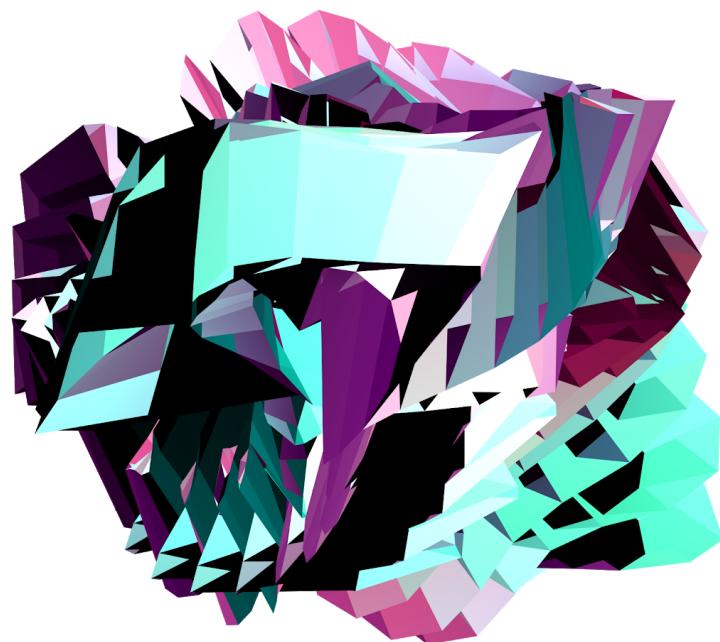


Erweiterte Audioanalyse und menschliche Kinetik als Datensatz für Echtzeit-Visualisierung



**Wettbewerbsarbeit für Schweizer Jugend forscht von Andrin Rehmann
Kantonsschule Wettingen, 2017**

Betreut durch Peter Skrotzky

Inhaltsverzeichnis

Vorwort	4
Einleitung	5
1. Arbeitsmethoden	6
1.1 Auswahl einer Entwicklerumgebung	6
1.2 TouchDesigner	6
1.2.1 Ressourcen	8
1.3 Programmieren	9
1.3.1 Python	9
1.3.2 OpenGL Shading Language	9
1.4 Ausstellung	10
2. Datenverarbeitung	11
2.1 Musikanalyse	11
2.1.1 Grundlegende Musikanalyse	11
2.1.1.1 Spektrumanalyse	12
2.1.1.2 Simples Filtern der Daten	12
2.1.1.3 Nachteile der simplen Datenfilterung	14
2.1.2 Neuer Ansatz zur Musikanalyse	14
2.1.2.1 Lernfähigkeit	14
2.1.2.2 Aufbau der lernenden Musikanalyse	15
2.1.2.3 Testphase	15
2.1.2.4 Häufungspunkte Methode	16
2.1.2.5 Umsetzung der Häufungspunkte Methode	20
2.1.2.6 Zweite Testphase	21
2.1.3 Differenzierung zwischen einzelnen Liedteilen	22
2.1.3.1 Annahme	22
2.1.3.2 Ausführung	22
2.1.3.3 Testphase	22
2.1.4 Auswertung	23
2.1.5 Verbesserungsmöglichkeiten	23
2.2 Kinetische Bewegungsanalyse	24
2.2.1 Hardware	24
2.2.2 Daten	25
2.2.3 Filtern der Daten	25
2.2.4 Eigene Positionsverfolgung	26
2.2.5 Auswertung	26

3. Visualisierung	28
3.1 Grundlagen zur 3D Visualisierung	28
3.1.1 Digitalisierte 3D Objekte	28
3.1.2 Belichtung von 3D Objekten	29
3.1.3 OpenGL	30
3.1.4 GLSL	31
3.2 Generierung von RGB Noise auf dem GPU	32
3.3 Strukturen	33
3.3.1 "Digitale Topografie"	33
3.3.2 "Tunnel"	35
3.3.3 "Sphere"	38
3.3.4 "Block Szenerie"	40
3.3.5 "Oberfläche"	41
3.4 Belichtungssystem	42
3.5 Integration von kinetischen Daten	43
3.6 Übergänge	44
3.7 Auswertung	44
3.8 Verbesserungsmöglichkeiten	44
4. Optimierung	45
4.1 CPU	45
4.2 GPU	46
4.3 Perform Mode	47
5. Anwendungsmöglichkeiten	49
5.1 Ausstellung "Klangform" im Salzhaus Brugg	49
5.2 Flexibilität	51
Fazit	52
Schlusswort	53
Anhang	54
Glossar	54
Zeitplan	55
Quellenverzeichnis	57
Bildverzeichnis	60

Vorwort

Seit ich 13 Jahre alt bin, setze ich mich als Hobby mit digitalen Medien auseinander. Durch Neugierde angetrieben entdeckte ich Methoden zur Bild- und Videobearbeitung sowie zur Erstellung von virtuellen 3D Objekten. Während ich anfangs mit Bildern und Videomaterialien aus der realen Welt arbeitete, verschob sich mein Fokus zunehmend auf computergenerierte Werke. Ich begann, mit Hilfe von Audioanalysen Musik in Bildsprache umzusetzen, ohne dabei in mühsamer Arbeit jeden Ton aus dem Lied heraushören zu müssen. Je länger ich mich mit dem Medium beschäftigte, desto wichtiger wurde das Arbeitsinstrument Computer. Dieser Weg führte mich zur generativen Kunst, der auch diese Arbeit im Rahmen der Maturaarbeit zuzuordnen ist.

Generative Kunstwerke sind Anleitungen, die von einem Menschen erschaffen werden. Die Ausführung erfolgt aber durch einen Computer. Der Erschaffer arbeitet dabei geschickt mit Zufallswerten, sodass einmalige Bilder, Objekte oder Melodien generiert werden. Jedes Mal, wenn der Computer den Algorithmus respektive die Anleitung ausführt, entsteht so ein neues Kunstwerk. Jedes dieser Kunstwerke ist ähnlich einmalig, wie ein Gemälde eines Malers. Einzig der Computer kann die Anleitung mit gleichen Werten speisen, um das Werk zu reproduzieren. Generative Kunst wirft die Frage auf, inwiefern der Computer zum Kunstschaffenden werden kann.^{1 2 3}

¹ Celestino Soddu, "Generative Art", Generativeart.com, <http://www.generativeart.com> (Zugriff: 29.10.16)

² John McCormack und Mark Guglielmetti, "Creative Coding", Future Learn,
<https://www.futurelearn.com/courses/creative-coding> (Zugriff: 7.15 - 8.15)

³ Brad Hammond, "Make art now", facebook Gruppe <https://www.facebook.com/groups/makeartnow/> (Zugriff: 21.10.16)

Einleitung

In dieser Arbeit möchte ich eine fortgeschrittene Methode zur Musikanalyse entwickeln, und diese visuelle umsetzen. Um einen zusätzlichen Reiz für den Betrachter zu schaffen, will ich eine anwendbare Methode zur Bewegungsanalyse entwickeln und diese Daten sinnvoll aufbereiten. Letztlich werde ich auf den Daten der Musik und Bewegungsanalyse basierend Darstellungsformen erarbeiten, die ästhetisch und bildhaft wirken und gleichzeitig eine grosse Dynamik aufweisen.

Das Endprodukt ist eine eigenständige Software, die Musik und die Bewegungen der Betrachter in Echtzeit analysiert und visualisiert. Ich werde das Produkt im Rahmen einer Ausstellung testen und grobe Fehler korrigieren.

In der Dokumentation beschränke ich mich vor allem auf Entscheidungen und Ideen. Ich verzichte bewusst auf die genaue Beschreibungen zur Umsetzung. Die Übersetzung der Ideen in Computersprache erfordert genaue Kenntnisse über die verwendete Entwicklerumgebung sowie auch deren grundlegenden Logik. Es ist auch nicht sinnvoll die Pfade der Daten über das gesamte Projekte mitzuverfolgen, da die Software mehr als 2500 Operationen beinhaltet. Darum sind nur interessante Ausschnitte der Software beschrieben, die signifikante Anstrengungen meinerseits erforderten.

1. Arbeitsmethoden

Das Konzipieren und Erstellen einer Applikation erfordert einen enormen Zeitaufwand. Professionelle Applikation werden meistens in einem Team erstellt und werden auf früheren Methoden und Arbeiten aufgebaut. Die direkte Kommunikation mit der Hardware eines Computers ist nicht ökonomisch. Ein Programm ist vergleichbar mit Werkzeugen der realen Welt. Beispielsweise gibt es kaum einen Fotografen, der seine Kamera selber entwickelt und konstruiert, obwohl diese einen wesentlichen Einfluss auf den Ausgang seiner Arbeit hat. In dem Sinne verwende ich eine Entwicklerumgebung, die mir die Arbeit erleichtert und das Produkt somit entscheidend beeinflusst.

Im Gegensatz zur Erschaffung einer soliden Struktur spielt sich die Entwicklung einer Software im digitalen Raum ab, weshalb dieser Prozess für viele Leser ohne Fachkenntnisse weniger offensichtlich erscheint. In der digitalen Welt stehen einem anstelle von physischen Hilfsmitteln digitale Werkzeuge zur Verfügung. Bereits eine Programmiersprache ist ein stark vereinfachter Weg, mathematische Befehle an die Hardware eines Computers weiterzugeben.⁴

1.1 Auswahl einer Entwicklerumgebung

Die Auswahl des richtigen Werkzeuges ist essentiell für ein erfolgreiches Resultat. Meine Entscheidung zur Auswahl basiert auf folgenden Kriterien:

- Die Applikation muss für Echtzeit-Berechnungen ausgelegt sein. Pro Bild stehen dem Programm lediglich 16 bis 33 Millisekunden zur Verfügung.
- Die Applikation sollte den Arbeitsprozess gegenüber herkömmlichem Programmieren verschneidern.
- Die Applikation sollte Module zur Audioanalyse und Erstellung von 3D Grafiken beinhalten.
- Die Applikation sollte einfach erweiterbar sein. Somit können unvorhersehbare Engpässe verhindert werden.
- Es sollten genug Ressourcen zum Erlernen dieser Applikation vorhanden sein.

1.2 *TouchDesigner*

TouchDesigner ist ein Programm mit visueller Entwicklerumgebung zur Erstellung von Software. Im Gegensatz zu konventionellen Umgebungen, die hauptsächlich Text basiert aufgebaut sind, arbeite man in *TouchDesigner* mit Nodes. Nodes sind Code-Fragmente, die bereits als fertige Blöcke für den Entwickler vorhanden sind oder vom Entwickler erstellt werden. Die Blöcke können untereinander kommunizieren. Mit Hilfe von verschiedenen Parametern bestimmt das Programm die Funktionalität der Nodes. Der Entwickler kann die Nodes in einer grafischen Oberfläche anordnen und so die Struktur des Programms

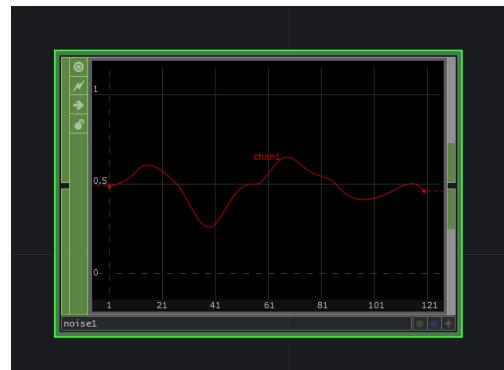
⁴ Unbekannt, “programming language”, business dictionary
<http://www.businessdictionary.com/definition/programming-language.html> (Zugriff: 13.10.16)

definieren. Dieser Prozess ist für einzelne Anwendungen effizienter als das rein textbasierte Programmieren.

“TouchDesigner” stellt Daten, wenn möglich, mit Hilfe eines sogenannten “Node Viewers” dar. Diese Eigenschaft ist einzigartig, auch für Node basierte Programme. Das vergleichbare Konkurrenz Produkt VVVV kann lediglich Matrizen anzeigen.⁵

Abb. 1

Das Bild zeigt einen Ausschnitt der Entwicklerumgebung von TouchDesigner. Abgebildet ist ein “Noise” Node, der dem Nutzer zufällige Werte zur weiteren Verarbeitung bereitstellt.



In den Parametereinstellungen kann der Entwickler Konstanten definieren. Alternativ können Referenzen zu anderen Nodes oder Variablen eingefügt werden.

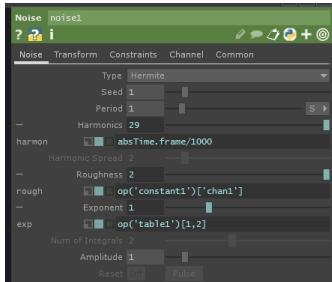


Abb. 2

Das Bild zeigt die Parameter Darstellung eines “Noise” Nodes.

Visuelle Entwicklerumgebungen besitzen oft beschränkte Anwendungsmöglichkeiten. Ein Entwickler wird durch die Vereinfachung limitiert. *TouchDesigner* hat den Vorteil, dass der Benutzer auch selber Code Fragmente programmieren und diese direkt in die grafische Architektur des Programms einbinden kann. Die Code Fragmente lassen sich mit externen Editoren bearbeiten. *TouchDesigner* verfügt über einen eigenen Compiler (*siehe Glossar*), der *Python*, *C++* und *OpenGL Shading Language* (*siehe 1.3, Glossar*) auf Fehler überprüfen kann.

Für die Generierung bewegter Bilder führt *TouchDesigner* generell für jedes Einzelbild, auch “Frame” genannt, den ganzen Algorithmus mit allen Nodes aus. Alternativ können Node Blöcke in definierten Zeiträumen deaktiviert werden. Somit ist es auch möglich, die Code

⁵ Joreg, “a multipurpose kit”, vvvv, <https://vvvv.org/> (Zugriff: 9.9.16)

Fragmente nur zu bestimmten Zeitpunkten auszuführen. Somit kann die Effizienz des Programms kontrolliert verbessert werden.⁶ ⁷

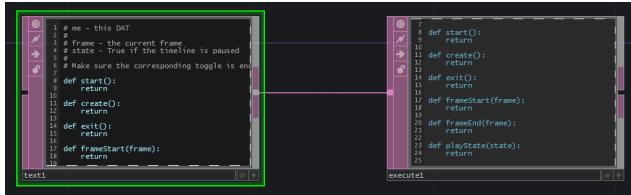


Abb. 3

Ein Code Fragment wird direkt in die Oberfläche integriert.

TouchDesigner stellt dem Entwickler mehr als 400 verschiedene Node Blöcke zur Verfügung. Um die Orientierung zu vereinfachen, werden diese in folgende Kategorien eingeteilt:

- COMP's dienen zum Organisieren der Programmstruktur
- TOP's ermöglichen grafische Operationen
- CHOP's sind für Rechnungen Funktionen und andere Operationen
- DAT's zur Verarbeitung von grossen Datenmengen in Tabellen
- SOP's generieren 3D Objekte
- MAT's Darstellung von 3D Objekten

Solange man sich innerhalb einer Kategorie bewegt, lässt sich beinahe jeder Node mit jedem anderen Node verbinden. Damit ein Kategoriewechsel möglich ist, muss der Node zuerst konvertiert und somit in ein neues Format gebracht werden.⁸

1.2.1 Ressourcen

Die Entwickler von *TouchDesigner* haben eigens für das Programm eine Wiki Seite erstellt, die detaillierte Erklärungen zur Programmoberfläche und jedem Node-Modul liefert. Zudem hostet *derriative.ca* ein eigenes Forum. Des weiteren gibt es eine Facebook-Gruppe mit dem Namen: "TouchDesigner Help Group". Diese reagiert äusserst schnell. Unter den Mitgliedern sind viele Entwickler der Software sowie auch Mitarbeiter grosser Studios zu finden. Die Hilfestellungen sind zuverlässig und weisen auf professionelle und effiziente Lösungen hin. Einige der Nutzer stellen ihre Projektdateien gratis zur Verfügung oder nehmen sich sogar die Mühe, Video Tutorials zu einzelnen Problemlagen zu publizieren.⁹ ¹⁰

⁶ Malcolm, "Write a CPlusPlus DLL", *derriative*,

https://www.derivative.ca/wiki088/index.php?title=Write_a_CPlusPlus_DLL (Zugriff: 14.10.16)

⁷ Rob, "Info Dat class", *derriative*, https://www.derivative.ca/wiki088/index.php?title=Info_DAT (Zugriff: 14.10.16)

⁸ Unbekannt, "About TouchDesigner" <http://www.derivative.ca/> (Zugriff: 15.10.16)

⁹ Richard Burns, "TouchDesigner Help Group", facebook Gruppe,

<https://www.facebook.com/groups/touchdesignerhelp/> (Zugriff: 10.10.16)

¹⁰ Matthew Ragan, "TouchDesigner", <https://matthewragan.com/> (Zugriff: 10.10.16)

1.3 Programmieren

Im Schweizer Schulsystem wird wenig Wert auf frühzeitiges Erlernen von Programmiersprachen gelegt. Obwohl wir uns zunehmend in einer digitalen Welt befinden, entfremden wir uns immer mehr von deren Grundbausteinen. Moderne Programmiersprachen sind viel näher an unserem logischen Denken als die mathematischen Befehle, die auf der Hardware ausgeführt werden. Programmiersprachen wurden so konzipiert, dass sie möglich einfacher lernbar sind. Trotzdem scheint das Programmieren in den Augen vieler Menschen eine Fähigkeit zu sein, die ausschliesslich Freaks, Hacker oder Nerds in Science Fiction Erzählungen beherrschen.

