Web Software Visualization Using Extensible 3D (X3D) Graphics

Craig Anslow, James Noble, Stuart Marshall*
Victoria University of Wellington, New Zealand

Robert Biddle†
Carleton University, Canada

Abstract

3D web software visualization has always been expensive, special purpose, and hard to program. Most of the technologies used require large amounts of scripting, are not reliable on all platforms, are binary formats, or no longer maintained. We can make web software visualization of object-oriented programs cheap, portable, and easy by using Extensible (X3D) 3D Graphics, which is a free open standard. In this paper we outline our experience with X3D and discuss the suitability of X3D as an output format for software visualization.

CR Categories: D.2.6 [Programming Environments]: Graphical Environments—;

Keywords: Software Visualization, Extensible 3D Graphics

1 Introduction

The Web3D Consortium's Extensible 3D (X3D) Graphics standard [Brutzman and Daly 2007] provides a technology for deploying interactive 3D graphical content over the web. This technology has many possible applications and one we are interested in is deploying software visualizations over the web.

Our current work in developing a web-based software visualization architecture [Marshall et al. 2001] requires a technology that can deliver visualizations over the web. For this reason we decided to evaluate X3D to see if it is a viable graphics technology for use in the development of software visualizations. We have created a prototype tool for creating X3D software visualizations [Anslow et al. 2006b], described techniques for visualizing software with X3D [Anslow et al. 2007], and presented preliminary results of evaluating X3D [Anslow et al. 2006a]. In this paper we present our final evaluation results for using X3D in software visualization.

2 Evaluation of X3D

We want to see if X3D can support a range of 3D software visualization techniques to determine if the technology is viable for use in software visualization. More precisely we want to experiment with automatically creating X3D software visualizations over the web, evaluate X3D's animation and interactivity aspects, examine the text, layout, and extensibility features, test the integration capabilities, and analyse the performance display capabilities of X3D.

Figure 1 shows bubble, selection, and insertion sort algorithms all animating at once. The combined views allow a user to see both the current state of the array and the history of an algorithm's execution. The animation replicates a similar example by [Najork and Brown 2001].

Figure 2 shows a documentation related visualization that has a 3D UML class diagram of a Java program and the associated

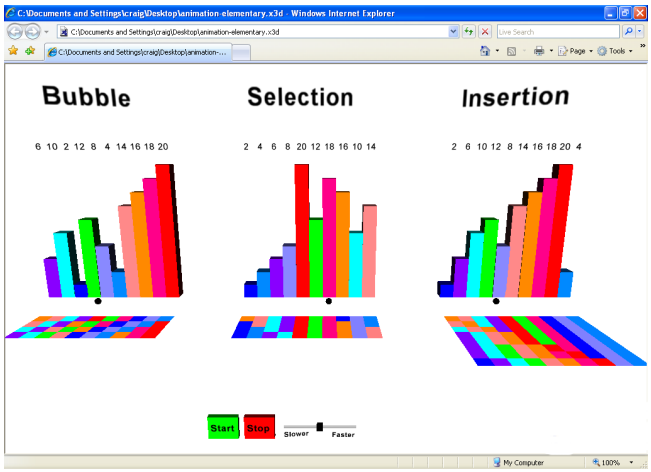

Figure 1: *X3D Elementary Sorting Algorithm Animation.*

Javadoc. When a user clicks on a class or package in the visualization in the left frame the associated Javadoc entity is displayed in the right frame. The UML diagram replicates a similar example by [McIntosh et al. 2005]. We now discuss our evaluation results.

