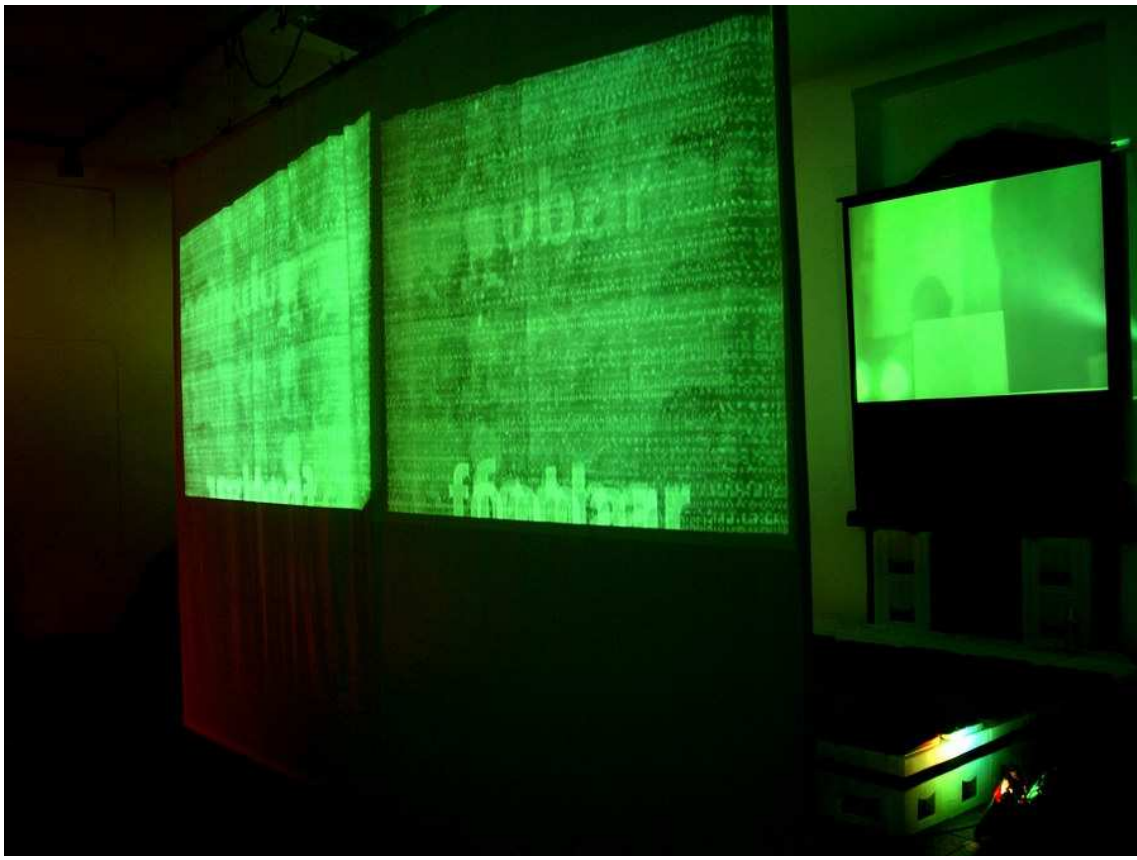
VJ - towards an electronic art
Forschungsprojekt an der Bauhaus Universität Weimar



Revision of 10. August 2003

Sebastian Heymann [1]

SS 2003

[1]heymann@uni-weimar.de

In dieser Arbeit soll ein Bild vermittelt werden. Ein Bild, das die vielen Aspekte des Projekts, den steinigen Weg bis zum Ende beschreibt.

Inhaltsverzeichnis

Einleitung Am Beginn stand die Idee, ein VJ-System zu schreiben. Weitere Feinheiten existierten zunächst nicht. Es soll in dieser Arbeit der Entscheidungsprozess, der zu dem führte, was am Ende steht. Es werden Implementationen näher beleuchtet, um dem Leser einen einigermassen tiefen Eindruck in die von mir geleistete Arbeit zu vermitteln. Desweiteren soll eine Idee beschrieben werden, die zu beschreiben sich als sehr schwierig erwiesen hat, die am Ende aber unterhaltsam und *cool* war.

