Performance Comparison between Unity and D3.js for Cross-Platform Visualization on Mobile Devices

Lorenz Kromer, Markus Wagner, Kerstin Blumenstein,
Alexander Rind, Wolfgang Aigner

St. Poelten University of Applied Sciences, Austria

Abstract

Modern data visualizations are developed as interactive and intuitive graphic applications. In the development process, programmers basically pursue the same goal: *creating an application with a great performance.* Such applications have to display information at its best way in every possible situation. In this paper, we present a performance comparison on mobile devices between D3.js and Unity based on a Baby Name Explorer example. The results of the performance analysis demonstrated that Unity and D3.js are great tools for information visualization. While Unity convinced by its performance results according to our test criteria, currently Unity does not provide a visualization library.

1 Introduction & Related Work

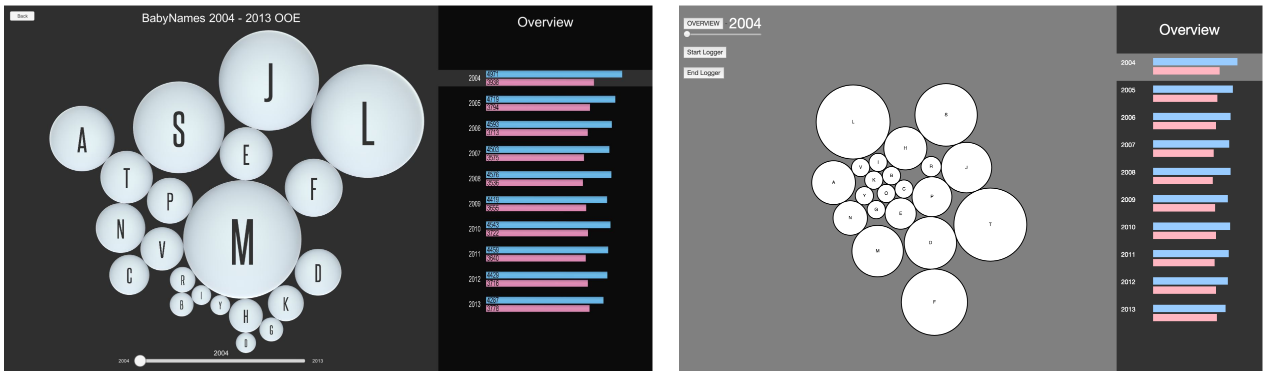
Visualization systems provide interactive, visual representations of data [8] designed to help people understand complex phenomena and augment their decision-making capabilities [18]. Given the interconnectedness of the current age and the increasing volumes of collected data, there is a dire need for such support. While many usage scenarios can be identified in scientific research and business management, systems for personal visualization [11] and casual information visualization [19] serve exceptionally broad audiences. These visualizations focus less on task-driven activities and more on curiosity and enjoyment while exploring personally relevant data. Showing trends of popular baby names, the Name Voyager [21] is a typical example of a casual visualization.

A main challenge faced by the developers of casual visualization systems is the heterogeneity of devices and platforms they should support. In particular for the casual context, mobile phones and tablets are more suitable than classical desktop computers [5, 4, 11, 15]. Native systems, e.g., apps for Android or Apple, are only runnable on the platform for which the code is compiled for. Cross-platform support requires the development on top of different software stacks and to maintain separate code bases. One approach to address this challenge are web-based visualizations, i.e. using web technology such as D3.js [7] within the browser. However, a wide-spread concern is that web-based systems lack performance. For example, Baur stated in a 2013 interview [3] that for big visualization systems such as TouchWave [2], going native cannot be avoided because "in the web it looks like a slide show". Besides the negative effects of interactive latency [16], performance overheads negatively affect battery load of mobile devices. Alternative approaches are cross-compilers such as Unity [1], which can deploy a single code base to native systems for multiple platforms. Yet, a limitation of Unity is that it does not include a software library for visualizing data [20]. These two approaches for cross-platform visualization work very differently during both implementation and runtime. The choice will largely depend on the respective application scenario but empirical data on their performance is needed to inform such a decision.