Glücklicherweise gibt es heute viele online Plattformen, die Kurse für das Erlernen von verschiedensten Programmiersprachen gratis anbieten. Auf der Seite codeacademy.com habe ich Kurse in *Python*, *Javascript*, *HTML* und *CSS* abgelegt. Auf der Seite futurelearn.com besuchte ich einen Kurs über kreatives Programmieren mit Java und zu guter Letzt habe ich auf Udacity.com einen Kurs über *WebGL* und *Three.js* absolviert. Profitieren konnte ich auch von meinen Erfahrungen im Praktikum bei der Firma *artificial rome* in Berlin. Das Projekt an dem ich in dieser Firma hauptsächlich arbeitete, basiert auf zwei Programmiersprachen: *Python* und *OpenGL Shading Language*.^{11 12 13 14}

1.3.1 Python

Python ist eine “High-Level” Programmiersprach, das heisst, der Abstraktionsgrad gegenüber reinen Hardware Befehlen ist gross. Die Programmiersprache wird als einfach und benutzerfreundlich angepriesen. Python ist Open-Source, was bedeutet, dass das Programm gratis heruntergeladen werden kann. Benutzer können Änderungen vorschlagen oder beantragen. Die Python-Foundation entscheidet, ob Änderungen in die offizielle Version von Python eingebunden werden.^{15 16}

1.3.2 OpenGL Shading Language

Diese Programmiersprache verwendet viele Elemente aus der Sprache C und erweitert diese mit Vektor- und Matrixtypen, welche den 3D Grafiken zugrunde liegen. Um die Sprache benutzerfreundlicher zu gestalten, wurden viele Funktionen der C++ Sprache implementiert.
¹⁷

¹¹ Ben, “Main Page”, derriative, http://www.derivative.ca/wiki088/index.php?title=Main_Page (Zugriff: 8.9.16)

¹² John McCormackt und Mark Guglielmetti, “Creative Coding”, Future Learn, <https://www.futurelearn.com/courses/creative-coding> (Zugriff: 7.15 - 8.15)

¹³ Eric Haines, “Interaktive 3D Grafiken”, Udacity, <https://de.udacity.com/> (Zugriff: 10.15 - 11.15)

¹⁴ Verschieden Autoren, “Learn to code”, Codeacademy, <https://www.codecademy.com/learn/learn-java> (Zugriff: 7.15 - 9.15)

¹⁵ Unbekannt, “What is Python”, Python Foundation, <https://www.python.org/doc/essays/blurb/> (Zugriff: 10.10.16)

¹⁶ Unbekannt, “About Python”, Python Foundation, <https://www.python.org/about/> (Zugriff: 10.10.16)

¹⁷ Unbekannt, “OpenGL Shading Language”, OpenGL, <https://www.opengl.org/documentation/glsli/> (Zugriff: 3.10.16)

1.4 Ausstellung

Von der Ausstellungsgruppe des *Salzhaus Brugg* bekam ich die Möglichkeit, ihre Räumlichkeiten für zwei Wochen in Anspruch zu nehmen. Die Ausstellung erlaubt mir Reaktionen von Betrachtern auf die Software zu sammeln. Die Betrachter sehen dabei lediglich die Projektionen und haben keine Ahnung vom gesamten Prozess hinter den Visualisierungen. Die Perspektive die Betrachterinnen und Betrachter ist somit viel neutraler. Sie sehen, was wirklich ausschlaggebend ist. Das fertige Bild. Die Projektionen müssen den Räumlichkeiten angepasst werden und gleichzeitig muss eine kleine Werbekampagne erstellt werden, um Leute anzuziehen. Die Ausstellung sollte auch ein intensiver Test für die Bewegungs und Audioanalyse sein. Eine kleine Musikformation wird live spielen und direkt mit der Software gekoppelt.

2. Datenverarbeitung

Die Datenverarbeitung beinhaltet das Sammeln und Weiterverarbeiten von Informationen.

Für mein Projekt erhalte ich Daten aus drei Quellen: ¹⁸

- Musikanalyse
- Bewegungsanalyse
- Zufallszahlen

Diese Datenquellen können nicht direkt in ein Bild umgewandelt werden. Als erster Schritt ist darum eine Weiterverarbeitung notwendig, die möglichst viele Informationen aus den Quellen herausliest und diese so umformt, dass sie visualisierbar werden.

2.1 Musikanalyse

Musik ist eine äusserst komplexe Kunstform, die in ein vielfältiges Spektrum an Stilrichtungen unterteilt wird. Die Musikanalyse sollte auf möglichst unterschiedliche Musik anwendbar sein. Zentral in der Interpretation von Ton in Form von Bildern wird der Rhythmus des Liedes sein, zumal dies ein einfacheres Element ist als die Melodie. Eine Analyse der Melodie würde eine zeitaufwendige Recherche erfordern.

Das Programm sollte fähig sein, Übergänge zwischen verschiedenen Liedern und Liedteilen zu erkennen. Die Musikanalyse erhält nicht mehr Informationen über ein Lied als dessen Audiosignal zum jetzigen Zeitpunkt. Dies korrespondiert mit den Daten, die ein Mikrofon über das Audiokabel an einen Lautsprecher sendet. Die Daten sollten schlussendlich in einer Form verfügbar sein, die ein Verarbeiten im grafischen Bereich ermöglicht.

2.1.1 Grundlegende Musikanalyse

Es existieren bereits viele Konzepte zur Audioanalyse. Der einfachste Ansatz besteht darin, die Intensität der einzelnen Frequenzen als Zahlenwerte darzustellen. Dies ermöglicht ein Unterteilen des Audiosignals auf verschiedene Tonhöhen. Ich werde mich bei der Musikanalyse auf zwei Frequenzbereiche konzentrieren. Einerseits tiefe Töne, in denen Bassläufe und Paukenschläge vorzufinden sind und andererseits hohe Töne, welche Gesang, andere Instrumente und die restliche Perkussion beinhaltet. Die Anzahl dieser Bereiche könnte sich beliebig erweitern, jedoch erhöht sich damit auch schnell die Komplexität des Programms. ¹⁹

¹⁸ Carl French, "Data Processing and Information Technology", Cengage Learning EMEA, 1996, Unbekannter Verlagsort

¹⁹ Richard Burns, "A simple Audio Reactive Sphere", Vimeo, <https://vimeo.com/15850663> (Zugriff: 20.10.16)

2.1.1.1 Spektrumanalyse

In erster Linie verwende ich eine digitale Spektrumanalyse. Das Audiosignal wird dabei in verschiedene Tonhöhen aufgeteilt, welchen dann je nach Intensität verschiedene Zahlenwerten zugeteilt werden. So entsteht eine Tabelle, die den Frequenzen zwischen ca. 20 Hz bis 20,000 Hz verschiedene Werte zuordnet. Die Beschränkungen des Spektrums orientieren sich an der Leistung des menschlichen Hörsystems. So können wir Werte ausserhalb dieses Bereiches nicht mehr wahrnehmen. Folglich werden Musikstücke für diesen Frequenzbereich komponiert und produziert.²⁰ ²¹

2.1.1.2 Simples Filtern der Daten

Die Erkennung der Bassläufe und Paukenschläge beschränkt sich auf einen Bereich von 20 bis 100 Hertz, davon wird der Maximalwert berechnet. Ein Problem ist die Tatsache, dass dieser Frequenzbereich noch viele andere Instrumente und Töne beinhaltet. Der Bassschlag kann nur durch seine Intensität und Regelmäßigkeit aussortiert werden. In diesem spezifischen Frequenzbereich ist der Bassschlag meist die stärkste Geräuschquelle. Mithilfe eines geschickt gesetzten Grenzwert kann der Paukenschlag eines Liedes extrahiert werden.

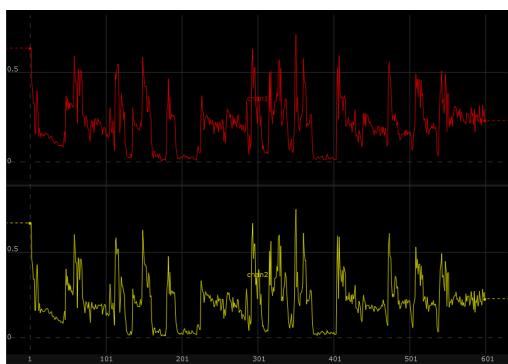


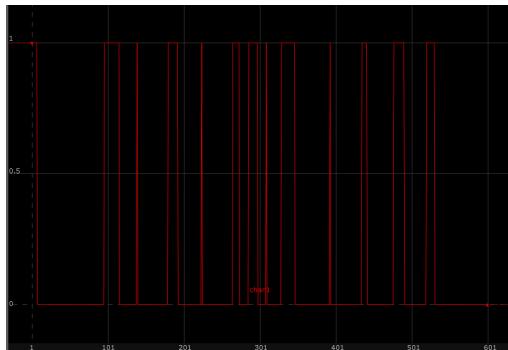
Abb. 4

Das Bild zeigt den Maximalwert zwischen 20 bis 100 Hertz eines Hip-Hop Stückes über zehn Sekunden. Der rote Kanal zeigt die linke und die gelbe die rechte Hälfte des Stereo Audio Signals.

In der Abbildung 4 ist eine Rhythmus Struktur nur vage zu erkennen. Im nächsten Bild ist dasselbe Lied unter Anwendung eines Grenzwerts dargestellt. Dieser wird als konstante Zahl gesetzt. Falls der Maximalwert diesen Grenzwert überschreitet, ist das Ausgabesignal 1, andererseits 0.

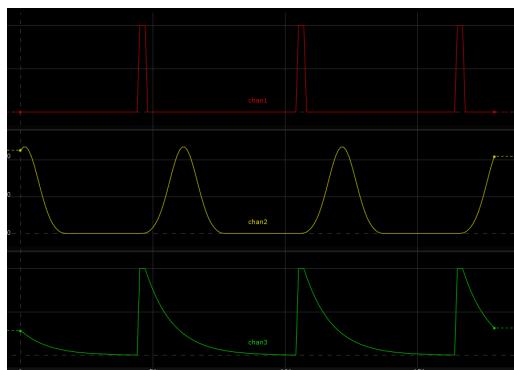
²⁰ Christoph Rauscher, "Grundlagen der Spektrumanalyse", Seite 9, Rohde & Schwarz GmbH & Co. KG, 2000, München

²¹ Verschiedene Autoren, "Audio Frequency", Wikipedia, https://en.wikipedia.org/wiki/Audio_frequency (Zugriff: 20.9.16)

**Abb. 5**

Das Bild zeigt Überschreitungen eines konstanten Grenzwerts. Hier wurde dasselbe Lied verwendet wie in Abbildung 2.

Eindeutige Signale sind vorteilhaft zur Datenverarbeitung, für eine Visualisierung aber viel zu hektisch. Demnach ist es sinnvoll einen Filter darauf anzuwenden, der die Verränderung abrundet. Es gibt verschiedene Methoden, beispielsweise kann ein Gauss Filter verwendet werden. Dies erwies sich aber nicht als zielführend. Die Signale werden in Echtzeit erarbeitet und der Gauss Filter kann seine Arbeit erst mit dem Eintreffen des Signals beginnen. Somit ist je nach Stärke des Filters eine signifikante Zeitdifferenz zwischen dem ursprünglichen Signal und dem Peak, der nach dem Filtern entsteht zu beobachten. Die sinnvollste Variante scheint eine simple Limitierung der Fallgeschwindigkeit zu sein.^{22 23}

**Abb. 6**

Das Bild zeigt das ursprüngliche Signal (Rot), einen darauf angewendeten Gauss Filter (gelb) und eine Limitierung der Abfallgeschwindigkeit (grün).

²² Patrice Delmas, "Gaussian Filtering", Seite 1, University of Auckland, 2012, Neuseeland

²³ Verschiedene Autoren, "Distributed Lag", Wikipedia, https://en.wikipedia.org/wiki/Distributed_lag (Zugriff: 8.9.16)

2.1.1.3 Nachteile der simplen Datenfilterung

Das in 2.1.1.2 beschriebene Verfahren ergibt sinnvolle Daten, wenn die Konstanten von vornherein auf ein Lied angepasst werden. Allerdings stösst das System auch mit benutzerdefinierten Konstanten oftmals an seine Grenzen. Viele Instrumente verfügen über eine grosse Dynamik in deren Intensität. Ist Beispielsweise der Paukenschlag im Refrain eines Liedes viel feiner als im restlichen Lied, erkennt die Musikanalyse im Refrain keinen Bassschlag. Der Grenzwert wurde somit zu hoch gesetzt. Alternativ kann das System auch sehr sensibel eingerichtet werden, was einem tiefen Grenzwert entspricht. Dies kann zu einer konstanten Überschreitung des Grenzwertes führen. Die Problematik verschärft sich, wenn das System auf andere Lieder angewendet wird, ohne vorher die Konstanten anzupassen.

2.1.2 Neuer Ansatz zur Musikanalyse

Die gesammelten Erfahrungen der simplen Datenfilterung haben gezeigt, dass ein statischer Grenzwert nicht sinnvoll für eine Musikanalyse ist. Die Lösung ist ein variabler Grenzwert, der autonom vom Programm bestimmt wird.

Der folgende Abschnitt beschränkt sich auf die Aussortierung des Rhythmus im Bass-Bereich. Letztlich sollte das System fähig sein, den Rhythmus, soweit er in diesem Frequenzbereich erkennbar ist, auszusortieren.

2.1.2.1 Lernfähigkeit

Damit ein Programm eine Variable selbstständig sinnvoll setzen kann, muss zuerst "sinnvoll" in diesem Kontext definiert werden. Das Programm sollte letzten Endes fähig sein, einen Rhythmus zu extrahieren, der keine Störgeräusche beinhaltet und folglich eine regelmässige Struktur aufweist. Falls kein Rhythmus erkennbar ist, sollte sich das Programm auf andere markante Geräusche im selben Frequenzbereich fokussieren.

Regelmässigkeit lässt sich durch die Differenz zwischen dem Jetzt-Wert und dem Durchschnittswert definieren. Zumal das System keine Erlaubnis hat, die Musikdateien im Voraus zu analysieren, kann es sich lediglich auf gespeicherte Ereignisse aus der Vergangenheit beziehen. Somit muss der Durchschnittswert anhand gespeicherten Daten erschlossen werden.²⁴

Anfängliches Grundkonzept

Falls die detektierten Paukenschläge in einem bestimmten Zeitraum nicht konstante Zeitabstände aufweisen, muss der Grenzwert angepasst werden. Falls die Zeitabstände zu gering sind, wird der Grenzwert nach oben korrigiert, andererseits nach unten. Dieser Vorgang wird für jeden Programmdurchlauf wiederholt, bis konstante Zeitabstände erreicht werden und das System nicht mehr aufgerufen wird.

²⁴ Shai Shalev-Shwartz und Shai Ben-David, "Understanding Machine Learning: From Theory to Algorithms", Seite 7, Cambridge University Press, 2014, England

Das Prinzip dahinter ist, den durchschnittlichen Zeitabstand zwischen den Paukenschlägen als Referenzwert zu definieren. Falls der jetzige Zeitabstand geringer ist, geht das System davon aus, dass der Grenzwert zu tief gesetzt wurde. Ein zu tief gesetzter Grenzwert hört auch auf Störgeräusche und verringert somit die Zeitabstände zwischen den Paukenschlägen. Falls der jetzige Zeitabstand grösser ist, wurden einzelne Paukenschläge übersehen, da sie den Grenzwert nicht erreicht haben.

2.1.2.2 Aufbau der lernenden Musikanalyse

Da viele Lieder keinen Basslauf haben der konstant identische Zeitdifferenzen aufweist, vergleiche ich einen Durchschnittswert, der über eine lange Zeit gemessen wird, mit dem einer geringen Zeitspanne.

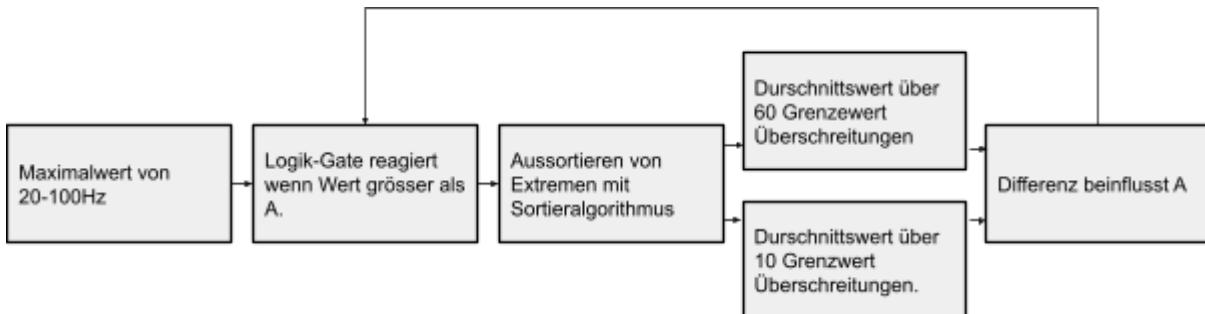


Abb. 7

Visuelle Darstellung des ersten Konzeptes einer lernenden Musikanalyse

2.1.2.3 Testphase

Das Ergebnis ist kaum zufriedenstellend. Folgende Problemlagen haben sich ergeben:

- **1. Problem:**
Wenn der Grenzwert zu tief definiert wird, kann dies ein konstantes Überschreiten desselben zur Folge haben.

Lösung:

Der Grenzwert wird auch durch den Faktor Länge der Schläge beeinflusst. Der Zeitabstand zwischen dem Überschreiten und Unterschreiten wird gemessen. Umso länger dieser Wert ist, desto tiefer wird die Empfindlichkeit gesetzt.

- **2. Problem:**
In vielen Liedern gibt es Phasen, in denen keine Bassschläge zu erkennen sind. In einer solchen Phase sollte das System auf andere Signale im Bassbereich zurückgreifen.

Lösung:

Das System misst die Zeitabstände zwischen jetzt und dem letzten Überschreiten des Grenzwerts. Sobald dieser Wert eine signifikante Grösse erreicht hat, wird der Grenzwert verringert und das System reagiert auf andere Geräusche.

- **3. Problem:**

Die Zeitabstände in den meisten Rhythmen sind variabel. Nur simple elektronische Lieder verfügen über einheitliche Zeitabstände. Das herkömmliche System arbeitet mit nur einem Durchschnittswert. Es wäre sinnvoller, mit mehreren Durchschnittswerten zu arbeiten.

Lösung:

Es müssen mehrere Durchschnittswerte aus einer Liste von Daten gefunden werden. Diese sind nicht mehr Durchschnittswerte sondern Häufungspunkte. Nach längerer Recherche konnte ich keinen passenden schon existierenden Lösungsansatz dazu finden. Mein persönlicher Lösungsansatz habe ich im nächsten Kapitel erläutert.

Die Testresultate zeigen, dass der Einfluss aus der lernenden Musikanalyse auf den Grenzwert alleine nicht zufriedenstellend ist. Das System muss sich selber überwachen, um Fehler zu vermeiden. Letzten Endes ist es notwendig, das System zu limitieren. Auch bei weitaus professioneller erarbeiteten lernfähigen Systemen besteht das Problem der Unberechenbarkeit. Das System kann nie mit garantierter Sicherheit operieren, da deren Eingangswerte nicht vorausschaubar sind. Um Fehler zu vermeiden, wird der Ausgabewert des Systemes limitiert.²⁵

2.1.2.4 Häufungspunkte Methode

Die folgende Methode ist erforderlich damit ich die Häufungspunkte der Abstände zwischen einzelnen Paukenschläge finden kann. Die Methode findet aus ungenauen Daten, Punkte an denen sich die Werte ähneln.

