TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

School of Mathematics, Statistics and Computer Science
Computer Science

X3D Web Based Algorithm Animation

Craig Anslow, James Noble, Stuart Marshall, and
Robert Biddle

Technical Report CS-TR-07/1
May 2007

School of Mathematics, Statistics and Computer Science
Victoria University
PO Box 600, Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

School of Mathematics, Statistics and Computer Science
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

X3D Web Based Algorithm Animation

Craig Anslow, James Noble, Stuart Marshall, and Robert Biddle

Technical Report CS-TR-07/1
May 2007

Abstract

3D web software visualization has always been expensive, special purpose and hard to programme. Flash requires proprietary authoring tools and involves large amounts of scripting to produce 3D effects, Java applets are slow to load and are not reliable on all platforms, and VRML is no longer maintained. We can make 3D software visualization cheap, portable and easy by using X3D, which is a lightweight open standard for web 3D graphics. We have replicated three example algorithm animations from Najork and Brown in X3D which unite multiple views, capture the history of execution, and display additional information. We outline our experience, and discuss the suitability of X3D as an output format for algorithm animation software.

Author Information

James Noble is a Professor of Computer Science and Stuart Marshall is a lecturer at Victoria University of Wellington (VUW). Robert Biddle is a Professor in Human Computer Interaction at Carleton University. Craig Anslow is a MSc thesis student at VUW.

X3D Web Based Algorithm Animation

Craig Anslow, James Noble, Stuart Marshall
School of Mathematics, Statistics, and Computer Science
Victoria University of Wellington, New Zealand
{craig,kjx,stuart}@mcs.vuw.ac.nz

Robert Biddle
Human Oriented Technology Lab
Carleton University, Canada
robert_biddle@carleton.ca

Abstract

3D web software visualization has always been expensive, special purpose and hard to programme. Flash requires proprietary authoring tools and involves large amounts of scripting to produce 3D effects, Java applets are slow to load and are not reliable on all platforms, and VRML is no longer maintained. We can make 3D software visualization cheap, portable and easy by using X3D, which is a lightweight open standard for web 3D graphics. We have replicated three example algorithm animations from Najork and Brown [3] in X3D which unite multiple views, capture the history of execution, and display additional information. We outline our experience, and discuss the suitability of X3D as an output format for algorithm animation software.

1 Introduction

Algorithm animation communicates how an algorithm works by graphically displaying its fundamental operations [4] and has been quite useful for education and for research into the design and anlysis of algorithms. Several systems have explored the use of 3D graphics for algorithm animation including Polka3D [5], Zeus3D [1], and JCAT [3]. Our software visualisation system [2] requires a web based technology for representing visualisations.

We can make 3D algorithm animation cheap, portable and easy by using X3D [6] which is the Web3D Consortium's lightweight open standard for web 3D graphics. Najork and Brown [3] identified several reasons for integrating 3D graphics into an algorithm animation system. The third dimension can be used for uniting multiple views of an object, capturing the history of a 2D view, and expressing fundamental information about structures that are inherently 2D. In this paper we present three replicated examples from Najork and Brown [3] and discuss how suitable X3D is for algorithm animation.

2 Elementary Sorting

Figure 1 shows bubble, selection, and insertion sort all executing at the same time in X3D. When an element is sorted the previous state of the array is drawn in the chips view. The animation can help to show the relative performance and the different execution states of the algorithms.

Ronald Baecker introduced the sticks view in *Sorting Out Sorting*. This view shows the array of elements as a row of sticks, where the height of each stick is proportional to the element in the array. Najork and Brown [3] extended the sticks view by creating the chips view which captures the history of execution of algorithms and is drawn in the xy plane at increasing values of z. The combined views allow a user to see both the current state of the array and the history of execution.

We implemented the chips view as an indexed faced set which represents a 3D shape formed by constructing faces (polygons) from vertices. Each face of the indexed faced set has separate colours for each position in the array. We also tried using boxes instead of the indexed faced set for the chips set view but found that they obsured the view of the current animation step. We then tried positioning the boxes behind and below the animation, but found that it was harder to grasp the history of execution.

We used a sphere object to iterate through the array rather than flashing each element because the elements are all different colours and the flashing effect would not have been as effective. We positioned the sphere on top of each element as it iterates but felt that it distracted the user when animating as it constantly changed height for each element. The sphere was moved to below the elements where it could be viewed from any viewpoint.

Text labels were grouped in a billboard node which modifies its coordinate system so that the node's local z-axis turns to point at the viewer. A dashboard was created which has user controls for start, stop, pause, and change the speed of an animation. The buttons were implemented with a mouse touch sensor, while the slider with a plane sensor to detect mouse dragging to change the velocity.

