ALGORITHM VISUALISATION

Group 79: David Olsen and Nathan Pitman
Supervisor: Michael Dinneen

Objective

To develop a tool for automatically animating algorithms in action, focussing on:

- Ease of use – animations should be highly comprehensible for students and highly configurable by lecturers.
- Versatility – our framework should not be biased towards any family of algorithms.

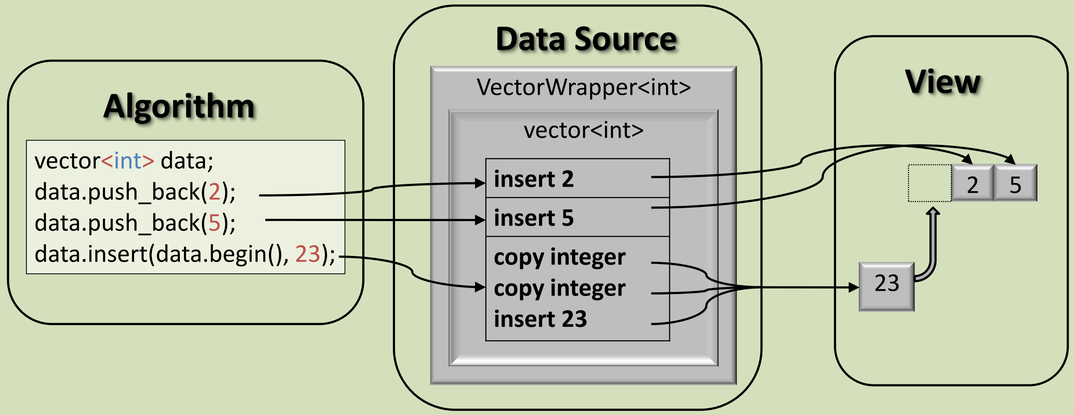
Motivation

Typically, lecturers will use some sort of diagram to show an example of an algorithm in action. This might just be a quick sketch, an animation, or even an interactive program.

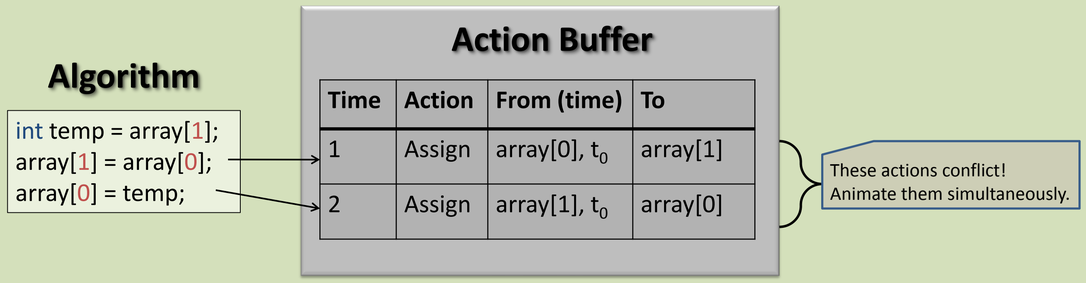
There are existing databases of animated GIFs and Java applets which are tailored only towards specific algorithms. Furthermore, because they are developed independently using different tools/languages, there is much repeated work between them.

The converse approach taken by debuggers is to display as much of the program state as possible at discrete points in time. This is problematic, as not only does it not emphasize the data which is crucial to comprehending the algorithm at hand, it fails to provide the transitions between program states which animations depict so well.

Architecture



The C++ Data Source intercepts changes to a data structure and causes animations to be played in the view.



User code instructions are converted into higher level actions and clustered together to form visually meaningful animations. In this case, three code instructions are converted into two actions which are treated as one swap animation.

Our Approach

- Developing a framework which works out of the box. In order for a user to have their algorithm visualised, they need simply write it as normal in C++ , include our header file and link against our library.
- Automatically detecting all instantiated data structures and deciding which ones should be displayed.
- Allowing the user to manipulate and drag these viewable objects around on screen.
- Tracking the movement of data throughout data structures ranging from integer types to hash tables. This allows our code to analyse complex data dependencies over time and compress them into useful and relevant actions.
- Providing functions which allow the user (in code) to perform useful tasks such as enforcing the display or suppression of specific data structures, or giving the View hints as to how it should lay out viewable objects according to the problem at hand.

Challenges

- Presenting the data structures in a space-efficient layout which is useful over a range of algorithms
- Performing introspection of data structures without affecting the expected operation of the original code or included libraries.
- Reading ahead through user code instructions and identifying sequences which constitute a single logical action.

Future Work and Conclusions

We have developed a generic framework for algorithm visualisation which can be used effectively in computer science education. Possible future work includes:

- Adding more wrappers and viewable equivalents so that more data structures can be visualised; e.g. graph data structures and matrices.
- Building upon the use-case of this framework as a visual debugger. This would involve writing a data-source component in another language which is more amenable to introspection.
- Allowing algorithm animations to be saved in a video format for archiving and future use.
- Adding a statistics module which displays useful data about the operations performed on various data structures.

SE Software Engineering The University of Auckland

THE UNIVERSITY OF AUCKLAND NEW ZEALAND
Te Whare Wānanga o Tāmaki Makaurau