# 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. Die CPU verarbeitet alle Hauptfunktionen des Computers, während die GPU viele kleinere Aufgaben gleichzeitig ausführt.[3]

| **CPU** | **GPU** |
|---|---|
| Generalistische Komponente - Verarbeitet die wichtigsten Verarbeitungsfunktionen eines Computers | Spezialisierte Komponente - Verarbeitet Grafik- und Video-Rendering |
| Kernanzahl- 2-64 (die meisten CPUs) | Kernanzahl- Tausende |
| Führt Prozesse seriell aus | Führt Prozesse parallel aus |
| Besser bei der Verarbeitung einer großen Aufgabe zur gleichen Zeit | Besser bei der Verarbeitung mehrerer kleinerer Aufgaben zur gleichen Zeit |

Table 1: CPU vs GPU

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.[10]

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.

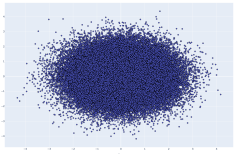

Figure 1: Random dots

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[8]: Response, Animation, Idle und Load. Im Kontext zur Visualisierung betrachte man die Scripting, Rendering und Painting Kategorien.[6]

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

Table 2: 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.

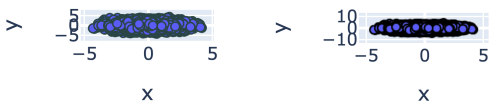


Figure 2: Qualität Unterschiede links SVG, rechts WebGL

## 1.2 Tiling

Tiling wurde von Google Maps erfunden, um die Rechenleistung zu optimieren. Die Karte wird in kleinere Teile unterteilt und nur die benötigten Daten für den jeweiligen Zoomlevel werden angezeigt. Diese kleineren Kacheln komprimieren nicht nur die Daten, sondern geben das Rendering an den Webbrowser ab, wodurch die für die Karte erforderliche Nutzlast weiter reduziert wird.[7] 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.[2]

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 3: Newyork Uber Data

Figure 3: Taxifahrten in Newyork (Plotly links, Datashader Tiles 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 Tile** | **%Δ** |
|---|---|---|---|
| Scripting | 1241ms | 492ms | -39% |
| Rendering | 6ms | 6ms | ±0% |
| Painting | 12ms | 12ms | ±0% |

Table 4: Performance Unterschiede

Spannender bei diesem Fall sind die Unteschiede des genutzen Speichers. Wenn wir den JavaScript Heap betrachten sehen wir einen signifikanten Unterschied.[9] Plotly braucht fast 130MB um die Daten zu visualisieren, der Datashader Tile braucht nur 23MB.

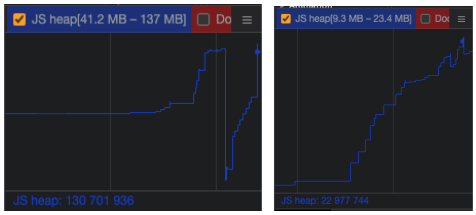


Figure 4: Memory Heap Plotly vs. Datashader Tile

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 Tiles und können so sehr schnell und performant visualisieren.[4]

# 2 LE2: Grundsätze für die Gestaltung von Dashboards

Ein wichtiger Aspekt bei der Entwicklung von Dashboards ist das Konzept. Dabei hat der Benutzer Zugriff auf verschiedene Steuerelemente und mehrere verknüpfte Visualisierungen. Diese Designprinzipien sind entscheidend für die Entwicklung eines erfolgreichen Dashboards.

## 2.1 Schneiderman's Mantra

Schneiderman's Mantra ist ein Organisationsprinzip für die Erstellung von Visualisierungssystemen. Es lautet wie folgt:

**Overview first**: Im Überblicksschritt wird der gesamte Datensatz in einer geeigneten Anzeigemethode dargestellt. Dies bietet einen Überblick auf hoher Ebene und gibt Kontext für die nächsten Schritte in der Visualisierung.

**Zoom and filter**: Zoomen auf einen Abschnitt des Datensatzes ermöglicht das Entfernen von überflüssigen Daten anhand der angezeigten Koordinaten. So erhalten die relevanten Daten mehr Auflösung und Detail. Das Filtern entfernt überflüssige Daten basierend auf gewünschten Attributen, vereinfacht die Anzeige Ihrer Daten und schafft mehr Platz für Details.

**Details on demand**: Details auf Abruf geben dem Benutzer Kontrolle über die Daten und ermöglichen es, ohne Überladen des Bildschirms weiter zu erkunden. Das Tooltip ist die häufigste Implementierung. Eine andere Möglichkeit besteht darin, ein Feld auszuwählen und die Daten hervorzuheben.[5]
Eine mögliche Umsetzung mit diesem Prinzip könnte wie folgt aussehen. Im ersten Schritt wird eine Übersicht über die Anzahl Filme pro Genre gegeben.

