1 Introduction

The Processing language [Casey Reas 2007], first introduced by Ben Fry and Casey Reas in 2001, is a simple, elegant language for data visualization that is already being used by artists, educators and commercial media groups to produce rich graphical content called sketches. Because Processing is implemented in Java, delivering Processing sketches via a web page requires the user to install a Java plug-in. Processing.js, in comparison, is an open source, cross browser JavaScript port of the Processing language; it translates Processing sketches into JavaScript using the <canvas> element for rendering. No additional plug-ins are required to view a Processing sketch delivered with Processing.js. Furthermore, Processing.js is much more than just a Processing parser written in JavaScript: it enables the embedding of other web technologies into Processing sketches and vice versa. Processing.js seamlessly integrates web technologies with the Processing language to provide an excellent framework for rich multimedia web applications.

John Resig released a proof of concept to show how the <canvas> element could be used to render Processing sketches in 2007 as an open source project named Processing.js. While it was able to render the 2D sketches found on the Processing website, it was not optimized and far from complete. In 2009, the authors of this paper took up the task to bring Processing.js to parity with Processing. The 3D functionalities, which had not existed in the original Processing.js demonstration, were implemented. Numerous bugs were fixed and Processing.js was re-engineered to be more reliable and efficient. In addition to contributing to the project’s code base, the authors also organized the developer community by managing the workflow, test systems, issue tracker, and repositories. In November 2010, Processing.js 1.0 was released.

2 From Processing to Processing.js

At the time the original Processing language was developed, Java was expected to become the lingua franca of web programming, while JavaScript was a client-side scripting language for small tasks. As the web has matured, JavaScript has become the native programming language of the web. While many misconceptions regarding its speed and capabilities persist, recent advances in JavaScript engine technology have greatly improved its performance. In particular, speed increases from its just-in-time (JIT) compilers and use of hardware acceleration for graphics help make it fast enough to support real-time interactive multimedia web applications. Furthermore, current versions of Firefox and Chrome, along with upcoming versions of Opera and Safari browsers, all include implementations of WebGL, a hardware accelerated JavaScript API based on OpenGL ES 2.0. While the initial version of Processing.js was limited to Processing’s 2D functions, the current version now takes advantage of WebGL to implement Processing’s 3D functionality.

Processing.js works by translating Processing code found in a web page into JavaScript and renders it using the <canvas> element on the page. Any JavaScript code encountered is left intact and executed as JavaScript. This method allows for the translation of existing Processing sketches and the injection of JavaScript into Processing sketches without explicit labelling. Processing.js can correctly render sketches written in pure Processing while also easily handling sketches that make use of common JavaScript libraries such as jQuery. It also enables the sketch to easily interact with other elements of the web page or pull in resources from web services such as Twitter, Google Maps, and Flickr, without any plug-ins or additional software.

Another important feature of Processing.js is that it masks the differences between browsers. Web standards are often loosely defined and variations can exist, not only between different browsers but even between versions of the same browser on different platforms. The behaviour of something as simple as key events can vary widely between browsers and platforms. When creating interactive content on the web, these small differences can create huge problems for the developers. Processing.js solves this problem and others like it by behaving the same way regardless of the browser or platform. This standardization makes applications developed with Processing.js highly portable, letting developers reach the widest possible audience with the least effort.

3 Conclusion

Processing.js seamlessly integrates elements of a web page, web services, and interactive graphics. It was written to take advantage of modern browser features such as an optimized JavaScript engine and hardware accelerated 3D graphics, handling events in a consistent manner between different browsers, and making it easier for web developers to create content for a wide audience. All of these features make Processing.js an ideal platform for rich multimedia applications on the web.

4 Acknowledgements

We want to thank: David Humphrey, John Resig, Alistair MacDonald, Corban Brook Riley, Yuri Delendik, Matthew Lam, Dominic Baranski, Mike Kamermans, Tom Brown and the Processing.js community for all their work on the project. A special thanks to Marius Watz for letting us use his beautiful sketch abstract01.js. Also thanks to Evan Weaver, Mike Hoye, Dawn Mercer, and NSERC CCIP for their support.

References