TechTalk Zu Beginn des Projekts waren die Vorstellungen sehr unterschiedlich und gingen an einigen Stellen stark auseinander. Grundgedanken, die sich relativ schnell in der Diskussionsphase herausbildeten waren die folgenden. Es bildete sich eine Dreiteilung heraus:

- Selektor
- Engine
- Performance

Der Selektor bildetet das Konfigurationswerkzeug des Projekts. An dieser Stelle soll jedoch nicht näher darauf eingegangen werden, da dieser Teil nicht, oder nur zu kleinen Teilen in meinen Zuständigkeiten zählte. Es soll an dieser Stelle auf die Gemeinschaftsdokumentation verwiesen sein.

Die Engine und der Performanceteil fallen mehr in meinen Bereich und werden daher hier etwas tiefer beleuchtet.

Zunächst sollen jedoch kleine technische Details und Hürden der Anfangsphase erläutert werden.

Gstreamer Am Anfang der technischen Diskussionen stand die Frage nach Plattform und Framework auf denen das System aufzusetzen ist. In zahlreichen Gesprächen und als Ergebnis von verschiedenen Aufgaben, haben wir uns auf der *Gstreamer Linux Media Framework*[1] geeinigt. Dies hatte unter anderem diese Gründe:

- OpenSource - Das hatte für die Entwicklung den großen Vorteil, daß wir im Quellcode des Systems nachsehen konnten um Fehler zu finden und Ideen zu entwickeln. Ausserdem ist dies sehr lehrreich um eine Vorstellung davon zu bekommen, wie ein solches Framework aufgebaut ist. Diese Punkte kann eine proprietäre Lösung nicht bieten.
- “Schöner Quellcode” - Nach Sichtung verschiedener Codebeispiele von Gstreamer und DirectShow gaben wir Gstreamer den zweiten Punkt.
- CrossPlattform - Gstreamer ist auf jeder Linux Plattform, sowie qnx, Solaris, hp-ux, MacOS X etc. getestet und gröstenteils portiert worden.
- Unterstützung von Seiten der Entwickler via irc oder Mailinglisten
- Linux als freie Entwicklungsplattform

[1] www.gstreamer.net

Dieses Framework hat so seine Probleme, aber ist dabei zu einem Linux-Standard heranzuwachsen.

Nach dieser Entscheidung hatte die Einarbeitung in das Gstreamer-System höchste Priorität. Ich habe mich daraufhin intensiv mit Gstreamer beschäftigt und versucht die in den Workshops geforderten Aufgaben mit dem Gstreamer-Framework zu realisieren um die Einarbeitungsphase schnell zu bewältigen. Dabei ging es nicht nur um Programmiertechnische Details, sondern auch um das Begreifen einiger Grundkonzepte Node-Basierter Videobearbeitung, Farbräume, etc.

Gstreamer-Filter

Um eine Arbeitsgrundlage für die Pluginentwicklung zu schaffen, wurde zunächst ein Plugintemplate entwickelt. Dieses Plugin basiert auf einem Plugin-Template des Gstreamer-Projekts. Dieses Template war für Audiodaten ausgelegt und musste daher umgebaut werden.

Für die wöchentlichen Projekttreffen sind folgene Filter entstanden:

- Five-Minuter - bekommt zwei Videoströme, belegt diese zufällig mit verschiedenen Effekten, mix sie übereinander und gibt einen Strom wieder heraus. Dieser Filter entstand als Lösung für die Aufgabe “Pixeltapete”. Erster Filter mit zwei Eingabepads für Videoströme.
- Colorchange - ermittelt die Bewegungen in einem Video durch Differenzenbildung und berechnet die Intensität der Bewegung als Integerwert[2].
- Colormix - Entstanden als Vorlage für parametrisierbare Filter. Dieses Plugin wurde weiterhin als Basis genutzt und steht so Pate für viel folgende Plugins. Es kann dazu benutzt werden die einzelnen RGB-Farbkanäle zu gewichten und somit den Farbeindruck eines Bildes zu verändern.