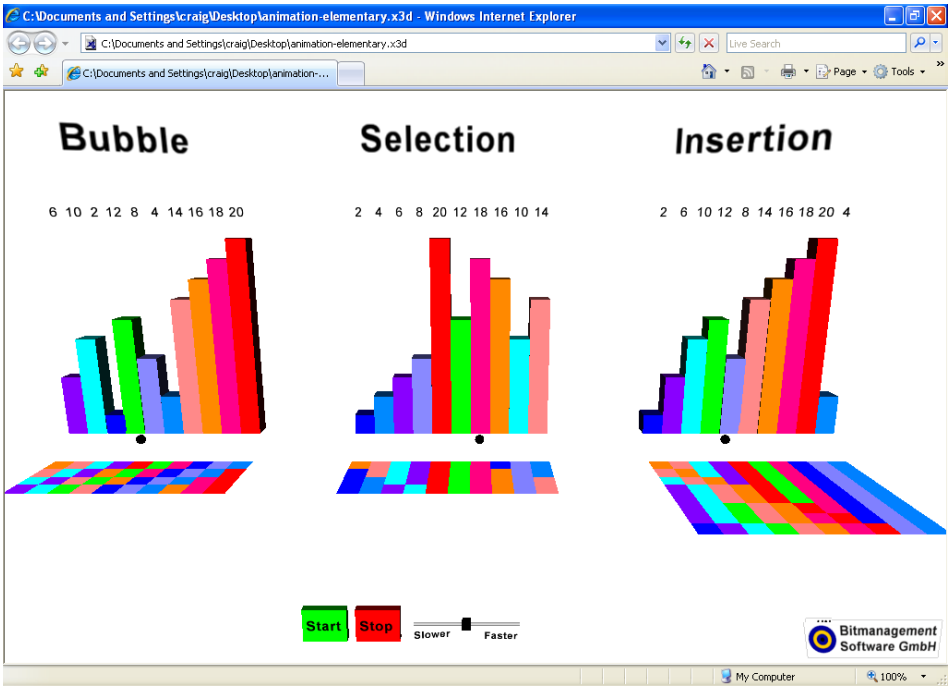

Figure 1. Sorting Algorithm Animation - bubble, selection and insertion sort

3 Heap Sort

Figure 2 shows the heapsort algorithm animation with a united view of the heap and the array in X3D. The top image shows elements in the array being sorted where each element has depth (z dimension) proportional to its value. The bottom image shows one element being flashed a white colour to signify that it has been ordered in the heap and then two elements swapping positions in the heap.

The traditional way of displaying the heapsort algorithm animation is to have two views, one showing the sticks like array and the other the heap. Najork and Brown [3] united the multiple views to alleviate the problem where users must mentally integrate the different views in order to understand the overall algorithm.

Elements are implemented as boxes and paths as cylinders. We originally implemented the paths as 2D lines but they were not visible once the current viewpoint was rotated and were not supported by some browsers. Using colour in this example is not crucial because the value of a stick is encoded by its length, but it is useful to distinguish the elements during the animation. The text labels for the elements could be embedded inside the boxes but would not be visible. A separate text object was created that is offset from the boxes but nested inside a transform grouping node that defines a coordinate system for its children. When the box moves in the heap the text also moves.

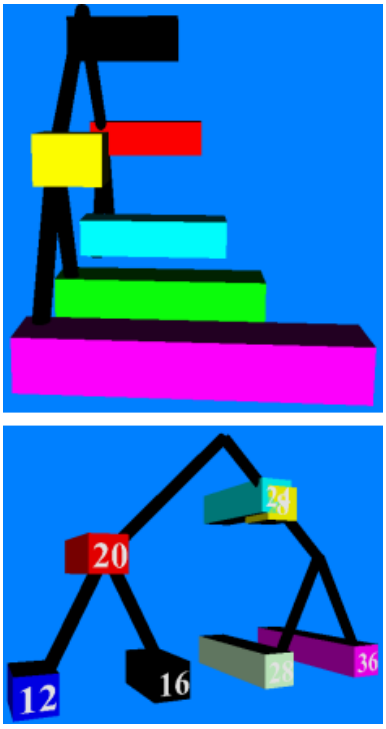

Figure 2. Heapsort Algorithm Animation

4 Shortest Path

Figure 3 shows Dijkstra's shortest path algorithm animation in X3D. The shortest path are green edges while explored edges are purple and unexplored edges are white. The top image shows the completed algorithm where a 2D graph is shown below to help with vertex labelling. The second image shows the first step in the algorithm where the costs from the starting node are noted. At this stage vertex F is unreachable from A.

