Hexaposer

Technische Dokumentation

Von:

Bager Bingöl (2239338)
Wladimir Nabok (2238778)
Klaus Bochmann (2250591)

Betreuer:

Prof. Dr. Andreas Plaß
Jakob Sudau

13. Januar 19

Audio-Video Programmierung
Studiengang Media Systems (B.Sc.)

Hochschule für angewandte Wissenschaften Hamburg /
Hamburg University of Applied Sciences

Department Medientechnik
Fakultät Design, Medien und Information

Vorwort

Das Projekt Hexaposer entstand im Rahmen der Veranstaltung Audio-Video Programmierung an der Hochschule für angewandte Wissenschaften Hamburg.
Ziel des Kurses war es, erste Erfahrung mit C++ und der WebAudio API zu erlangen und diese in unserem Projekt umzusetzen und vorzustellen.

Unser Ziel war es ein Projekt mit der Programmiersprache C++ und JavaScript zu realisieren. Zur Visualisierung wird eine Webseite, mit HTML und CSS, erstellt.
Das Team ist entsprechend unserer Fähigkeiten gut aufgestellt.

Technische Umsetzung

Frontend

Das Frontend besteht aus drei Spalten, vier Hauptbestandteilen und wurde zuerst mit HTML/JS/CSS realisiert und anschließend von HTML auf PUG umgeschrieben.

Logo

Das Logo besteht aus zwei Teilen. Einmal dem gelblichen Hexagon und dem Projekttitel “Hexaposer”.
Es wurde in Adobe Photoshop CC 2018 gestaltet und realisiert.

Hexagon

Die Hexagone werden in der “hexagon.css” definiert und sind ausschließlich im CSS geschrieben. Ein Hexagon besteht aus drei Rechtecken, die, mit den Pseudoelementen “::before” und “::after” um 60°, 0° und -60° rotiert wurden, sodass ein Hexagon abgebildet wird.

Es wird eine Grundmelodie (MediaElement. Source) abgespielt, die mit jedem gelegten Hexagon verändert werden kann.
Jedes Hexagon ist einer Node aus der WebAudio API zugeordnet. Insgesamt gibt es sieben Hexagone die gelegt werden können, wobei eine Mindestanzahl von vier Hexagone gelegt werden müssen. Legt man das blaue, grüne, rote und türkise Hexagon, so werden die nicht gelegten Hexagone (Grau, Orange und Lila) durch eine if-Anweisung in der “hexHidden.js” auf “visibility=hidden;” gesetzt.

Play-/Pause Knopf

Mit einem Klick auf den Play-Knopf wird nach dem Legen der Hexagone eine Grundmelodie abgespielt. Über das legen eines Hexagons kann diese Grundmelodie verändert werden. Je nachdem welches Hexagon man gelegt hat, variiert die Grundmelodie. Dementsprechend wird bei dem Pause-Knopf die Musik pausiert.

Sidebar/Legende

Mit einem Klick auf die “BurgerBox” öffnet sich die Sidebar und es ist eine Legende zu sehen. Die Legende zeigt eine Auflistung von allen Hexagone und erklärt deren Funktion.

