X3D Software Visualisation

Craig Anslow[1], Stuart Marshall[1], James Noble[1], and Robert Biddle[2]

[1] School of Mathematics, Statistics, and Computer Science,
Victoria University of Wellington, New Zealand
[2] Human Oriented Technology Laboratory, Carleton University, Canada
{craig,stuart,kjx}@mcs.vuw.ac.nz

Abstract. We have a software visualisation architecture that requires tools to develop visualisations from XML execution traces and integrate the visualisations into user's web environments. Most existing web software visualisation systems create 2D visualisations and if they do use 3D they are using technologies that are outdated, not designed for the web, and hard to extend. We are building a tool that transforms XML execution traces into X3D – the Web3D Consortium's open standard for web 3D graphics – web enabled visualisations and exploring how suitable X3D is for use in software visualisation. Our tool and visualisations will help developers to understand the structure and behaviour of software for reuse, maintenance, re-engineering, and reverse engineering.

Key words: Software Engineering, Software Visualisation, Graphics, X3D

1 Introduction

Software visualisation is the use of the crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software [1]. In a recent survey [2] based on questionnaires completed by 111 researchers from software maintenance, re-engineering and reverse engineering, 80% found software visualisation either absolutely necessary or important (but not critical) for their work.

We have a visualisation architecture [3] for deploying software components over the web. The design supports components in multiple languages and configurations (e.g. complete programs or just code fragments). Our architecture requires tools to develop and deliver visualisations from XML execution traces.

Our research group have created a tool that can produce 2D Scalable Vector Graphics (SVG) visualisations over the web from our execution traces [4]. Ware [5] demonstrates that displaying object oriented software in three dimensions instead of two can make it easier for users to understand the data. We want to create 3D software visualisations for software reuse, maintenance, re-engineering, and reverse engineering. We have decided to explore how suitable X3D [6] – the Web3D Consortium's open standard for web 3D graphics – is for creating visualisations from our XML execution traces and use in software visualisation.

Identifying the advantages and disadvantages of 3D visualisation techniques is beyond the scope of this paper. The rest of this paper is organised as follows. In section 2 we describe our tool for visualising execution traces in 3D. In section 3 we describe what X3D is. We then show our X3D software visualisations in section 4. In section 5 we discuss related work and conclude our ideas in section 6.

2 VARE3D

We are developing a web-based software visualisation tool called VARE3D, based on our Visualisation Architecture for REuse (VARE) [3]. VARE3D currently transforms XML execution traces into X3D visualisations, see Figure 1. Users make queries from a web browser. Users can test drive software components by specifying a sequence of method invocation and field access/modifications and then executing the sequence on a component. The output of a test drive is an XML execution trace. Users can transform XML execution traces using XSLT [3] into X3D visualisations. Users can then display X3D visualisations in a web browser.

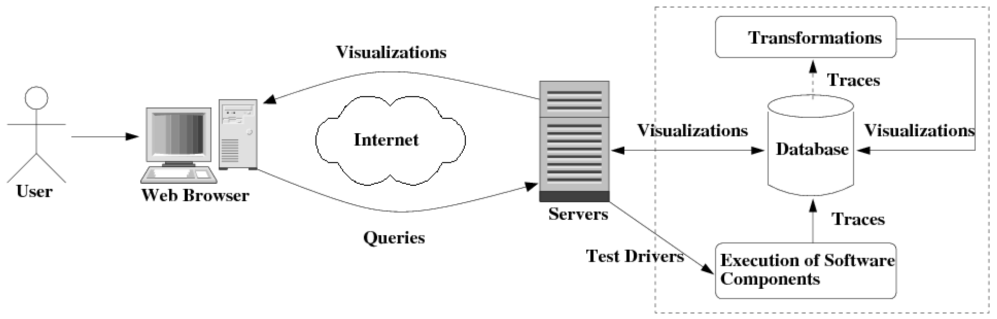


Fig. 1. VARE3D architecture.

The XML execution traces contain static and dynamic information of software components including the events that happened during the execution of a component. Separating the test driving and creation of visualisation steps allows users to test drive components and then create X3D visualisations in the future. Users can also view stored X3D visualisations without having to test drive remote software or transform XML execution traces.

[3] Extensible Stylesheet Language Transformations (XSLT) is a language used for transforming XML documents into other XML documents.