While some research has been carried out to compare the performance of different web-based visualization technologies [14, 12, 13], no studies have been found which compare the performance of web-based and cross-compiled visualization approach. Neither could we identify performance results obtained from different target platforms.

Thus, the paper at hand, contributes a performance

In: W. Aigner, G. Schmiedl, K. Blumenstein, M. Zeppelzauer (eds.): Proceedings of the 9th Forum Media Technology 2016, St. Pölten, Austria, 24-11-2016, published at http://ceur-ws.org



(a) Unity

(b) D3.js

Figure 1: Shows a screenshot of the Baby Name Explorer interface implemented with (a) Unity and (b) D3.js. Shows the circle packing chart (left) with the corresponding grouped bar chart (right) representing the frequency per year for male (blue) and female (pink) names.

comparison between Unity (cross-compiled to native) and D3.js (web-based) on four mobile devices. For this, we created two implementations of a casual visualization system to explore popular baby names as described in Section 2. Section 3 covers the implementation details and test setup. After the test results in Section 4, we conclude our work in Section 5 and outline future work.

2 Visualization Design

As proof of concept we started with implementing a simple interactive visualization setup using an open data set of the regional government of Upper Austria on the 50 most often used male and female baby names from 2004 to 2013. The dataset includes the variables *name* (nominal), *gender* (categorical), *year* (quantitative) and *count* (quantitative). All these data are merged together into a table provided as *.csv file. As visualization concepts we combined a circle packing chart [10] with grouped bar charts [9].

Initially, the circle packing chart shows the first letters of the baby names as bubbles and its diameter matches with the number of babies per year. A slider is positioned at the bottom of the screen for selecting the year to display.

By tapping a bubble, the bubble expands and the names which are related to the first letter are shown inside the big bubble (see Figure 1). The color of a name bubble is related to the gender (pink := female, blue := male) and the diameter matches the number of babies with the name for the selected year. During the layout phase, the bubbles are placed using physics-based movement like gravity and the biggest bubble is set to the center of the screen. The circle packing chart is linked with a grouped bar chart. The bar chart initially shows the number of babies for all names grouped per year, split into female and male names (using the same colors as for the bubbles). When selecting a first letter bubble, the grouped bar chart shows the number of babies for names starting with the selected letter. When selecting a name bubble (e.g., “Leonie”), the grouped bar chart changes to a single bar chart presenting the number for the name per year.

3 Implementation and Test Setup

To introduce the implementation and test setup, we describe the used tools for implementation D3.js and Unity, the four test devices and environments, the performance criteria and desired results as well as the measured values and methods.

3.1 Test Devices and Environments

Since we focus on cross-platform visualization, the test devices cover a range from tablets (Nexus 9 and iPad Air) to Smartphones (iPhone 6S+ and Galaxy S6 Edge). Both visualization systems are investigated on the devices shown in Table 1.

When selecting the mobile test devices, we deliberately choose devices with larger screen sizes, since the presentation of the tested visualization (see Section 2) on a screen size of 5” or small is not optimal.

The visualization is tested under Android 5.1 (Nexus 9 and Galaxy S6 Edge) and iOS 9 (iPad Air and iPhone 6S+). In addition to the requirements of the devices, the test concept of this paper also examines the dependencies of both visualization versions of external components such as libraries and plug-ins, which were used during the development process.

Unity: With the development environment of Unity it is possible to make a project accessible for multiple platforms. The Unity version of the Baby Name Explorer (Figure 1a) is exported in two versions

Table 1: Overview of the dimensions of the test devices.

Device	Type	Screen size	Screen resolution	Processor	RAM	Graphics processor
Nexus 9	Tablet	8.9"	$2048 \times 1536px$	NVIDIA Tegra K1	2 GB	NVIDIA GeForce ULP
iPad Air LTE	Tablet	9.7"	$2048 \times 1536px$	Apple A7	1 GB	PowerVR G6430
iPhone 6S+	Smart-phone	5.5"	$1920 \times 1080px$	Apple A9	2 GB	PowerVR GT7600
Galaxy S6 Edge	Smart-phone	5.1"	$2560 \times 1440px$	Samsung Exynos 7 Octa 7420	3 GB	Mali-T760 MP8

(Android and iOS). The rich development environment of the game engine Unity includes a sufficient repertoire of physics components and 3D elements. Therefore, we did not have to use external libraries.

