LED-Installation zur Visualisierung von Sortieralgorithmen

lab:prepare WS1920

Lysanne Passek Stefan Insam

29. April 2020

1 Kurzbeschreibung

Sortierte Listen sind in der Computertechnik sehr wichtig. Sie ermöglichen beispielsweise das effiziente Durchsuchen von Datenmengen und dienen dem Menschen zur Übersichtlichkeit. Es existieren einige verschiedne Vorgehensweisen, die eine ungeordnete in eine geordnete Liste überführen, die diese Installation veranschaulichen soll.

Die verschiedenen Charakteristika unterschiedlicher Sortieralgorithmen werden visualisiert durch RGB-LEDs, die einzeln ansteuerbar sind. Aus der zufälligen Anfangsverteilung der RGB-Farben der einzelnen Pixel entsteht Schritt für Schritt eine farbliche Sortierung. Für die Anordnung diser Sortierung ergeben sich durch die Geometrie des Objektes verschiedene Möglichkeiten sowie auch für die Sortierung der dreidimensionalen Elemente (bestehend aus ihrem jeweiligen Anteil für rot, grün und blau) selbst, da sich hierfür beliebige Ordnungsrelationen definieren lassen. Weiterhin lassen sich interessante Grenzfälle betrachten. Zum Beispiel gibt es Algorithmen, die eine bereits sortierte Liste sofort als solche erkennen und somit in diesem Fall eine sehr kurze Laufzeit haben, aber auch welche, die in diesem Fall besonders lang brauchen.

2 Theorie

2.1 Vergleichsbasierte Sortieralgorithmen

Vergleichsbasierte Sortieralgorithmen sortieren eine Liste mithilfe paarweiser Vergleichs- und Tauschoperationen (compare & swap).

Ein Listenelement kann im Allgemeinen beliebigen Datentyps sein, solange sich ihm ein *Sortierschlüssel* zuordnen lässt, also ein numerischer Wert, der entweder das Element selbst sein kann, ein Attribut dessen oder sich aus seinen Charakteristika berechnen lässt. Eine Liste aus Zahlen wird vermutlich nach der Größe der Zahlenwerte sortiert werden, während mehrdimensionale Objekte mehrere Möglichkeiten für den Sortierschlüssel bieten, beispielsweise den Mittelwert mehrerer numerischer Dimensionen oder eine bestimmte einzelne Dimension.

Für den Vergleich zweier Elemente bedarf es außerdem einer geeigneten *Ordnungsrelation*. Diese Relation definiert für jeweils zwei gegebene Elemente, welcher Sortierschlüssel der größere bzw. kleinere ist.

2.2 Sortierung von RGB-LEDs

In diesem Projekt sind RGB-LEDs die zu ordnenden Listenelemente. Eine solche LED setzt sich aus jeweils einer roten, grünen und einer blauen LED zusammen, deren Mischungsverhältnis die LED in einer bestimmten Farbe leuchten lässt. Die digitale Repräsentation der Farbe des Lichtes der LED ist ein dreidimensionaler Vektor (i_R, i_G, i_B) mit ganzzahligen Einträgen $i_{\{R,G,B\}} \in [0, 255]$. Es gibt verschiedene Möglichkeiten, diesem Vektor nun eine Größe zuzuordnen, anhand derer er sich mit anderen vergleichen lässt. Mithilfe dieser Zuordnung kann eine beliebige Anordnung von LEDs als eine Liste von Zahlen aufgefasst werden, was die Anwendung algorithmischer Sortierung ermöglicht.

Es ist natürlich auch möglich, nach nur einer der drei Grundfarben zu sortieren, also die LED beispielsweise nur nach ihrem jeweiligen Blauanteil zu sortieren.

2.3 Stabilität und Komplexität

Ein Sortierverfahren heißt *stabil*, wenn bei gleichen Sortierschlüssel zweier Elemente die vorherige Reihenfolge dieser beiden Elemente beibehalten wird. [2]

Ein wichtiges Charakteristikum eines Sortierverfahrens ist seine Laufzeit- und Speicherkomplexität. Je weniger Zeit und Speicher ein Verfahren benötigt, desto effizienter ist es. Zur Abschätzung der Laufzeit betrachtet man den *worst case*, *average case* bzw. den *best case*, die ein Maß für den Laufzeitzuwachs bei größerem Input darstellen. Für die benötigte Menge an Speicher lassen sich auch Grenzen bezüglich worst-, best- und average-case angeben sowie ob er *in-place* operiert. Das bedeutet, dass der Speicherbedarf zusätzlich zum Input nicht von diesem abhängig ist. Ein in-place-Algorithmus darf also einzelne Elemente in einer Variablen abspeichern, jedoch nicht die Liste oder Teile dern Liste in den Speicher kopieren.

