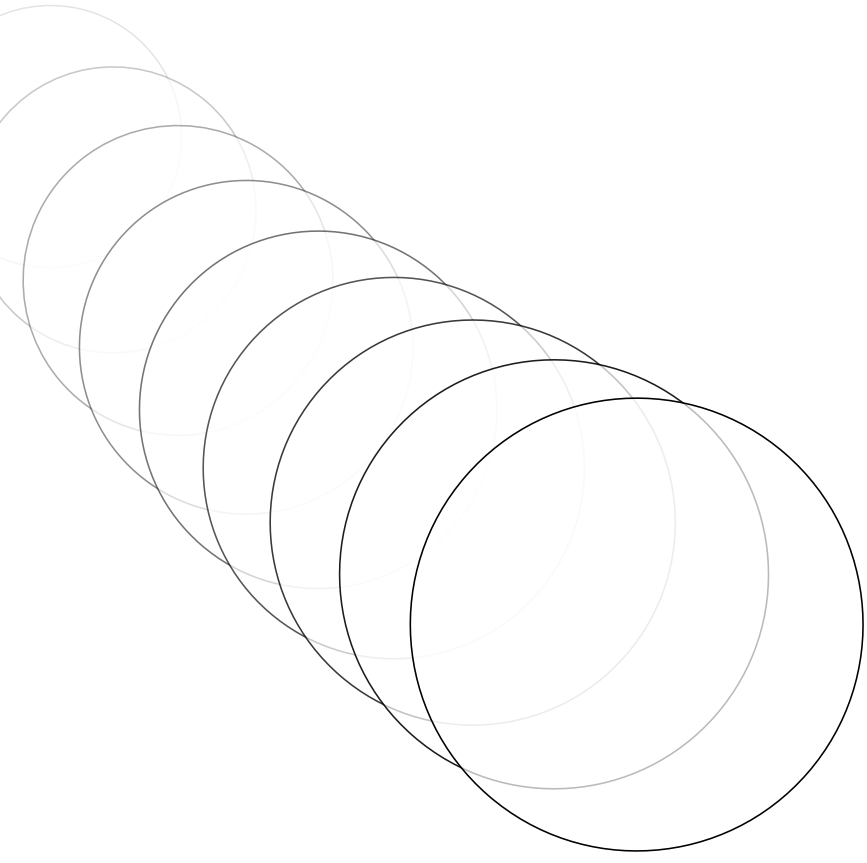
decrescendo

Benjamin Milde | Anna-Sophie Meyer



Benjamin Milde
Anna-Sophie Meyer
MD0511
6. Semester | MediaLab

Dozenten:
Eduard Mittermaier
Jakob Thomsen

Die Arbeit beschäftigt sich mit dem Zusammenspiel von Bewegung, Form und Klang. Dabei versuchen wir über generative Prozesse eine Synästhesie dieser Komponenten zu schaffen. Ein frei schwingendes Pendel erzeugt hierbei eine Soundstruktur, die zeitgleich über graphische Elemente visualisiert wird.

Die Bewegungen einer Tänzerin werden mit 3 Microsoft Kinect aufgezeichnet. Danach werden die Daten in 3DS Max[1] geladen und verfremdet.
Das 3D Model wird durch Punkte ersetzt, welche im Zusammenspiel noch immer einen sich bewegenden Menschen erkennen lassen.
Der Partikeleffect folgt den Bewegungen, passend zum Lied zerfallen die einzelen Elemente aber auch wieder, was dem Video seinen Flair gibt.

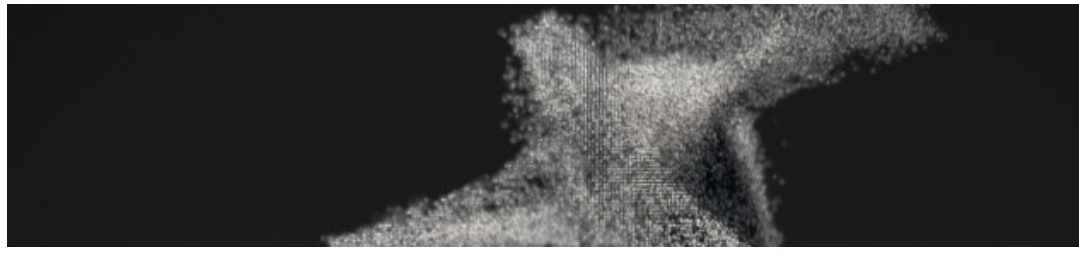
Ein für Nike entwickeltes System arbeitet ebenfalls mit den Bewegungen des Betrachters. Mit einer Kinect werden die Umrisse erkannt und weiter verfremdet, jeh nachdem wie extrem sich die Person bewegt. Durch die Tiefenempfindlichkeit der Kinect sind abstrahierte Höhen und Tiefen, dargestellt durch verschieden große Punkte, sichtbar. Diese Insterlation ist im Vergleich zum vorherigen keine Nachbearbeitung sondern geht direkt auf den Betrachter ein.