3 What is X3D?

X3D [6] is an XML language for 3D content delivery on the web. X3D is the successor to the Virtual Reality Modeling Language (VRML97). X3D combines both geometry and runtime behaviour into a single XML file. X3D can be displayed in a native X3D browser or a web browser that has an X3D plug-in. X3D content can be created using purpose built X3D authoring tools, text editors, transformed from XML, and converted or exported from third party applications. X3D allows scripting in ECMAScript or Javascript.

The X3D runtime environment is the scene graph which is a directed, acyclic graph containing the objects represented as nodes and object relationships in the 3D world. Nodes within the scene graph can have descriptive fields and can contain one or more child nodes. X3D is organised as a set of components where each component is a set of related functionality. Profiles are built from components and are a standardised set of extensions to meet specific application needs.

Navigation in X3D is specified by the navigation info node which defines the navigation paradigms (walk, slide, examine, fly, and look-at), the speed at which a user can move, and the range of visibility within the 3D world. These options can also be dynamically changed in a web browser. A user can also navigate to predefined locations called viewpoints. A user can interact with nodes in X3D by clicking, dragging, or moving the mouse over them. Animation is controlled by time sensors, colour, coordinate, orientation, and position interpolators. Lighting components can be used to illuminate or hide objects depending upon their location and grouping. Collision and gravity can also be applied to give real world effects.

4 X3D Software Visualisations

We want to find out how effective X3D is for use in software visualisation. In particular we want to visualise the static structure and dynamic behaviour of software from execution traces. We now present a representative sample of software visualisation techniques including algorithm animation, source code related visualisations, and execution trace visualisations.

Algorithm Animation. Brown and Najork [1] identified several reasons for integrating 3D graphics into an algorithm animation system. Figure 2 shows how the third dimension can be used for expressing fundamental information about Dijkstra's Shortest Path Algorithm. The shortest path is the green edges (A-B and A-C-E-D-F), while explored edges are purple and unexplored edges are white. Columns are used to represent the cost of getting to each vertex. The length of a path is the sum of all the weights of the edges along a path.

The third dimension in this animation provides state information about the cost of vertices and weight of edges. Animation is used to show fundamental operations of the algorithm: lifting an edge represents addition, lowering a highlighted edge indicates the outcome of a comparison, and shortening a column shows assignment.

The columns and edges are implemented as indexed faced sets. A column has eight points while an edge has four points. When the cost to a vertex changes the top face of the column is increased or decreased in the z dimension by changing four coordinate points. When an edge changes only two coordinate points are changed. Coordinate points are moved using coordinate interpolators, while the colour of a column or an edge is controlled by colour interpolators. A time sensor is used to start and keep the animation going, which loops every 20 seconds. Start, stop, and pause buttons can also be added to give a user more control over the animation. Using indexed faced sets as opposed to boxes and cylinders allowed smoother transitions and required less calculation effort.