Gstreamer-Applikationen

- Balloon-App - Vorstufe zum ovjDemon/Pakt. Werte konnten von ausserhalb via Netzwerk gesetzt werden.
- Five-Minuter-App - Erste Gstreamer Applikation mit fest einkompilierterer Videopipeline.
- ovjDemon / openEyed / gshift
- Warsaw Pakt

inter-pipeline Kommunikation

Es hat sich für einige Konzepte als sehr sinnvoll erwiesen, die Möglichkeit zu besitzen, aus einer Videopipeline “Messdaten” zu erzeugen und damit andere Filter/Parameter zu steuern. Um dieses zu ermöglichen benötigt man eine Sammlung von Gstreamer-Plugins, die eine bestimmte Schnittstelle implementieren. Beispiele für Datenerzeuger:

[2] Die Colorchangeversion für die Balloon-App schickte seine Werte direkt über Netzwerk an eine entfernte Balloon-App

- Fast Fourier Transformation von Audiodaten.
- Farbanalyse von Videodaten
- Bewegungsanalyse von Videodaten

Vertiefendes Beispiel: Gstreamer besitzt ein Plugin zur spektralen Analyse von Audiodaten über eine Fast Fourier Transformation. Die Ausgabe dieses Plugins besteht nur aus den Intensitäten der einzelnen Frequenzanteile. Die Anzahl der gewünschten Frequenztrennungen ist dabei frei wählbar. Es wird verständlig, dass für eine möglichst generische Verwendung beliebiger Werte ein einheitliches Interface geschaffen werden muss. Im Laufe des Projekts sind daraus mehrere Plugins entstanden, die jeweils einen Teil der benötigten Funktionalität implementieren.

- *colorchange* - Registriert Farbänderungen/Bewegungen in einem Video und gibt diese in Intensitätswerten zurück.
- *motionmatrix* - Registriert Bewegungen in einem bestimmten Bildbereich. Im allgemeinen Fall ist das Bild in neun Vierecke geteilt, welche einzeln auf Veränderungen untersucht werden. Zurückgegben werden
  - die höchste Intensität aller Bereiche
  - die Nummer des Bereichs mit der höchsten Bewegungsintensität
- *track2d* - ermittelt den hellsten Punkt ein einem Videobild und errechnet die Postition im Bild[3].
- *spectrum2osc* - Transformiert die vorher erwähnten Spektralwerte eines Audiostroms in ein osc[4] -ähnliches Protokoll, welches zur internen Übertragung genutzt wird.
- *ovjSink* - Schickt die empfangenen Daten an einen Filter in einem entfernten ovjDemon
- *paktSink*[5] - Schickt die empfangenen Daten an einen Paktserver
- *oscSink*[6] - Schickt die empfangenen Daten an ein Programm mit osc-Interface

oscdispatcher

Um die in der Videopipeline erzeugten Daten nutzbar zu machen und zu verschicken, wurde das osc-Protokoll[7] benutzt. Dieses Protokoll besteht aus eine sehr kleinen, festen Spezifikation[8] zum verschicken verschiedener Datentypen zwischen medienverarbeitenden Programmen. Beispiele für osc-Implementationen in Programmen sind:

- MaxMsp
- Reaktor

[3] Diese Plugin wurde beim Projekt Handgriff genutzt und für diesen Fall implementiert.
[4] Open Sound Control
[5] noch nicht fertig implementiert
[6] noch nicht fertig implementiert
[7] Open Sound Control
[8] http:cnmat.cnmat.berkeley.eduOSCOSC-spec.html

- Director
- Flash

Da das osc-Protokoll ein existierender “Standard” für Cross-Media Kommunikation ist, bot es sich an, diesen Standard auch als internes Protokoll zu verrwenden, da das Erfinden eines neuen Protokolls an dieser Stelle nicht sinnvoll wäre und desweiteren die Integration in bestehende Software vereinfacht. Die oben erwähnten Plugins implementieren dieses Protokoll. Dabei kann ein internes osc-Packet beliebig viele Daten der folgenden Datentypen enthalten.