Die Problemstellung ist folgende: Aus einer Reihe von Zahlen sollen Häufungspunkte extrahiert werden. Die Musikanalyse kann erst darauf basierend weitergeführt werden. Die Audioanalyse ist niemals korrekt. Audiospuren haben eine Datendichte von 44 bis 96 Tausend abgespeicherten Pegeln pro Sekunde. Die Audioanalyse detektiert Unterschiede mit einer Rate von 60 Abtastungen pro Sekunde. Gleichzeitig fliessen immer auch Fehlangaben in die Daten mit ein. Das System muss folglich fähig sein, diese Ungenauigkeiten auszusortieren. Das Konzept meines Lösungsansatzes basiert auf einer visuellen Problemlösung. Hierbei wird jeder Zeitabstand als Gradient abgebildet. Dieser Gradient ist in der Mitte dunkel und verblasst dann nach links und rechts ins Weisse. Die Gradienten weisen eine hohe Transparenz auf und der Farbwert wird dann mit darunterliegenden Gradienten zusammen addiert. Zum Bestimmen der verschiedenen Mittelwerte müssen lediglich die dunkelsten Abschnitte auf der Grafik gefunden werden.²⁶

²⁵ Matthew Kirk, "Thoughtful Machine Learning: A Test-Driven Approach", Seite 6, O'Reilly, 2015, U.S.A

²⁶ Unbekannt, "Samplerate", Baumannmusic,
<http://www.baumannmusic.com/de/2012/sampleratehz-und-khz-aufloesung-bit-und-bitrate-kbits/> (Zugriff 5.10.16)

Abb. 8

Visuelle Darstellung der Häufungspunkte Funktion.

Diese Methode lässt sich auch mithilfe von Funktionen anstelle einer Grafik umsetzen. Für jeden Wert in einer Liste von Zahlen wird eine Funktion erstellt. Der Abstand zwischen dem Nullpunkt und der X-Position des einzigen Maximums dieser Funktion entspricht der Zeit zu der ein bestimmter Peak gemessen wurde. Addiere man nun solche Funktionen, die geschickt genug gewählt wurden, entstehen nur an den grob gemittelten Häufungspunkten neue Maximas.

Eine Gausskurve erfüllt die erwarteten Kriterien. Sie besitzt einen einzelnen Punkt als Maximum, gibt dem Punkt aber trotzdem einen Toleranzbereich und flacht dann relativ schnell ab. Zudem kann ihr Maximum einfach verschoben werden und die Ableitung der Funktion ist dank den Eigenschaften der Eulerschen Zahl einfach. An einem einfachen Beispiel versuche ich den aus diesem Konzept entstandenen Algorithmus zu erklären.²⁷

Die Variablen bedeuten folgendes:

e = Eulersche Zahl

P = Zeitdifferenz oder auch X Koordinate im Koordinatensystem

γ = Formfaktor

Die Grundfunktion ist so:

$$f(x) = e^{-(x-P)^2 \times \lambda}$$

Die Variablen werden exemplarisch folgendermaßen definiert.

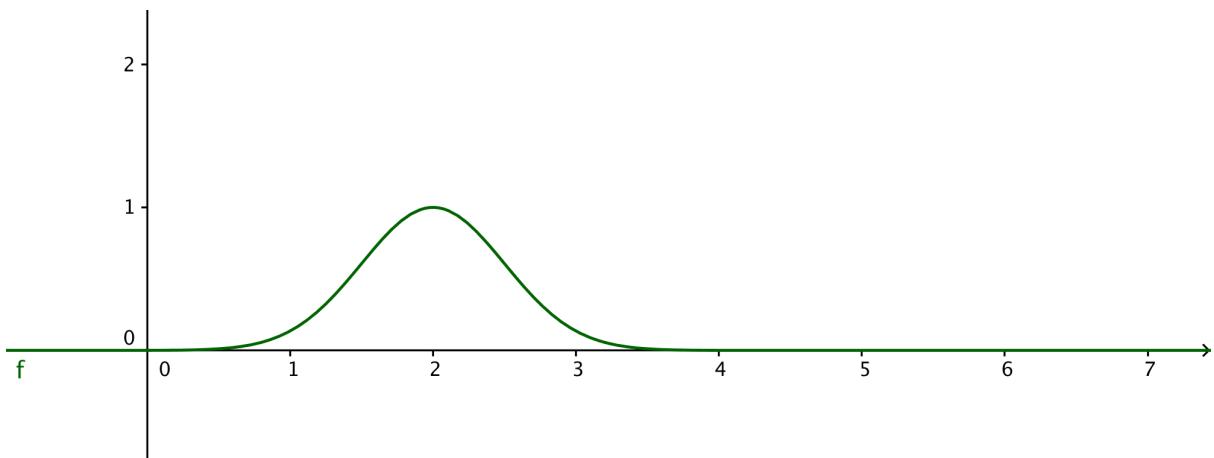
$$P = 2$$

$$\gamma = 2$$

Die Gleichung lautet also:

$$f(x) = e^{(-(x-2)^2 \cdot 2)}$$

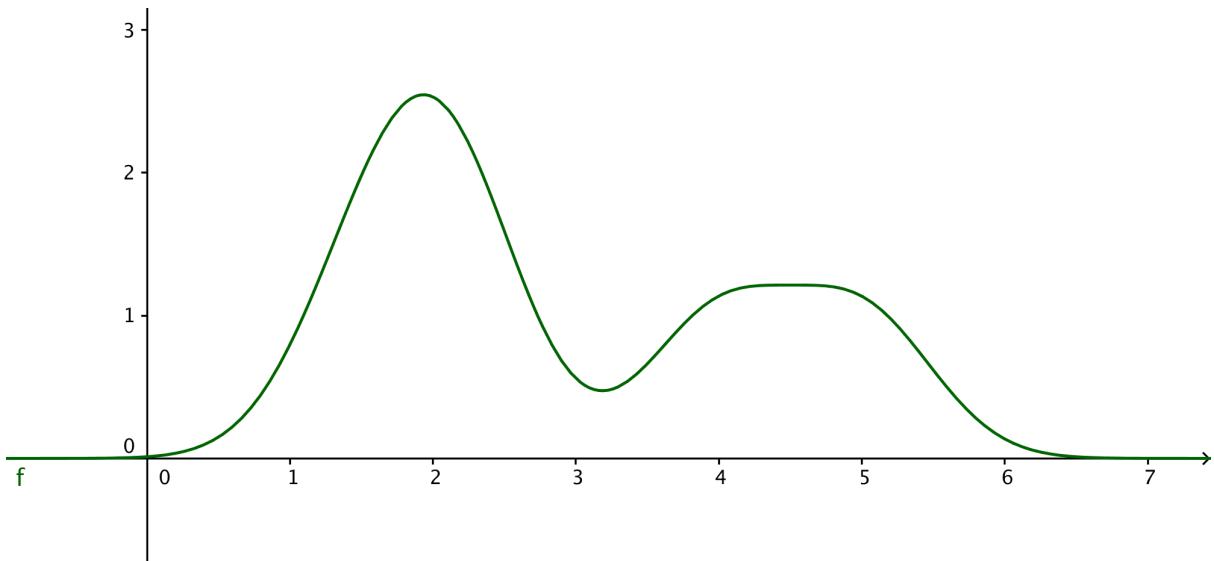
²⁷ Quo R, "Dichtefunktion", Wikipedia, <https://commons.wikimedia.org/wiki/File:Dichtefunktion.png> (Zugriff: 7.7.16)

Abb. 9

Geplottete Version der Gleichung mit den exemplarischen Werten.

Dieser Vorgang wird wie beschrieben für jeden Wert wiederholt. In diesem Falle verwende ich für P(1) bis P(5) die Werte: 2, 4, 1.5, 5, 2.2 Der Formfaktor ist immer noch derselbe. Die Gleichungen werden dann addiert und es ergibt sich folgende Formel.

$$f(x) = e^{-(x-2)^2/2} + e^{-(x-4)^2/2} + e^{-(x-1.5)^2/2} + e^{-(x-5)^2/2} + e^{-(x-2.2)^2/2}$$

Abb. 10

Geplottete Version der oben beschriebenen Gleichung.

Die ungefährten Positionen der Häufungspunkte sind jetzt als Extremas zu sehen. Damit die Positionen der Extremas ermittelt werden kann, wird die Gleichung abgeleitet und mit Null gleichgesetzt.²⁸

Die Ausgangsfunktion ist wieder:

$$f(x) = e^{g(x)}, \text{ mit } g(x) = -(x - P)^2 \cdot \lambda$$

Ausmultiplizieren von $g(x)$

$$g(x) = -(x - P)^2 \cdot \lambda$$

$$g(x) = -(x^2 - 2Px + P^2) \cdot \lambda$$

$$g(x) = -\lambda x^2 + 2\lambda Px - P^2 \lambda$$

Ableiten:

$$f(x)' = e^{g(x)} \cdot g'(x) = e^{-\lambda x^2 + 2P\lambda x - P^2 \lambda} \cdot (-2\lambda x + 2P\lambda)$$

Aufsummieren aller einzelnen Ableitungen der Funktionen f_i mit passend positionierten Maximas durch die P_i . Gleichsetzen mit Null zu Evaluierung der Maximas:

$$0 = \sum_{i=1}^n (e^{-\lambda x^2 + 2P_i \lambda x - P_i^2 \lambda} \times (-2\lambda x + 2P_i \lambda))$$

Die Nullstellen dieser Gleichung können numerisch gefunden werden. Beispielsweise mit dem Befehl `fsolve` aus der *Python*-Library `scipy.optimize`. Numerische Verfahren zum Finden von Nullstellen garantieren in den meisten Fällen nicht, dass sie alle Nullstellen finden. Mit mehrmaligem Ausführen des Befehls mit verschiedenen Startwerten im zu überprüfenden Intervall bekommt man wahrscheinlich akzeptable Resultate.

²⁸ Ina de Brabandt, "Ableitung der Exponentialfunktion", Mathematik-Oberstufe, <http://www.mathematik-oberstufe.de/analysis/e/e-funktion-ableiten.html> (Zugriff: 8.7.16)

2.1.2.5 Umsetzung der Häufungspunkte Methode

Unglücklicherweise wurde die Entwicklerumgebung TouchDesigner nicht für solche Anwendungen konzipiert. Der grösste Teil dieses Algorithmus müsste in der Sprache Python geschrieben werden. Aufgrund des Zeitdruckes und fehlender Kenntnisse habe ich mich für eine andere Lösung entschieden. Ein vorgefertigtes TouchDesigner Element, das für benutzerdefinierte Animationen konzipiert ist, kann auch verwendet werden um Funktionen zu definieren. Diese Funktionen basieren nicht auf einer Gausskurve. Der Benutzer setzt sogenannte Keyframes, welche die Position der Funktion bestimmen. Zudem bestimmt er, ob ein Punkt eine Konstante darstellt oder eine quadratische Steigung einleiten soll. Die folgende Tabelle erstellt eine Funktion, die nahezu identisch zur Funktion mit der unten beschriebenen Gleichung ist:

$$f(x) = e^{(-(x - 2)^2 / 2)}$$

id	x	y	expression
0	0	0	cubic()
0	1	2	constant()
0	4	0	cubic()

Der Nachteil dieser Vorgehensweise ist, dass die Funktion mit einer limitierten *Samplerate* abgetastet wird. Das heisst der Entwickler definiert, welcher Bereich der Funktion ausgegeben werden soll und mit welcher Auflösung. Diese Auflösung ist stark durch die Prozessorleistung limitiert.

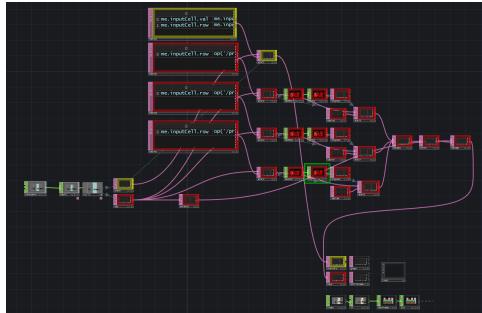


Abb. 11

Dieses Netzwerk generiert innerhalb weniger Millisekunden eine beliebige Anzahl Funktionen.

Die fertige Funktion eines Liedes sieht dann folgendermassen aus:

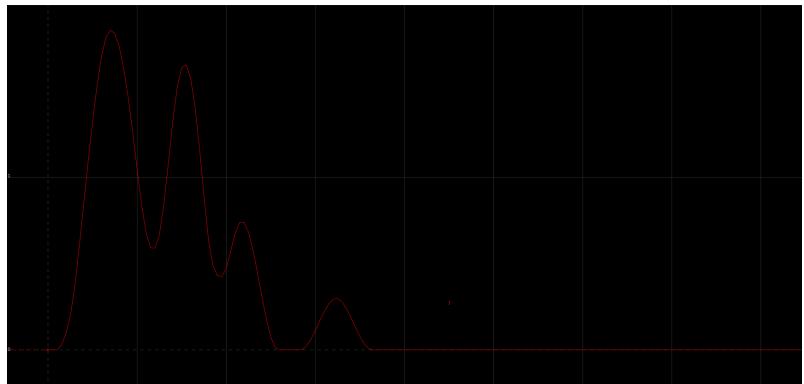


Abb. 12

Diese Funktion wurde mit den Daten eines Hip-Hop Liedes generiert.

Die Funktion zeigt eine relativ exakte Darstellung der Rhythmusstruktur. Die Höhe der verschiedenen Maxima ist mit der Häufigkeit des Zeitabstandes gleichzusetzen. Das Ergebnis der Funktion ist zufriedenstellend. Statt die Funktion abzuleiten, kann ich die Maximas einiges effizienter finden. Da die Funktion limitiert ist auf eine gewisse Auflösung, lassen sich die Maximas durch ein simples "If Statement" finden. Ein Maxima ist nur dort zu finden, wo die Werte links und rechts davon tiefer sind.

Da jetzt mehrere Referenzwerte vorhanden sind, muss das in Kapitel 2.1.2.2 beschriebene System adaptiert werden. Zuerst werden die Unterschiede zwischen dem zuletzt gemessenen Zeitabstand und den neu errechneten Zeitabständen gemessen. Der Wert wird nun dem Maxima zugeordnet, das den geringsten Unterschied zum gemessenen Wert aufweist. Falls sich dieser Wert innerhalb eines Toleranzbereiches befindet, ist keine Veränderung notwendig. Andernfalls wird nach dem in Kapitel 2.1.2.2 beschrieben Verfahren vorgegangen.

2.1.2.6 Zweite Testphase

Das System verhält sich nun sehr stabil. Der Grenzwert pendelt sich innert wenigen Sekunden ein und bleibt dann ziemlich konstant während eines bestimmten Liedteiles. Die Veränderungen während eines bestimmten Liedteiles sind nur noch im tausendstel Bereich zu erkennen. (Der Grenzwert wird zwischen 0 und 1 gesetzt).

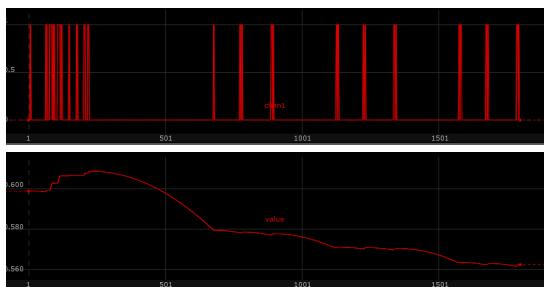


Abb. 13 & 14

Diese Graphen wurden mit den Daten eines Hip-Hop Liedes über 30 Sekunden generiert. Die obere Grafik zeigen die Bassimpulse. Unten ist der Grenzwert dargestellt. Dabei entspricht eine Rasterlinie auf der y-Achse einem Wert von 0.1.

Der Grenzwert der Bassfrequenzen wird von drei verschiedenen Werten beeinflusst. Einmal vom Unterschied zu den berechneten Idealwerten, dann der Zeitdifferenz zwischen dem letzten registrierten Überschreiten des Grenzwertes und dem gegenwärtigen Wert. Und als drittes von der Länge des Impulses.

**Abb.15**

- Rot = Unterschied zu Idealwerten
- Gelb = Warten auf Signal
- Grün = Impulslänge
- Blau = Finaler Grenzwert

2.1.3 Differenzierung zwischen einzelnen Liedteilen

Wie bereits in 2.1 erwähnt, ist eines der Ziele dieses Programms einzelne Liedteile von anderen und somit auch verschiedene Lieder zu unterscheiden. Der Software wird nicht direkt mitgeteilt, wann ein neues Lied Startet. Dies muss sie selber eruieren können. Auch hier konnte ich keinen bestehende Methode ausfindig machen und musste ein eigenes Prinzip entwickeln.

2.1.3.1 Annahme

Beobachtungen haben gezeigt, dass sich die Struktur der Funktion aus Abbildung 10 je nach Lied und Liedteil unterscheidet. Das Ziel besteht nun darin einen Kennwert zu definieren, der verschiedene Funktionen so gut charakterisiert, dass er sich signifikant ändert, falls das Lied wechselt, oder ein neuer Liedteil beginnt.

2.1.3.2 Ausführung

Wenn man die Höhe der Maximas mit deren Position multipliziert, diesen Vorgang für alle Maximas wiederholt und schlussendlich alle Ergebnisse zusammenaddiert, erhält man einen Wert, der die Funktion gut zusammenfasst und Veränderungen meist klar widerspiegelt. Ein Python Script sendet ein Signal sobald der Unterschied des gegenwärtigen Wertes zum letzten Wert einen Grenzwert überschreitet. Die Methode erfordert aber eine Schutzfunktion, die verhindert, dass ein Übergang mehrere Impulse auslöst. Die Funktion enthält nach einem Übergang immer noch Daten aus dem alten Liedteil, somit verändert sich die Funktion über eine gewisse Zeitspanne. In der finalen Version wird der Impuls für 20 Sekunden nach einem Übergang gesperrt.

2.1.3.3 Testphase

Die Methode erweist sich als erstaunlich zuverlässig. Es ist nun möglich, meistens innerhalb weniger als einer Sekunde, ein neues Lied oder einen neuen Liedteil zu erkennen. Die Zuverlässigkeit des Systems wurde von mir nicht ausgiebig getestet, jedoch schätze ich sie

auf etwa 80%. Diese subjektive Einschätzung bestätigt sich in den vielfältigen Übergängen innerhalb der Liedstrukturen.

2.1.4 Auswertung

Das Programm setzt eigenständig einen Wert anhand von gewonnenen Daten um. Die Lernfähigkeit des Programms funktioniert gut und schnell. Obwohl die Analyse der Regelmässigkeit (2.1.2.1, *Anfängliches Grundkonzept*) einen merkbaren Einfluss auf den Grenzwert hat, funktioniert das System auch nur mit den zwei anderen Einflüssen (2.1.2.2). Um die hohen Töne zu extrahieren, verwende ich erfolgreich dasselbe System aber ohne den Einfluss der Rhythmusanalyse. Die Erstellung der Häufungspunkte Methode (2.1.2.4), welche ich speziell für die Rhythmusanalyse erstellt habe, war trotzdem sinnvoll. Denn die Erkennung neuer Liedteile (2.1.3) baut auch auf der Häufungspunkte Methode auf.

2.1.5 Verbesserungsmöglichkeiten

Die Lernfähigkeit des Programms liesse sich durch einen variabel extrahierten Frequenzbereich verbessern. Es gibt einzelne Lieder, in welchen die Paukenschläge aus dem Frequenzbereich 20 bis 100 Hertz nicht extrahierbar sind.