Design: X3D is a free open standards file format and run-time architecture to represent, communicate, and deploy 3D scenes and objects over the web using XML. The X3D specification is comprised of components which contain nodes (e.g. geometry) that are declared in a scene graph. Content can be created using text editors, X3D editors (e.g. X3D-Edit a Netbeans plug-in), digital content creation tools (3DS Max, Maya, Blender), or XSLT transformations. There are language bindings to ECMAScript and Java. We used three of the main X3D browser free version implementations including (in order of preference): BS Contact VRML/X3D Player (6MB download), Octaga Player (5MB), and Flux Player (1.5MB). Each of these X3D browsers operate on Windows and can be plugged into Mozilla Firefox and Microsoft Internet Explorer or operate as stand alone. There is also the Web3D Consortium's stand-alone open source test-bed implementation Xj3D (12MB). X3D content can be rendered in either OpenGL or DirectX. Some of these X3D browsers do not implement all of the X3D specification nor do they make the Scene Access Interface (SAI) run-time API available. This makes it hard for developers to create consistent X3D software visualizations.

Graphical Capability: A rich set of graphical elements exist to create high quality visual pictures required in software visualizations. Points and lines can be implemented using nodes from the 2D geometry component, and areas and volumes from the 3D geometry component. Shapes have size, height, radius, colour, and transparency fields, and can be animated to change in a software visualization. Shapes can be orientated in any order (e.g. translated then rotated) and change position during a software visualization. When nodes that are connected in a graph visualization are moved in a scene, scripting is required to preserve the node-link relationships. Text can also be displayed using the shape node. Lighting

*E-mail: {craig,kjx,stuart}@mcs.vuw.ac.nz
†E-mail: robert_biddle@carleton.ca

SOFTVIS 2008, Herrsching am Ammersee, Germany, September 16–17, 2008.
© 2008 ACM 978-1-60558-112-5/08/0009 $5.00

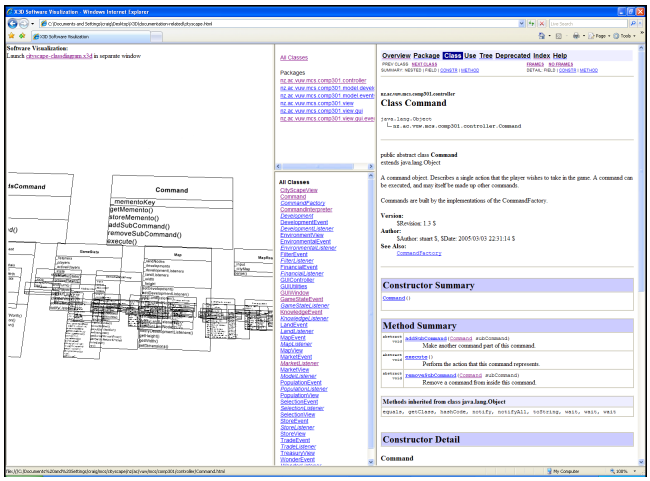

Figure 2: *X3D UML Class Diagram showing 100 classes integrated with Javadoc.*

(directional light) and environment (background) effects can be applied to a scene. Textures can be applied to shapes using images (.png, .gif, and .jpg), sound (.wav, .mp3), or video (.mpg). Sound is used in Figure 1 to signify ordering of elements. Videos have been used for providing additional information in our UML diagrams when an entity is selected. Node prototyping can be used to extend X3D. Developers specify node prototypes and then use prototype instances which are then mapped to geometry nodes.

Performance: Using our prototype tool [Anslow et al. 2006b] our smaller XML execution traces take less than a few seconds[1] to generate a X3D software visualization and our larger traces (10-50MB) take less than two minutes to produce 10K-100K nodes. The longest time spent in creating our web software visualizations is the rendering of the X3D scenes rather than the stylesheet transformation. It takes less than 10 seconds to render about 10K nodes, three minutes for 50K nodes, but up to 10 minutes to render 100K nodes. The size of the files ranged from less than 100KB for our small visualizations to 10-18MB for our large visualizations. We converted the visualizations to the X3D binary format which reduced the files by about 75%.

Visualization Techniques: There is no specific software visualization component or library. We replicated a range of software visualization techniques in X3D including algorithm animations (Figure 1), 3D UML diagrams (class, package, and sequence diagrams), documentation related visualizations (source code, API Javadoc – Figure 2, and video visualizations), and execution trace visualizations (3D compound shapes and 3D information visualization metaphors). The data for our visualizations have been encoded using three different approaches: in the X3D scene, transformed from XML execution traces into X3D geometry primitives, and as node prototype instances. Our visualizations can display Java and C++ programs. X3D can be used to represent program synchronization (Figure 1). Multiple views can be accomplished by displaying the data in different positions in the scene (Figure 1) or integrating external web pages (Figure 2).