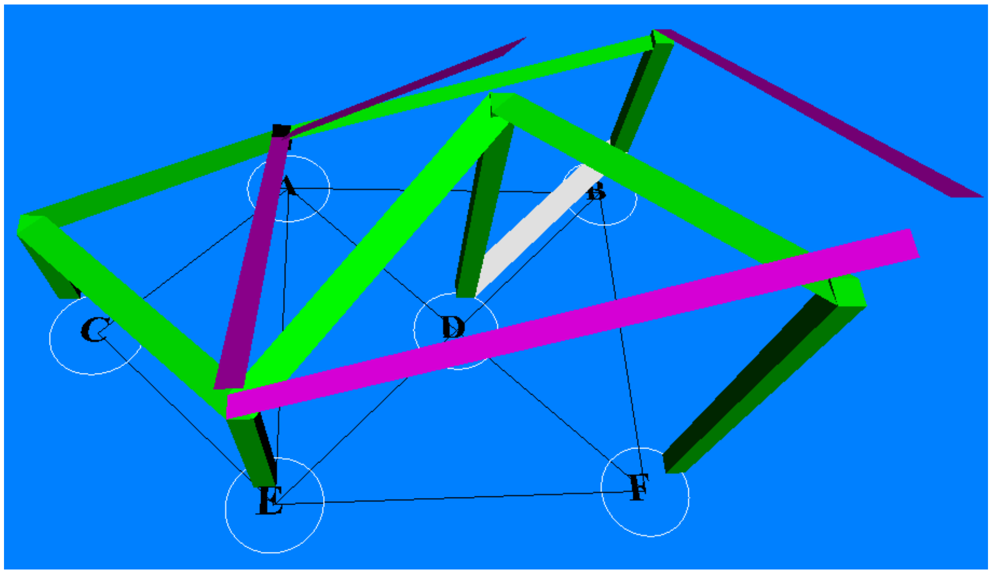
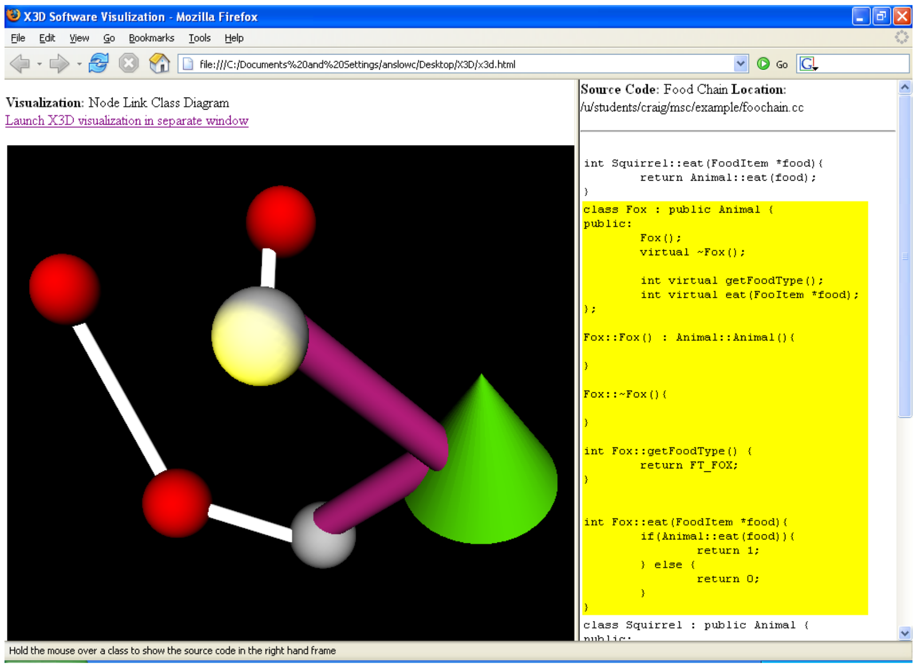


Fig. 2. Shortest Path Algorithm Animation

Source Code Related Visualisation. Figure 3 shows a class diagram of a C++ program represented as a node-link visualisation. The image has two displays, the left display shows the class diagram and the right display shows the original source code of the program which was used as the information for the visualisation. The source code is rendered in HTML. Classes are represented as spheres and the base class is represented as a cone. The two classes that inherit from the base class are coloured grey while the other classes are red. The cone is animated to change colour from blue to red to purple to green continuously every second. Larger purple cylinders represent inherited relationships from the base class while smaller white cylinders represent other class relationships.

A user can examine the visualisation by rotating the view which gives different view points of the class diagram to gain a greater understanding. A user can manipulate nodes by dragging them to show different parts of the visualisation. A user can click on a class which shines a light on the node in the visualisation and then some Javascript is executed that highlights the associated class declaration from the source code in the right display. In the visualisation the user has selected the class in the middle of the left display (grey sphere) which has highlighted the associated class declaration (highlighted yellow) in the source code (the Fox class).


X3D Software Vizulization - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/anslowc/Desktop/X3D/x3d.html
Go

Visualization: Node Link Class Diagram
Launch X3D visualization in separate window

Source Code: Food Chain **Location**:
/u/students/craig/msc/example/foochain.cc

```
int Squirrel::eat(FoodItem *food){
        return Animal::eat(food);
}
class Fox : public Animal {
public:
        Fox();
        virtual ~Fox();

        int virtual getFoodType();
        int virtual eat(FooItem *food);
};

Fox::Fox() : Animal::Animal(){

}

Fox::~Fox(){

}

int Fox::getFoodType() {
        return FT_FOX;
}


int Fox::eat(FoodItem *food){
        if(Animal::eat(food)){
                return 1;
        } else {
                return 0;
        }
}
class Squirrel : public Animal {
```

Hold the mouse over a class to show the source code in the right hand frame

Fig. 3. Source code related visualisation.

Execution Trace Visualisations. Wiss et al. [7] found if it were possible to predict what structure the data will have for a visualisation then this will help determine what metaphor is most applicable for visualising large data structures. Wiss et al. [8] performed an empirical study on three 3D information visualisation designs. The results indicated that the subjects were significantly faster with the information landscape followed by the cam tree and then the information cube. We have customised VARE3D to allow a user to display alternate 3D information visualisation metaphors from the same execution trace. We next show our information landscape and information cube visualisations.