Die Anwendung der Häufungspunkte Methode lässt sich in diesem Kontext verbessern, indem nach einem Erkennen eines neuen Liedteils die dazu verwendeten Daten zurückgesetzt werden. Zusätzlich könnte man die gesamte Musikanalyse auf mehr als nur zwei Frequenzbereiche ausbauen.

Die Lernfähigkeit des Programms könnte man stark ausbauen. Zurzeit ist der Datensatz klein und es wird nur ein Wert vom Programm gesetzt. Es wäre möglich komplexe lernfähige Algorithmen wie beispielsweise *Tensor Flow* von *Google* auf Musikstücke anzuwenden. Mithilfe passender Datenbanken könnten aus den Liedern Informationen über Stimmung, Geschwindigkeit und Musikrichtung entnommen werden.²⁹

²⁹ Unbekannt, "About TensorFlow", TensorFlow, <https://www.tensorflow.org/> (Zugriff: 16.10.16)

2.2 Kinetische Bewegungsanalyse

Die kinetische Bewegungsanalyse baue ich auf den Daten eines *XBOX Kinect Sensor's* auf. Dieser wurde für die Spielkonsole von *Microsoft* konzipiert, fand aber auch grossen Anklang bei Software Entwicklern und im Bereich der wissenschaftlichen Datenerstellung. Microsoft verkauft nun einen Adapter, der es ermöglicht, den Kinect über ein *USB 3.0* Kabel mit dem Computer zu verbinden. Die Entwicklerumgebung *Touch Designer* ermöglicht direkten Zugriff auf die Daten des *Kinects*.^{30 31}

Ich werde die Bewegungsanalyse während der öffentlichen Ausstellung testen und verbessern. Es ist schwierig allfällige Problemlagen alleine zu testen.

2.2.1 Hardware

Der *Kinect* verfügt über eine *1080p* Kamera sowie eine Infrarot Kamera mit einer Auflösung von 512×512 . Die *TOF* (siehe *Glossar*) Kamera ermöglicht dem *Kinect* eine dreidimensionale Ausmessung des Raumes mit Hilfe von Lichtimpulsen. Die Lichtimpulse werden in einem bestimmten Muster ausgesandt, das sich nie wiederholt und somit jedem Punkt eine Identität gibt. Eine Kamera, die leicht anders positioniert ist als der Infrarot Projektor, erkennt diese Strukturen und berechnet mithilfe von Trigonometrie dessen Distanz zum Sensor. Zudem nutzt der Sensor eine Methode, die sich "Depth from focus" nennt. Diese nutzt die Tatsache, dass sich Objekte die weiter von der Kamera entfernt sind, eine gewissen Tiefenschärfe aufweisen. Durch eine astigmatische Linse werden Kreise in Ellipsen umgewandelt. Diese Ellipsen ausserhalb vom Fokus weisen eine andere Form auf als Objekte im Fokus.^{32 33 34}

FIG. 4

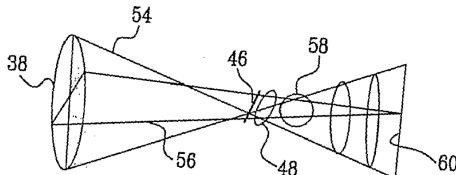


Abb 16

Unterschrift³⁵

³⁰ Adam Mann, "Scientists Hack Kinect to Study Glaciers and Asteroids", *Wired*, <https://www.wired.com/2011/12/hacked-kinect-science/> (Zugriff: 15.10.16)

³¹ Unbekannt, "Kinect", *Microsoft*, <https://developer.microsoft.com/en-us/windows/kinect> (Zugriff 6.8.16)

³² Unbekannt, "Overview for 3D Depth Sensing", *Texas Instruments*, <http://www.ti.com/lscds/ti/sensors/3d-imaging-sensors-overview.page> (Zugriff 23.8.16)

³³ Unbekannt, "Kinect for XBOX ONE", *Microsoft*, <http://www.xbox.com/de-CH/xbox-one/accessories/kinect-for-xbox-one> (Zugriff: 6.8.16)

³⁴ John McCormick, "How does the Kinect Work", <http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf> (Zugriff: 6.8.16)

³⁵ Barak Freedman, Alexander Shpunt und Yoel Arieli, "Distance-Varying Illumination and Imaging Techniques for Depth Mapping", *Prime Sense LTD*, <http://www.google.com/patents/US20100290698> (Zugriff: 12.8.16)

2.2.2 Daten

Der Kinect V2 hat die Fähigkeit die Umwelt als grafischer Z-Buffer wiederzugeben. Dieser grafische Z-Buffer nennt man auch Depth Map. Das heisst, nahe Objekte werden weiss dargestellt und Objekte weiter entfernt schwarz. Zudem erstellt der Kinect ein virtuelles Skelet mit 26 Gelenkpunkten. Auf geringe Distanzen ist der Sensor sogar in der Lage, 30 Gesichtspunkte in Echtzeit zu verfolgen. Der Kinect kann höchstens sechs Personen gleichzeitig erkennen. Die Bewegungsanalyse sollte Daten liefern, die keine störenden Werte aufweisen. Zudem wäre es ideal, wenn mehrere Menschen gleichzeitig mit dem System interagieren könnten. ^{36 37}

2.2.3 Filtern der Daten

Der Kinect liefert die sogenannte "World Space Position" der Personen. Die "World Space" ist ein dreidimensionales Koordinatensystem, das jeweils von -1 bis 1 reicht. Das heisst die maximale Reichweite in jeder Achse wird als 1 oder eben -1 definiert. Falls der Sensor eine Person erkennt, überarbeitet er die Ursprungswerte von 0 auf die jetzige Position. Damit die Daten weiterverwendet werden können, dürfen keine abrupten Veränderung auftreten. Zudem tritt ein Problem auf, wenn mehrere Personen vom Sensor erkannt werden. Die verschiedenen Werte können aufaddiert werden, wenn aber eine Person bei (0.8/0.4/-0.1) steht und die andere bei (0.6/0.1/0.1) addiert sich dies zu (1.4/0.5/0). Je nach der Anzahl erkannter Menschen ändert sich mit diesem System auch die Intensität der Veränderung. Daten von sechs Personen können sich zu einem Wert von -6 bis 6 aufaddieren. Es ist sinnvoll lediglich die Grösse der Veränderung der Werte zu extrahieren. Diese wird aber limitiert, um unerwünschte Veränderungen auszuschneiden.

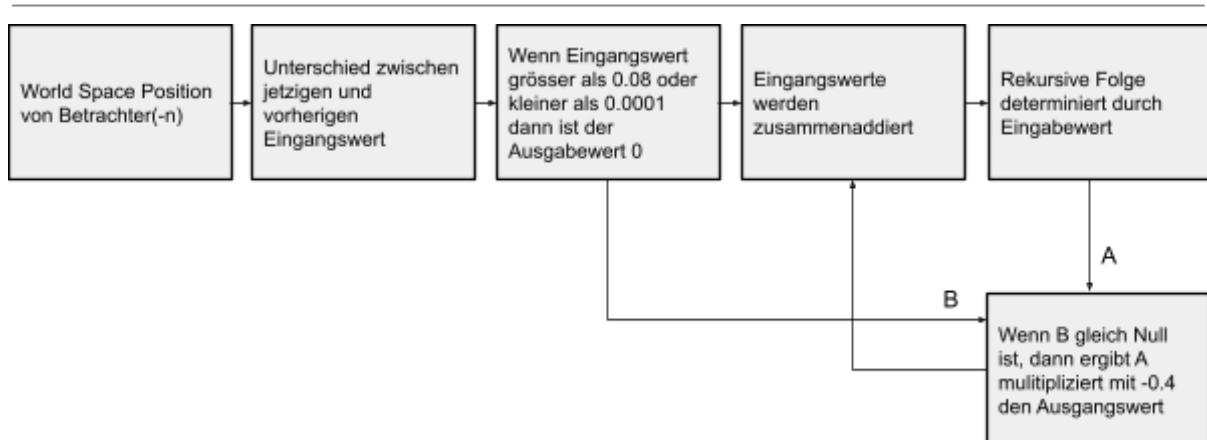


Abb.17

Schematische Darstellung der Filterung kinetischer Daten.

³⁶ Margaret Rouse, "z-buffering", techtarget, <http://whatis.techtarget.com/definition/z-buffering> (Zugriff: 23.7.16)

³⁷ Matthew Szymczyk, "How Does The Kinect 2 Compare To The Kinect 1?", Zugara, <http://zugara.com/how-does-the-kinect-2-compare-to-the-kinect-1> (Zugriff: 23.7.16)

In der Abbildung 17 ist ein Filtersystem zu erkennen. Das Modul in der Mitte lässt nur Veränderungen zwischen 0.08 und 0.0001 durch, somit werden Störsignale, die erfahrungsgemäss grösser sind, aussortiert und wenn sich die Person kaum mehr bewegt, wird der Ausgangswert Null gleichgesetzt. Das Modul unten rechts reagiert nur falls der Wert Null ist. In diesem Fall, nähert es sich dem finalen Wert Null an. Somit wird das System langsam wieder zurückgesetzt.

2.2.4 Eigene Positionsverfolgung

Da die Positionsverfolgung von der Kinect auf 6 Personen limitiert ist, habe ich ein alternatives System aufgebaut. Die Depth Map vom Kinect wird auf einen Streifen der nur einen Pixel hoch ist limitiert. Dieser wird auf einer Höhe gewählt, in der die Betrachter gut ersichtlich sind, idealerweise auf der Höhe vom Hals. Mithilfe eines Farbfilters wird der Streifen so getrimmt, dass die Personen um einiges intensivere Farben haben als der Hintergrund. Später wird er, immernoch auf dem GPU, verwischt, damit keine scharfen Konturen mehr zu sehen sind. Dies ist notwendig, da der vom Kinect ausgehende Depth Map vielerorts Störsignale aufweist. Dank dem Verwischen sind kleine Störsignale nicht mehr zu sehen, ausserdem werden die Bewegungen der Personen weicher. Dieser Streifen kann wiederum in eine Funktion umgewandelt werden. Die Ableitung davon ergibt die Position der Maximas, oder auch die Position der Personen.

2.2.5 Auswertung

Ich habe die Schwierigkeit der Bewegungsanalyse leicht unterschätzt. Entgegen meinem Zeitplan begann ich mit der Erarbeitung der Bewegungsanalyse erst kurz vor Beginn der Ausstellung. Die Bewegungsanalyse war zu Beginn der Ausstellung immer noch fehleranfällig. Das grösste Problem ist, dass dem Kinect nicht gesagt werden kann, welche Personen er anvisieren soll. Wäre dies möglich, könnten Personen ausgewählt werden, die sich in einem bestimmten Bereich der Ausstellung befinden. Das Auswahlverfahren vom Kinect ist leider unberechenbar.

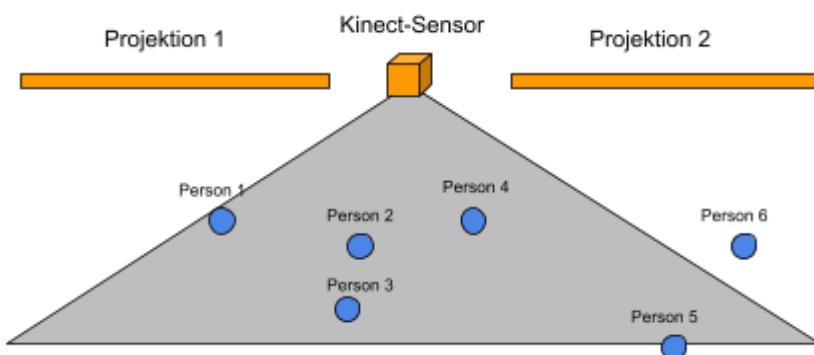


Abb 18
Vogelperspektive vom
Aufbau der Ausstellung

In der schematischen Darstellung (Abbildung 18) ist der Grundriss der Ausstellung zu sehen. Das graue Dreieck in der Mitte symbolisiert den Sichtbereich des Sensors. Personen ausserhalb dieses Bereiches werden nicht erkannt. Person 5 und Person 1 befinden sich auf der Grenze und führen zu stark oszillierenden Werten. Zudem befindet sich Person 3 gleich hinter Person 2 und verdeckt somit die Sicht vom Kinect, was auch zu Störwerten führt. Obwohl die Methode hinter der Datenfilterung durchaus Sinn macht, können damit nicht alle Störwerte ausgefiltert werden.

Die eigene Positionsverfolgung funktioniert, jedoch hatte ich keine Zeit mehr eine spezielle Datenfilterung für dessen Daten zu programmieren. Somit bleibt die eigene Bewegungsverfolgung nur als Konzept in der Software implementiert.

Nach Herumprobieren und Adaptieren der Bewegungsanalyse konnte ich gegen Ende der Ausstellung doch sinnvolle Resultate erzielen. Die fertige Bewegungsanalyse sendet die Daten der Köpfe und Hände an die Visualisierung weiter.

2.2.6 Verbesserungsmöglichkeiten

Um eine genaue Position der Personen zu erhalten, könnte der Kinect an der Decke befestigt werden. Weil die Passanten weniger weit Weg vom Kinect sind als der Boden, kann daraus die Position einzelner Passanten extrahiert werden. Der Nachteil ist, dass daraus keine Daten über die Position der Hände extrahiert werden können. Andernfalls müssten mehrere Kinect Sensoren verwendet werden, um genauere Daten zu bekommen oder es muss auf einen anderen Sensor zurückgegriffen werden.

3. Visualisierung

Das finale Programm beinhaltet 15 verschiedene Visualisierungen, die ausgewerteten Audiodaten in bewegte Bilder umsetzen. Teilweise basieren Visualisierungen auf ähnlichen Konzepten, daher werde ich nicht auf alle Visualisierungen eingehen. Eine Visualisierung benötigt je 150 - 400 Linien OpenGL Shading Code und 20 -150 Node-Elemente.

3.1 Grundlagen zur 3D Visualisierung

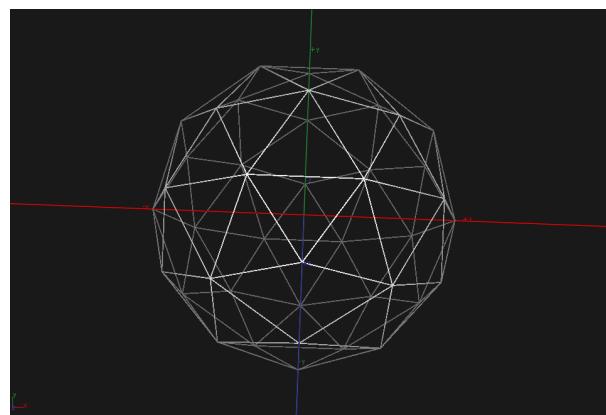
Ein dreidimensionales Objekt digital replizieren und abbilden ist ein sehr rechenintensives Unterfangen. Manche Formen haben Tausende von Eckpunkten. Da ein CPU eine Architektur hat, mit der er nur wenige Operationen aufs Mal ausführen kann, wird für 3D Grafiken die Grafikkarte verwendet. Diese kann tausende Operationen gleichzeitig ausführen. Um mit dieser komplexen Hardware zu kommunizieren, gibt es verschiedene Werkzeuge, welche einem die Kommunikation erleichtern. TouchDesigner ermöglicht und vereinfacht dem Entwickler mit Hilfe von *GLSL* Prozesse auf dem GPU auszuführen.³⁸

3.1.1 Digitalisierte 3D Objekte

Um dreidimensionale Objekte digital zu erfassen und abzuspeichern, braucht es ein sinnvolles System. Da bei Objekten meist nur die Oberfläche zu sehen ist, macht es Sinn nur diese digital zu erfassen. Die einfachste Methode eine Fläche darzustellen ist die Beschreibung in Form von Dreiecken. Aus Dreiecken gebaute Objekte erlauben die höchste Komplexität mit der kleinsten Datengröße. Von einem Dreieck werden nur die Punkte und die jeweiligen Eckpunkte eines Dreieckes abgespeichert, die sogenannten Vertices.³⁹

Abb. 19

Das Bild zeigt eine Kugel, die durch Dreiecke angenähert wird. Dieses Beispiel verwendet 80 Dreiecke, welche durch 42 Vertices abgespeichert werden.

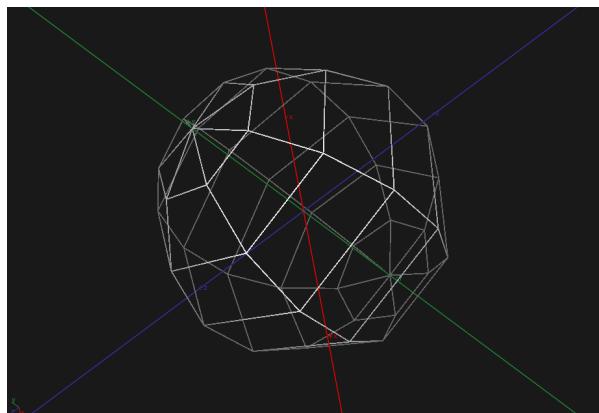


³⁸ Unbekannt, What's the difference between a cpu and a gpu?", nVIDIA, <https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/> (Zugriff: 28.9.16)

³⁹ Máté Kovács, "Why does Graphics hardware only render triangles?", Quora, <https://www.quora.com/Why-does-graphics-hardware-only-render-triangles> (Zugriff: 1.10.16)

Abb. 20

Das Bild zeigt eine Kugel, die durch Quadrate angenähert wird. Diese Beispiel verwendet 56 Quadrate, welche durch 44 Vertices abgespeichert werden.



Die Abbildungen 19 und 20 zeigen, warum Dreiecke eine effizientere Speichermethode sind. Beide Formen haben annäherungsweise eine gleich grosse Datenmenge. (42 und 44 Punkte). Trotzdem können mithilfe der Dreiecke 24 Flächen mehr dargestellt werden. Während in der Kugel aus Dreiecken ein Vertex Punkt der gemeinsame Eckpunkt von sechs Dreiecken ist, sind die Punkte in der Quadratform nur Eckpunkte von vier Punkten. Auch ist zu beobachten, dass die Pole der Quadratform nicht durch Quadrate darzustellen sind und es muss wieder auf Dreiecke zurückgegriffen werden.