Animation: X3D relies too heavily on the routing event model (Figure 3) for animation. ROUTE directives are used between each event in the model. A ROUTE directive takes input from one node's field and outputs values to another node's field (e.g. timer to a position interpolator then to a geometry node such as a cube). The design of the model is very cumbersome as ROUTE directives require the explicit name of each node as opposed to an instance based model for object-oriented methods.

[1]Performance transformation and rendering timings of both client and server were measured using a Dell 610 laptop with Windows XP and 512MB of RAM.

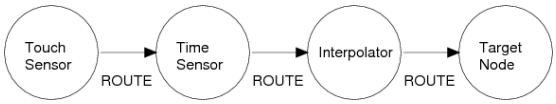

Figure 3: *X3D Animation Routing Event Model.*

User Control and Navigation: X3D supports basic user controls such as start and stop buttons for temporal control in algorithm animations (Figure 1). No specific software visualization user controls exist in X3D. More complicated controls such as speed, pause, fast forward, rewind, and step in algorithm animations require the use of scripts. X3D has very good 3D user navigation support for software visualization including the follwing techniques: walk, examine, fly, look-at, slide, and pan. A user can change the navigation type, the speed of navigation, and viewpoint at run-time using the X3D browsers' user control menus.

User Tasks: Users can gain an overview of the entire software visualization if a viewpoint is defined that contains the whole data set. Users can zoom using the navigation controls or by selecting a pre-defined viewpoint. Filtering can be achieved using the boolean filter field or changing the transparency, scale, or size values of a node once a user clicks or moves a user control. Details-on-demand can be provided through the level-of-detail or switch nodes which displays additional information about a node once a user clicks or moves within a certain distance from a node in the scene. Showing relationships among items in a visualization requires the use of scripting. There is no built in support for creating a history of user actions nor extracting sub-collections of information.

In this paper we have reported on our experience with X3D for use in software visualization. Our detailed evaluation can be found elsewhere [Anslow 2008]. In summary, the advantages of X3D for software visualization are rich graphics, extensibility, and XML integration. The disadvantages of X3D are lack of software visualization user controls, a primitive animation model, and weak support for filtering and layout. Nonetheless we encourage software visualization developers to adopt X3D if they need 3D for the web.

References

ANSLOW, C., MARSHALL, S., NOBLE, J., AND BIDDLE, R. 2006. Evaluating X3D for use in software visualization. In *Proc. of SOFTVIS*, 161–162.

ANSLOW, C., MARSHALL, S., NOBLE, J., AND BIDDLE, R. 2006. VET3D: a tool for execution trace web 3D visualization. In *Companion to OOPSLA*, 655–656.

ANSLOW, C., MARSHALL, S., NOBLE, J., AND BIDDLE, R. 2007. X3D software visualisation. In *Proc. of NZCSRSC*.

ANSLOW, C. 2008. *Evaluating Extensible 3D (X3D) Graphics For Use in Software Visualisation*. Master's thesis, VUW.

BRUTZMAN, D., AND DALY, L. 2007. *X3D: Extensible 3D Graphics for Web Authors*. Morgan Kaufmann.

MARSHALL, S., JACKSON, K., BIDDLE, R., MCGAVIN, M., TEMPERO, E., AND DUIGNAN, M. 2001. Visualising reusable software over the web. In *Proc. of INVIS*, 103–111.

MCINTOSH, P., HAMILTON, M., AND VAN SCHYNDEL, R. 2005. X3D-UML: enabling advanced UML visualisation through X3D. In *Proc. of Web3D*, 135–142.

NAJORK, M. A., AND BROWN, M. H. 2001. Three-dimensional web-based algorithm animations. Tech. Rep. SRC-RR-170, Compaq Systems Research Centre.