Durch das berühren eines Tuches wird der Sound in Dreiklängen verändert. Auch hier wird mittels einer Kinect erkannt ob jemand die gespannte Decke berührt, und wenn ja wie extrem hineingedrückt wird. Jeh nach Tiefe verändert sich der Klang stark oder schwach, allerdings immer in harmonischen Dreiklängen. Eine Visualisierung unterstützt die Musik und gleicht sich an, sobald man Einfluss auf die Melodie nimmt.

Fluid ist ein Projekt bei dem sich stark verdicktes Wasser auf Berührungen reagiert. Unter dem Tisch befindet sich ein Lautsprecher der Bass verschieden stark abspielt sobald jemand das Brett anfasst. Hierbei ist die Instalation so sensibel, dass sowohl die Position als auch die Anzahl der Finger berücksichtigt wird. Je größer die berührte Fläche ist umso lauter wird der Bass während die Position der Hand über die feinen Tonlagen bestimmt.

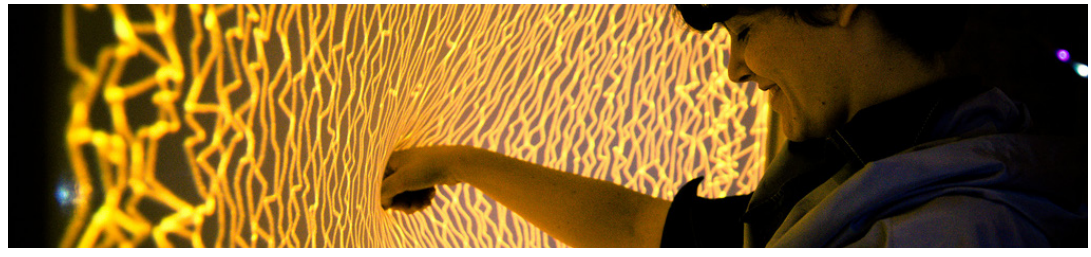
1 3Ds Max: von Autodesk entwickeltes 3D Programm

http://vimeo.com/38874664



http://vimeo.com/44391801

http://vimeo.com/54882144



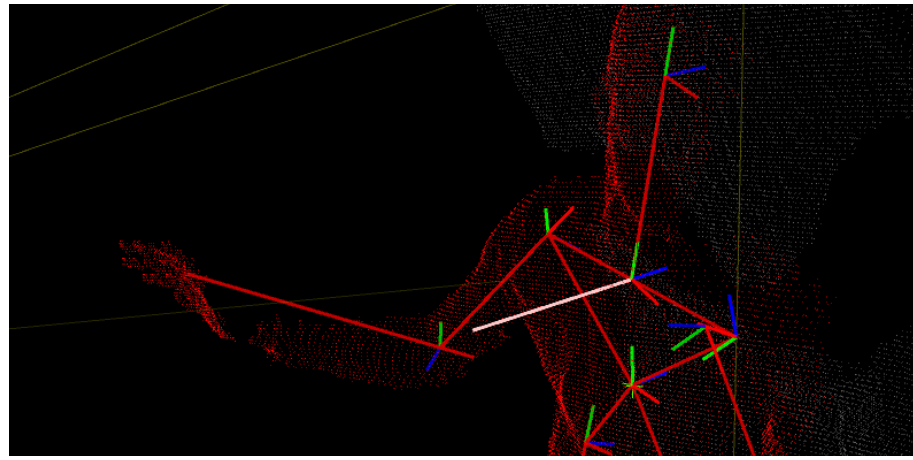
http://vimeo.com/58940104

Auf der Suche nach unserem Thema für die Arbeit sind uns viele Installationen und Werke begegnet, die Interaktion und Visualisierung mit Hilfe von Kameras aus ihrer 2 Dimensionalen Bindung an Bildschirme und Projektionen befreit haben.[1] Diese Ideen überzeugten uns sehr schnell ein Projekt zu machen, dass sich ebenso im dreidimensionalen Raum befindet und dort auch mit den Betrachtern interagiert.

Nach einigen eher technischen Versuchen mit einer Microsoft Kinect[2], die uns einen kleinen Überblick, für uns fassbaren Möglichkeiten gab, versuchten wir über verschiedene Musikstücke eine Idee für eine Visualisierung zu finden. Die Stimmung von „Jamie Woon - Shoulda Instrumental)“[3] inspirierte uns durch seine fließend-schwingenden Tonalitäten zur Nutzung eines Pendels. Das Pendel soll die entscheidende Rolle der Interaktion mit den Betrachter spielen. Im einfachsten Fall, dass er aktiv mit einwirkt, in dem er es neu anstößt wenn es zum Stillstand kommt.

Grafisch sollte die Bewegung des Pendels visualisiert werden. Nach ersten Überlegungen tendierten wir zu fließenden oder auch pulsierenden Formen, passend zur Musik. Die Darstellung sollte dann durch Parameter des Pendels, wie zum Beispiel Geschwindigkeit oder Ausschlag, verändert werden. Gleichzeitig sollten diese Parameter auch die Präsentation des Liedes verändern.