- Float32
- Int32
- Boolean
- String

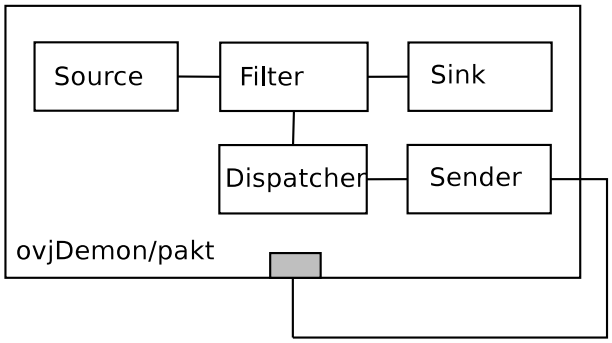


Abbildung 1: Dispatcher

Am Anfang des Pakets steht dabei die Zieladresse (Filter/Parameter) gefolgt von einer Auflistung der verwendeten Datentypen. Den größten und abschliessenden Block des Packets bilden die Daten selbst.

Da diese Packete wie erwähnt beliebig viel Daten unterschiedlicher Datentypen enthalten können, bestand die Notwendigkeit zu einem Dispatcher. Realisiert wurde dies im *oscdispatcher*.

Beispiel:

```
... ! oscquelle ! oscdispatcher out1=1 range1=i,0,300
drange1=f,0.0,1.0 out2=1 range2=i,0,300 drange2=f,0.0,40.
! ovjsink host=localhost port=8888 node=sinesrc0
param=volume oscdispatcher0.src2!sink ovjsink host=192.16
node=motionblur0 param=strength port=1234
```

Das Beispiel zeigt, wie Daten eine osc-Quelle aufgeteilt und an verschiedene Zielrechner geschickt werden.

- Daten werden für den ersten Sender aus einer Range von *int* 0 bis 300 werden auf eine Range von *float* 0.0 bis 1.0 umgerechnet

- Zieldaten werden an *localhost port 8888* Filter *sinesrc0* Parameter *volume* geschickt.
- Daten werden für den zweiten Sender aus einer Range von *int* 0 bis 300 werden auf eine Range von *float* 0.0 bis 40.0 umgerechnet
- Zieldaten werden an *192.168.0.51 port 1234* Filter *motionblur0* Parameter *strength* geschickt.

Dieses Beispiel zeigt, wie mit einem aus einem Video generierten Datensatz zwei verschiedene Filter gesteuert werden können. Der eine Filter wird lokal und der andere ferngesteuert. Verändert werden kann jedes beliebige GStreamer Plugin.

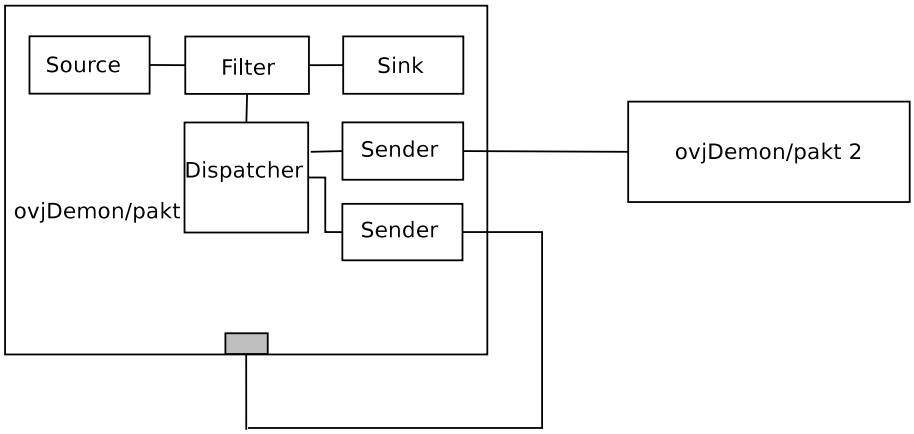


Abbildung 2: osc-Dispatcher

Um verschiedene Systeme steuern zu können, bestehen verschiedene Implementationen von Sendeplugins.