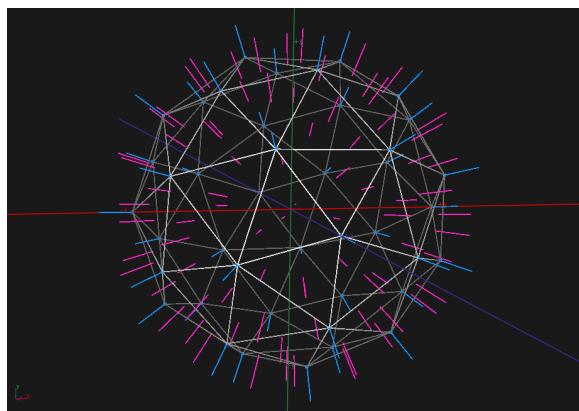
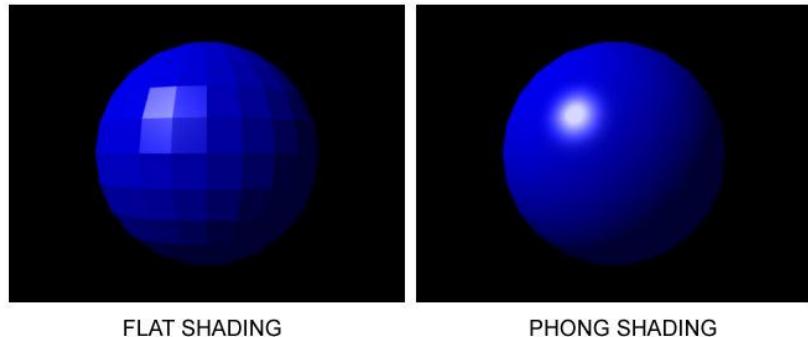
3.1.2 Belichtung von 3D Objekten

Bei komplexen Objekten ist die Anzahl der Punkte, die für ein 3D Objekt verwendet werden kann, schnell durch die Hardware limitiert. Eine realistische Belichtung ist kaum möglich. Um ein Objekt wirklich rund aussehen zu lassen, muss die Farbgebung der Oberfläche für beinahe jedes Pixel eines Objektes einen anderen Wert aufweisen. Da ein Polygon meist durch mehrere Pixel dargestellt wird, interpoliert man beim Berechnen des Ausfallwinkels vom Licht die Punkte zwischen den Punkt Normalen. Das Interpolieren erlaubt dem Programm eine viel realistischere Darstellung einer Oberfläche. In der Computergrafik verwendet man vor allem das *Phong* Verfahren sowie auch das *Gouraud Shading*.⁴⁰

⁴⁰ Eric Haines, "Interaktive 3D Grafiken", Udacity, <https://de.udacity.com/> (Zugriff: 10.15 - 11.15)

Abb. 21

Das Bild zeigt eine Kugel, die durch flaches Schattieren abgebildet wird und eine, die durch das *Phong* verfahren verfeinert wurde.

Abb. 22

Dargestellt ist eine Kugel mit ihren Punkt Normalen (Blau) und ihren Polygon Normalen (Pink).

3.1.3 OpenGL

OpenGL ist eine grafische Programmbibliothek, die sich vor allem für 3D Darstellungen eignet. Eine Programmbibliothek beinhaltet eine Sammlung von Lösungswegen für konkrete Probleme. Beispielsweise muss der Einfallswinkel einer Lichtquelle auf ein Objekt und die daraus resultierenden Farbgebungen nicht mehr vom Entwickler selber berechnet werden. Der Entwickler ruft eine Funktion auf, die speziell für Lichtberechnungen konzipiert wurde. Diese Funktion speist er mit den Informationen einer oder mehreren Lichtquellen. So wird dem Entwickler viel Arbeit abgenommen.

Auch perspektivische Verzerrung, Z-Buffer und weitere Grundfunktionen der Computergrafik sind bereits in der Programmbibliothek enthalten. *OpenGL* wird standartmäßig mit dem Grafikkartentreiber mitinstalliert.^{41 42 43}

⁴¹ Martin Christen, "Lighting", OpenGL, <https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/lighting.php> (Zugriff: 13.10.16)

⁴² Unbekannt, "Lighting", GL programming, <http://www.glprogramming.com/red/chapter05.html> (Zugriff: 13.10.16)

⁴³ Unbekannt, "What is OpenGL", OpenGL, https://www.opengl.org/wiki/FAQ#What_is_OpenGL.3F (Zugriff: 13.10.16)

3.1.4 GLSL

GLSL ist eine erweiterte Version von *OpenGL*. *GLSL* baut auf dem gleichen Prinzip wie *OpenGL* auf. Der einzige Unterschied ist, dass der Entwickler auf verschiedene Prozesse der Render-Pipeline Einfluss nehmen kann. In der Standard Version von *OpenGL* wird ein 3D Objekt als fertiges Objekt in die Render Pipeline gegeben. Das heißt, die Struktur dieses Objektes lässt sich nur vor der RenderPipeline verändern. Ein solcher Prozess ist lediglich auf dem CPU möglich und daher extrem ineffizient (3.1). Der Vertex Shader erlaubt dem Entwickler die Position eines Punktes auf der Grafikkarte zu verändern. Mithilfe des Geometry Shaders kann der Entwickler für einen Punkt bis zu 30 weitere Punkte hinzufügen. Der Fragment Shader, erlaubt einem, alles am Bild zu verändern, was nicht die Grundstruktur ist. Dieser ist auch unter dem Namen "Pixel Shader" bekannt. Mithilfe des Fragment Shaders können beispielsweise Reflexionen berechnet werden. Mit *GLSL* ist es möglich Texturen und andere Daten in jeden Shader einzuspeisen. Die Codes für den Fragment Shader können von einem Compiler auf Fehler überprüft werden, jedoch können keine Zahlenwerte die Render Pipeline verlassen. Die einzigen Daten die aus der Pipeline kommen sind die fertigen Bilder. Wenn auf dem CPU gerechnet wird, können Zwischenschritte als Zahlenwerte im Compiler dargestellt werden. Falsche Endresultate ohne Fehler in der Programmstruktur sind also viel schwieriger aufzuspüren.⁴⁴

⁴⁵

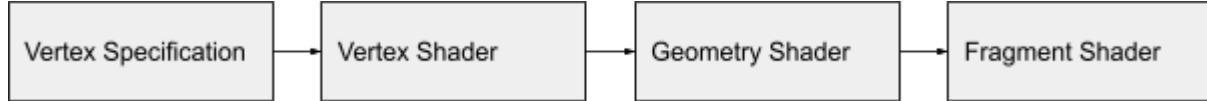


Abb. 23

Dargestellt eine stark vereinfachte Darstellung der Elemente, die durch *GLSL* in der *OpenGL* Render-Pipeline veränderbar sind.

⁴⁴ Alfonse, "Rendering Pipeline Overview", OpenGL, https://www.opengl.org/wiki/Rendering_Pipeline_Overview (Zugriff: 17.10.16)

⁴⁵ Unbekannt, "OpenGL Shading Language", OpenGL, <https://www.opengl.org/documentation/glsl/> (Zugriff: 17.10.16)

3.2 Generierung von RGB Noise auf dem GPU

Damit Strukturen immer wieder neu generiert werden können, wird eine grosse Menge an Positionsdaten benötigt. Hier tritt ein ähnliches Problem auf wie auch schon bei der Berechnung von Geometrien. Es müssen tausende Operationen gleichzeitig ausgeführt werden. Somit ist es auch hier sinnvoll den GPU für diese Aufgabe zu verwenden. In diesem Falle verwende ich *Simplex 4D Noise*. Dieses Programm eignet sich für animierte Noise Funktionen. Damit sich die Strukturen mit der Musik verändern, arbeite ich mehrheitlich mit rekursiven Werten der Musikanalyse. Eine Noise Textur unterscheidet sich von zufälligen Werten. Sie basiert zwar auch auf Zufallswerten. Diese werden jedoch grobmaschig angelegt und die Werte dazwischen daraus interpoliert. Somit sind je nach Einstellungen runde oder harte Formen und Strukturen erkennbar, was viel eher einer natürlichen Struktur ähnelt.^{46 47}

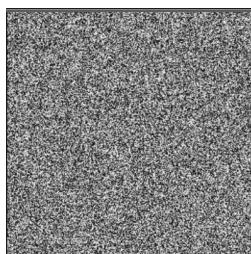
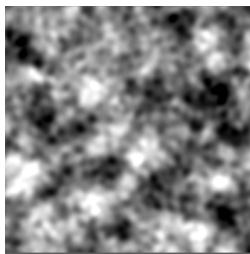


Abb. 24 & 25
Zufällig und Noise

Damit eine dreidimensionale Struktur definiert werden kann, wird ein dreidimensionaler Vektor benötigt. Auf dem GPU können auch farbige Noise Texturen berechnet werden. Um damit eine dreidimensionale Struktur abzubilden, kann der Rotwert der X-Komponente zugeordnet werden. Das Gleiche gilt für Grün, die Y-Komponente und für Blau, die Z-Komponente. Normalerweise verwenden Bilder ein 8-bit Pixel Format. 8-Bit bedeutet, die Daten werden auf acht binären Speichereinheiten gespeichert. Jede dieser Speichereinheiten kann entweder auf 0 oder 1 gesetzt werden. Damit entstehen $2^8 = 256$ Abstufungen für jeden der RGB Farben pro Pixel. Bei normalen Bildern macht eine solche Auflösung Sinn, jedoch resultiert dieses Format in abgerissenen Bewegungen, wenn es auf 3D Objekte angewendet wird. Daher verwende ich ausschliesslich ein 32-bit Format. Dies ermöglicht $2^{32} = 4294967296$ Abstufungen.^{48 49}

⁴⁶ Louis, "Noise TOP", derivative, https://www.derivative.ca/wiki088/index.php?title=Noise_TOP (Zugriff: 9.10.16)

⁴⁷ Ken Perlin, "Noise Hardware", Seite 2, UMBC, 2002, Unbekannter Ort

⁴⁸ Gabe, "fixed point vs floating point number", stackoverflow,

<http://stackoverflow.com/questions/7524838/fixed-point-vs-floating-point-number> (Zugriff: 7.10.16)

⁴⁹ Matt Ball, "jpg bits per pixel", stackoverflow, <http://stackoverflow.com/questions/4305101/jpg-bits-per-pixel> (Zugriff: 7.10.16)

3.3 Strukturen

In diesem Abschnitt werden ein paar der 15 Visualisierungen genauer beschrieben. Jede Visualisierung beinhaltet ein Kamerasystem, verschiedene Lichtquellen und vertiefte GLSL Einstellungen.

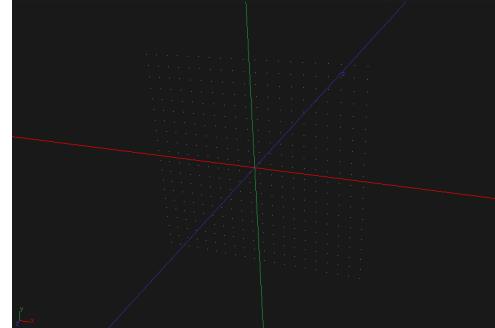
Die Visualisierungen stellen eine Verbindung zwischen Natur und Technik dar. Die Bewegungen stehen für das ästhetische Chaos das in der Natur zu finden ist. Diese Bewegungen werden wiederum von stark abstrahierten Formen aufgenommen. Durch surreale Belichtung und provokative Farben ziehen die Visualisierungen einen Betrachter in ihren Bann.

3.3.1 “Digitale Topografie”

Das Ziel dieser Visualisierung ist eine verpixelte Landschaft darzustellen. Diese Visualisierung ist auf technischem Level die einfachste. Da ich im GLSL Shader keine neue Geometrie kreiere, ist dafür nur der Vertex Shader und der Pixel Shader notwendig. Die hier verwendete Grundgeometrie wird auf dem CPU berechnet, danach aber im Cache Speicher abgelegt. Somit beansprucht die Geometrie während der Ausführung des Programms keine Prozessorleistung. Die Geometrie ist ein Raster aus 200 x 150 Vertices. Diese Geometrie wird zu Partikel konvertiert, das heißt der Typ der Geometrie wird verändert. In der Render-Pipeline können Partikel, im Gegensatz zu anderen Geometrien, ohne in Dreiecke aufgeteilt zu werden, berechnet werden. Der Nachteil ist, Partikel sind immer zweidimensional und können somit nicht perspektivisch verzerrt werden.

Abb. 26

400 rasterförmig angeordnete Partikel



Die Position und die Größe der Partikel kann aber im Vertex Shader beeinflusst werden. Der Shader lädt eine Noise Textur mit denselben Massen. Diese Textur wurde speziell adaptiert, damit eine spannende Oberfläche entsteht. Somit kann jeder Vertex Punkt einem Pixel auf

der Textur zugeordnet werden. Die RGB Werte der Textur werden zu den XYZ Koordinaten der Partikel addiert.⁵⁰ ⁵¹ ⁵²

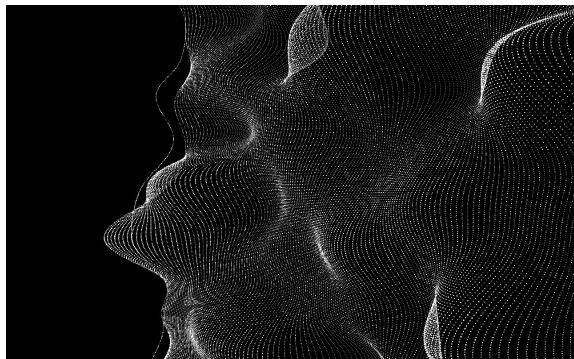


Abb. 27

Für dieses Bild wurden rechteckige, weisse Flächen als Partikel benutzt.

Um die Visualisierung spannender zu gestalten, verwende ich Farbringe als Partikel. Diese Farbringe vergrössern sich mit einer Geschwindigkeit, die linear abhängig zum Ausgabewert der Musik ist. Damit sich auch die Struktur der Oberfläche ebenfalls verändert, deformiert sich die Noise Textur auch proportional zum Ausgabewert der Musik.

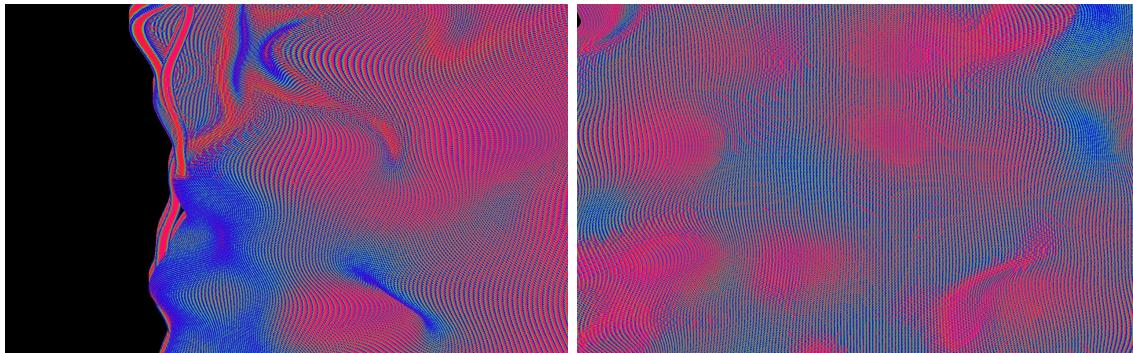


Abb. 28 & 29

Farbige Partikel in einer perspektivischen Ansicht und in der Vogelperspektive

⁵⁰ Louis, "Convert SOP", derriative, https://www.derivative.ca/wiki088/index.php?title=Convert_SOP (Zugriff: 13.10.16)

⁵¹ Malcolm, "Write a GLSL Material", derriative,

https://www.derivative.ca/wiki088/index.php?title=Write_a_GLSL_Material (Zugriff: 15.10.16)

⁵² Unbekannt, "Particles / Instancing", OpenGL tutorial,

<http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/> (Zugriff: 19.9.16)

3.3.2 “Tunnel”

Diese Visualisierung sollte einen Tunnel darstellen der sich um den Betrachter bewegt. Die hier verwendete Grundgeometrie ist ein n -Eck ($2 < n < 7$) das noch auf dem CPU, heisst ausserhalb der GLSL-Pipeline, 100 Mal repliziert wird. Die Anzahl (n) der Ecken wird durch die Liedgeschwindigkeit determiniert. Die Anzahl n der Ecken verändert sich nur sporadisch. Jeder Replikation wird einer Index Zahl zugewiesen. Die Ausrichtung und Skalierung der Formen wird mit Hilfe von Sinuswellen verechnet.

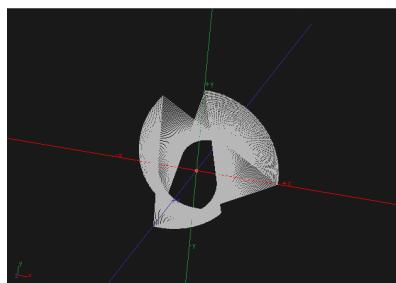


Abb. 30
100 Replikationen eines Dreiecks

Der Shader lädt diese Grundgeometrie in Linien-Form. Das heisst, auf dem Geometrie-Shader sind jeweils Anfangs und Endpunkte des Vektors abrufbar. Damit die Linien dargestellt werden können, müssen sie wiederum in Polygone konvertiert werden. Im ersten Schritt müssen also neue Vertices generiert werden. Zuerst berechne ich die drei Punkt Normalen der Fläche. Diese lassen sich mit dem Kreuzprodukt berechnen. Da ich im Geometrie Shader in diesem Falle aber nur Zugriff auf die zwei Punkte der Linie habe, reicht die Information zum Berechnen des Kreuzproduktes nicht. Die Punkt Normalen werden also auf dem CPU berechnet.

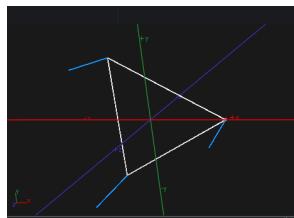
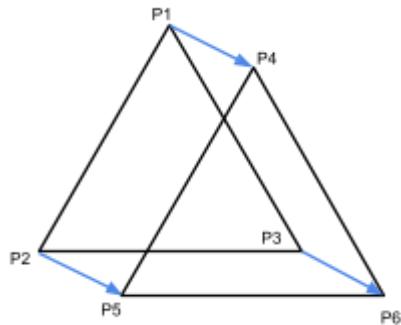


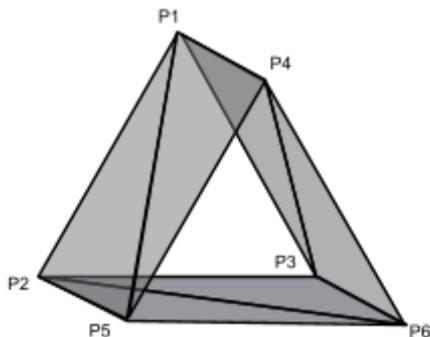
Abb. 31
Blau abgebildet: die Punkt Normalen eines Dreiecks.

Für das Darstellung des n -Eckes benötige ich $n \cdot 2$ Punkte. Die Punkte P_1 bis P_n sind bereits durch die Grundgeometrie gegeben. Die Punkte P_{n+1} bis P_{2n} g, die die Punkt Normalen repräsentieren, geben sich durch Addieren der Punkt Normalen Vektoren zu den Punkten P_1 bis P_n .

Abb. 32

P_1 bis P_3 sind die Eckpunkte eines Dreiecks. Diese werden mithilfe der Punkt Normalen erweitert.