1 http://www.onformative.com/work/nike-fuel-station/
2 Microsoft Kinect: Infrarot- und Webkamera, die 3D Informationen auslesen und wiedergeben kann.
3 http://www.youtube.com/watch?v=iw5bancYmBk
4 Processing: Entwicklungumgebung für Software, die speziell für die Ansprüche von Designern und Grafikinteressierten entwickelt wurde.

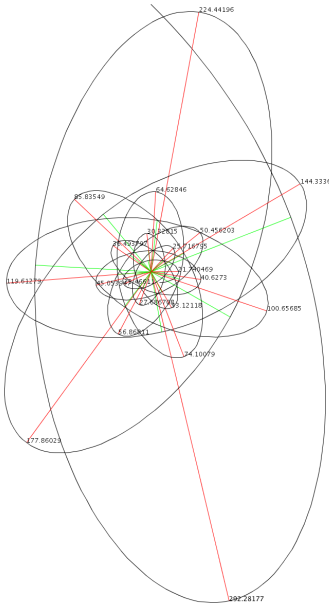


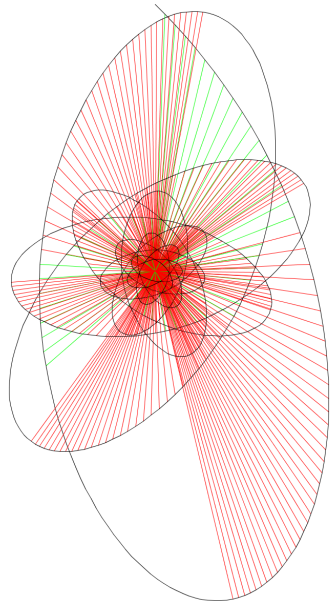
Die Kinect sollte uns die Bewegung des Pendels aufzeichen. Obwohl die Kinect ein 3D Abbild der Umgebung erstellt, haben wir die Kamera über dem Pendel positioniert, um Ungenauigkeiten in der Positionsbestimmung zu minimieren. Uns interessierte nur die Laufbahn des Pendels, somit war der Höhenunterschied des Pendels wärend der Schwingung zu vernachlässigen.

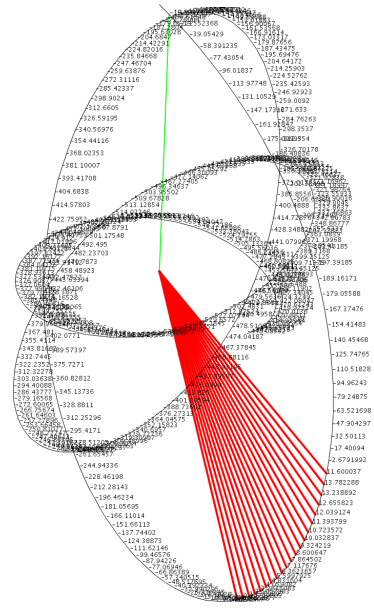
Für die Programmierung haben wir eine Open Source Bibliothek für Processing benutzt: SimpleOpenNI[1]. Diese bietet schon viele Funktionalitäten in Verbindung mit Kinect, jedoch ist diese im ersten Moment für die Xbox und deren Nutzer gedacht. Somit sind die nützlichsten Funktionen nur in Verbindung mit dem Körper des Spielers vorhanden. Deshalb mussten wir selbst einen Weg finden, das Pendel aus dem 3D Bild der Kinect zu extrahieren und die Umgebung auszublenden.
Da sich mit dem Pendeltracking über die Kinect schon ein paar Hürden gezeigt haben, haben wir für die ersten Experimente ein Programm entwickelt, dass versucht diese Pendelbahn zu simulieren.

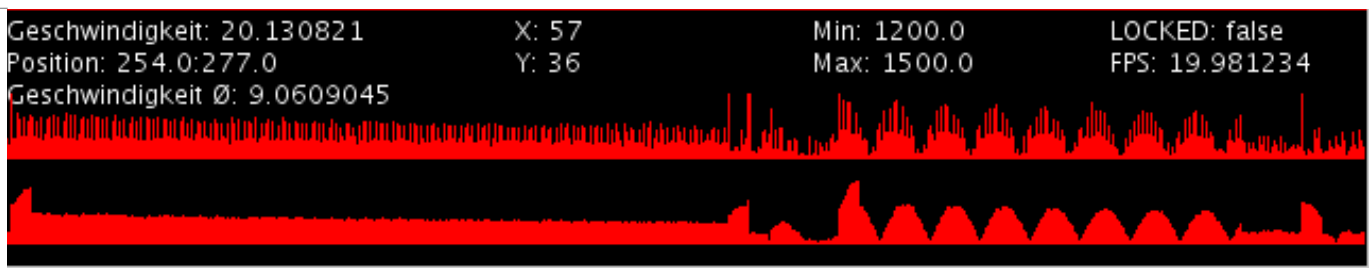
Anhand der Simulation konnten wir bereits testen, wie die Soundverarbeitung und Visualisierung funktionieren könnte. Dafür haben wir eine weiter Bibliothek benutzt: Minim[2]. Eine rudimentäre Version davon ist bereit in der Installation von Processing enthalten. Jedoch bietet die neue Version einige neuen Möglichkeiten, weswegen wir diese getestet haben. Erste Änderungen der Lautstärke liesen sich beispielwiese recht schnell implementieren.
Die visuellen Elemente unserer Simulation sind zunächst dafür bestimmt, Daten der Soundbibliothek für uns sichtbar und bewertbar zu machen.