Mithilfe des *divide and conquer-Prinzips* lässt sich durch rekursives Aufteilen der Liste eine deutliche Laufzeitverbesserung erzielen. Es ist jedoch nicht möglich, eine Liste aus n Elementen schneller als in $n \log(n)$ Operationen zu sortieren, sodass $n \log(n)$ eine untere Schranke für die Laufzeitkomplexität vergleichsbasierter Sortieralgorithmen darstellt. [1]

2.4 Nicht vergleichsbasierte Algorithmen

Nicht vergleichsbasierte Algorithmen stellen spezielle Anforderungen an die Struktur der zu sortierenden Elemente; es lassen sich somit auch Sortierungen

in linearer Zeit erzielen. Oft basieren sie auf der Gruppierung der Elemente.

3 Bauplan

Das Grundgerüst der Installation bilden vier Holzplatten in Form gleichseitiger Dreiecke, die zu einem Tetraeder zusammengesetzt werden. Drei dieser Dreiecksflächen weisen möglichst gleichverteilte Löcher auf, in denen individuell ansteuerbare RGB-LEDs vom Typ WS2812b eingefasst sind. Das vierte Dreieck bildet den Boden der Konstruktion und das Innere bietet Platz für die Elektronik. Über jeder der LEDs befindet sich ein Tischtennisball, der die Funktion eines Lampenschirmes übernimmt, durch seine Beschaffenheit jeden Pixel größer wirken lässt und diffuses Licht erzeugt, welches angenehmer für die Augen ist als direktes Licht.

Damit lassen sich die Sortierungen der Pixel in vielfältiger Weise anhand ihrer dreier Farbkomponenten anordnen. Für die Nutzung der drei Dreiecksflächen kommen folgende Szenarien infrage:

1. Auf jeder Fläche wird ein anderer Algorithmus gezeigt.
2. Auf jeder Fläche wird synchron derselbe Algorithmus gezeigt.
3. Die drei Flächen bilden eine gemeinsame Oberfläche, auf denen ein Sortieralgorithmus gezeigt wird.

Mithilfe des Skriptes calc_led_nmbr.py lässt sich die Anzahl benötigter LEDs in Abhängigkeit von der Kantenlänge des Tetraeders bestimmen.

Für die Steuerung benutzen wir einen ESP32. Die LEDs, die wir verwenden sind vom Typ WS2812b, die sich in Reihe geschaltet durch ein individuelles Clockmuster einzeln ansteuern lassen. Bei 1000 in Reihe geschalteten LEDs muss vermutlich wegen des Spannungsabfalls über größere Distanz mehrfach frischer Strom zugeführt werden. Im folgenden (sehr schematischen) Schaltplan (siehe Abbildung 1) sind diese LED-Intervalle durch i bezeichnet.

4 Materialien

- 4 dreieckige Holzplatten
- WS2812b, 1000 Stück
- ESP32
- Stromversorgung
- Tischtennisbälle ø 25 cm, 1000 Stück
- sonstige Elektronikbauteile