Um die Flächen zu definieren, müssen drei Punkte zusammen als Polygon definiert werden. Somit definiere ich P_1 , P_2 und P_{n+1} als erstes Dreieck. Das zweite schliesst die rechteckige Fläche und beinhaltet somit die Punkte P_{n+1} , P_{n+2} und P_2 . Dieser Prozess wird durchgeführt bis n^*2 .

Abb. 33

Abgebildet sind die Dreiecke, die zur Erstellung der 3D Form notwendig sind.

Nun sind alle Formen als 3D Objekte auf der Grafikkarte, jedoch sind alle an derselben Position. Dies ist die einzige Visualisierung, die nicht auf einer Noise Textur basiert. Hier sollten alle Objekte den gleichen Abstand zueinander haben. Somit ist ein Zufallswert kaum von Nutzen. Trotzdem speichere ich die Positionswerte auch wieder auf der Grafikkarte ab. In diesem Falle verwende ich einen einfachen, zur Musik animierten, Farbverlauf. Dieser Farbverlauf hat eine Dimension von $1 \cdot 100$ Pixel, somit lässt sich jede Instanz des n-Eckes einem Pixel zuordnen.

Für dieses Objekt, müssen keine Normalen definiert werden, da im Pixelshader keine Lichtquellen verwendet werden. Im Pixel wird lediglich die Geometrie als Weiss deklariert. Korrekte Normalen müssen nur berechnet werden, wenn Schattierung von Lichtquellen erwünscht sind.

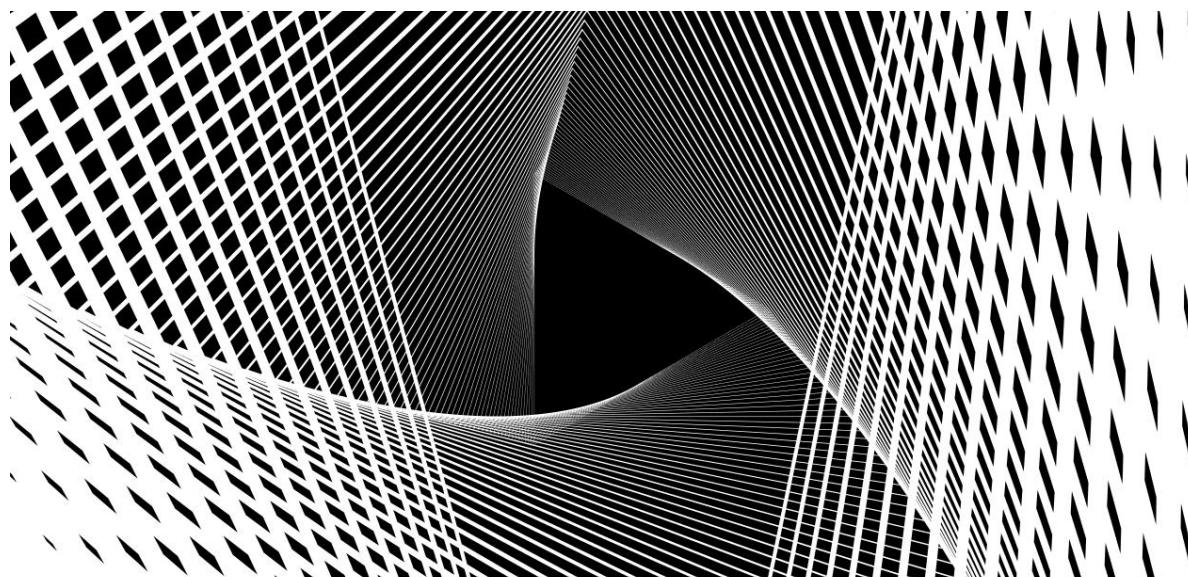


Abb 34

Übereinander gelegte Linien bilden unwirkliche Formen.

3.3.3 “Sphere”

Das Ziel dieser Visualisierung ist ein Kugelförmiges Gitterraster, das relativ zur Musik deformiert wird. Die Grundgeometrie ist eine einfache Sphäre. Zuerst wird die Distanz zum Mittelpunkt der individuellen Punkte durch eine Noise-Textur verändert. Diese Noise-Textur wird wiederum durch die Audioanalyse verzerrt. Der Shader lädt die Linien, die für die Konstruktion eines Polygons notwendig sind. Jetzt sollten diese Linien als dreidimensionale Röhren abgebildet werden. Der Ausgangsvektor ist wieder die Punkt Normale. Diese Punkte Normalen richten sich standartmäßig vom Mittelpunkt der Sphäre weg.

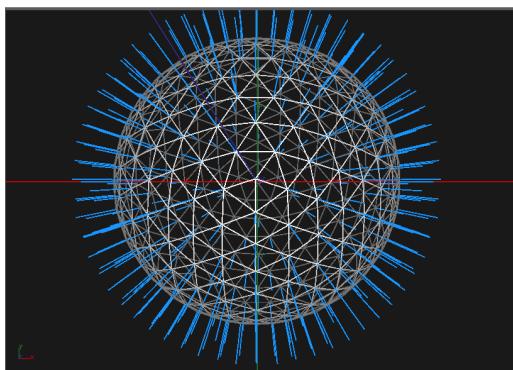


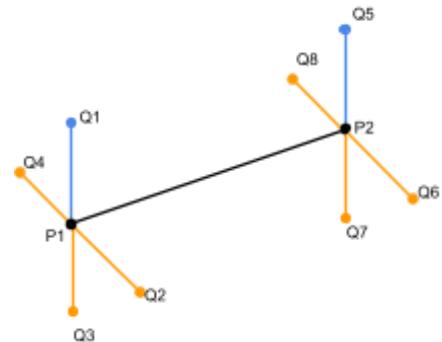
Abb.35

Eine Sphäre mit deren Punkt Normalen (Blau)

Um die Geometrie so simpel wie möglich zu halten, werden für die Röhre nur vier Eckpunkte definiert. Mithilfe der Normale kann der Punkt des ersten Eckpunktes bestimmt werden. Dazu muss der normalisierte Vektor der Punkt Normalen mit der gewünschten Länge multipliziert werden. Diese Länge ist abhängig von der selben Noise Textur. Somit weist eine Röhre je nach Farbigkeit der Noise Textur eine andere Dicke auf. Zudem wird die Länge auch noch durch eine Sinuswelle beeinflusst. Diese basiert wiederum auf einem Rekursiven Folgewert der Audioanalyse.

Abb.36

Eine Linie einer Sphäre. Die Punkt Normalen sind wieder blau eingezeichnet.



Das erste Dreieck wird mit den Eckpunkten Q_1 , Q_2 und Q_5 generiert. Das nächste komplettiert die Fläche und besteht aus Q_2 , Q_5 und Q_6 . Dieser Prozess wird für jede Seitenfläche des gestreckten Würfels angewandt. Damit die Röhre rund wirkt, werden die Punkt Normalen so definiert wie die Vektoren zum Erreichen der Punkte. Das heisst, die Punkte Normale von Punkt Q_1 entspricht dem Vektor P_1 nach Q_1 . Das Reflektionsmodell nach Phong interpoliert die Normalen zwischen den Punkt Normalen.



Der Shader arbeitet hier mit vier verschiedenen Lichtquellen. Mehrere Lichtquellen machen eine Szene sehr schnell visuell ansprechend. Zudem werden hier auch die Schatten berechnet, auch dies führt zu einem höheren Detailgrad.

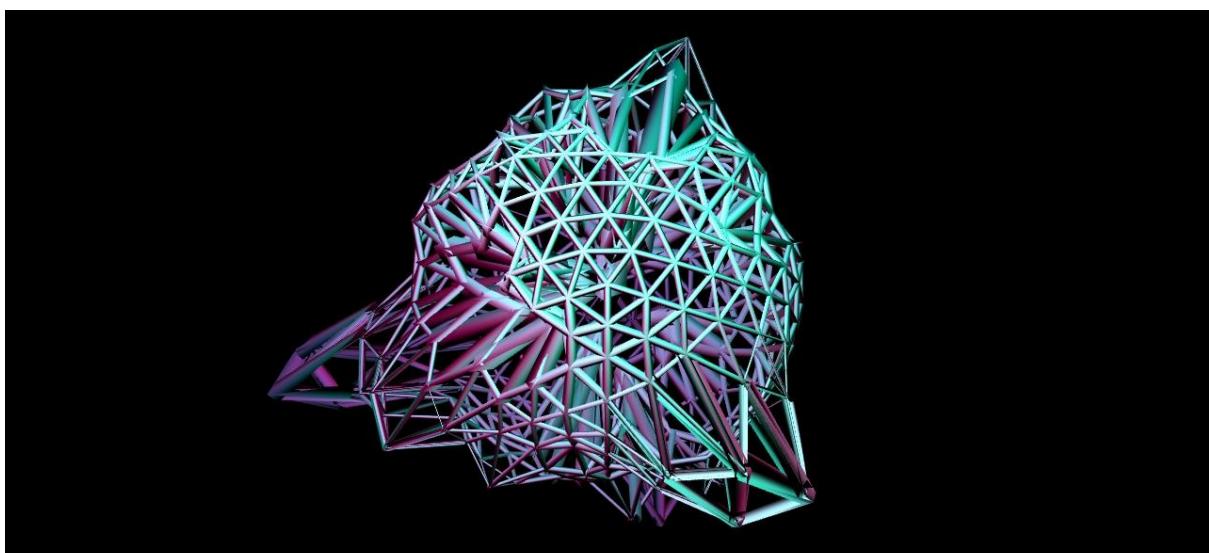


Abb. 38
Ein deformiertes Gitterraster stellt einen Körper im Raum dar.

3.3.4 “Block Szenerie”

Ich will hier Berge und Täler erstellen, welche durch viereckige Säulen anstelle von Flächen dargestellt werden . Die Grundgeometrie ist hier, wie bei 3.3.1, eine Rasterfläche, die zu Punkten konvertiert wird. Der Shader lädt diese Punkte und erstellt für jeden Punkt eine viereckige Säule. Die Säulen müssen wiederum mit Polygonen aufgebaut werden. Der grosse Unterschied zur Visualisierung 3.3.3 ist, dass die Anfänge und Enden der Säulen auch erstellt werden müssen. Zudem sollten diese Säulen eckig und nicht rund erscheinen. Damit die Form eckig erscheint, müssen die vom Phong Modell interpolierten Normalen identisch sein zu den Punkt Normalen. Dazu müssen die Punkt Normalen je nach Polygon neu definiert werden. Wie auf der Abbildung 30 zu erkennen ist, wird ein Punkt von 5 Polygonen geteilt. Die Punkt Normalen müssen also für jede Fläche neu berechnet und definiert werden.

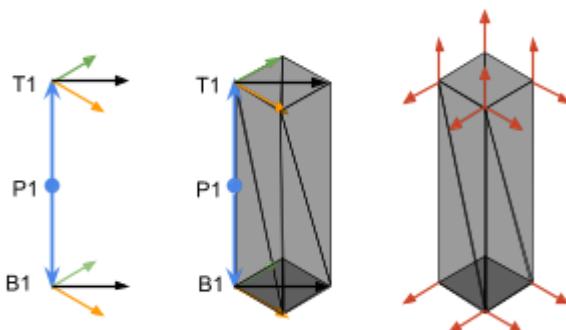


Abb.39
Schematische Darstellung der Vertex und Polygon Struktur einer Säule. In der letzten Darstellung sind die Punkt Normalen für das Belichtungsmodell rot dargestellt.

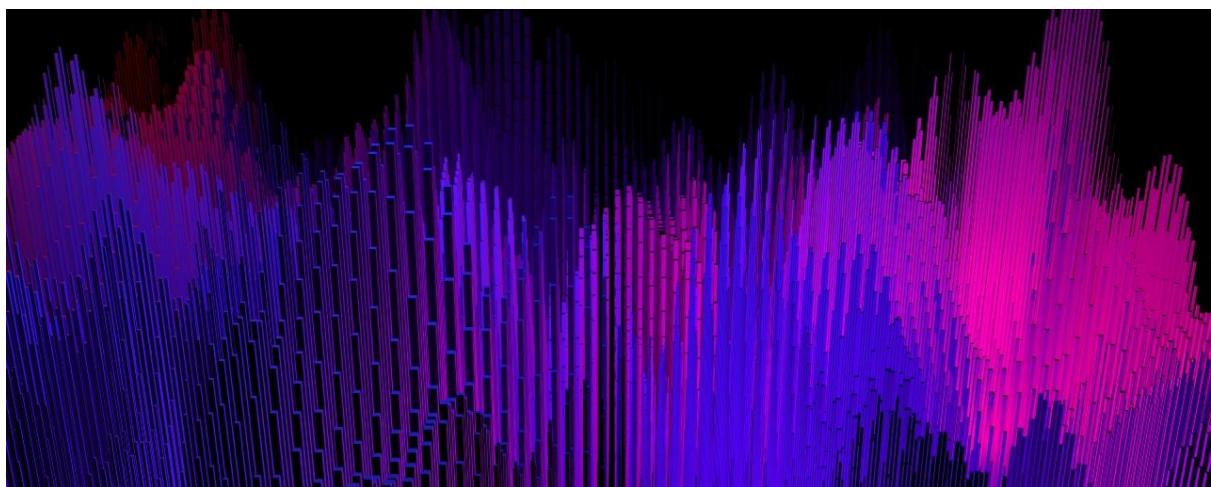


Abb.40

Hunderte Säulen bilden eine surreale Landschaft.

3.3.5 “Oberfläche”

Das Prinzip hinter dieser Visualisierung ist dasselbe wie bei 3.3.1, mit dem Unterschied, dass hier eine Oberfläche dargestellt wird und nicht lose Partikel. Ich will also eine geschlossene Fläche erreichen, die in drei Dimensionen verzerrt wird. Damit dies erreicht wird, muss für jeden Punkt ein Polygon berechnet werden. Punkte mit identischen Koordinaten müssen auch den selben Wert aus der Noise Textur extrahieren. Nur so lässt sich eine nahtlose Fläche generieren.

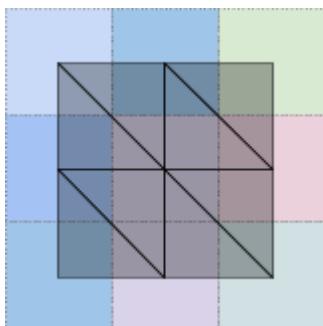


Abb. 41

Schematische Darstellung der Pixel-Vertex Zuordnung.

Interessanterweise kann für eine Oberfläche von Beispielsweise 20 x 20 Punkten auch eine unterdimensionierte Noise Textur verwendet werden. In der Computergrafik werden unterdimensionierte Objekte mit Hilfe von Interpolation hochskaliert.^{53 54}

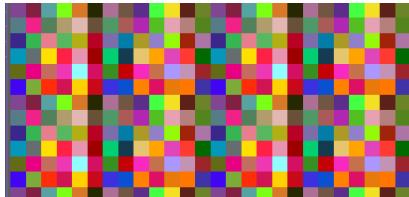


Abb. 42 & 43

Unterskalierung mit und ohne Interpolation.

Wenn die Vertex Punkte nun den Pixeln zugeordnet werden, weisen mehrere Punkte sehr ähnliche Werte auf, da die Interpolation nicht zu sehr weichen Ergebnissen führt. Somit sind in der fertigen Version kantige Flächen zu sehen.

⁵³ Sean McHugh, “image interpolation”, cambridge in colour,

<http://www.cambridgeincolour.com/tutorials/image-interpolation.htm> (Zugriff: 12.19.16)

⁵⁴ JeGX, “OpenGL Interpolation Qualifiers (GLSL)”, geeks 3D,

<http://www.geeks3d.com/20130514/opengl-interpolation-qualifiers-glsl-tutorial/> (Zugriff: 12.10.16)

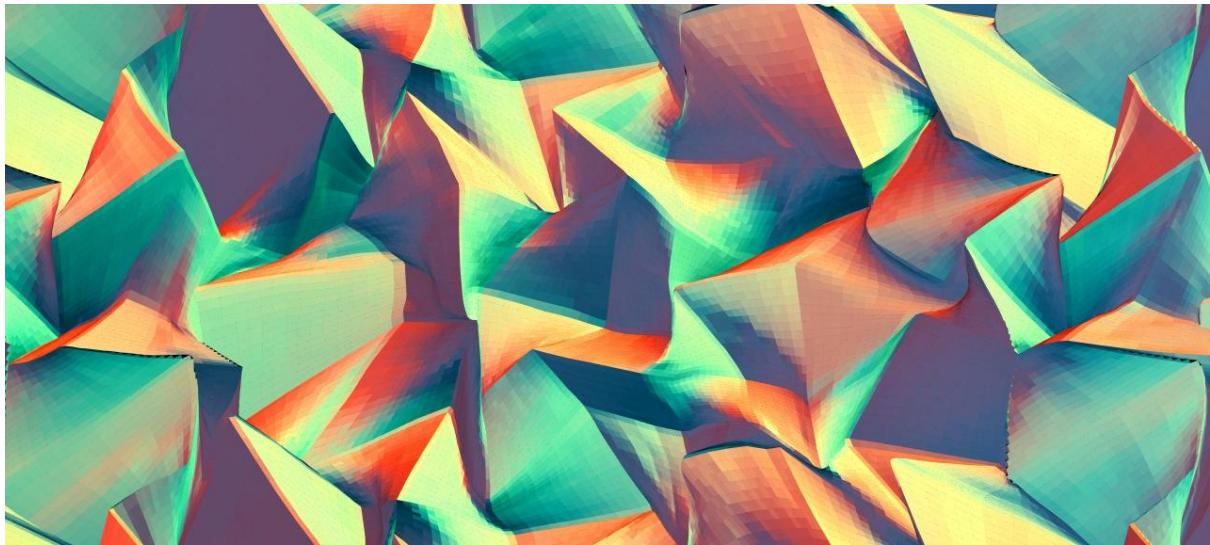


Abb. 44

Ein surreale Oberfläche getaucht in intensives Licht verschiedener Lichtquellen.

3.4 Belichtungssystem

Jede Visualisierung enthält ein Belichtungssystem bestehend aus zwei bis vier Lichtquellen. Die Lichtquellen verwenden alle eine distanzbasierte Abschwächung. Das bedeutet, desto weiter die Lichtquelle entfernt ist, desto geringer ist deren Einfluss auf das Objekt. Diese Lichtquellen können innerhalb der Node Umgebung deklariert und definiert werden. Die GLSL Pipeline lädt dann im Pixel Shader die Daten und berechnet die Farbgebungen und Schattierungen. Die Materialeigenschaften werden im Pixel Shader definiert. Alle Visualisierungen verwenden hier nur eine simple diffuse Lichtbrechung. Es werden weder Texturen noch Reflektionen benutzt.