1 http://code.google.com/p/simple-openni/
2 http://code.compartmental.net/tools/minim/


224.44196
144.3336
64.62846
83.83549
59.456203
40.6273
100.65685
24.10079
177.86023
292.28177






Geschwindigkeit: 20.130821
Position: 254.0:277.0
Geschwindigkeit Ø: 9.0609045
X: 57
Y: 36
Min: 1200.0
Max: 1500.0
LOCKED: false
FPS: 19.981234

Bei der Weiterentwicklung der Kinect-Schnittstelle konnten wir das Problem der Pendelerkennung lösen, indem wir die Tiefeninformationen des 3D Bildes analysiert und eingeschränkt haben. Somit wurde nurnoch alles auf der Höhe des Pendels erkannt, während andere Dinge ignoriert wurden. Für ein freischwingendes Pendel sollte dort eh nichts stehen. Mit dieser Grundlage ließen sich Position des Pendels herausfinden. Über die Zeit der Aufnahme lassen sich mit dieser Information weitere Daten, wie Geschwindigkeit oder Größe des Ausschlages berechnen.

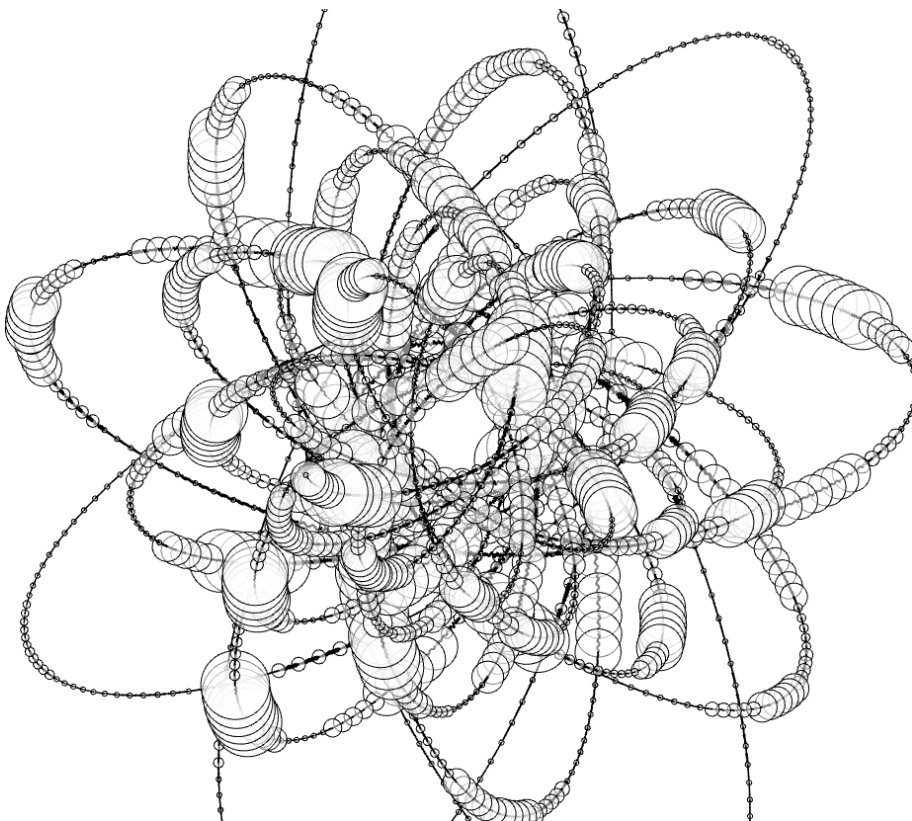
Um den Zugriff auf diese Daten möglichst intuitiv zu gestalten, ist die programmierte Logik für die Berechnungen in einer Klasse[1] versteckt, die Funktionen bereitstellt, die einfach nur den gewünschen Parameter zurückgibt. In Processing gibt das die Möglichkeit diese Klassen in eigene Dateien auszulagern, wodurch es einfacher wird die Teile des Programmes logisch abzutrennen.

Bei den Versuchen unser Lied nach diesen Parametern zu verzerren, ist uns jedoch schnell klar geworden, dass dies nicht zu interessanten Soundstrukturen führt, sondern eher die Anmutung eines kaputten CD Players schafft. Desweiteren ist uns das Lied trotz seiner sanften Tonal-ität viel zu lebendig für die Pendelbewegung. Der Sound sollte viel monotoner und einfacher werden, um Veränderungen daran deutlicher herausarbeiten zu können.

Da wir während unserer Recherche auch einige Beispiele mit programmierter Musik[2] gefunden hatten, war es die logischte Alternative dies auch selbst auszuprobieren.