D3.js: Since the implementation of the visualization in D3.js (Figure 1b) is web browser based, we used the Google Chrome web browser as test environment which is available on all tested devices (see Table 1). Thus, the visualization is represented under the same technological conditions. For the implementation of the web based version, we did not need additional JavaScript libraries, because D3.js contains all functionalities.

3.2 Measured Values and Methods

To compare a number of software applications, common metrics and measurement points have to be defined [17]. Subsequently the used methods are:

- **FPS:** For measuring the frames per second (FPS) rates, time logging functions are added around rendering methods in the code, logging the results via logfiles or the console.
- **CPU utilization:** To show the difference between the hardware components, the CPU utilization was observed while performing both visualizations in a specific scenario and five minutes in idle mode. Therefore, it was ensured that no other processes were running on the device.
- **Loading time of raw data:** Both version (Unity and D3.js) contain an explicit function to load the raw data. In order to compare the raw data loading from a CSV file, the elapsed time was measured between the explicit function call and end.

In relation to the technical implementation, Unity and D3.js are strongly different. To overcome this issue, we recorded the system parameters and console logs with OS specific development systems, because there are no uniform functions available to detect the previously listed system parameters.

With the aforementioned measured values, both visualization systems were tested in a specific user scenario. In this case, the Baby Name Explorers usage was simulated over 60 seconds by a regular interaction with the respective system. To reduce the effects of operating system and other processes beyond user control, this user scenario was repeated five times on each visualization system per tested device.

4 Results

The results of the performance comparison of both versions are separated into the three measured parameters, which were presented before. All the measured values of the different test devices were compared into an Excel sheet for preprocessing. By using MS Excel, we processed the calculation of the median values to eliminate outliers and exported the result for each parameter as grouped bar chart.

4.1 CPU Usage Analysis

Based on the performed measurements, Unity generates less CPU usage than D3.js. Calculating the median across all measured devices, Unity takes 22% and D3.js takes 38%. Figure 2 illustrates a diagram to compare the CPU usage between the tested devices in idle mode and while performing both versions.

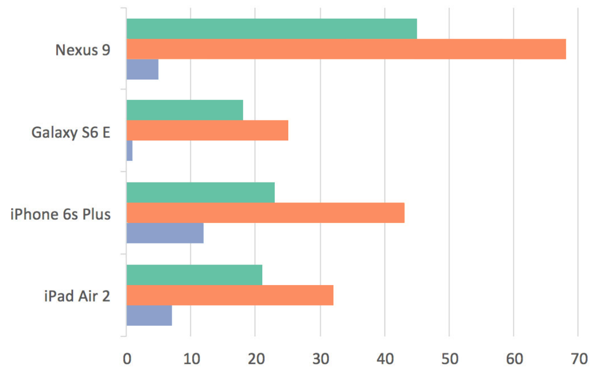


Figure 2: CPU usage in % in Unity (green), D3.js (orange) compared to idle mode (blue) [lower is better].

During the performance analysis it was very interesting to see, that the Nexus 9 tablet got noticeable warmer than the other devices. This effect mirrors in the device's CPU usage. However, no temperature measurements were carried out to investigate this effect. In general, less CPU usage is a big benefit from the perspective of smart devices because less energy consumption results in more battery time.

4.2 FPS analysis

The evaluation of the FPS data shows that Unity reaches a median of 57 FPS and D3.js version achieves a median of 51 FPS. Unity can be seen as the winner of this criteria of the performance comparison. The detailed median values of the evaluation part are shown in Figure 3.

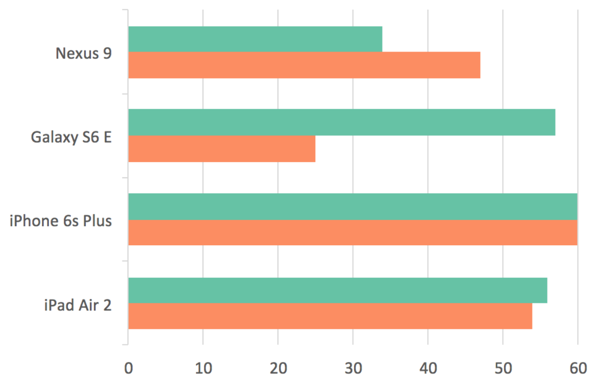


Figure 3: FPS rate while performing with Unity (green) and D3.js (orange) [higher is better].