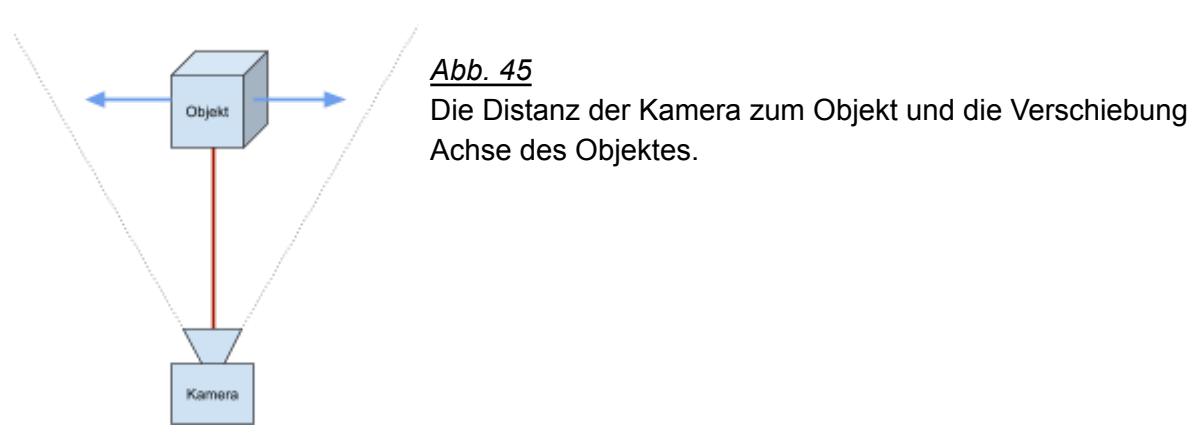
⁵⁵ ⁵⁶

⁵⁵ Louis, "Light COMP", derivative, https://www.derivative.ca/wiki088/index.php?title=Light_COMP (Zugriff: 28.10.16)

⁵⁶ Malcolm, "Write a GLSL Material", derivative, https://www.derivative.ca/wiki088/index.php?title=Write_a_GLSL_Material (Zugriff: 1.11.16)

3.5 Integration von kinetischen Daten

Die Kameraposition wird bei allen Visualisierungen ausschliesslich von den kinetischen Daten beeinflusst, ansonsten ist die Kamera statisch. Das Ziel der kinetischen Manipulation ist es, die 3D Strukturen so zu deformieren, dass sie sich der Perspektive des Betrachters anpassen und somit einen 3D Effekt haben. Es stellte sich schnell heraus, dass dieser Effekt mit einem klassischen Kamerasytem nicht erreicht werden kann und auch ineffizient ist. Das Kamerasytem wird in TouchDesigner auf dem CPU und nicht dem GPU berechnet. Es ist also von Vorteil die Geometrie zu deformieren und nicht die Kameraposition zu ändern. So kann mehr arbeit auf den GPU verlagert werden und es sind weniger Rechenschritte notwendig. Objekte, die keine neuen Daten erhalten, werden auf dem CPU nicht neu berechnet.⁵⁷



Es macht mehr Sinn die Vertex Punkte des Objektes zu verschieben. Das System verwendet hier keine physikalisch akkurate Berechnungen. Die Berechnung der betrachterspezifischen Vertex Verschiebung sieht folgendermassen aus:

$$d = \text{Distanz von Kamera zu Vertex Punkt}$$

$$p.x = X - \text{Koordinate vom Vertex Punkt}$$

$$B.x = X - \text{Koordinate vom Betrachter}$$

$$p.x_{\text{neu}} = p.x + (B.x * d^2 * - 1)$$

In Worten beschrieben verändert sich die Position der Vertex Punkte entgegen der Position vom Betrachter. Je weiter der Punkt von der Kamera entfernt ist, desto stärker ist die Veränderung. d wird mit einem Exponent von zwei verrechnet damit die Veränderung surrealer und interessanter wirkt.

⁵⁷ Louis, "Camera COMP", derivative, https://www.derivative.ca/wiki088/index.php?title=Camera_COMP (Zugriff: 9.10.16)

Da die kinetische Analyse auch Hände, Füsse und andere Körperteile verfolgen kann, sind die Möglichkeiten der Integration fast unbegrenzt. Im Verlaufe der Ausstellung habe ich die Bewegungen der Hände mit anderen Parametern der Geometrien verbunden. Diese waren besser ersichtlich als die Veränderungen durch die Kamera.

3.6 Übergänge

In 2.1.3 wird beschrieben, wie mit der Audioanalyse Liedteile differenziert werden. Die 15 Visualisierungen sind nach Aussehen in vier verschiedenen Kategorien eingeteilt. Ähnliche Strukturen landen in der selben Kategorie. Diese Kategorien werden in einer fixen Reihenfolge durchlaufen. Wenn alle 15 Visualisierungen durchgelaufen sind, werden die Mitglieder innerhalb einer Kategorie untereinander neu gemischt. Durch dieses System sind Wiederholungen auszuschliessen, und das Bild bleibt spannender. Der Betrachter kann nie wissen, was als nächstes zu erwarten ist. Für den Betrachter wirkt diese Verfahren "zufälliger" als "echter" Zufall.

Die Übergänge zwischen den Visualisierungen sind in der Geschwindigkeit abhängig von der Rhythmusgeschwindigkeit. Somit sind in hektischen Liedteilen auch schnellere Übergänge zu beobachten. Die Übergänge bestehen aus stetigem Auf oder Abbauen der Grundgeometrie.

3.7 Auswertung

Auf technischem Level funktionieren die Visualisierungen einwandfrei. Die Darstellungsformen wurden aber von einzelnen Besuchern als unverständlich und zu hektisch bezeichnet.

Die Integration der kinetischen Daten hatte nicht den erwünschten Effekt. Für die Besucher der Ausstellung war schwierig zu erkennen, was sie mit ihren Bewegungen wirklich beeinflussen.

3.8 Verbesserungsmöglichkeiten

Damit die Darstellungen besser auf den Betrachter wirken, wäre es sinnvoll, die Bewegungen der Formen so natürlich und organisch wie möglich zu gestalten. Zudem sind die Visualisierungen meist sehr grob und weisen wenige Details auf. Die Darstellungen können schnell erfasst werden und langweilen den Betrachter dementsprechend. Speziell mit erweiterten Licht und Schattierungseffekten, die eher den Belichtungsbedingungen aus der Natur entsprechen, könnte man den Visualisierungen eine gewisse Tiefe verleihen. Nur die Perspektive mit den kinetischen Daten zu verändern, scheint nicht ausreichend zu sein. Es ist sinnvoller die Bewegungen mit Lichtquellen und anderen Parametern der Geometrie zu verbinden.

4. Optimierung

Damit eine Software optimal läuft, muss die Anzahl ausgeführter Operationen so stark wie möglich reduziert werden, ohne die Funktionalität der Software zu beeinträchtigen. Dieses Programm hat einen Komplexitätsgrad der den CPU und GPU eines leicht überdurchschnittlichen PC's an seine Grenzen bringt.

4.1 CPU

TouchDesigner kann nicht mehr als einen Kern des PC Chips verwenden. Moderne Computer haben jedoch fast immer mehrere Kerne. Die Entwicklerumgebung erlaubt es, mehrere Instanzen gleichzeitig laufen zu lassen. Dadurch können auch mehr Kerne des CPU's verwendet werden. Diese Software läuft auf drei Instanzen. Eine Instanz ist für die Audioanalyse zuständig, eine andere für die Bewegungserkennung und eine Dritte Instanz berechnet die Bilder.⁵⁸



Abb.46

Die drei Instanzen in der Netzwerkansicht. Zu sehen ist nur das Basis-Level.

Ein Programm läuft immer mit einer bestimmten Bildrate. Diese Bildrate bestimmt den Zeitaufwand, die die Software benötigt, um alle Operationen durchzuführen. Dann wird der gesamte Vorgang für das nächste Bild wiederholt. Falls die Operationen mehr Zeit in Anspruch nehmen als vorgegeben, wird die Bildrate gezwungenermaßen gesenkt. Das vollständige Durchlaufen aller Operationen hat also oberste Priorität.

⁵⁸ Hany Galal, "Multi Core Processor Advantages", technoop, <http://technoop.blogspot.ch/2009/07/multi-core-processor-advantages.html> (Zugriff: 23.10.16)

Gerade die Audioanalyse profitiert stark von einer höheren Bildrate. Mit einer höheren Bildrate können genauere Messungen der Zeitabstände gemacht werden, was sich dann auch auf die Visualisierung auswirkt. Die Audioanalyse läuft stabil auf 60 Bildern pro Sekunde, die Bewegungsanalyse und Visualisierung laufen auf 30 Bildern pro Sekunde. Die Kameras der Kinect wurden für diese Bildwiederholungsraten erbaut.⁵⁹ ⁶⁰

4.2 GPU

Die Visualisierung läuft nahezu konstant mit 30 Bildern pro Sekunde. Eine höhere Bildwiederholungsrate wäre idealer, damit die Bewegungen flüssiger auf den Betrachter wirken. Der GPU kommt jetzt schon an seine Grenzen. In den ersten Versionen der Software, wurden die kompletten Systeme einer Visualisierung heruntergefahren, sobald die Visualisierung nicht mehr benötigt wurde. Dies erwies sich als sehr ineffizient. Beim Aufstarten einer Visualisierung musste die gesamten Struktur in den Grafikkarten Cache geladen werden, bevor das nächste Bild berechnet werden konnte. Dies führte zu einem Einfrieren der Grafik, was etwa 0.5 bis 1 Sekunde dauerte. Der Nachteil von TouchDesigner ist, dass man kann nicht direkt kontrollieren kann, was wann in die RenderPipeline geladen werden sollte. Die Lösung ist, alle Render Pipelines immer laufen zu lassen. Die Auflösung der nicht benutzten Visualisierungen wird auf eine minimale Auflösung von 2x2 Pixel heruntergeschraubt. Die Kameras, Lichter und Geometrien bleiben somit immer im Grafikkarten Cache. In den nicht aktiven Visualisierungen werden die Vertex Daten aber nicht mehr an den Geometrie beziehungsweise den Pixel Shader weitergegeben. Somit kann der Leistungsanspruch der nicht sichtbaren Strukturen minimiert werden.

Des weiteren werden die einzelnen Noise Textur Nodes zusammengefasst. Damit sind nur noch vier Noise Textur Nodes notwendig anstelle der 20 ursprünglichen. Diese Zahl kommt zustande, da jede Visualisierung mindestens einen Noise Textur Node verwendet. Damit dies funktioniert, musste in die Software ein System eingebaut werden, welches die Parameter eines Nodes in eine Liste speichert und diese falls nötig wieder in einen Node speist.⁶¹ ⁶²

⁵⁹ Louis, "Time Slicing", derivative, https://www.derivative.ca/wiki088/index.php?title=Time_Slicing (Zugriff: 9.10.16)

⁶⁰ Ben, "Frame Rate", derivative, https://www.derivative.ca/wiki088/index.php?title=Frame_Rate (Zugriff: 13.10.16)

⁶¹ Alfonse, "Information for "GLAPI/glMemoryBarrier", OpenGL, <https://www.opengl.org/wiki/GLAPI/glMemoryBarrier> (Zugriff: 20.10.16)

⁶² Richard Burns, "TouchDesigner Help Group", facebook Gruppe, <https://www.facebook.com/groups/touchdesignerhelp/> (Zugriff: 10.10.16)

4.3 Perform Mode

In der Netzwerkansicht muss die gesamte Darstellung der Nodes auch berechnet werden, was die Leistung stark beeinträchtigt. Wenn das Programm nicht bearbeitet werden muss, kann es im sogenannte Perform Mode ausgeführt werden. Darin werden nur vom Entwickler ausgewählte Elemente gerendert. TouchDesigner vereinfacht dem Entwickler ein grafisches Interface für seine Applikation zu gestalten.⁶³



Abb. 47

Das Interface der Audioanalyse

Das Interface der Audioanalyse zeigt von links rechts folgende Elemente:

- Reset Brain: Damit kann der Lernalgorithmus auf Standardwerte zurückgesetzt werden.
- 61: Die jetzige Bildwiederholungsrate.
- Prev / Next / Mute / Reset Playlist: Dies sind alle Knöpfe zum Steuern der Playlist
- Die Werte ganz rechts geben das Ausgangssignal sowie die Schwellwerte der Audioanalyse an.
- Unten ist der Pfad zum aktuellen Musikstück angegeben.



Abb. 48

Das Interface der Bewegungsverfolgung

Das Interface der Bewegungsverfolgung zeigt von links rechts folgende Elemente:

- 31: Die aktuelle Bildwiederholungsrate.
- 0: Die Anzahl vom Kinect Sensor erkannter Personen.
- -0.003...: Der Ausgabewert der eigenen Bewegungsverfolgung.
- Ganz rechts wird angegeben, welche Daten an die Visualisierung weitergegeben werden.

⁶³ Greg, "Perform Mode", derivative, https://www.derivative.ca/wiki088/index.php?title=Perform_mode (Zugriff: 26.10.16)



Abb.49

Das Interface der Visualisierungen

Das Interface der Visualisierungen zeigt von links nach rechts folgende Elemente:

- fakeTrigger: Damit kann ein Liedwechsel Indikator simuliert werden.
- StoreValues: Überschreiben der Einstellungen, nur notwendig falls Änderungen an bestimmten Operation vorgenommen wurden.
- Open/Close: Damit lässt sich die finale Animation als Fenster öffnen und schliessen.
- 31: Die jetzige Bildwiederholungsrate.
- Ganz rechts zu sehen ist der Grafikkarten Cache, die Anzahl aktiver und aller Operationen.

5. Anwendungsmöglichkeiten

Für die fertige Applikation gibt es durchaus Anwendungsmöglichkeiten. Viele Nachtclubs stellen einen sogenannten Visual Jockey an. Dieser kauft sich aus dem Internet Videomaterial zusammen und mischt dieses passend zur Musik übereinander. Diese Applikation kann ohne menschliche Eingabe den visuellen Inhalt der Musik mit einer grossen Dynamik entsprechend anpassen. Zudem ermöglicht die Echtzeitberechnung eine musikbasierte Deformierung der 3D Objekte. Viele Visual Jockeys verwenden nur Videomaterialien die im vorhinein auf bestimmte Liedgeschwindigkeiten angepasst wurden. Die Applikation ermöglicht also eine Automation eines seltenen Berufszweiges und ist dem menschlichen Visual Jockey in vielen Punkten überlegen. Neben Clubs kann das Programm auch für Visualisierung an Konzerten verwendet werden. Dabei sind vor allem Bands angesprochen, die ihre Musik an Konzerten in verschiedenen Varianten spielen wollen. Die visuelle Umsetzung der Musik passt sich mit diesem Programm laufend an.⁶⁴ ⁶⁵

5.1 Ausstellung "Klangform" im Salzhaus Brugg

In der Ausstellung habe ich zwei Beamer und einen Kinect Sensor installiert. Die Beamer zeigten mit einer Auflösung von 1280 x 720 Pixel jeweils eine andere Ansicht des gleichen Objektes. Die aufaddierte Auflösung betrug also 2560 x 720 Pixel.

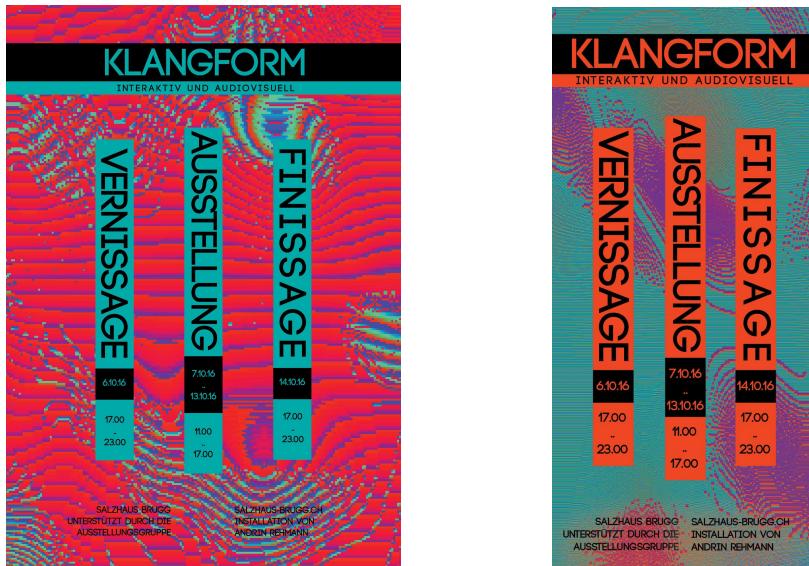


Abb.50

Printmedien der Ausstellung "Klangform"

⁶⁴ Mike Winkelmann, "VJ-Clips", beeple, <http://www.beeple-crap.com/vjclips.php> (Zugriff: 3.11.16)

⁶⁵ Chris Chrismond, "VJ-Union Europe", facebook group, <https://www.facebook.com/groups/vjunioneurope/> (Zugriff: 2.11.16)

Die Software wurde während der Ausstellung erfolgreich mit einer Musikformation gekoppelt. Mehrere Zuschauer konnten gleichzeitig mit den Bildern interagieren.

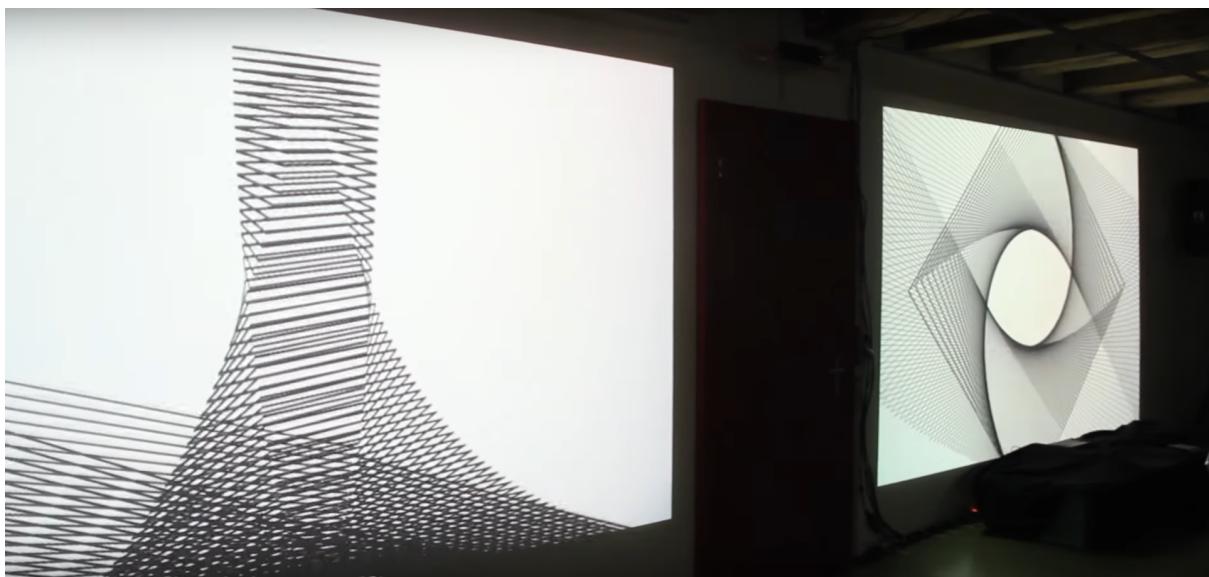


Abb. 51

Foto der Ausstellung "Klangform"

5.2 Flexibilität

Das Programm ist sehr flexibel anwendbar. Die finalen Bilder können in beliebigen Größen dimensioniert werden. Anstelle von zwei einzelnen Bildern können auch ganze Wände eines Raumes zu einer Leinwand werden. Die einzige Einschränkung ist die Hardware. Mithilfe einer leistungsfähigeren Grafikkarte sind der Applikation kaum mehr Limiten gesetzt. Es ist sogar möglich, die Visualisierungen mithilfe einer Virtual Reality Brille zu erkunden. Zudem können die visuellen Darstellungen beliebig ausgebaut werden. Die Bewegungs- und Audioanalyse der Software liefern bereits zuverlässige Daten, welche einen guten Grundstein für weitere Visualisierungen bieten.