1 Klasse: Struktur in der Programmierung, die Eigenschaften und Funktionen von einem Objekt logisch gruppiert.
2 http://aaron-sherwood.com/works/firewall/


Startabstand: 1200.0 (LEFT - RIGHT)
Endabstand: 1500.0 (UP - DOWN)
Geschwindigkeit: 3.354102
Position: 306.0:258.0

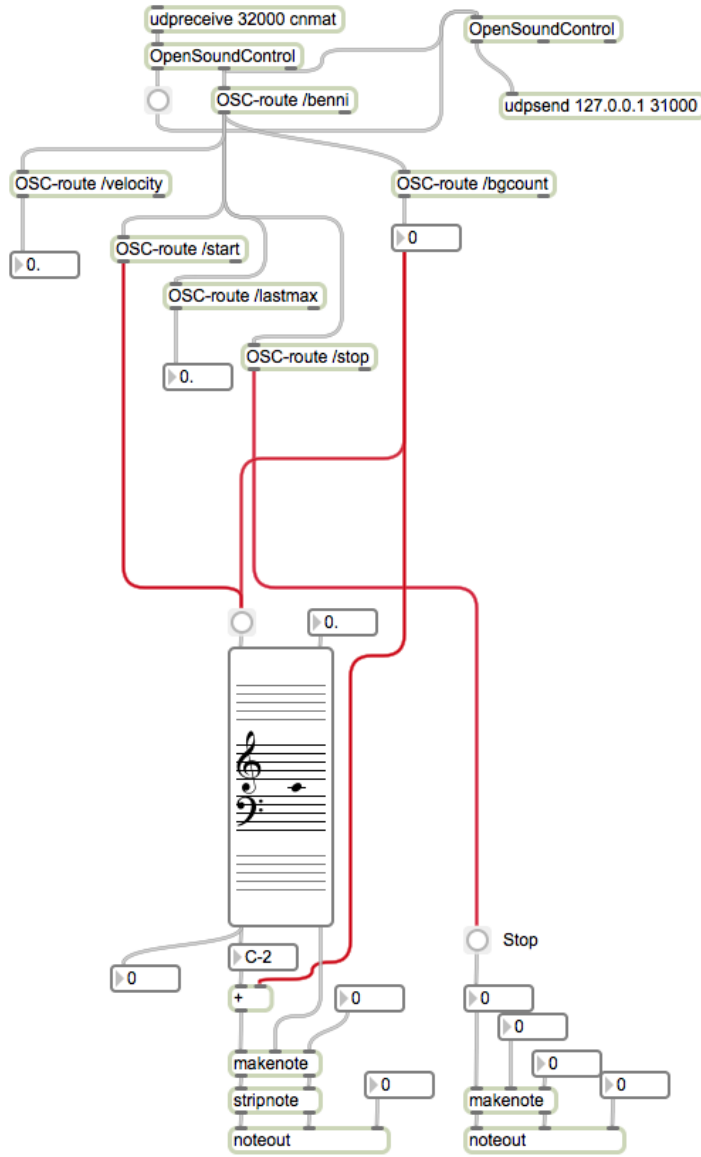


Der nächste Schritt bestand zunächst in der Suche verschiedener Bibliotheken zur Soundprogrammierung in Processing. Jedoch stellten wir fest, dass sich für Anfänger auf dem Gebiet keine geeignete Variante finden ließ. Deshalb suchten wir nach einer grafischeren Lösung und fanden das Programm Max 6[1]. Max, auch Max/Msp genannt, ist eine knotenbasierte Programmierumgebung, die ihr Spezielgebiet auf Sound gesetzt hat. In dem Programm lassen sich Noten über ihre Parameter Tonhöhe (Frequenz), Anschlagsstärke (Amplitude) und Dauer[2] bestimmen und abspielen.

Für die Kommunikation zwischen Processing, also dem Pendel, und Max gibt es verschieden Möglichkeiten. MIDI[3] ist ein Standart für die Übertragung von Noten zwischen Digitalen Instrumenten. Jedoch ist MIDI zunächst nicht dafür ausgelegt andere Daten außer Noten auszutauschen. Eine weitere Technik ist das OpenSoundControl-Protokol[4], kurz OSC. Dieses überträgt viele Formen von Daten anhand frei wählbarer Schlüsselwörter.

Wir haben uns zunächst für OSC entschieden, um die Kommunikation Processing – Max herzustellen. Da Max jedoch intern nur ein virtuelles Klavier zu Verfügung stellt, benötigten wir noch Software, um die generierten Töne auch abzuspielen. Ableton Live[5] und Reaktor[6] sind beides professionelle Soundsuiten. Ableton Live ist dabei eher klassisch ausgestattet mit allen möglichen Instrumenten, während Reaktor eher auf Synthesizer spezialisiert ist. Beide Suiten lassen sich über MIDI steuern, Reaktor kann zusätzlich auch über OSC bedient werden. Die Kommunikation von Max zu den beiden Programmen war somit auch gesichert.