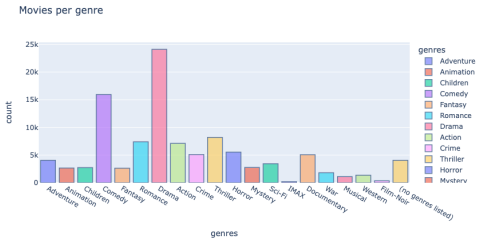


Figure 5: Genre Übersicht

Wenn der Benutzer nun auf ein Genre klickt oder auf ein Genre zoomt wird die Anzahl Filme pro Jahr angezeigt.

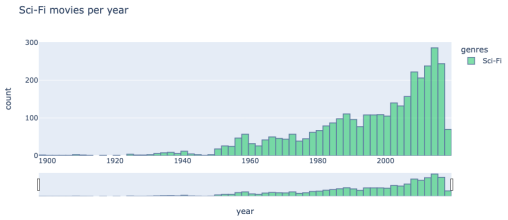


Figure 6: Filtering/Zooming: Anzahl Filme pro Jahr

Mit einem Tooltip kann der Benutzer nun die Details zum einzelnen Film einsehen.

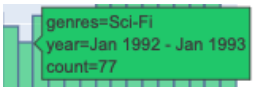


Figure 7: Tooltip: Details on demand

Diese Interaktionen können mit Hilfe Plotly.js Chart Events realisiert werden. Wenn ein solches Event getriggert wird kann die Funktion `Plotly.restyle` aufgerufen werden. Diese Funktion ermöglicht es die Daten eines Charts zu ändern.

## 2.2 Verknüpfte Ansichten

Das Paradigma der verknüpften Ansichten ist eine Methode, die mehrere einfache Ansichten von Daten verwendet. Wenn Sie mit einer Ansicht interagieren, ändert sich die Anzeige der Daten in allen verknüpften Ansichten. Ein einfaches Beispiel wäre, dass die Auswahl eines Datums in einer Ansicht die Daten in allen anderen Ansichten auch ändert. Kurz gesagt wenn sich eine Ansicht ändert, ändern sich alle anderen Ansichten auch.[11]

## 2.3 Brushing

Brushing ist eine Methode, die es ermöglicht, Daten in einer Ansicht zu selektieren und diese dann in anderen Ansichten zu verwenden. Ein Beispiel wäre, dass Sie eine Ansicht mit einem Scatterplot haben, in der Sie einen Bereich selektieren. Diese Selektion wird dann in einer anderen Ansicht mit einem Histogramm verwendet.[1]

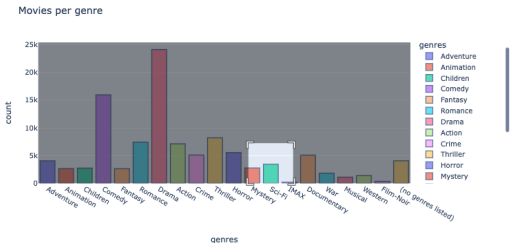

Figure 8: Brushing

# References


[1] Richard A. Becker and William S. Cleveland. "Brushing Scatterplots". In: *Technometrics* 29.2 (May 1987), pp. 127–142. ISSN: 0040-1706, 1537-2723. DOI: 10.1080/00401706.1987.10488204. URL: http://www.tandfonline.com/doi/abs/10.1080/00401706.1987.10488204 (visited on 04/13/2023).

[2] 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).

[3] *CPU vs. GPU: What's the Difference?* URL: https://www.cdw.com/content/cdw/en/articles/hardware/cpu-vs-gpu.html (visited on 04/13/2023).

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

[5] HAMPDATAVISUALIZATION. *Schneiderman's Mantra*. Data Visualization. Feb. 26, 2016. URL: https://hampdatavisualization.wordpress.com/2016/02/26/schneidermans-mantra/ (visited on 04/13/2023).

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

[7] *Map Tiles: Everything You Need To Know*. URL: https://carto.com/blog/map-tiles-guide (visited on 04/13/2023).

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

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

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

[11] Graham Wills. "Linked Data Views". In: *Handbook of Data Visualization*. Ed. by Chun-houh Chen, Wolfgang Härdle, and Antony Unwin. Springer Handbooks Comp.Statistics. Berlin, Heidelberg: Springer, 2008, pp. 217–241. ISBN: 978-3-540-33037-0. DOI: 10.1007/978-3-540-33037-0_10. URL: https://doi.org/10.1007/978-3-540-33037-0_10 (visited on 04/13/2023).