Fazit

Dass meine Arbeit als gelungen bezeichnet werden kann, zeigt sich im grossen Echo sowohl vom Publikum meiner Ausstellung, wie auch im Internet. Im Verlauf des Arbeitsprozesses galt es einige Engpässe zu überwinden und für diverse Probleme, Lösungen zu finden. Trotz allem Aufwand sind im Endprodukt immer noch verbesserungswürdige Aspekte zu finden.

Grosses Verbesserungspotenzial ist vor allem in der Bewegungsanalyse und deren Koppelung an die Visualisierung vorhanden. Es ist fragwürdig ob eine Bewegungsanalyse auf ein grosses Publikum anwendbar ist. Damit das Programm erfolgreich in Konzertlokalen oder Clubs angewendet werden kann, müsste vorerst auf die Bewegungsanalyse verzichtet werden.

Die Audioanalyse wurde zum Schwerpunkt der Arbeit. Vor Beginn der Arbeit war geplant, eine simple Audioanalyse zu verwenden. Angesichts der Wichtigkeit einer soliden Basis für die Visualisierung macht es durchaus Sinn, die Audioanalyse soweit auszubauen. Auch die online Community hatte grosses Interesse an der erweiterten Audioanalyse, speziell an der Methode zur Erkennung neuer Liedteile.

Schlusswort

Die gesammelten Erfahrungen im technischen Bereich sind für mich die grösste Errungenschaft dieses Programms. Das Programm selber hat zwar Potenzial, ob ich aber wirklich noch daran weiter arbeiten werde, ist zum jetzigen Zeitpunkt noch unklar. Dank den ausführlichen Rückmeldungen, die ich während der Ausstellung erhalten habe, weiss ich besser, was die Menschen wirklich anzieht. Sie wollen mehr Tiefe, Details und Formen, die an die natürlich wirken.

Die Arbeit war manchmal anstrengend und erforderte viel Durchhaltevermögen, doch meistens war es äusserst interessant und abwechslungsreich. Die Aufgabe war umfangreicher, als ich es am Anfang eingeschätzt hatte. Darum musste ich die Qualität des Programms in der Finalisierungsphase anpassen. Ich konnte jedoch mit der komplexer werdenden Aufgabestellung viel mehr neue Teilaufgaben lösen, als ich geplant hatte.

Die Arbeit ist eine Datenvisualisierungsarbeit, ich verarbeite Abstrakte Daten und verwandle sie in visuell ansprechende Bilder. Zahlen und mathematische Befehle alleine bedeuten wenig für einen Betrachter. Wenn diese abstrakte Datenwelt mit Form, Farbe und Bewegung kommuniziert werden, sehen wir Bedeutung darin. Wir verstehen die Bilder ohne grössere Anstrengung. Gleichzeitig nimmt die Intelligenz von Maschinen stark zu. Die Kommunikation zwischen Maschine und Mensch wird immer wichtiger. Digitale Daten und Vorgänge müssen in verständlicher und ansprechender Form kommuniziert werden. Nur so kann die zunehmende Kluft zwischen der Denkweise von Mensch und Maschine minimiert werden.

Interessant ist, wie ich mich im Verlaufe der Arbeit immer mehr mit dem Computer als Gegenüber identifiziert habe. Ich habe realisiert, wie immens komplex und gleichzeitig schwer fassbar ein Computer heute schon ist. Durch diesen unberechenbarkeit entstehen viele Ideen durch das Zusammenspiel zwischen Mensch und Maschine. Natürlich sind wir die Schöpfer dieser digitalen Welt, jedoch glaube ich an eine Zukunft, in der auch Maschinen mit dem Adjektiv "kreativ" bezeichnet werden können.

Anhang

Glossar

Compiler	Der Compiler wandelt Programmiersprache in maschinelle Sprache um. Diese kann dann direkt von der Hardware verwendet werden. Gleichzeitig kann ein Compiler den Code auf Fehler überprüfen. ⁶⁶
CPU	CPU steht für Central Processor Unit. Der Prozessor ist die Zentrale Komponente eines Computers. Sie führt Operationen aus und überwacht diese. ⁶⁷
GPU	GPU steht für Graphics Processing Unit. Die Grafikkarte ist zuständig für die Berechnung von zwei und dreidimensionalen Grafikberechnungen. ⁶⁸
TOF	TOF steht für Time of Flight. Ein TOF Sensor liefert die Distanzen der sichtbaren Objekte zur Kamera. ⁶⁹
Normale	Die Normale ist eine Gerade, die in einem bestimmten Punkt senkrecht auf eine Funktion oder geometrische Figur steht. ⁷⁰
Node	Ein node ist ein Objekt dass in ein Netzwerk geschlossen wird und die Fähigkeit hat eingehende Daten zu verrechnen und weiterzugeben. ⁷¹
Z-Buffer	Z-Buffering ist ein Prozess der notwendig ist zur Erkennung ob ein virtuelles Objekt in einer 3D Szene sichtbar ist oder nicht. ⁷²

⁶⁶ Margaret Rouse, "What is a compiler?", techtarget, <http://whatis.techtarget.com/definition/compiler> (Zugriff: 13.8.16)

⁶⁷ Klaus Lipinski, "CPU", itwissen, <http://www.itwissen.info/definition/lexikon/central-processing-unit-CPU-Zentraleinheit.html> (Zugriff: 3.11.16)

⁶⁸ Klaus Lipinski, "GPU", itwissen, <http://www.itwissen.info/definition/lexikon/GPU-graphics-processing-unit-Grafikprozessor.html> (Zugriff: 3.11.16)

⁶⁹ Unbekannt, "Overview for 3D Depth Sensing", Texas Instruments, <http://www.ti.com/lit/sensors/3d-imaging-sensors-overview.page> (Zugriff: 23.8.16)

⁷⁰ Unbekannt, "Normale", serlo, <https://de.serlo.org/entity/view/1789> (Zugriff: 3.11.16)

⁷¹ Margaret Rouse, " network node", <http://searchnetworking.techtarget.com/definition/node> (Zugriff: 1.11.16)

⁷² Unbekannt, "What is z-buffering?", <http://www.computerhope.com/jargon/z/zbuffering.htm> Zugriff: 1.11.16)

Zeitplan

	April	Mai	Juni	Juli	August	September	Oktober	November	Dezember
Planung									
1.									
2.									
3.									
Informatik									
1.									
2.									
3.									
4.									
5.									
Planung Installation									
1.									
2.									
Aufbau Installation									
1.									
2.									
3.									
Dokumentation									
1.									
2.									
3.									

• Planung

1. Erstellen Zeitplan, ausformulieren Vereinbarung
2. Planung, Ideensammlung Visualisierung
3. Technische Umsetzung Visualisierung

• **Informatik**

1. Funktionsfähige Bewegungsanalyse
2. Funktionsfähige Audioanalyse
3. Erstellung einer einfache Darstellung zur besseren Überprüfung der Daten
4. Kombination der Daten von der Bewegungsanalyse und der Audioanalyse
5. Erarbeitung erster komplexeren Visualisierungsmethoden

• **Planung Installation**

1. Organisieren von Räumlichkeit
2. Organisieren von technischen Mitteln

• **Aufbau Installation**

1. Installieren
2. Testen
3. Adaptieren

• **Dokumentation**

1. Log Führen
2. Hauptteil schreiben
3. Gegenlesen

Quellenverzeichnis

Webdokumente:

Adam Mann, "Scientists Hack Kinect to Study Glaciers and Asteroids", Wired, <https://www.wired.com/2011/12/hacked-kinect-science/> (Zugriff: 15.10.16)

Alfonse, "Information for "GLAPI/glMemoryBarrier", OpenGL, <https://www.opengl.org/wiki/GLAPI/glMemoryBarrier> (Zugriff: 20.10.16)

Alfonse, "Rendering Pipeline Overview", OpenGL, https://www.opengl.org/wiki/Rendering_Pipeline_Overview (Zugriff: 17.10.16)

Barak Freedman, Alexander Shpunt und Yoel Arieli, "Distance-Varying Illumination and Imaging Techniques for Depth Mapping", Prime Sense LTD, <http://www.google.com/patents/US20100290698> (Zugriff: 12.8.16)

Ben, "Frame Rate", derriative, https://www.derivative.ca/wiki088/index.php?title=Frame_Rate (Zugriff: 13.10.16)

Ben, "Main Page", derriative, http://www.derivative.ca/wiki088/index.php?title>Main_Page (Zugriff: 8.9.16)

Brad Hammond, "Make art now", facebook Gruppe <https://www.facebook.com/groups/makeartnow/> (Zugriff: 21.10.16)

Celestino Soddu, "Generative Art", Generativeart.com, <http://www.generativeart.com> (Zugriff: 29.10.16)

Chris Chrismond, "VJ-Union Europe", facebook group, <https://www.facebook.com/groups/vjunioneurope/> (Zugriff: 2.11.16)

Eric Haines, "Interaktive 3D Grafiken", Udacity, <https://de.udacity.com/> (Zugriff: 10.15 - 11.15)

Gabe, "fixed point vs floating point number", stackoverflow, <http://stackoverflow.com/questions/7524838/fixed-point-vs-floating-point-number> (Zugriff: 7.10.16)

Greg, "Perform Mode", derriative, https://www.derivative.ca/wiki088/index.php?title=Perform_mode (Zugriff: 26.10.16)

Hany Galal, "Multi Core Processor Advantages", technoop, <http://technoop.blogspot.ch/2009/07/multi-core-processor-advantages.html> (Zugriff: 23.10.16)

Ina de Brabandt, "Ableitung der Exponentialfunktion", Mathematik-Oberstufe, <http://www.mathematik-oberstufe.de/analysis/e/e-funktion-ableiten.html> (Zugriff: 8.7.16)

JeGX, "OpenGL Interpolation Qualifiers (GLSL)", geeks 3D, <http://www.geeks3d.com/20130514/opengl-interpolation-qualifiers-glsl-tutorial/> (Zugriff: 12.10.16)

John McCormack und Mark Guglielmetti, "Creative Coding", Future Learn, <https://www.futurelearn.com/courses/creative-coding> (Zugriff: 7.15 - 8.15)

John McCormick, "How does the Kinect Work", <http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf> (Zugriff: 6.8.16)

Joreg, "a multipurpose kit", vvvv, <https://vvvv.org/> (Zugriff: 9.9.16)

Klaus Lipinski, "CPU", itwissen, <http://www.itwissen.info/definition/lexikon/central-processing-unit-CPU-Zentraleinheit.html> (Zugriff: 3.11.16)

Klaus Lipinski, "GPU", itwissen, <http://www.itwissen.info/definition/lexikon/GPU-graphics-processing-unit-Grafikprozessor.html> (Zugriff: 3.11.16)

Louis, "Camera COMP", derriative, https://www.derivative.ca/wiki088/index.php?title=Camera_COMP (Zugriff: 9.10.16)

- Louis, "Convert SOP", derriative, https://www.derivative.ca/wiki088/index.php?title=Convert_SOP (Zugriff: 13.10.16)
- Louis, "Light COMP", derriative, https://www.derivative.ca/wiki088/index.php?title=Light_COMP (Zugriff: 28.10.16)
- Louis, "Noise TOP", derriative, https://www.derivative.ca/wiki088/index.php?title=Noise_TOP (Zugriff: 9.10.16)
- Louis, "Time Slicing", derriative, https://www.derivative.ca/wiki088/index.php?title=Time_Slicing (Zugriff: 9.10.16)
- Malcolm, "Write a CPlusPlus DLL", derriative, https://www.derivative.ca/wiki088/index.php?title=Write_a_CPlusPlus_DLL (Zugriff: 14.10.16)
- Malcolm, "Write a GLSL Material", derriative, https://www.derivative.ca/wiki088/index.php?title=Write_a_GLSL_Material (Zugriff: 15.10.16)
- Margaret Rouse, "network node", <http://searchnetworking.techtarget.com/definition/node> (Zugriff: 1.11.16)
- Margaret Rouse, "What is a compiler?", techtarget, <http://whatis.techtarget.com/definition/compiler> (Zugriff: 13.8.16)
- Margaret Rouse, "z-buffering", techtarget, <http://whatis.techtarget.com/definition/z-buffering> (Zugriff: 23.7.16)
- Martin Christen, "Lighting", OpenGL, <https://www.opengl.org/sdk/docs/tutorials/ClockworkCoders/lighting.php> (Zugriff: 13.10.16)
- Máté Kovács, "Why does Graphics hardware only render triangles?", Quora, <https://www.quora.com/Why-does-graphics-hardware-only-render-triangles> (Zugriff: 1.10.16)
- Matt Ball, "jpg bits per pixel", stackoverflow, <http://stackoverflow.com/questions/4305101/jpg-bits-per-pixel> (Zugriff: 7.10.16)
- Matthew Ragan, "TouchDesigner", <https://matthewragan.com/> (Zugriff: 10.10.16)
- Matthew Szymczyk, "How Does The Kinect 2 Compare To The Kinect 1?", Zugara, <http://zugara.com/how-does-the-kinect-2-compare-to-the-kinect-1> (Zugriff: 23.7.16)
- Mike Winkelmann, "VJ-Clips", beepie, <http://www.beepie-crap.com/vjclips.php> (Zugriff: 3.11.16)
- Quo R, "Dichtefunktion", Wikipedia, <https://commons.wikimedia.org/wiki/File:Dichtefunktion.png> (Zugriff: 7.7.16)
- Richard Burns, "A simple Audio Reactive Sphere", Vimeo, <https://vimeo.com/15850663> (Zugriff: 20.10.16)
- Richard Burns, "TouchDesigner Help Group", facebook Gruppe, <https://www.facebook.com/groups/touchdesignerhelp/> (Zugriff: 10.10.16)
- Rob, "Info Dat class", derriative, https://www.derivative.ca/wiki088/index.php?title=Info_DAT (Zugriff: 14.10.16)
- Sean McHugh, "image interpolation", cambridge in colour, <http://www.cambridgeincolour.com/tutorials/image-interpolation.htm> (Zugriff: 12.19.16)
- Unbekannt, "About TensorFlow", TensorFlow, <https://www.tensorflow.org/> (Zugriff: 16.10.16)
- Unbekannt, "What is Python", Python Foundation, <https://www.python.org/doc/essays/blurb/> (Zugriff: 10.10.16)
- Unbekannt, "About TouchDesigner" <http://www.derivative.ca/> (Zugriff: 15.10.16)
- Unbekannt, "Kinect", Microsoft, <https://developer.microsoft.com/en-us/windows/kinect> (Zugriff 6.8.16)
- Unbekannt, "Kinect for XBOX ONE", Microsoft, <http://www.xbox.com/de-CH/xbox-one/accessories/kinect-for-xbox-one> (Zugriff: 6.8.16)
- Unbekannt, "Lighting", GL programming, <http://www.goprogramming.com/red/chapter05.html> (Zugriff: 13.10.16)
- Unbekannt, "Normale", serlo, <https://de.serlo.org/entity/view/1789> (Zugriff: 3.11.16)

- Unbekannt, "OpenGL Shading Language", OpenGL, <https://www.opengl.org/documentation/glsl/> (Zugriff: 3.10.16)
- Unbekannt, "Overview for 3D Depth Sensing", Texas Instruments,
<http://www.ti.com/lsm/ti/sensors/3d-imaging-sensors-overview.page> (Zugriff 23.8.16)
- Unbekannt, "Particles / Instancing", OpenGL tutorial,
<http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/> (Zugriff: 19.9.16)
- Unbekannt, "programming language", business dictionary
<http://www.businessdictionary.com/definition/programming-language.html> (Zugriff: 13.10.16)
- Unbekannt, "Samplerate", Baumannmusic,
<http://www.baumannmusic.com/de/2012/sampleratehz-und-khz-aufloesung-bit-und-bitrate-kbits/> (Zugriff 5.10.16)
- Unbekannt, "What is OpenGL", OpenGL, https://www.opengl.org/wiki/FAQ#What_is_OpenGL.3F (Zugriff: 13.10.16)
- Unbekannt, "What is Python", Python Foundation, <https://www.python.org/doc/essays/blurb/> (Zugriff: 10.10.16)
- Unbekannt, "What is z-buffering?", <http://www.computerhope.com/jargon/z/zbuffering.htm> Zugriff: 1.11.16)
- Unbekannt, What's the difference between a cpu and a gpu?", nVIDIA,
<https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/> (Zugriff: 28.9.16)
- Verschieden Autoren, "Learn to code", Codecademy, <https://www.codecademy.com/learn/learn-java> (Zugriff: 7.15 - 9.15)
- Verschiedene Autoren, "Audio Frequency", Wikipedia, https://en.wikipedia.org/wiki/Audio_frequency (Zugriff: 20.9.16)

Bücher:

- Christoph Rauscher, "Grundlagen der Spektrumanalyse", Rohde & Schwarz GmbH & Co. KG, 2000, München
- Ken Perlin, "Noise Hardware", UMBC, 2002, Unbekannter Ort
- Matthew Kirk, "Thoughtful Machine Learning: A Test-Driven Approach", O'Reilly, 2015, U.S.A
- Patrice Delmas, "Gaussian Filtering", University of Auckland, 2012, Neuseeland
- Shai Shalev-Shwartz und Shai Ben-David, "Understanding Machine Learning: From Theory to Algorithms"
Cambridge University Press, 2014, England

Bildverzeichnis

Abbildungen 1 - 20 und 22 - 53 sowie das Titelbild, wurden von Andrin Rehmann im Verlauf seiner Maturaarbeit erstellt.

Abbildung 21:

https://en.wikipedia.org/wiki/Phong_shading#/media/File:Phong-shading-sample.jpg

zeitpunkt