1 http://cycling74.com/products/max/
2 http://de.wikipedia.org/wiki/Parameter_(Musik)
3 kurz für Musical Instrument Digital Interface, http://de.wikipedia.org/wiki/Musical_Instrument_Digital_Interface
4 http://de.wikipedia.org/wiki/Open_Sound_Control
5 https://www.ableton.com/de/live/
6 http://www.native-instruments.com/de/products/komplete/synths-samplers/reaktor-5/


udpreceive 32000 cnmat
OpenSoundControl
OpenSoundControl
OSC-route /benni
udpsend 127.0.0.1 31000
OSC-route /velocity
OSC-route /bgcount
0.
OSC-route /start
0
OSC-route /lastmax
OSC-route /stop
0.
0.
C-2
0
+
0
makenote
stripnote
noteout
0
Stop
0
0
0
0
makenote
noteout

Die Suche nach einer geeigneten Melodie ging trotz der Hilfestellung eines befreundeten Musikstudenten nicht so voran wie gewünscht. Die Idee Dreiklänge zu verwenden, um harmonische Tonfolgen zu bekommen war auch nicht das was wir suchten. Bisher hatten wir bei der Suche hauptsächlich mit Ableton Live gearbeitet, da das in der Testversion die größere Auswahl an Instrumenten bietet.

Da wir bisher eher auf der Suche nach einer Melodie waren, fielen die Synthesizer von Reaktor zunächst nicht so sehr ins Suchmuster. Jedoch fanden wir dann in den vielen Presets der drei freien Synthesizer eine Vorlage, die uns sofort zusagte. Der Ton weckte die Assoziationen von Weltall, Athmosphäre und Schwerelosigkeit, was sehr gut zu der Anmutung eines Pendels passte.
Nach einigem Ausprobieren der vielen verschiedenen Regler fanden sich auch einige Interessante, an die man die vom Pendel generierten Parameter knüpfen könnte.

Während der Suche nach dem geeigneten Sound hat das Tracking über die Kinect noch einige Verbesserungen bekommen, wobei die größte eine Vorbereitung für den graphischen Teil der Arbeit war. Diese beinhaltet, dass in dem laufendem Processing Sketch[1] über Tastatureingabe ein Bedienungsmenü geöffnet werden kann, dass das Bild der Kinect zur Überprüfung anzeigt, sowie eine Darstellung verschiedener errechneter Parameter. Über Tastur und Mauseingabe lassen sich dort auch statische Einstellungen tätigen, wie der Mittelpunkt der Pendelbewegung oder die Werte für die Tiefenfilterung, sollte das Pendel höher oder tiefer, als in unserem Versuchsaufbau hängen.
Nachdem diese Bedienungselemente in einem schließbaren Menü versteckt waren und auch ein geeigneter Ton für den Sound gefunden war, konnten wir mit der grafischen Visualisierung beginnen.

1 Sketch: Name für ein Programm in Processing. Processing will das Sketchbook des generativen Designers darstellen.


Reaktor 5 – Carbon 2.rkplr
REAKTOR
061 Atmospherics*
106.2 BPM
Select File
untitled Audiofile.aif
PLAYER
DEMO
USER
DISKS
PLAYER
Reaktor Factory Selection
Carbon 2
File Name
Carbon 2.rkplr
Newscool.rkplr
SpaceDrone.rkplr
CARBON2
REAKTOR LIBRARY
OSCILLATORS
FILTER
MODULATION / MASTER
Routing
Noise
Ring
Osc1
Osc2
Osc3
Level
PreAmp
Mode
LP4 MD
Resonance
Cutoff
Pitch Shifter
Shift
Grain
FB
Mix
Phaser
Center
Rate
Phs
Dpth
Res
Chorus
Delay
Depth
EQ
Bass
Mid
MFrq
High
HFrq
Fine
Swap
HP
PITCH
WAVE
SYNC
GLOBAL
EFFECTS
SOURCE

Für die grafische Visualisierung haben wir uns für ein Partikelsystem[1] entschieden, dessen einzelne Partikel auf das Pendel und somit auch die Musik reagieren jedoch auch eine Eigenbewegung besitzt. Jeder Partikel wird dabei zufällig nach bestimmten Vorgaben auf der angezeigten Fläche generiert und bewegt sich zunächst für jeden Schritt in eine zufällige Richtung.

Stößt man nur das Pendel an kommen alle Komponenten zusammen. Das Pendel liefert zwei interessante Werte: Geschwindigkeit des Pendels, sowie das Erreichen und die Position des größen Ausschlages.

Die Geschwindigkeit bedingt dabei eine Erhöhung der Geschwindigkeit mit der sich die Partikel bewegen. Außerdem wird zu dem Grundton, der dauerhaft abgespielt wird ein heller Zusatzton eingespielt. Die wellenförmige Änderung der Geschwindigkeit wird somit hörbar.

Das Erreichen des maximalen Ausschlages bedingt mehrere verschiedene Elemente. Beim Erreichen eines Punktes pulsieren die Elemente des Partikelsystems für einen kurzen Moment auf. Des weiteren wird über die Größe des Ausschlages die Lautstärke des Sounds beeinflusst.

Zudem ist die Größe der Pendelschwingung in vier Teile eingeteilt. Wechselt der größte Ausschlag von einem Bereich in den nächstkleineren, wird schrittweise der Hintergrund, auf dem das Partikelsystem ist abgedunkelt, sowie die Tonhöhe des Sounds verringert.

