Interactive Visualizations (IVI) Bericht

Nicola Rohner

SE FE 2023

Repository: `https://github.com/nicolarohner1337/ivi`

Abstract

In diesem Bericht werden die Lernergebnisse des Moduls IVI zusammengefasst.
Für alle ausser LE1 wurden folgende Daten verwendet:
GroupLens Research hat Bewertungsdatensätze von der MovieLens-Website `https://movielens.org` gesammelt und zur Verfügung gestellt. Die Datensätze wurden über verschiedene Zeiträume hinweg gesammelt, abhängig von der Größe des Satzes. Das ZIP findet man unter folgendem Link: `https://files.grouplens.org/datasets/movielens/ml-latest.zip` Die Daten umfassen 27.000.000 Bewertungen und 1.100.000 Tag-Anwendungen für 58.000 Filme von 280.000 Nutzern. Beinhaltet Tag-Genom-Daten mit 14 Millionen Relevanz-Scores für 1.100 Tags.

Die Visualisierungen in diesem Bericht wurden alle selbst erstellt. Der Datensatz wurde bereits für die Challenge CDS1 genutzt und für die Erarbeitung von GDV.

Contents

1 LE1: Performance

In Zeiten von Big Data ist es wichtig, dass Visualisierungen schnell und performant sind. In diesem LE wird untersucht, wie sich die Performance von Visualisierungen auf verschiedene Arten beeinflussen lässt.

1.1 WebGL

WebGL ist eine leistungsstarke Technologie, die Berechnungen beschleunigt. Allerdings gibt es einige Einschränkungen. WebGL benötigt eine GPU (Grafikkarte), die nicht immer in allen Browsern verfügbar ist. Mit WebGL gerenderte Daten werden als Pixelraster gezeichnet, was in einigen Fällen zu verpixelten oder unscharfen Bildern führen kann. Browser begrenzen die Anzahl der WebGL Kontexte, auf die ein bestimmtes Webdokument zugreifen kann, was die Darstellung von WebGL-Visualisierungen einschränken kann. Browser setzen Grenzen für die Höhe und Breite von Visualisierungen, die WebGL verwenden, was bei extrem großen Plots zu Problemen führen kann.[6]

Um die Performance Unterschiede zu untersuchen habe ich eine Visualisierung erstellt, welche 100000 zufällige Punkte in einem 2D Koordinatensystem darstellt. Einmal wurde die Visualisierung mit dem Modus SVG und einmal mit WebGL gerendert. Was bereits qualitativ auffällt ist, dass die WebGL Visualisierung deutlich schneller angezeigt wird im Chrome Browser.

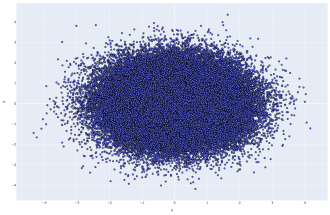

Figure 1: Random dots

Die Chrome Entwickler Tools verfügen über ein Performance Profiler Tool, welches die Performance einer Webseite analysieren kann. Es unterteilt die Laufzeitleistung in das RAIL-Modell[4]: Response, Animation, Idle und Load. Im Kontext zur Visualisierung betrachte man die Scripting, Rendering und Painting Kategorien.[3]

Kategorie	**SVG**	**WebGL**	**%Δ**
Scripting	3778ms	651ms	-580%
Rendering	2026ms	14ms	-1447%
Painting	197ms	2ms	-985%

Table 1: Performance Unterschiede

Zu beobachten ist ein signifikanter Unterschied in den Rendering und Painting Zeiten der beiden Visualisietungen. Auch die subjektive Wahrnehmung der Performance ist deutlich besser bei der WebGL Visualisierung. Auch die Tradeoffs zwischen Performance und Qualität sind deutlich zu erkennen. Die SVG Visualisierung ist deutlich schärfer und die Punkte sind deutlich besser zu erkennen.

