xml2vrml

A Library for Easy 3D Graphing

William Baucom

Department of Computer Science, Furman University

william.baucom@gmail.com

ABSTRACT

A dearth exists in the realm of easy-to-use 3D graphing solutions for the web. Many have used XML to generate VRML in the past, but these implementations have suffered from a very narrow focus and aging web scripting technologies. This paper details a modular approach to XML graph definition that incorporates customizable templates for power users while abstracting the complexities of VRML generation for less savvy users.

INTRODUCTION

As data sets grow larger and larger with the advent of increasingly complex data aggregation techniques, deriving useful information from visual representations can be challenging. Two-dimensional representations can be useful for conveying limited amounts of information, but adding a third dimension to data display gives that representation a new level of descriptive power. It has been shown that 3D representations of graphs that provide depth cues can enable an unskilled observer to see and interpret paths between nodes in graphs containing as many as 333 nodes, while observers who spent more time with the experiment could work with as many as 1000 nodes. The increased capacity for conveyance of useful information is increased by orders of magnitude when a third dimension of data is incorporated[1]. In today's world of waning attention spans, a flashy, three-dimensional graph is sure to provide a longer-lasting, more useful impression than its two-dimensional counterpart. Creating multi-dimensional representations of data has been a trend in computer science for ten years now, yet applications produced over those years have generally been very focused, narrow implementations that are designed to take a particular type of data, use a particular processing language for formatting, and display that data using a proprietary descriptive three-dimensional language. These applications have failed to embrace the trend in web-based solutions toward open standards and extensible implementations. The aim of this project is to provide a simple, XML-based solution for describing sets of data in three dimensions and to offer an implementation of a parser in a common, open-sourced language that can be distributed freely and used to create visually stimulating graphical representations of data.

BACKGROUND AND PREVIOUS WORK

The Virtual Reality Modeling Language has its origins in the early days of the web3d movement. Established as an ISO standard in

1997, VRML was created to facilitate the description of three dimensional scenes that would be rendered using plug-ins to common browsers. It was the standard for three dimensional web modeling until Extensible 3D (X3D) superseded it, beginning in 2004[2]. The X3D movement has been slow to catch on due to the entrenchment of VRML code and the backwards-compatibility of X3D rendering agents. Therefore, it was considered appropriate to maximize usefulness and backwards compatibility of this project by implementing a VRML-based parser rather than an X3D parser.

VRML is a highly structured, deeply nested, and unwieldy language. All objects within a scene are given explicit descriptions that often require multiple containers for placement, coloration, texture application, and interconnection. Only specific properties of objects are exposed to the coder, and deviance from the code structure often results in complete program failure. Thus, implementation of VRML code is an endeavor best left to either a skilled worker or a machine – hence the usefulness of this project.

XML is the Swiss army knife of the Internet. Though some contend that the hype surrounding XML technologies is excessive, the sheer market permeation it exhibits is proof that the disgruntled few are just that – very few. Technically, XML is a data format specification. It imposes a strict tree-based structure on data, which facilitates parsing and using that data. In a broader sense, XML represents the overwhelming trend in computing to extract form from function, representation from underlying data, and fluff from meat. XML encoding offers no presentational markup. In fact, it specifically avoids anything other than giving meaning to content. XML mirrors the SGML prototype—enclosing tags within angle brackets—but in contrast to fluid languages like HTML, imposes a strict standard of case-sensitivity, matched tags, and well formed attributes. However, beyond these code form issues, XML offers no direction. Herein lies the elegance of the language: the user defines his own set of tags that best describe his own data. Yet, because the structure of those tags is very rigid, an abundance of libraries, programs, and interfaces have been created to utilize and manipulate that data. For this project, XML was the natural choice for an easy to use, descriptive language because of its widespread compatibility. XML is also an excellent choice because, unlike other SGML-based languages, it does not require a Document Type Definition. Rather, code can be validated by error checking in the parser[3].

The use of VRML with XML is not a new idea. One early blending of these technologies was described by Arun and Ganguly in their paper on Chemistry's applications of the VRML-XML nexus. Their subject was the visual display of molecular data, which was already encoded in XML-like form. Chemists were using the Chemical Markup Language while XML was still in its infancy, and transitioning this data to a VRML representation was an excellent solution for visualizing the complex nature of chemical

interactions[4]. XML played the role of value-adder in this case, taking the basic CML data and extending it to include information for representing movement and interaction between atoms. This early implementation reveals the limitation characteristic of many others to follow: the language of implementation is no longer the preferred coding medium for the web. Using the XML::Parser interface to the C expat library, the Perl script implementation of this application is no longer common or easily maintained. Thus, this implementation offers little value to current and new users.