1 http://de.wikipedia.org/wiki/Partikelsystem

```
  showCon = false;

  //Start Kinect Tracking
 //  context = new SimpleOpenNI(this);
 //  if (context.isInit() == false)
 //  {
 //    println("Can't init SimpleOpenNI, maybe the camera is not connected!");
 //    exit();
 //    return;
 //  }
  context = new SimpleOpenNI(this, "test1.oni");

  ptrack = new Pendeltracking(context, 1200, 1450);

  //Start OpenSoundConroller
  osc = new OscP5(this, 12000);
  oscIP = new NetAddress("127.0.0.1", 32000);

  //Start Sound
  OscMessage msg = new OscMessage("/benni/start");
  msg.add(1);
  osc.send(msg, oscIP);

  //Gestaltung
  bgCount = 0;
  bg = new color[4];
  bg[0] = color(0,0,0);
  bg[1] = color(0.02,0.04,0.05);
  bg[2] = color(0.04,0.06,0.09);
  bg[3] = color(0.07,0.10,0.14);
  part = new Particlesystem(2000);
}

void draw()
{
  //Tracking ( + Interface )
  if(showCon) ptrack.toggleRender();
  context.setPlaybackSpeedPlayer(1.0);
  ptrack.track();
  if(int(bgCount) < 4) background(bg[int(bgCount)]);
  text(frameRate, 10, 10);
  if(showCon){
    ptrack.drawController();
    translate(640, 0);
  }

  //Controller
  //Geschwindigkeit
  OscMessage msg = new OscMessage("/benni/velocity");
  //msg.add(map(ptrack.getSmoothVel().mag(), 0, ptrack.getMaxVel(), .5, 1));
  msg.add(map(ptrack.getSmoothVel().mag(), 0, 30, .5, 1));
  osc.send(msg, oscIP);

  //LastMax
  OscMessage msg2 = new OscMessage("/benni/lastmax");
  msg2.add(constrain(map(ptrack.getLastMax().mag(), 0, 150, 0, .8), 0, 1));
  osc.send(msg2, oscIP);

  //LastMax
  OscMessage msg3 = new OscMessage("/benni/bgcount");
  msg3.add(floor(bgCount));
  osc.send(msg3, oscIP);

  //Gestaltung
  bgCount = constrain(map(ptrack.getLastMax().mag(), 0, 170, 0, 3.999), 0, 3.99);

  part.addMovement(ptrack.getSmoothVel().mag());
  part.generate(ptrack.foundLastMax());
}

void stop()
{
  //Start Sound
  OscMessage msg = new OscMessage("/benni/stop");
  msg.add(1);
  osc.send(msg, oscIP);
```

```
class Particle{
 PVector pos;
 PVector start;
 color col;
 float size;
 int lastWay;

 Particle(int x, int y, color c, float s){
  pos = new PVector(x,y);
  col = c;
  size = s;
  this.init();
 }

 Particle(PVector v, color c, float s){
  pos = v;
  col = c;
  size = s;
  this.init();
 }

 void init(){
  start = new PVector(0, 0);
  lastWay = -1;
 }

 void addMovement(PVector v){
  pos.add(v);
 }

 void addMovement(float multi){
  PVector move = new PVector();
  int way;
  if(lastWay != -1 && random(1) >= .5) way = lastWay;
  else way = floor(random(3.99));
  float speed = .4 * multi/5 * size/2 + .4;
  if(way == 0) move.set(speed, 0);
  else if(way == 1) move.set(-speed, 0);
  else if(way == 2) move.set(0, speed);
  else if(way == 3) move.set(0, -speed);
  pos.add(move);
 }

 void resetStart(PVector v){
  if(PVector.dist(pos, start) == 0 || start.x == 0 && start.y == 0){
   start.set(v);
  }
 }

 void resetStart(){
  if(PVector.dist(pos, start) == 0 || start.x == 0 && start.y == 0){
   start.set(0, 0);
  }
 }

 void resetStart(PVector v, boolean hard){
  if(hard){
   start.set(v);
  }
 }

 PVector getStart(){
  return start;
 }

 PVector getPos(){
  return pos;
 }

 color getColor(){
  return col;
 }

 float getSize(){
```

```
Particlesystem(int i){
 p = new Particle[i];
 mid = new PVector(width/2, height/2);
 this.generateParticles();
}

void generateParticles(){
 for(int i = p.length - 1; i >= 0; i--){
  boolean found = false;
  while(!found){
   PVector v = new PVector(random(width), random(width) - (width - height) / 2);
   if(
    (PVector.dist(mid, v) <= 125 && random(1) >= 0.1) ||
    (PVector.dist(mid, v) <= 250 && random(1) >= 0.6) ||
    (PVector.dist(mid, v) <= 500 && random(1) >= 0.9) ||
    (PVector.dist(mid, v) <= 1000 && random(1) >= 0.95) ||
    (random(1) >= 0.99)
   ){
    color c = color(noise(0.1, millis()%100) * 0.7 + .3,
                    noise(0.5, millis()%100) * 0.7 + .3,
