Programmdokumentation MRT Praktikum 3: Animation

Gliederung:

1. Einführung
2. Projektablauf (zeitlicher Ablauf, Aufgabenverteilung)
3. Vorstellung des Programmes
4. Probleme / Schwierigkeiten bei der Bearbeitung der Aufgabenstellung
5. Einführung

(kurz Aufgabenstellung vorstellen)

1. Projektablauf:

Die Praktikumsaufgabe wurde um den Jahreswechsel 22/23 bearbeitet. Im ersten Schritt machten sich alle Teammitglieder mit der Aufgabenstellung vertraut. Anschließend wurden in einer Gruppenbesprechung am 22.12.2022 grundlegende Punkte zu Bearbeitung der Aufgabe besprochen.

Dabei wurde die Datenstruktur „laufzeitdaten“ festgelegt und die Interpretation des Arrays für den Puffer besprochen. Weiter wurde der grundsätzliche Ablauf des Programmes diskutiert und festgelegt.

Abschließend wurde die Praktikumsaufgabe in kleinere Arbeitspakete unterteilt und gleichmäßig im Team verteilt:

* Hauptprogramm und Benutzereingabe: Justus Krenkel
* Einlesen der Datei: Kleon Dingeldein
* Berechnung des nächsten Puffers: Martin Blümel
* Ausgabe des Bildes: Theo Röhl

In den folgenden Tage wurden die jeweiligen Arbeitspakete von den Teammitglieder bearbeitet. Dabei fand ein regelmäßiger Austausch in der Gruppe statt, um Probleme und Schwierigkeiten bei der Programmierung gemeinsam zu besprechen.

Am 02.01.2023 war eine letzte Besprechung geplant. In dieser wurden die einzelnen Programme zusammengefügt wurden und ein finaler Funktionstest durchgeführt wurde.

1. Vorstellung des Programmes:

3.1. Main und Benutzereingabe

3.2. Datenstruktur „laufzeitdaten“

3.3. Einlesen der Datei

Das Einlesen einer Konfigurationsdatei ist im C-Modul config.c implementiert. Über den Funktionsaufruf *x=einlesen(“filename.txt“)* wird die Variable x des Types „laufzeitdaten“ mit den eingelesen Informationen aus der filename.txt befüllt.

Nach Funktionsaufruf springt das Programm über die Headerdatei *config.h* in das gleichnamige C-Module. Dort ist der grundlegende Programmablauf in der Funktion *struct laufzeidaten einlesen (const char\* filename)* implementiert:

1. Öffnen der Datei (ausschließlich zum Einlesen von Informationen). Der Name der Datei wird über den Funktionsaufruf im Hauptprogramm an *config.c* übergeben. Misslingt das Öffnen wird über *perror* eine Fehlermeldung angezeigt.
2. Einlesen der Informationen zu Spalten, Zeilen, Schritt, Gesamtschritte, Pausen zwischen den Animationsschritten in der Funktion *nummerneinlesen(input\_file):*

Diese Funktion bekommt den Pointer auf das geöffnete Dokument übergeben. Der Curser befindet sich an erster Stelle. In einer while-Schleife wird ein String eingelesen und dieser mit den Schlüsselbegriffen („Zeilen:“, „Spalten:“, usw.) verglichen. Anschließend wird eine funktionsinterne Variable der Struktur *laufzeitdaten* mit der nun eingelesenen Zahl beschrieben. Dadurch wird bei mehrfacher Angabe einer Information die letzte Angabe abgespeichert. Die Schleife wird abgebrochen sobald die Funktion erkennt, dass als nächstes der Animations-Puffer eingelesen werden soll.

1. Einlesen des Puffers in der Funktion *puffereinlesen(data.X, data.Y, input\_file):*

Diese Funktion liest den Puffer der Datei ein und hinterlegt ihn in einem Zwischenspeicher. Dafür werden die maximalen Maße des Bildschirmes übergeben und der Pointer auf die Einlesedatei. Dessen Curser befindet sich zu Funktionsbeginn vor dem ersten Zeichen des Puffers.

Der Zwischenspeicher ist von der Datenstruktur *eingelesenerpuffer* – einer Struktur die *laufzeitdaten* ähnlich ist. Sie umfasst jedoch nur die für diesen Schritt benötigte Zeilen- und Spaltenlänge sowie ein Array für den eingelesen Puffer. Sie ist in *config.h* deklariert, da nur *config.c* diese Datenstruktur verwendet.

Zunächst wird in das dynamisch erzeugte Array *einlesespeicher* die erste Zeile eingelesen und dessen Länge ermittelt.

Anschließend wird Speicherplatz für den Zwischenspeicher allokiert. Hintergrund dieser etwas umständlicheren Programmierung ist, dass der Puffer deutlich kleiner als der gegebene Bildschirm sein kann. Dadurch wird weniger Speicher zum Einlesen benötigt.

Anschließend wird der Animations-Puffer zeilenweise in *einlesespeicher* eingelesen und im Zwischenspeicher (*zs.ptrzs*) abgespeichert.

Der nicht mehr benötigte *einlesespeicher* wird deallokiert und die Funktion gibt den eingelesenen Puffer sowie dessen Abmaße zurück.

1. Schließen der Einlesedatei.
2. Meldung falls der eingelesen Puffer nicht zentrierbar ist: Aus den Maßen des eingelesen Puffers und den Maßen des Bildschirmes wird nun geprüft, ob das Bild mittig platziert werden kann. Es wird eine Meldung ausgegeben, falls dies nicht möglich ist.
3. Zentrieren des Puffers in der Funktion *zentrieren(zwischenspeicher, data.X, data.Y):*

Diese Funktion setzt den eingelesen Puffer mittig in den eingelesen Bildschirm. Dafür wird der benötigte Speicher allokiert und in einer doppelten for-Schleife jedes Pixel des Bildschirmes geprüft, ob es von einem Pixel des eingelesen Puffers beschrieben werden muss.

Rückgegeben wird ein Array das den Bildschirm mit dem zentrierten eingelesen Puffer beschreibt. Das Array berücksichtig außerdem den zur weiteren Berechnung benötigten Rand.

1. Rückgabe der befüllten Datenstruktur an das Hauptprogram.

3.4. Berechnung des nächsten Animationsschrittes

3.5. Ausgabe des Bildes

4. Probleme / Schwierigkeiten bei der Bearbeitung der Aufgabenstellung

1. VM, Git einrichten
2. (Übergang von VS zu VM)
3. Umgang mit Pointer (bei strinugs, Dateien,…)
4. .c dateien nicht in .h dateien (Umgang mit c. und h. dateien)