Najork and Brown [3] extended the animation to provide additional state information about the cost of vertices and weight of edges. Columns are used to represent the cost of getting to each vertice. While edges are linked from the top of one column to the top of the next column along the path.

The columns and edges are implemented as indexed faced sets. When the cost to a vertex changes the height of a column is either increased or decreased in the z dimension by changing four coordinate points. When an edge changes only two coordinate points are adjusted. Coordinate points are moved using coordinate interpolators, while the colour of a column or an edge is controlled by colour interpolators. Boxes and cylinders were not used for columns and edges as adjusting their dimensions, positions, and rotations required extra effort.

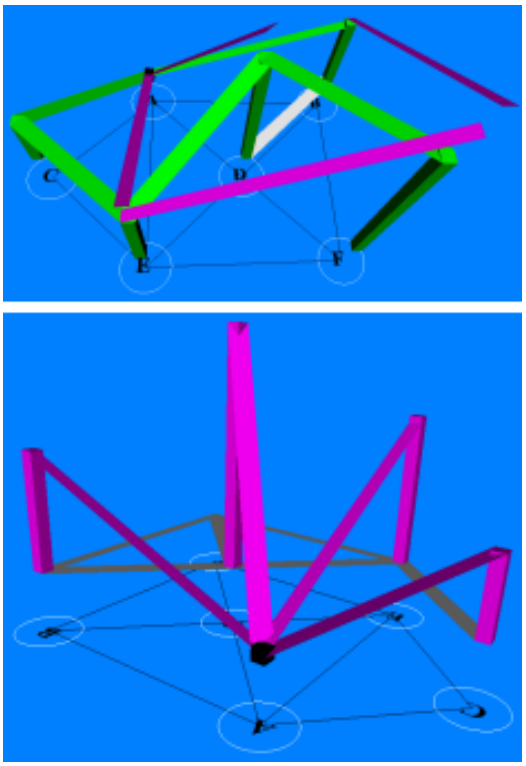

Figure 3. Shortest Path Algorithm Animation

5 Discussion

We were attracted to using X3D for algorithm animation because of its powerful animation routing event model and that it is a lightweight XML web based 3D graphics format. The following code is a sample from the insertion sort algorithm animation that demonstrates the routing event model.

A time sensor is defined to loop and last for 20 seconds. A position and colour interpolator are defined which states at what time in the animation an object moves or changes colour. The box geometry and the appearance of the box are nested within a transform node. Finally, the routing events are the glue which link the timer with the interpolators and then the transform and material objects.

```
<X3D>
<Scene>
<TimeSensor DEF="TIME" cycleInterval="20" loop="true"/>
<PositionInterpolator DEF="MOVE16" key="0.1 0.2
0.3 0.4" keyValue="2 8 0, 4 8 0, 4 8 0, 6 8 0"/>
<ColorInterpolator DEF="FLASH16" key="0.4 0.5 0.6"
keyValue="1 0.5 0, 1 1 0, 1 0.5 0"/>
<Transform  DEF="ITEM16" translation="2 8 0">
 <Shape>
  <Box size="2 16 2"/>
  <Appearance>
  <Material DEF="COLOR16" diffuseColor="1 0.5 0"/>
  </Appearance>
 </Shape>
</Transform>
<ROUTE fromNode="TIME" fromField="fraction_changed"
toNode="MOVE16" toField="set_fraction"/>
<ROUTE fromNode="MOVE16" fromField="value_changed"
toNode="ITEM16" toField="set_translation"/>
<ROUTE fromNode="TIME" fromField="fraction_changed"
toNode="FLASH16" toField="set_fraction"/>
<ROUTE fromNode="FLASH16" fromField="value_changed"
toNode="COLOR16" toField="set_diffuseColor"/>
</Scene>
</X3D>
```

An issue with the X3D routing event model is that for each 3D geometry object that needs to be animated, a separate position interpolator and two route events are required, likewise for the material colour object. In order to create continuous smooth animations we had to create pauses for each object just before they moved. Without the pauses objects that had a long distance to travel in the scene would be moving while other objects are animating, hence causing confusion for the user. This can be seen in the keyValue of the position interpolator where item 16 moves from position 2 8 0 to 4 8 0 and then stays at 4 8 0 for another second.