- ovjSink - Implementation abgeschlossen und getestet. Der ovjSink Sender übersetzt das interne Protokoll in das ovjDemon-Protokoll und ist in Adresse, Port, Filter und Parameter frei konfigurierbar.
- paktSink - Implementation noch nicht abgeschlossen. Integriert die beschriebenen Funktionalitäten in das Pakt-System.
- oscSink - Implementation noch nicht abgeschlossen. Schickt die Werte an osc Clients. Die Motivation für das Projekt war zunächst nicht so gross, da an anderen Stellen des Projekts noch keine osc-Funktionalität bestand.

Entstandene Programme/Plugins

Insgesamt entstandene Applikationen

ovjDemon Der ovjDemon ist an anderer Stelle dieser Dokumentation schon ausführlicher erklärt worden. An dieser Stelle soll daher nicht näher darauf eingegangen werden.

pakt siehe ovjDemon.

openeyed siehe ovjDemon.

ovjGui/ovjRemote In der Entwicklungsphase zum ovjDemon kam schnell das Verlangen nach einer Testumgebung auf. Diese wurde einfach in Python realisiert und deckte einen großteil der ovjDemon Funktionalitäten ab.

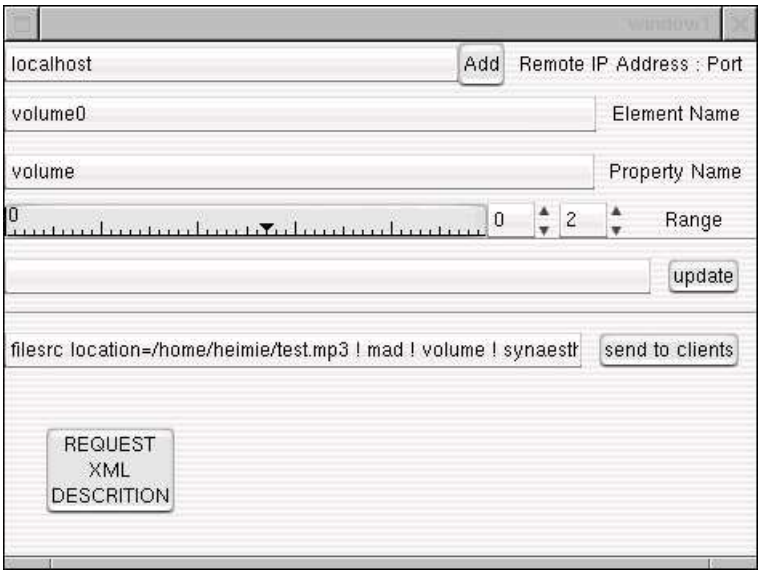


Abbildung 3: Screenshot der ovjGui

Wie auf dem Bild zu sehen ist, konnten numerische und String-Parameter eingestellt und zur Laufzeit verändert werden. Dieses kleine Programm diente somit als praktische Test- und Debughilfe.

Die Gui ist weitesgehend selbsterklärend und es soll daher nicht näher darauf eingegangen werden. Einfach mit *python -i ovjRemote* starten. Notwendige Bibliotheken:

- Glade
- PyGlade

FiveMinuterApp Am Anfang existierten kleine Probleme mit der Kommandozeilensyntax für *gst-launch*. Diese waren die erste Motivation ein *Gstreamer-Programm* zu schreiben, in dem die Syntax fest einkompiliert ist. Das war bei mir der Five-Minuter. Es ging darum zwei Videoströme in ein Filter-Plugin zu leiten und einen ausgehenden Videostrom zu erhalten.

Insgesamt entstandene Plugins

Video Plugin Template Dieses Plugin bildet die Grundlage aller entstandenen Videoplugins. Es ist eine Adaption des im *Gstreamer-CVS* liegenden Plugin-Templates zur Verarbeitung von Videodaten. Es wird nichts am Bild verändert. Auskommentiert ist jedoch eine Farbreduktion auf drei Stufen Rot, um dem interessierten Leser ein Beispiel für pro-Pixel-Operationen zu geben.