Figure 2: Qualität Unterschiede links SVG, rechts WebGL

1.2 Layering

Datashader ist ein Grafik-Pipeline-System zur schnellen und flexiblen Erstellung aussagekräftiger Darstellungen grosser Datensätze. Datashader unterteilt die Erstellung von Bildern in eine Reihe expliziter Schritte, die es ermöglichen, Berechnungen an Zwischendarstellungen vorzunehmen.[1]
Wir visualisieren hier die räumliche Verteilung der Taxifahrten in New York City. Die Daten werden von Plotly zu Verfügung gestellt und sind unter folgendem Link zu finden: `https://raw.githubusercontent.com/plotly/datasets/master/uber-rides-data1.csv`

Date/Time	**The date and time of the Uber pickup**
Lat	**The latitude of the Uber pickup**
Lon	**The longitude of the Uber pickup**

Table 2: Newyork Uber Data

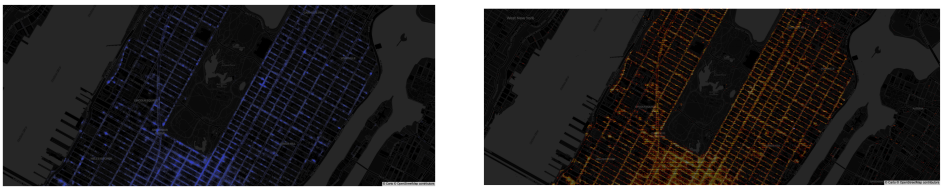

Figure 3: Taxifahrten in Newyork (Plotly links, Datashader Layer rechts)

Zu beobachten ist das die unprozessierten Daten mit Plotly zwar schöner dargestellt werden. Aber viel länger brauchen um angezeigt zu werden. Die Datashader Visualisierung ist deutlich schneller und die Daten werden deutlich besser dargestellt. Diese Beobachtungen wiederspiegeln sich auch in den Performance Daten. Zwar wird nur die Scripting Zeit signifikant verkürzt.

Kategorie	**Plotly**	**Datashader Layer**	**%Δ**
Scripting	1241ms	492ms	-39%
Rendering	6ms	6ms	±0%
Painting	12ms	12ms	±0%

Table 3: Performance Unterschiede

Spannender bei diesem Fall sind die Unteschiede des genutzen Speichers. Wenn wir den JavaScript Heap betrachten sehen wir einen signifikanten Unterschied[5]. Plotly braucht fast 130MB um die Daten zu visualisieren, der Datashader Layer braucht nur 23MB.

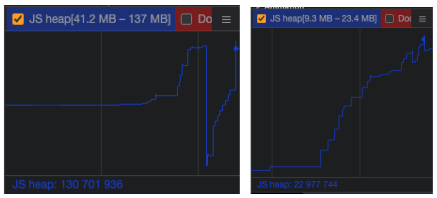

Figure 4: Memory Heap Plotly vs. Datashader Layer

Einen ähnlichen Ansatz verfolgen auch Framworks wie Deck.gl. Diese Frameworks sind jedoch auf WebGL basiert und können somit nur auf Browsern mit WebGL Unterstützung verwendet werden. Sie prozessieren für verschiedene Ansichte voragregierte Layers und können so sehr schnell und performant visualisieren.[2]

References

[1] Joseph A. Cottam, Andrew Lumsdaine, and Peter Wang. "Abstract rendering: out-of-core rendering for information visualization". In: IS&T/SPIE Electronic Imaging. Ed. by Pak Chung Wong et al. San Francisco, California, USA, Dec. 23, 2013, 90170K. DOI: 10.1117/12.2041200. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2041200 (visited on 03/18/2023).

[2] deck.gl. *Home — deck.gl*. URL: https://deck.gl/ (visited on 03/19/2023).

[3] Kayce Basques. *Analyze runtime performance*. Chrome Developers. URL: https://developer.chrome.com/docs/devtools/performance/ (visited on 03/18/2023).

[4] *Measure performance with the RAIL model*. web.dev. URL: https://web.dev/rail/ (visited on 03/19/2023).

[5] *Memory management - JavaScript — MDN*. Feb. 23, 2023. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_Management (visited on 03/19/2023).

[6] Plotly. *Webgl vs svg in Python*. WebGL vs SVG in Python. URL: https://plotly.com/python/webgl-vs-svg/ (visited on 03/18/2023).