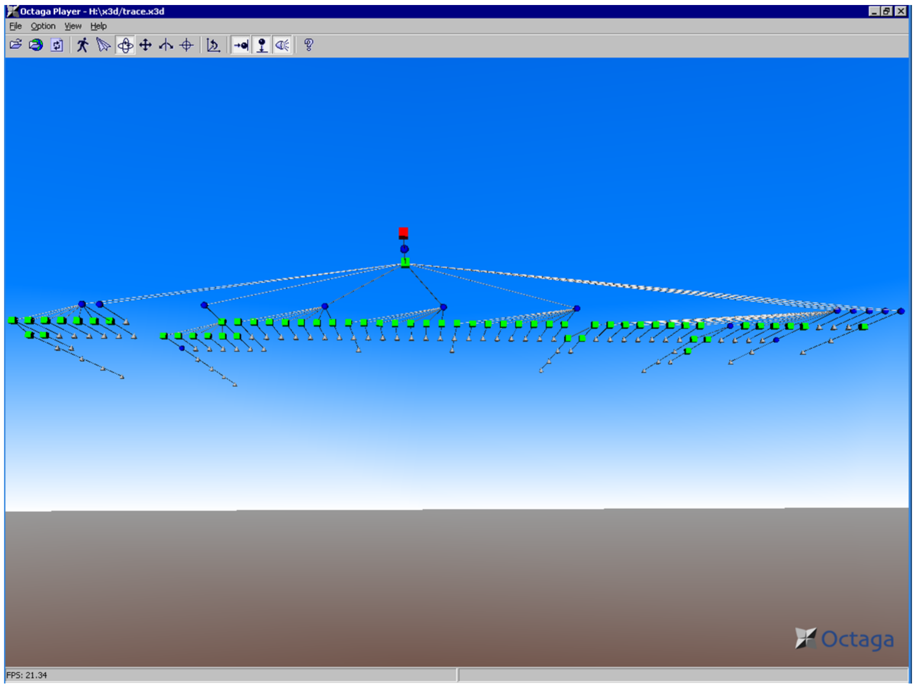


Fig. 4. Execution trace visualisation - information landscape.

Figure 4 shows the layout of all the events from an execution trace of a Java program as an information landscape. The information landscape is essentially 2 1/2 D rather than 3D. A red box represents the main class, blue spheres object creation, green boxes as method calls, white cones as method returns and end of the object. The image starts at the red box (the highest node in the image) and is animated from right to left.

Figure 5 shows the same information as Figure 4 but displayed as an information cube. The information starts with the main method outer red box, followed by the first object creation blue sphere at the top right of the cube then continuing along the links and down to each of the object creation blue spheres. The spheres are transparent and the events that an object executes are encompassed within the sphere.

Implementing the information landscape was more difficult than the information cube as it required more complex calculations for the links between nodes and the number of lines of code was greater. A linear layout approach was used however using formal graph layout algorithms should improve the creation and the actual visualisations. Finally transparency in the information cube was limited to only one level.

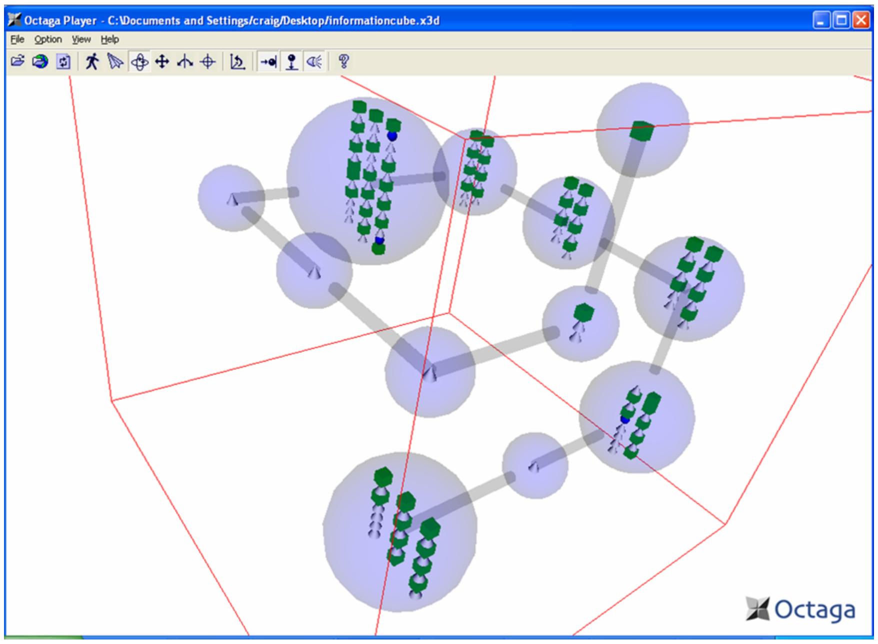

Octaga Player - C:\Documents and Settings\craig\Desktop\informationcube.x3d
File Option View Help
Octaga

Fig. 5. Execution trace visualisation - information cube.