A second intriguing implementation of an XML to VRML tool is Hong Kong Polytechnic University's *Dynamic Structuring of Web Information for Access Visualization*. Researchers attacked the problem of the endless link hierarchy on the web to find relevant data from sites. Referencing the issue of limited bandwidth and screen space for mobile browsers, these researchers endeavored to encapsulate the relationships between pages of a site in XML documents. These so-called Document Cluster Graphs are then fed into different visualization formats based on the request and capabilities of the user's browser. Often, this means that the relationships are translated into three-dimensional models of the cluster using VRML and a mobile-browser compatible plug-in. These VRML views consist of collections of spheres and cylinders that represent the nodes (web pages) and links between them, respectively. Incoming and outgoing links are color-coded, and can be weighted according to click counts[5]. Data parsing is handled in this implementation using the Oracle XML Parser for C++ and C++ libraries. Interestingly, the conversion to VRML is generally done on the end-user's device. Here again, in addition to the potential issues of offloading processing to the user, this project suffers the limitation of a proprietary, Microsoft-owned language.

PROJECT MOTIVATIONS

There is a dearth of 3D libraries in currently popular and relevant web scripting languages, and it is hoped that this library can be made available under an open-source license and improved and extended by others. This project intends to make the third dimension available to the masses by providing an easy way to structure data while encapsulating the verbose and difficult VRML coding.

XML2VRML IMPLEMENTATION

As previously discussed, the xml2vrml library is implemented in PHP. As a language, PHP bridges the gap between the total lack of structure embodied by most scripting languages and the object-oriented, strongly-typed structure of Java. The result is a hybrid that allows the declaration and assignment of completely type-less variables for easy learning and use by new coders, yet still provides for type hinting and class-based scope definitions for use by more serious developers.

This was the author's first foray into the structured side of PHP and it was quite a success in that regard. PHP is kind enough to follow

many of Java's trailblazing conventions, such as implementing a garbage collector instead of requiring manual disposal of objects. Also, access controls for classes are specified in the manner of Java. One frustrating difference is the deviation from the dot-notation of variable and method access. Because the '.' is already used in PHP as the concatenation operator, the character sequence '->' was instead chosen to access class properties. In the same vein, PHP strangely requires that the '$this' keyword be used when accessing properties of the class from within the class's methods. While this certainly makes scope clearer, it is a rather annoying change to make for someone who was trained in Java, where the 'this' keyword is often optional. Finally, the declaration of classes is not required to follow file-naming conventions. While it is convenient to declare an exception class in the same document that uses it, even the possibility of holding all your classes in one text file is a frightening prospect for any Java coder worth his salt.

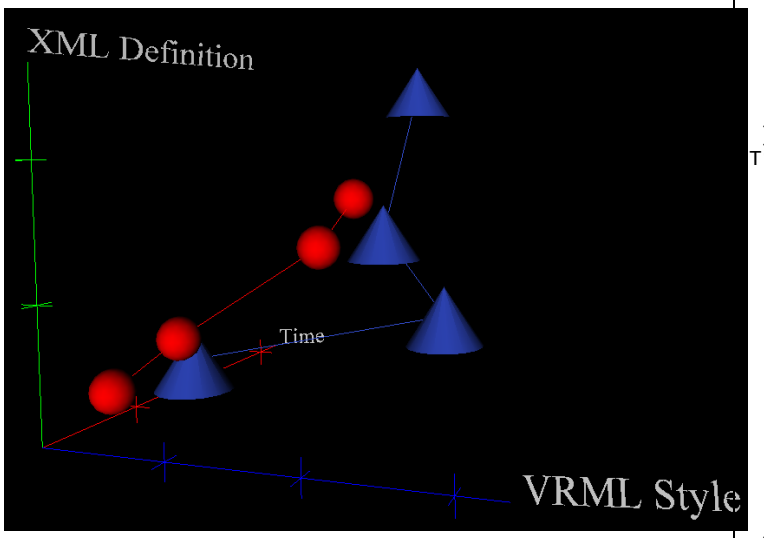
The most important aspect of the PHP class system to the success of this endeavor is the SimpleXML parser library. SimpleXML is quite aptly named, and allows an XML document to be loaded from a string or file with a single command. The resulting SimpleXML object is a composite of many more SimpleXML objects which hold all of the original object's attributes and children. This hierarchy continues through the XML document, encapsulating all of the data into iterable objects. For example, the following code snippet accesses the child elements of the <points> element which is a child of the <options> element, which is a child of the <graph>, or root element.

```
if( count(
    $this->xml->options->points->children()) > 0)
        foreach ( $this->xml->options->points-
              >children() as $optn)
        {
          $psObj->setAttribute($optn-
                    >getName(), $optn);
        }
```

