Springer.
Polys NF (2003) Stylesheet transformations for interactive visualization: towards a Web3D chemistry curricula. In *Proceedings of the Web3D 2003 Symposium*. ACM SIGGRAPH.
Schneiderman B (1996) The eyes have it: a task by data type taxonomy for information visualizations. *Proceedings of IEEE Visual Languages*, 336–343.
Sun Microsystems Java and Java Server Pages. Available: http://java.sun.com/products/jsp/
Tufte E (1990) *Envisioning Information*. Cheshire, CT: Graphics Press.
Walsh A and Sévenier M (2001) *Core Web3D*. Upper Saddle River, NJ: Prentice-Hall.
White C (2002) *Mastering XSLT*. San Francisco: Sybex.

The Web3D Consortium (2002)
Specifications:
Extensible 3D (X3D-ISO/IEC 19775:200x), Virtual Reality Modelling Language (VRML- ISO/IEC 14772:1997): http://www.web3d.org/fs_specifications.htm

X3D TaskGroup and X3DEdit: http://www.web3d.org/x3d.html
Software Development Kit: http://sdk.web3d.org
Xj3D Open Source X3D/VRML toolkit: http://www.web3d.org/TaskGroups/source

The World Wide Web Consortium (2002)
Specifications:
Extensible Markup Language (XML): http://www.w3.org/XML
Extensible Stylesheet Transformations (XSLT): http://www.w3.org/TR/xslt11