It is very prominent, that the FPS rate of the D3.js version was pretty low on the Galaxy S6 Edge, despite the fact that the CPU usage on this device also stayed slightly. In contrast, the Nexus 9 tablet was the only device which reaches higher FPS with D3.js.

4.3 Loading Time Analysis

The result of the CSV data loading time measurement shows, that D3.js takes a median of 5.17ms. In contrast, Unity requires significantly more time for the raw data loading which results in a median of 15.17ms. Figure 4 shows the gap between both versions.

The measured time depends on the internal implementation of the loading methods of the visualizations which is the reason of the serious differences at the cycle times of these functions.

5 Conclusion

This study compared two different approaches for implementing cross-platform visualizations: cross-compilation to native code and web technology, i.e. usage within a web browser.

For this, the Baby Name Explorer, as example of a realistic casual visualization design, was implemented in both Unity and D3.js. Our experimental comparison on four devices showed that FPS were comparable, D3.js was faster in initial data transformations, and Unity resulted in a lower CPU utilization.

In terms of developer experience, Unity's IDE supports C# as well as JavaScript for development. The cross-compilation and deployment of the Baby Name Explorer for all tested platforms worked seamlessly.

D3.js code is typically developed for a web environment. Due to the variety of web browsers, web based visualizations need to be tested on a wide selection before being released. During our experiment both implementations worked well.

Depending on our proof-of-concept, we demonstrated the benefits of the use of Unity for information visualization and cross-platform compilation in our field of research. In the next steps we will focus on the synchronization for collaboration and semantic zoom [20] and to show the ability to use this framework for visualization for the masses as called by Blumenstein et al. [6] as an easy to use system.

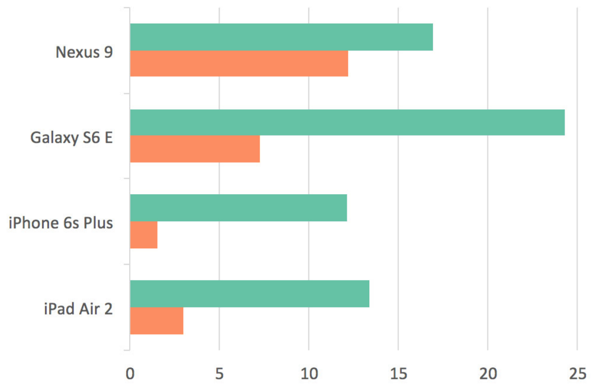


Figure 4: CSV loading times in ms while performing in Unity (green) and D3.js (orange) [lower is better].

Acknowledgements

This work was supported by the Austrian Science Fund (FWF) via the KAVA-Time and VisOnFire projects (no. P25489 and P27975), the Austrian Ministry for Transport, Innovation and Technology (BMVIT) under the ICT of the future program via the VALiD project (no. 845598) and under the Austrian Security Research Programme KIRAS via the project Courageous Community (no. 850196) as well as the project seekoi (no. 1154) funded by the Internet Foundation Austria (IPA).

References

[1] Unity – Game Engine, 2016. https://unity3d.com/.

[2] Dominikus Baur, Bongshin Lee, and Sheelagh Carpendale. TouchWave: kinetic multi-touch manipulation for hierarchical stacked graphs. In *Proc. 2012 ACM int. conf. Interactive Tabletops and Surfaces, ITS*, pages 255–264. ACM, 2012.

[3] Enrico Bertini, Moritz Stefaner, and Dominikus Baur. Visualization on Mobile & Touch Devices. datastori.es podcast, `http://datastori.es/data-stories-25-mobile-touch-vis/`, 00:41:49 to 00:46:08, July 2013.