Abbildung 4: Colormix

colormix Der Colormix ist die erste Implementation im Projekt, die variable Parameter in Plugins zur Verfügung stellt. Es kann eine Gewichtung der RGB-Kanäle über drei Parameter (optR, optG, optB) vorgenommen werden.

Abbildung 5: Colorchange

colorchange Der Colorchange bildet die Differenz aus dem momentanen und dem vergangenen Bild eines Videostroms. Das resultierende Bild ist schwarz an den Stellen, an denen sich nichts verändert hat und wird immer heller, je mehr sich änderte. Der Grad der Veränderungen kann in Verbindung mit dem osc-Dispatcher genutzt werden.

fiveMinuter Kunstplugin mit mehreren Überblendmodi für zwei Videos, die zufällig ausgelöst werden.

track2D Näheres zu diesem Plugin bitte unter *RundgangLaser-Rahmen*

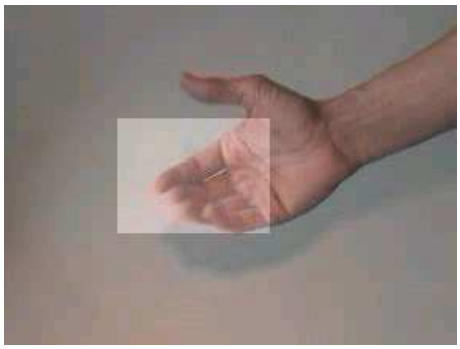


Abbildung 6: Motionmatrix

motionmatrix Die Motionmatrix bildet einen technischen Grundstein des *CMC-Prototypen*[9]. Es wird über Differenzenbildung[10] eine Bewegungserkennung vorgenommen. Diese geht jedoch nicht über das gesamte Bild, sondern über Teile. Die Motionmatrix teilt das Bild in 3x3 Teile und führt auf den einzelnen Quadern eine Bewegnugsanalyse durch[11].

motionblur Der Motionblurfilter addiert das momentane Bild mit einer gewissen Gewichtung zum vergangenen Bild dazu. Diese Addition zieht sich über mehrere Bilder hin, so entsteht der Eindruck einer verwischten Bewegung bei schnellen Bewegungen. Dieser Effekt ist im i420-Farbmodell realisiert und benutzt nur die Helligkeitskanäle der Bilder, da diese für die Erzielung des gewünschten Effekts ausreichend sind[12].

Abbildung 7: Motionblur

oscdispatcher Näheres zu diesem Plugin unter *Inter-Pipeline Kommunikation*

[9] siehe *RundgangCMC*
[10] Vgl. Colorchange
[11] Dazu: Inter-Pipeline-Kommunikation
[12] ...was grosse Performancegewinne verschafft

spectrum2osc Das *Spectrum2osc* Plugin dient lediglich als Konverter zwischen dem *spectrum*[13] Plugin und dem *oscDispatcher*. [14]

Abbildung 8: i420switch

i420switch Der *i420switch* ist eine Implementation eines vielseitigen Video-Crossfaders für zwei Videos. Die zu Projektende aktuelle Version beherrschte folgende MixModi:

- 0. Video0
- 1. Video1
- 2. Video0 und Video1 werden übereinandergeblendet
- 3. Horizontaler Wipe
- 4. Zufällige Zerteilung des Bildes in Streifen nach eingestellter Gewichtung.

```
gst-launch dffplay location=05802.avi ! i420switch
mode=2 strength=0.5 ! colorspace ! xvideosink
dffplay location="oversand.avi" src\!i420switch0.sink_2
```

Anmerkung: Bitte in einer Zeile auf der Konsole eingeben.[15]

loopbuffer Der Loopbuffer ist ein Ringpuffer für Video-Frames. Da im hochkomprimierten Videodaten ein Vor- und Zurückspulen sehr viel Rechenzeit erfordert, fanden wir darin die Lösung für Probleme wie:

- Loops
- Rückspulen
- Vorspulen
- Standbilder

Einstellbar sind dabei folgende Parameter:

[13] Gstreamer-Plugin
[14] Näheres zur Anwendung dieses Plugins unter *Inter-Pipeline Kommunikation*
[15] dffplay ist auf *cvs.subsignal.org/dplugins* zu finden