In the elementary sorting algorithm we used the inline node which can embed an X3D scene stored at a location on the web or local file system. However, for each sorting algorithm we had to physically position the elements in a different place in the 3D world so that they were not mapped onto each other.

We found that when implementing the shortest path animation we wanted objects to appear in the animation and then later disappear. We found no specific node for doing this. Instead, we had to hide objects inside other objects when we didn't require them to be visible.

We implemented the dashboard for user controls as a proximity sensor which generates events when the viewer enters, exits, and moves within a region in space (defined by a box). We found the dashboard to be very useful as it enables a user to rotate the viewpoint and always have the dashboard in front of them to control the animation.

The actual user controls in the dashboard were implemented using touch sensors, plane sensors, a time sensor, and ECMAscript. Start and stop were straightforward and required a route from a touch sensor to the timer. However, for changing the speed and pause, ECMAscript functions were required. The speed control either increased or decreased the time sensor cycle interval depending on which way a user drags the box attached to the cylinder. The pause button required a function call and an additional route for each element in the animation.

We tested our animations on a Dell Latitude laptop running Windows XP with a Intel Pentium 1.86GHz processor and 512MB ram, with free versions of the following browsers: Octaga Player, BS Contact, and Flux Player. The browsers operate as standalone browsers or as plugins for Micrsoft Internet Explorer and Mozilla Firefox. We found that occasionally the X3D browsers displayed a slightly different output for the same X3D file. Sometimes nodes weren't supported or objects in the world animated differently. BS Contact allows a user to capture a screen shot or video which would be useful for a user to diagnose an animation. BS Contact performed the best followed by Octaga and then Flux.

An important aspect in 3D is navigation. X3D allows full 3D navigation where a user can walk, fly, slide, pan, and examine a scene. Users can also look at nodes which allows a user to zoomin up close. The navigation options can be changed dynamically in a browser including the speed of movement and rotation. Some of the browsers use different notations for the same navigation operations. Specific viewpoints can be defined to allow a user to view an animation from a different perspective. However, creating dynamic viewpoints is not possible.

All of our animations were created using a simple text editor. We also used graphical tools such as X3D-Edit and Flux Studio but found they did not meet our expectations as they are designed for non-programmers. X3D-Edit is essentially a graphical tree version of XML and requires lots of mouse clicking to create text. A useful feature of X3D-Edit is that it has support for tool tips and a listing of all the nodes a user could select from to create an X3D world. Flux Studio is a tool for creating physical objects rather than dealing with the low level XML semantics. A nice feature of Flux is that allows a user to see multiple views of the objects when creating a scene and to preview them using the Flux Player. The size of each of the X3D files were less than 50KB and about 500 lines of X3D code.

We found creating our X3D animations by hand to be very time consuming as all data was encoded and physically laid out. Creating animations by separating the data from the presentation using XSLT would be a better solution. However, XSLT is primarily designed for transforming XML documents into other XML documents and would not be a suitable candidate for implementing complex layout algorithms. To implement complex layout algorithms with X3D we would recommend using the open source X3D Java implementation Xj3D.

6 Conclusion

We have replicated three examples from Najork and Brown [3] to see if X3D can graphically represent their reasons for integrating 3D graphics into algorithm animations. Of the examples we implemented X3D performs well at uniting multiple views (heap sort) and capturing the history of execution (elementary sorting), and not so well for displaying additional information (shortest path). The routing event model is the critical feature to create an X3D animation, however it does not scale well once there are more than 100 objects. In the future we will create a web application where users can select the type of animation and specify their own data set. We would also like to explore X3D's audio and video capabilities to reinforce animations.

References

[1] M. H. Brown and M. A. Najork. Algorithm animation using 3D interactive graphics. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pages 93–100. ACM Press, 1993.

[2] S. Marshall, K. Jackson, R. Biddle, M. McGavin, E. Tempero, and M. Duignan. Visualising reusable software over the web. In *Proceedings of the Australasian Symposium on Information Visualisation (INVIS)*, pages 103–111. Australian Computer Society, Inc, 2001.

[3] M. A. Najork and M. H. Brown. Three-dimensional web-based algorithm animations. Technical Report SRC-RR-170, Compaq Systems Research Centre, 2001.

[4] J. Stasko, M. Brown, and B. Price, editors. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, 1998.

[5] J. T. Stasko and J. F. Wehrli. Three-dimensional computation visualization. In *Proceedings of the IEEE Symposium on Visual Languages (VL)*, pages 100–107. IEEE Computer Society Press, 1993.

[6] Web3D-Consortium. X3d specification, 2004. http://www.web3d.org/x3d/specifications/.