[4] Kerstin Blumenstein, Christina Niederer, Markus Wagner, Grischa Schmiedl, Alexander Rind, and Wolfgang Aigner. Evaluating information visualization on mobile devices: Gaps and challenges in the empirical evaluation design space. In *Proc. 6th Workshop on Beyond Time and Errors on Novel Evaluation Methods for Visualization, BELIV*, pages 125–132. ACM, 2016.

[5] Kerstin Blumenstein, Markus Wagner, and Wolfgang Aigner. Cross-Platform InfoVis Frameworks for Multiple Users, Screens and Devices: Requirements and Challenges. In *Workshop on Data Exploration for Interactive Surfaces DEXIS 2015*, pages 7–11, 2015.

[6] Kerstin Blumenstein, Markus Wagner, Wolfgang Aigner, Rosa von Suess, Harald Prochaska, Julia Püringer, Matthias Zeppelzauer, and Michael Sedlmair. Interactive Data Visualization for Second Screen Applications: State of the Art and Technical Challenges. In *Proc. of the Int. Summer School on Visual Computing*, pages 35–48. Frauenhoferverlag, 2015.

[7] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-Driven Documents. *IEEE Trans. Vis. and Comp. Graphics*, 17(12):2301–2309, December 2011.

[8] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. *Readings in Information Visualisation. Using Vision to Think.* Morgan Kaufmann, 1999.

[9] William S. Cleveland and Robert McGill. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.

[10] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. A tour through the visualization zoo. *Comm. of the ACM*, 53(6):59, 2010.

[11] Dandan Huang, Melanie Tory, Bon Adriel Aseniero, Lyn Bartram, Scott Bateman, Sheelagh Carpendale, Anthony Tang, and Robert Woodbury. Personal visualization and personal visual analytics. *IEEE Trans. Vis. and Comp. Graphics*, 21(3):420–433, March 2015.

[12] Donald W. Johnson and T. J. Jankun-Kelly. A scalability study of web-native information visualization. In *Proc. Graphics Interface, GI*, pages 163–168, Toronto, 2008. Canadian Information Processing Society.

[13] Daniel E. Kee, Liz Salowitz, and Remco Chang. Comparing interactive web-based visualization rendering techniques. In *Poster Proc. IEEE Conf. Information Visualization, InfoVis*, 2012.

[14] Tim Lammarsch, Wolfgang Aigner, Alessio Bertone, Silvia Miksch, Thomas Turic, and Johannes Gärtner. A comparison of programming platforms for interactive visualization in web browser based applications. In *Proc. 12th Int. Conf. Information Visualisation, iV*, pages 194–199, July 2008.

[15] Tim Lammarsch, Wolfgang Aigner, Silvia Miksch, and Alexander Rind. Showing important facts to a critical audience by means beyond desktop computing. In Yvonne Jansen, Petra Isenberg, Jason Dykes, Sheelagh Carpendale, and Dan Keefe, editors, *Death of the Desktop—Workshop co-located with IEEE VIS 2014*, 2014.

[16] Zhicheng Liu and Jeffrey Heer. The effects of interactive latency on exploratory visual analysis. *IEEE Trans. Vis. and Comp. Graphics*, 20(12):2122–2131, December 2014.

[17] J. D. Meier, Carlos Farre, Prashant Bansode, Scott Barber, and Dennis Rea, editors. *Performance testing guidance for web applications: patterns & practices.* Microsoft, United States?, 2007. OCLC: ocn245241921.

[18] Tamara Munzner. *Visualization Analysis and Design.* A K Peters Ltd, 2014.

[19] Zachary Pousman, John T. Stasko, and Michael Mateas. Casual Information Visualization: Depictions of Data in Everyday Life. *IEEE Trans. Vis. and Comp. Graphics*, 13(6):1145–1152, 2007.

[20] Markus Wagner, Kerstin Blumenstein, Alexander Rind, Markus Seidl, Grischa Schmiedl, Tim Lammarsch, and Wolfgang Aigner. Native cross-platform visualization: A proof of concept based on the Unity3D game engine. In *Proc. Int. Conf. Information Visualisation, iV*, pages 39–44. IEEE Computer Society Press, 2016.

[21] Martin Wattenberg. Baby names, visualization, and social data analysis. In *Proc. IEEE Symp. Information Visualization, INFOVIS*, pages 1–7, October 2005.