Schallbild - Dokumentation

Struktur dieses Dokuments

Beschreibung des Projekts

Schall

Visualisierung

Technische Details

Beschreibung des Projekts

Schallbild ist eine interaktive Multi-Laptop Installation. Mittels eines Interfaces, ist es möglich simple Waveforms zu erstellen, deren Frequenz und Amplitude zuverändern und diese Signal an alle verfügbaren Client Laptops zu verschicken. In Abhängigkeit der generierten Parameter, wird von dem versendeten Signal eine Visualisierung des Schals erstellt. Der Aufbau enthält die Anordnung der Laptops in einer gegebenen Position. Dadurch kennt jeder Client seine Position im Raum. Dies ermöglicht der Visualisierung sich über alles Laptops zu zeichnen. Ein Gesamtbild entsteht.

Schall

Mit dem Kursthema im Hinterkopf, eine Art Orchester mithilfe verschiedener Laptops zu erschaffen, fingen wir an uns über mögliche Herangehensweisen zu unterhalten. Nach längeren Diskussionen entschieden wir uns, als Grundlage unserer Visualisierung das Thema „Schall“ unter Berücksichtigung seiner physikalischen Gesetze zu behandeln. Dabei ist das Ziel sowohl eine abstrakte als auch eine korrekte Darstellung umzusetzen. Die abstrakte Gestaltung der mechanischen Ausbreitung soll dabei jedoch im Fokus stehen.

Unsere Visualisierung der Schwingungen beschränkt sich auf drei primitive Wellenformen, die Sinus-Welle (sine wave), die Rechteck-Welle (square wave) und die Sägezahn-Welle (saw wave).

Es werden pro Phase Partikel entsandt, welche sich kreisförmig ausbreiten. Jede Wellenform hat ihre eigenen Partikel. Somit werden die der Sinus-Welle durch weiße Kreise visualisiert, die der Rechteck-Welle durch jeweils vier weiße Rechtecke und die Sägezahn-Welle durch ein Rechteck mit Farbverlauf von schwarz nach weiß. Die Ausbreitung der Partikel findet kontinuierlich statt, es sei denn, sie prallen von einer der Wände ab oder sie kollidieren miteinander.

Ausschlaggebend für unsere Visualisierung sind die Interaktionen der Partikel einer Schwingungsart. Beispielsweise interagieren Sinus-Wellen mit Sinus-Wellen, Rechteck-Wellen mit Rechteckwellen und Sägezahn-Wellen mit Sägezahnwellen.

Im Falle eines Aufeinandertreffens zweier Partikel derselben Art innerhalb eines bestimmten Frequenzbereichs verhalten sich diese abhängig von ihrer Amplitude. Treffen zwei positive Amplituden aufeinander, interferieren diese. Sie überlagern sich. In unserer Visualisierung wird dies beispielsweise durch zwei weiß-gefüllte Kreise (sine wave) symbolisiert, welche sich im Interferenzfall innerhalb des Frequenzbereichs vereinen und vergrößern. Überlagern sich zwei negative Amplituden, passiert dies mit weiß-konturierten Kreisen. Im Falle einer Kollision eines Partikels mit negativer und eines mit positiver Amplitude erfolgt wie in der Physik eine destruktive Interferenz, das heißt, dass sich die Partikel gegenseitig auslöschen.

Visualisierung

Um die Idee, die wir über die Visualisierungsmöglichkeiten von den verschiedensten Schallwellen hatten, eine größtmögliche Wirkung zu geben, entschieden wir uns früh in einem monochromatischen Raum zu arbeiten. Dabei waren weiß und schwarz innerhalb der Darstellung bei den Client-Laptops nur vorzufinden. Jedoch entschieden wir uns, den Master zwar auch eher grau zu halten, ihm aber gleichzeitig durch die Hinzugabe einer Akzentfarbe von den restlichen Laptops abzuheben. Damit wollten wir eine stärkere Trennlinie zwischen den Clients, die passiv in unserem virtuellen Raum stehen, und die aktive Rolle des Masters setzten. Mit dieser Trennlinie konnten wir auch so eine gewisse Dynanmik innerhalb der Perfomance erzielen. Jeder der Zuschauer und Nutzer unserer Perfomance wusste sofort, welchen Laptop er nutzen musste, um eine gewünschte Reaktion innerhalb des virtuellen Raumes zu erzeugen.

Diese rein technische Visualisierung, wurde durch die Form des Aufbaus noch weiter unterstützt. Hierbei waren wir uns schon sehr früh einig, dass sich eine umgedrehte V-Form am Besten dafür eignete, jeden der vielen Bildschirme sehen zu können, sowie die Hauptidee unseres Programmes wiederzugeben. Um dabei noch einen größeren Rahmen des virtuellen Raumes darzustellen, entschieden wir die Höhe der einzelnen Podeste zu unserem Vorteil zu nutzen. Für die Höhe entschieden wir uns für den ersten Laptop, der die Spitze der Form bildete, das höchste Podest zu nehmen. Je weiter die Laptops von der Spitze wegstehen, desto niedriger werden die Podeste. Der Maste bekommt, wie bei der digitalen Visualisierung, sein eigenes Podest. Dieses steht vor der Form, die die Clients bilden, damit sich der Master auch räumlich von den Clients abheben kann.

Technische Details

Master

OSC

Der Master bestimmt wie groß der Raum in Metern ist und wie viele Laptops an der Installation teilhaben. Im ersten Schritt empfängt der Master die IP Adresse der Clients und trägt sie in eine StringList ein. Dies passiert nur, wenn die IP Adresse noch nicht besagter List vorhanden ist.