Nodes

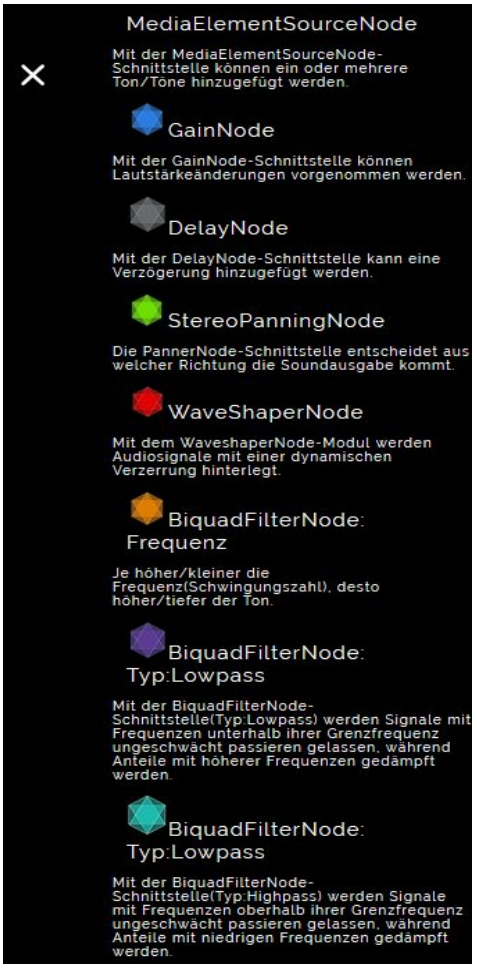

Abbildung 1: Funktionalität Hexagone

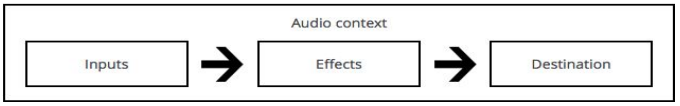

Abbildung 2: Workflow für WebAudio

Backend

Wird die index.js initialisiert werden im ersten Schritt folgende Module geladen: Express, die hextractor.exe als Kindprozess in der Variablen exec. (Siehe Image Processing) Sowie die colorDefine.js in der Variablen colorConv. Nun wird in der Variablen app express initialisiert und pug als engine festgelegt. Anschließend wird die von uns verwendete Ordnerstruktur (/images, /fonts, /css und /js) übergeben.

Daraufhin wird die Funktion getHexagons() aus der ColorDefine.js aufgerufen welche die vom Kindprozess ins Hauptverzeichnis gelegte .json einliest, parsed und den Inhalt als Hexagonobjekte im hexagons[] Array abspeichert. Jedes Hexagon hat hierbei einen String mit dem Hexadezimalwert für die Farbe, sowie zwei Integerwerte für jeweils die X- und Y-Position. Im Anschluss werden die Farbwerte in einer for-Schleife ausgelesen und mittels der colorPrepare() ihren später im Frontend verwendeten Farbwörtern zugeordnet.

Farbzuweisung

Die colorPrepare() ist eine Funktion die nacheinander folgende, in der colorDefine.js definierten Funktionen ausführt. Zuerst wird der hexadezimale String in der Variablen proc zwischengespeichert und anschließend über convertColor() in seine drei rgb Kanäle umgewandelt. Über die colorAdjust() wird nun jeder Kanal eingelesen und über closest() näherungsweise einem der im colorRange[] Array hinterlegten Absolutwerte zugewiesen. Abschließend wird ein leerer String newHex erstellt, in welchem dann über eine for-Schleife durch den Aufruf der rgbToHex() wieder ein hexadezimaler Wert gebildet wird. Dieser wird von colorPrepare() zurückgegeben.

Die anschließende Zuweisung der hexadezimalen Werte zu Farbwörtern erfolgt, da wir mit einer überschaubaren Farbmenge arbeiten, über eine einfache if-Abfrage. Das so aufbereitete hexgons[] Array wird an die dem Template dynamic.pug übergeben.

Template

Das Template erstellt unser Frontend über eine Standard HTML Formatierung. Es kommt JQuery zum Einsatz, es werden unsere css Dateien geladen und unser Logo hinzugefügt. Kern des Templates ist eine For-Schleife, welche für jedes Objekt aus der übergebenen hexagons[] einen div Container erstellt, das Farbwort sowie die X-/Y-Position ausliest und dem Container als id bzw. class zuweist. Danach wird noch der Play/Pause Button hinzugefügt, die soundPlayback.js und hexHidden.js geladen und die Legende als Burgermenü angelegt und die Texte hinzugefügt.

Image Processing

Version und Environment

Für den Image Processor benötigt man die openCV Library in Version 3.4.4 und nlohmann_json zum Schreiben von json files. Zum Generieren der Konfigurationsfiles habe ich Cmake verwendet und MSBuild.exe, zum erzeugen der Builds. Wir haben die x86 Architektur verwendet und die Cmake File wurde mit dem cxx Standard 17 geschrieben. Der Source Code wurde mit der Clion IDE, Version 2018.3.1 geschrieben.

Ablauf

In der Image Processing Komponente (Hextractor) unserer Applikation werden die Bildinformationen ausgewertet und daraus eine .json file geschrieben, welche dem HTTP Server Backend zugänglich ist.
Das folgende Flussdiagramm beschreibt den groben Ablauf der hextractor.exe:

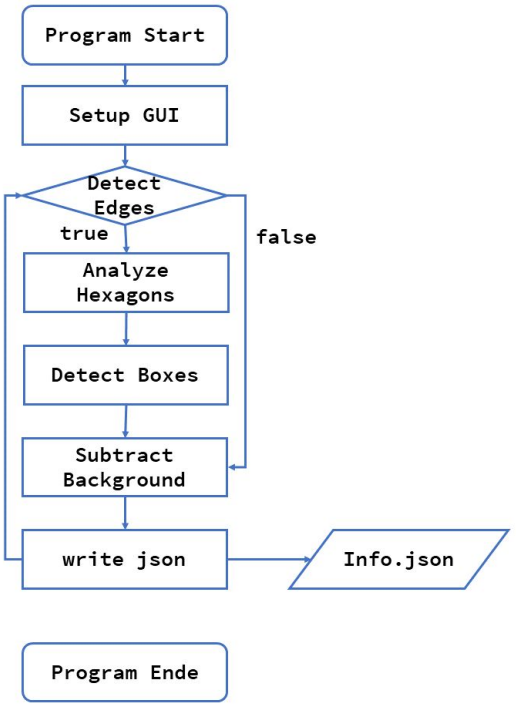

Abbildung 3: Flussdiagramm hextractor.exe

Zunächst wird die GUI mit den integrierten OpenCV GUI Funktionen wie **create-Trackbar()** zum Erstellen von diversen Sliders und **namedWindow(),** zum erstellen der Fenster ausgeführt. Unter anderem wird auch initial die **detectBoxes()** Funktion als Callback Funktion, beim Rendern eines Sliders ausgeführt. Dann wird die main loop des Image Processor ausgeführt, in welcher zunächst die Kameradaten abgegriffen werden. Anschließend werden nach einem erfolgreichen Detektieren der Kanten (**detectEdges()**), die Hexagone analysiert (**AnalyzeHexagons()**) und Boxen detektiert (**detectBoxes()**).
Der **Analyze Hexagons** Prozess setzt die RGB Darstellung in Hex der durchschnittliche Farbe einer Box als Farbe des jeweiligen Hexagons. Außerdem wird die Mitte dieser Box als Position für das jeweilige Hexagon gesetzt.
Nach diesem Schritt werden die Boxen erneut für die nächste Schleifeniteration ermittelt, im **Detect Boxes** Prozess. Dabei werden mit einem Schwellwert Konturen erkannt oder nicht erkannt, und Boxen um diese gelegt, wenn sie abgeschlossen sind.
Falls im anderen Fall die **Detect Edges** Kondition ein false ausgibt, dann werden diese zwei genannten Schritte ausgelassen.
Als nächstes erfolgt eine **Background Subtraction**, dabei wird wie aus der Vorlesung bekannt, der Pixel des aktuellen Frames vom Pixel des ersten Frames subtrahiert und der Schwerpunkt der Bewegung des aktuellen Frames ausgerechnet. Dieser Schritt ist relevant für einen Robusten Ablauf der Main Loop, wenn nämlich zu viel Bewegung im Bild erkennbar ist, dann wird über die Abweichung des Schwerpunkt zur Mitte, wird der Boolean isMoving gesetzt der die Prozesse Analyze Hexagons und Detect Boxes einschränkt.
Am Ende einer Iteration der main loop, wird im **write json** Prozess die Info.json Datei anhand des vektors hexagons mit ihren Farb- und Positionsdaten überschrieben.
Die Schleife beginnt nun wieder beim **Detect Edge** Prozess...
Beim forcierten Schließen der Applikation, wird dieses beendet, sei es durch den externen schließen des Konsolenfensters, indem das Programm ausgeführt wird oder dem Fehlen von Arbeitsspeicher.

Abbildungsverzeichnis

Quellenverzeichnis

fl1p, LordZnarf: Das Web Audio Konzept und Benutzung (2016),
URL:
https://developer.mozilla.org/de/docs/Web/API/Web_Audio_API (Stand: 23.10.2018)

connorshea, fscholz, chrisdavidmills, looshi, jpmedley, erikadoyle, dgashmdn, Sebastianz, teoli, andrew.punnett, padenot, ChrisMondok, kscarfone, Sheppy: BiquadFilterNode (2018),
URL:
https://developer.mozilla.org/de/docs/Web/API/BiquadFilterNode (Stand: 25.10.2018)

fscholz, chrisdavidmills, abbycar, Sheppy, jpmedley, teoli, Sebastianz, padenotmoz, kscarfone: StereoPanner (2017),
URL:
https://developer.mozilla.org/en-US/docs/Web/API/StereoPannerNode (Stand: 12.11.2018)

Sudau, Jakob: AVPRG (2018),
URL:
https://github.com/jakobsudau/AVPRG
(Stand: 23.10.2018)

ForbesLindesay: Pug - robust, elegant, feature rich template engine for Node.js (2018),
URL:
https://pugjs.org/api/getting-started.html
(Stand: 30.11.2018)