                    noise(0.6, millis()%100) * 0.7 + .3);
    p[i] = new Particle(v, c, random(1.5)+1);
    found = true;
   }
  }
 }
}

void addMovement(float speed){
 PVector move = new PVector();
 for(int i = p.length - 1; i >= 0; i--){
  p[i].addMovement(speed);
  p[i].resetStart();
 }
}

void addMovement(int i, PVector v){
 p[i].addMovement(v);
}

void attract(){
 for(int i = p.length - 1; i >= 0; i--){
  PVector toMid = PVector.sub(mid, p[i].getPos());
  PVector dir = PVector.fromAngle(toMid.heading());
  if(toMid.mag() > 1) dir.setMag(20 * 100 / toMid.mag());
  else dir.setMag(toMid.mag());
  p[i].resetStart(p[i].getPos());
  p[i].addMovement(dir);
 }
}

void detract(float speed){
 for(int i = p.length - 1; i >= 0; i--){
  if(p[i].getStart().x != 0 && p[i].getStart().y != 0){
   PVector toStart = PVector.sub(p[i].getStart(), p[i].getPos());
   PVector dir = PVector.fromAngle(toStart.heading());
   float mag = 20 * 100 / toStart.mag();
   if(mag <= toStart.mag()) dir.setMag(mag);
   else if(toStart.mag() < 0.2) p[i].addMovement(speed);
   else dir.setMag(toStart.mag());
   p[i].addMovement(dir);
  }
 }
}

void generate(boolean pulse){
 for(int i = p.length - 1; i >= 0; i--){
  pushStyle();
  stroke(p[i].getColor());
  strokeWeight(p[i].getSize());
  if(pulse) strokeWeight(2.5);
  PVector pos = p[i].getPos();
  point(pos.x, pos.y);
  popStyle();
```

```
void track(){
 k.update();

 int[] depthMap = k.depthMap();
 int index;
 PVector realWorldPoint;

 float minX = 640;
 float maxX = 0;
 float minY = 480;
 float maxY = 0;

 if(render){
  PImage save = new PImage(640, 480);
  PImage depth = k.depthImage();
  save.loadPixels();
  depth.loadPixels();

  for (int y=0; y < k.depthHeight();y++){
   for (int x=0; x < k.depthWidth();x++){
    index = x + y * k.depthWidth();
    if (depthMap[index] > 0){
     realWorldPoint = k.depthMapRealWorld()[index];
     if (realWorldPoint.z >= thresholdMin && realWorldPoint.z <= thresholdMax){
      if(x < minX) minX = x;
      if(x > maxX) maxX = x;
      if(y < minY) minY = y;
      if(y > maxY) maxY = y;
      if(render) save.pixels[index] = color(0, 1, 0);
     }
     else if(render) save.pixels[index] = depth.pixels[index];
    }
   }
  }
  save.updatePixels();
  newDepthImage = save;
 }else{
  for (int y=0; y < k.depthHeight();y++){
   for (int x=0; x < k.depthWidth();x++){
    index = x + y * k.depthWidth();
    if (depthMap[index] > 0){
     realWorldPoint = k.depthMapRealWorld()[index];
     if (realWorldPoint.z >= thresholdMin && realWorldPoint.z <= thresholdMax){
      if(x < minX) minX = x;
      if(x > maxX) maxX = x;
      if(y < minY) minY = y;
      if(y > maxY) maxY = y;
     }
    }
   }
  }
 }

 lastPos.set(pos.x, pos.y);
 pos.set((minX + maxX)*1.0/2, (minY + maxY)*1.0/2);
 vel = PVector.sub(pos, lastPos);
 this.saveVel();
 this.saveMinMax();
}

void saveMinMax(){
 foundLastMax = false;
 //Errechnung von Maximalwerten, beim Ersten ausführen letzten Wert als Minimalwert setzen
 if(PVector.dist(pos, wpoint) > tmpLastMax.mag()){
  if(!growing &&
     ( ( checkAngle(radians(50), lastMin.heading()+PI, tmpLastMin.heading()) && lastMin.mag() >= tmpLastMin.mag() ) ||
     lastMax.x == 0 && lastMax.y == 0 ) ||
     tmpLastMin.mag() - 30 >= lastMin.mag()
  ){
   lastMin.set(tmpLastMin.x, tmpLastMin.y);
  }
  tmpLastMax.set(pos.x - wpoint.x, pos.y - wpoint.y);
  tmpLastMin.set(pos.x - wpoint.x, pos.y - wpoint.y);
  growing = true;
```

Abschließend lässt sich sagen, dass es interessant war mit den drei verschiedenen Medien zu experimentieren. Dabei haben wir auch gesehen, wie sehr einzelne Elemente zusammenpassen müssen um ein erwünschte Wirkung zu erreichen. Mit der Suche nach der richtigen Musik haben wir uns entsprechend lange beschäftigt. Die graphische Arbeit wird jedoch noch nicht unseren Vorstellungen gerecht und auch technisch sind noch Verbesserungen möglich. Trotz allem ist es uns gelungen die Grundzüge unsere Idee zu verwirklichen.