UNIVERSITY OF AMSTERDAM

Efficient Abstractions for Visualization and Interaction
A.J. van der Ploeg

Summary

Abstractions, such as functions and methods, are an essential tool for any programmer. Abstractions encapsulate the details of a computation: the programmer only needs to know what the abstraction achieves, not how it achieves it. However, using abstractions can come at a cost: the resulting program may be inefficient. This can lead to programmers not using some abstractions, instead writing the entire functionality from the ground up.

In this thesis, we present several results that make this situation less likely when programming interactive visualizations. We present results that make abstractions more efficient in the areas of graphics, layout and events.

* Graphics abstractions

Graphics abstractions, which for example allow us to draw a line, fill a shape or rotate a drawing, are an essential part of programming interactive visualizations. We present a new declarative approach to resolution-independent 2D graphics that generalizes and simplifies the functionality of traditional frameworks, while preserving their efficiency. Our framework makes it easier to produce high image quality and makes non-affine transformations more efficient. As a real-world example, we show that the implementation of focus+context lenses gives higher image quality and better performance than the state-of-the-art solution, at a fraction of the code.

* Layout abstractions

Interactive visualizations often use abstractions that produce some kind of layout, such as a force directed graph layout or a treemap. When laying out trees in a node-link diagram, a classical algorithm exists that produces the layout in linear time, but the resulting layout takes more space than necessary. We present a novel algorithm that also runs in linear time, but produces more compact drawings.

* Event abstractions

Interactive visualizations need to deal with events such as mouse clicks and touch commands. Dealing with such events is traditionally done with either blocking I/O or callbacks. However, the former requires concurrency to compose reactive parts, which leads to non-determinism. The later leads to inversion of control: the control-flow of the program is dictated by the events that occur, not by the programmer.

An alternative that does not have these problems is Functional Reactive Programming (FRP). However, FRP often comes at the cost of efficiency: parts of the program are re-computed even though nothing changes. We present a novel FRP framework, called Monadic FRP, that has an efficient, incremental evaluation mechanism, hence preventing such redundant re-computations.

A general problem, which also manifests itself in Monadic FRP, is that for certain associative operators the number of steps it takes to evaluate an expression depends on how the brackets are placed. A solution is to use continuation passing style, but this again imposes a penalty if we alternate between using the associative operator and observing the results of that operator. We present a general solution that makes the performance of such operators efficient regardless of the placement of the brackets, while also providing efficient support for observing the result of that operator.

To conclude, we have shown that abstractions do not have to come at high costs: it is possible to create interactive visualizations by composing them from simple abstractions, without paying in terms of performance.