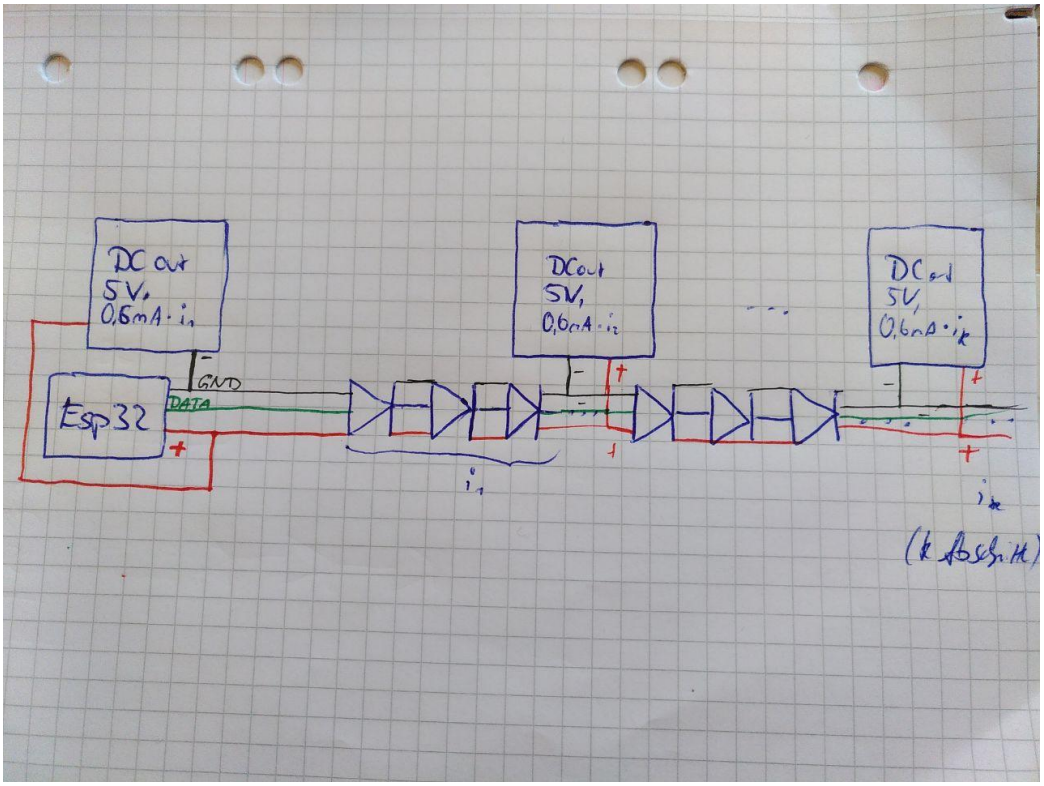

Abbildung 1: Schematischer Aufbau der elektrischen Schaltung

5 Beschreibung des Aufbaus

5.1 Software

Unser Vorhaben ist es, so viele vergleichsbasierte Sortieralgorithmen zu implementieren wie möglich. Die Visualisierung nicht-vergleichsbasierter Sortieralgorithmen erfordert eine Tabellenansicht, für die wir bisher noch keine adäquate Möglichkeit der Darstellung in unserem Format gefunden haben.

Für die vergleichsbasierten Sortieralgorithmen existieren Attribute und Prozeduren, die allen gemein sind (z. B. dass swap- und compare-Berechnungsschritte so lange durchgeführt werden bis die Liste sortiert ist). Dies ist abgebildet durch die sorter-Klasse. Die einzelnen Algorithmen selbst sind jeweils in Klassen implementiert, die von der sorter-Klasse erben und lediglich die individuellen Schritte implementieren. Sie bestehen jeweils aus den Funktionen:

- **setUp**: wird einmal am Anfang aufgerufen

- **keepGoing**: $\begin{cases} \text{True} & \text{falls Liste unsortierte Elemente enthält} \\ \text{False} & \text{falls Liste sortiert ist} \end{cases}$
- **step**: Implementiert jeweils einen Berechnungsschritt
- **tearDown**: wird einmal am Ende aufgerufen.

Die zu implementierenden Algorithmen müssen in diese Struktur umgeschrieben werden. Das Dokument main.py übersetzt die Einstellungen des Users in die Struktur des Programmes, instanziiert konkrete Sortieralgorithmen und führt sie aus. Im code-Ordner befindet sich der Code der bisher implementierten Sortieralgorithmen sowie das Skript zur Berechnung benötigter LEDs, um einen Tetraeder zu bestücken.

5.2 Hardware

Der Bau der Hardware musste (u. a.) wegen der Pandemie nach ersten Tests leider ausgesetzt werden, wird aber so bald wie möglich fortgesetzt. Dadurch werden sich auch Änderungen in der Materialliste oder bei Details im Aufbau ergeben. Diese Dokumentation wird dann entsprechend erweitert werden.

Wir haben die 1000 LEDs getestet und herausgefunden, dass alle funktionieren, einen Probe-Lasercut (siehe Abbildung 2) zum Einfassen der Tischtennisbälle angefertigt und die Sortieralgorithmen an einem (nicht verbauten) LED-Streifen getestet. Auch an visuellen Effekten wurde gearbeitet, wie das Aufblitzen jener LEDs, die grade verglichen oder getauscht werden oder softe Farbwechsel.

6 Probleme, good to know/Lessons Learned

Der Bau der Installation ist noch nicht vollführt worden. Nichtexistenz ist für eine Installation natürlich ein sehr fundamentales Problem. Es wird aber beseitigt werden, sobald wir die Möglichkeit haben, gemeinsam zu bauen und Zugang zu einem Lasercutter. Es hat sich als sinnvoll erwiesen, sich früh um eine vernünftige, modulare Programmstruktur zu bemühen. Dies ermöglicht besseres kollaboratives Arbeiten und auch beispielsweise automatisierte Tests von Anfang an.

Abbildung 2: Prototyp für den Lasercut

Literatur

[1] T.H. Cormen C.E. Leiserson R.L. Rivest C. Stein. *Introduction to Algorithms*. 2002.

[2] M. Dietzfelbinger Kurt Mehlhorn P. Sanders. *Algorithmen und Datenstrukturen*. Springer Vieweg, 1 edition, 7 2014.