Though it seems verbose, this is actually a very intuitive way of thinking about the XML data structure that few other libraries utilize. The children() function returns an array that can be iterated over, pulling out each element as `$optn` in turn and using that reference to set an attribute of another object.

The class structure of the xml2vrml library is an abstraction of a graph's properties, and it mirrors the XML declaration's structure. An Axis class encapsulates the minimum and maximum value of each axis, the tick mark attributes, the color, and the axis's label. The axes are accumulated in an Axes class which merely holds an array of the Axis class. For declaring the data to be graphed, Point and PointSet classes were used. The Point class offers an encapsulation of the x, y, and z coordinates of a particular datum, and the PointSet class aggregates Points and defines the color, shape, size, plot type, and name of a particular set of data. Breaking down groups of data into PointSets provides the ability to represent multiple themes on the same visual, providing a facility for comparison of trends. The figure below demonstrates two PointSets of random data, showing two different point shapes

and colors, with both using line graphs rather than scatter plots.



Each class shares the responsibility of producing its verbose VRML code with a template loaded using the CodeTemplate class. The CodeTemplate class loads a static template file into memory, then uses PHP's string manipulation functionalities to replace flags in the template, defined as `{_VARIABLE_NAME_}`, with the appropriate code, whether that be the point location, the axis color, or the array of numbers that define the line graph. Thanks to these templates, code that was as short as

```
<x>
  <color>red</color>
  <min>0</min>
  <max>10</max>
  <label>Time</label>
</x>
```

in XML now expands to:

```
Shape { # start x axis definition
   geometry IndexedLineSet {
     coord Coordinate {
       point [
            0 0 0, 10 0 0,
            3 -0.4 0, 3 0.4 0, 3 0 -0.4, 3 0 0.4,
            6 -0.4 0, 6 0.4 0, 6 0 -0.4, 6 0 0.4,
            9 -0.4 0, 9 0.4 0, 9 0 -0.4, 9 0 0.4
       ]
     }
     coordIndex [
          0, 1, -1,
          2, 3, -1,
          4, 5, -1,
          6, 7, -1,
          8, 9, -1,
          10, 11, -1,
          12, 13, -1
     ]
     color Color { color 1 0 0 }
     colorIndex [ 0 0 0 0 0 0 0  ]
     colorPerVertex FALSE
   }
 } # end Shape
Transform { # label section
   translation 10.2 0 0
   children Billboard {
     axisOfRotation 0 0 0
     children Shape {
       appearance Appearance {
         material Material {}
       }
       geometry Text {
         string "Time"
         fontStyle FontStyle {
           size 0.8
         }
       }
     }
   }
 }
```

in the full VRML. The Axis class accepts simply a color, minimum, maximum, and label value from the XML document. With this, it uses the tick mark interval and width to plot the array of points starting on line 4 of the code above. It then links these points to make lines using the "coordIndex" block just below. The color is specified and mapped to each of those line segments created by the coordIndex, and finally, the label is dropped into the Text feature of the "billboard transform". This template and encapsulation scheme accomplish the principle goal of this project—to provide an easy to use, yet extensible graphing solution. Only a basic knowledge of XML is required to use this library, yet a skilled VRML coder can manipulate the template files to unlock additional functionality.

APPLICATION OF THE PRODUCT

Without an application to show off its features, a library like this one is little more than a nice idea. In this case, we looked to the Timing

Analyzer project run by Dr. Chris Healy. Having seen the development of a web interface for this program the previous summer, the author found this programs output data exceptionally hard to understand and assimilate. The original data output from this project was a table with many columns of data. To make this more accessible to those untrained in reading this data, the author undertook the task of converting the tabular data to an XML format that could be read into a three dimensional graph. This allows analysis of multiple variables at the same time while reducing the sheer volume of data presented.