//Listen to hey

if (theOscMessage.checkAddrPattern("/hey") == true) {

ip = theOscMessage.get(0).stringValue();

println(ip);

//Check if current ip is already in StringList ipAdresses

for (int i = 0; i < ipAdresses.size(); i++) {

String currentId = ipAdresses.get(i);

//If yes (ipStatus = true) --> ip won't get in list again

if (ip.equals(currentId)) {

ipStatus = true;

}

}

//If ip is not in ipAdresses (ipStatus = false) --> append current ip to ipAdresses

if (ipStatus == false) {

aliveIds[ipAdresses.size()] = false;

ipAdresses.append(ip);

}

//save hey

hey = theOscMessage.get(1).stringValue();

println(hey);

//Print ipAdresses

println(ipAdresses);

}

Zeitgleich schickt der Master ein Paket an alle Clients mit der eigenen IP Adresse (zur Identifizierung), Position in der StringList, Raumbreite -und Länge und Anzahl der Computer.

void sendId() {

//send id

OscMessage sendIP = new OscMessage("/id");

for (int i = 0; i < ipAdresses.size(); i++) {

if (!aliveIds[i]) {

sendIP.add(ipAdresses.get(i)); //#0

sendIP.add(i); //#1

sendIP.add(roomWidth); //#2

sendIP.add(roomHeight); //#3

sendIP.add(numComputers); //#4

oscP5.send(sendIP, remoteLocation);

}

}

}

Um sicher zustellen das alle Laptops noch im Netzwerk sind, wird die StringList mit allen ip Adressen alle 5 Sekunden zurück gesetzt. Der Master wartet auf ein Lebenssignal der Clients und trägt diese dann wieder in die Liste eine.

//Listen to alive

if (theOscMessage.checkAddrPattern("/alive") == true) {

id = theOscMessage.get(0).intValue();

aliveIds[id] = true;

}

Bei Benutzen des Interfaces und drücken des "Senden"-Knopfes wird je nach dem welche Einstellung und Parameter bestimmt wurden ein Paket an die Clients geschickt. Dieses Paket enthält auch die adressierte IP Adresse, die besagt auf welchem Client die Visualisierung stattfinden soll.

//send data (0 = type of wave; 1 = frequency; 2 = amp)

void sendData() {

OscMessage sendData = new OscMessage("/data");

if (squareOn == true) {

sendData.add(adressedId); //The id to which you should send it #0

sendData.add("squarewave"); //is square active? #1

sendData.add(squareFunction()[0]); //send frequency of square #2

sendData.add(squareFunction()[1]); //send amp of square #3

sendData.add(global\_velocity); //global velocity #4

sendData.add(duration\_sq); //time (how long should it send) #5

sendData.add(echoCounter); //counter #6

oscP5.send(sendData, remoteLocation);

}

if (sawOn == true) {

sendData.add(adressedId); //The id to which you should send it #0

sendData.add("sawtooth"); //is square active? #1

sendData.add(sawFunction()[0]); //send frequency of square #2

sendData.add(sawFunction()[1]); //send amp of square #3

sendData.add(global\_velocity); //global velocity #4

sendData.add(duration\_sw); // time (how long should it send) #5

sendData.add(echoCounter); // counter #6

oscP5.send(sendData, remoteLocation);

}

if (sinOn == true) {

sendData.add(adressedId); //The id to which you should send it #0

sendData.add("sinewave"); //is square active? #1

sendData.add(sineFunction()[0]); //send frequency of square #2

sendData.add(sineFunction()[1]); //send amp of square #3

sendData.add(global\_velocity); //global velocity #4

sendData.add(duration\_sin); // time (how long should it send) #5

sendData.add(echoCounter); // counter #6

oscP5.send(sendData, remoteLocation);

}

}

Die globale Geschwindigkeit wird über ein separates Paket an alle Clients gerichtet gesendet.

void sendVelocity() {

//get global velocity

global\_velocity = cp5.getController("global\_velocity").getValue();

OscMessage sendData = new OscMessage("/global\_velocity");

sendData.add(global\_velocity);

oscP5.send(sendData, remoteLocation);

}

Interface

Das Interface des Masters enthält im oberen Bereich einen Vorschaubereich der Visualisierung. Darunter befinden sich drei Osciliatoren mit den eingestellen Waveform "Sinewave", "Sawtooth" und "Squarewave". Hier lassen sich die Parameter mittels Drehreglern Frequenz, Amplitude und Dauer der Sendung einstellen. Im mittleren Teil liegt ein Slider der die globale Geschwindigkeit der Visualisierung steuert. Im unteren Bereich befindet sich der Knopf zur Sendung des eingestellten Signals.

Die Interface-Elemente sind der Library controlP5 von Andreas Schlegel entnommen.

Git

GitHub

Unstaged Changes

Stage All

documentation\large\_doc.md

location\Interface\control\_interface\control\_interface.pde

location\Interface\control\_interface\controllers.pde

location\Interface\control\_interface\sound\_functions.pde

location\Interface\receive\receive.pde

location\client\echoNew2750328053669398684.autosave

Staged Changes

Unstage All

See All Staged Changes

Commit message

Commit to master

72

Zwischenstand Schallbeschreibung. Wird noch Korrektur gelesen und bearbeitet.

Undo

1d

updated large doc

1d

large document added

1d

aufräumung

1d

final

1M

master final hopefully

1M

ForPresentation

1M

FastFinal

1M

Alle Visualisierungen ohne Collision

1M

Update client.pde

1M

large\_doc.md

masterFetchGitHubGit (6)2 updates