5 Related Work

We now discuss some related systems and tools that use 3D technologies for visualising software. Further information on software visualisation and program visualisation can be located elsewhere [1].

sv3D [9] is a framework for visualising source code and related attributes in 3D. sv3D is implemented using Qt for the user interface and Open Inventor for the rendering components. This tool only works as a stand-alone desktop tool and does not consider dynamic run-time information.

Churcher et al. [10] use VRML and XML for software visualisation. They visualise inheritance structures, class hierarchies, class cohesion, object-oriented metrics, and class clusters. Charters et al. [11] created a tool that also uses VRML and XML to display visualisations of software components as 3D cities. Feijs and Jong [12] use VRML to visualise software architectures. However, none of these systems address dynamic information and both use the older VRML language.

UML visualisations have been explored using Java3D [13], VRML [14], and X3D [15]. However displaying UML in 3D does not scale well once there are many nodes in a world. Reading text in 3D once it has been rotated, repositioned rather than left to right, or partially obscured can make it hard for users to interpret what is being displayed.

6 Conclusion

We are building a tool that transforms our software XML execution traces into X3D visualisations. Our tool will help assist developers to understand the structure and behaviour of software for reuse, maintenance, re-engineering, and reverse engineering. We plan to implement other 3D information visualisation metaphors, create some more algorithm animations, provide visualisation filtering options, investigate the use of text in 3D, and integrate some test driving tools. We intend to conduct a comprehensive evaluation of X3D against some software visualisation frameworks and taxonomies to determine if X3D is applicable for use in software visualisation.

References

1. Stasko, J., Brown, M., Price, B.: Software Visualization. MIT Press (1998)
2. Koschke, R.: Software visualization for reverse engineering. In: Revised Lectures on Software Visualization, International Seminar. (2002)
3. Marshall, S., Jackson, K., Biddle, R., McGavin, M., Tempero, E., Duignan, M.: Visualising reusable software over the web. In: Proceedings of the Australasian Symposium on Information Visualisation. (2001)
4. Duignan, M., Biddle, R., Tempero, E.: Evaluating scalable vector graphics for use in software visualisation. In: Proceedings of the Australasian Symposium on Information Visualisation. (2003)
5. Ware, C.: Information Visualization: Perception for Design. Morgan Kaufmann (2004)
6. Web3D-Consortium: X3D specification (2004) http://www.web3d.org/x3d/.
7. Wiss, U., Carr, D., Jonsson, H.: Evaluating three-dimensional information visualization designs: A case study of three designs. In: Proceedings of the International Conference on Information Visualisation. (1998)
8. Wiss, U., Carr, D.: An empirical study of task support in 3D information visualizations. In: Proceedings of the International Conference on Information Visualisation. (1999)
9. Marcus, A., Feng, L., Maletic, J.: 3D representations for software visualization. In: Proceedings of the ACM Symposium on Software Visualization. (2003)
10. Churcher, N., Keown, L., Irwin, W.: Virtual worlds for software visualisation. In: Proceedings of the Software Visualisation Workshop. (1999)
11. Charters, S., Knight, C., Thomas, N., Munro, M.: Visualisation for informed decision making; from code to components. In: Proceedings of the Conference on Software Engineering and Knowledge Engineering. (2002)
12. Feijs, L., de Jong, R.: 3D visualization of software architectures. Communications of the ACM (1998)
13. Dwyer, T.: Three dimensional UML using force directed layout. In: Proceedings of the Australasian Symposium on Information Visualisation. (2001)
14. Gogolla, M., Radfelder, O., Richters, M.: Towards three-dimensional animation of UML diagrams. In: Proceedings of the International Conference on The Unified Modeling Language. (1999)
15. McIntosh, P., Hamilton, M., van Schyndel, R.: X3D-UML: enabling advanced UML visualisation through X3D. In: Proceedings of the International Conference on 3D Web Technology. (2005)