The first challenge in porting this web interface was to generate data that varied across multiple parameters. Because of the structure of the underlying program and of the web interface, it was impractical to get all of the data in a single page load. Therefore, a refreshing system was implemented, using query string variables and the <meta http-equiv="refresh"> header. The page refreshes once per second and gets the next iteration of data. When this data is fully generated, the user has the option to download the XML file for later viewing, or to immediately send that XML to the xml2vrml parser.

A second important challenge in creating a useful graph is the need to appropriately size the axes. The raw data produced by the timing analyzer has a enormous range of values. One axis might vary from .20 to .55, while another may vary from 2 to 256. Since the VRML code interprets all points literally, this meant that one axis stretched past the edges of the screen while the other was completely invisible. With the axes that far out of proportion, the data points become meaningless. Therefore an algorithm had to be devised to force the axes to lie within a reasonable range of each other. The following is code for the algorithm the author developed:

```
$try = 0; // initialize temporary maximum
$prevdiff = SCALE_VARIANCE; //variance threshold

for($i = 2; ; $i++) {
   if ($trueMax < $target_max)
               $try = $trueMax * $i;
   else
               $try = $trueMax / $i;
   $tempdiff = abs($try - $ target_max);
   if ($tempdiff < $prevdiff && $tempdiff <
                        SCALE_VARIANCE) {
               if ($trueMax < $ymax)
                    return $i;
        else
                    return 1 / $i;
   }
   elseif ($tempdiff < $prevdiff)
               $prevdiff = $tempdiff;
   else  // overshot ideal number
      return ($trueMax < $ymax) ? $i-- : 1 / $i--;
}
```

Starting with i=2, the algorithm scales up (by i) or down (by 1/i) towards the target until the absolute value of the difference between the two is within a predefined range. This has worked quite successfully and is important to the future work.

FUTURE WORK

Although the project was a success, the interface layer between the application and library layers contains more functionality than it should. Currently, the maximum and minimum for the axes values are determined using database queries from the timing analyzer's data cache. Also, the scaling algorithm is currently incorporated into this transitional layer. It would be much more in line with the projects goal of an easy-to-use interface if the scaling algorithm was incorporated directly into the XML parser. In this way, the

user would just feed the raw data into the XML input, rather than having to pre-scale it as is currently the case. Also, the determining of the minimum and maximum values should be left to the PointSet class, again so that the burden of providing this data is only on the user if they choose to set them specifically. A future revision of this parser should make the axis min & max parameters optional and automatically scale the input data and axis values.

Additionally, the data pulled from the timing analyzer is apparently not the most important and relevant data. Based on the author's conversations with the implementer of the original web interface, the execution time was assumed to be the data of interest. This is not the case, and future implementations should instead look at the simulation data, as this provides the most relevant and interesting information.

CONCLUSIONS

This project set out to create an easy interface for unleashing the power of three-dimensional graphing to capture peoples' interest. It has successfully accomplished that goal. A simple, flexible XML schema combined with plug and play library access (its as easy as this:)

```
$gr = new XMLGraph($xml);
$gr->publish();
```

make this a worthwhile tool for data visualization. Robust error checking and exception handling make using this tool easy for newcomers to PHP and to web scripting in general, and there is exciting potential for future development.

---

[1] C. Ware and P. Mitchell, 2005. *Reevaluating stereo and motion cues for visualizing graphs in three dimensions.* In *Proc of the 2nd Symposium on Applied Perception in Graphics and Visualization* (A Coroña, Spain, August 26 - 28, 2005). APGV '05, vol. 95. ACM, New York, NY, 51-58.

[2] The web3d Consortium. *X3D Standards and FAQs.,* http://www.web3d.org/x3d/specifications/

[3] T. Usdin and T. Graham, 1998. *XML: not a silver bullet, but a great pipe wrench.* StandardView 6, 3 (Sep. 1998), 125-132.

[4] Peter Murray-Rust, Henry Rzepa, & Christopher Leach. *Chemical Markup Language (CML),* http://www.ch.ic.ac.uk/cml/

[5] J. Y. Mak, H. Va Leong, and A. T. Chan, 2002. *Dynamic structuring of web information for access visualization.* In Proc of the 2002 ACM Symposium on Applied Computing (Madrid, Spain, March 11 - 14, 2002). SAC '02. ACM, New York, NY, 778-784.

Total word count: 3006