- Spielposition in Frames
- Größe des Puffers
- Spielgeschwindigkeit in fps[16]

Beispiel-Commandline zum Ausprobieren:

```
gst-launch filesrc location="05801.avi" ! avidemux
video_%02d! { queue ! ffmpegdecall ! colorspace !
loopbuffer speed=80 size=200 ! colorspace !
xvideosink }
```

Anmerkung: Aus Benutzbarkeitsgründen wurde eine Geschwindigkeitspufferung eingebaut. Die angegebene Commandline daher in wechselnden Standbildern resultieren. Die Benutzung ist daher nur sinnvoll, wenn zur Laufzeit wechselnde Werte für die Geschwindigkeit gesetzt werden.

dropnth Um für Übertragungen von Video über das Internet die Bandbreite der Übertragung zusätzlich zum Kompressionsaufwand zu minimieren, wurde diese Plugin entwickelt. Es gibt nur jedes *n*-te Bild durch und wirft den Rest weg.

sprobe Einfaches Strobe Plugin. Es wird ein Frame *n*-mal wiederholt. Dann wird zum nächsten aktuellen umgeschaltet. Ist der Parameter *strength* auf 1 gesetzt, dann läuft das Video normal ab. Bei ca. 25 sieht der Betrachter nur jede Sekunde ein neues Bild. Der Videostrom hat trotzdem eine Wiederholrate von 25Hz.

[16] fps - Frames per Second

PAKT - ToolKit for Art Performances

Geschichtliches

Von Anfang an lag ein Hauptaugenmerk bei der Entwicklung des Systems auf der Verteil- und Fernsteuerbarkeit des Kernsystems. Im folgender wird auf diese Punkte tiefer eingegangen werden. Die Grundidee für das Verteilungskonzept lag zunächst darin, eine Videopipeline aus der Ferne (Lokales Netzwerk oder Internet) oder lokal starten zu können. Stellt man sich ein Performancesetup vor, in dem ein oder mehrere Rechner in eine Saal verteilt stehen und an Projektoren angeschlossen sind, wird der Nutzen schnell klar. Der Nutzer ist auf diese Weise in der Lage, an einem von ihm festgelegten Platz zu arbeiten und muss nicht seinen Standort auf die Technik abstimmen. In weiterführenden Diskussionen ergab sich in Konsequenz, dass auch die Fernsteuerung der einzelnen Videoparameter und anderer Pipelineelemente von grosser Bedeutung für das Projekt sein werden. Es sollten mehrere Szenarien möglich sein:

- Ein Nutzer steuert seinen eigenen Rechner
- Ein Nutzer steuert einen/mehrere ferne Rechner
- Mehrere Nutzer steuern einen/mehrere ferne Rechner

ovjDemon

Aus diesen Anfangsüberlegungen entstand der *ovjDemon*[17] ("OpenVJ Demo Daemon"), in eben genannten Ziele prototypenhaft implementiert worden sind. Es ist dem Nutzer hier möglich eine Gstreamerpipeline lokal oder über einen Netzwerkport zu initialisieren, zu starten und zu verändern. Dies geschieht folgendermassen:

- *./ovjDemon -l pipeline* für eine lokale Pipelineinitialisierung mit Standardport.
- *./ovjDemon -r 8888* für eine Ferninitialisierung auf dem Port 8888.
- *./ovjDemon -l pipeline 8888* für eine lokale Pipelineinitialisierung auf Port 8888.

Ist eine Videopipeline erst angestossen, läuft sie unverändert. Der *ovjDemon* übernimmt hier also nur den Start. Um nun eine Änderung an bestehenden Parametern der Pipeline zu ändern, hat der *ovjDemon* einen offenen UDP-Port, welcher beim Start angegeben werden kann. Dieser Port versteht sehr einfache Nachrichten dieser Struktur :

- *sinesrc0 volume 0.43*

Diese Nachricht würde eine Änderung des Parameters *volume* des Filterelements *sinesrc0* auf den Wert *0.43* bewirken. Der *ovjDemon* ist in der Lage folgende Werte zu interpretieren:

[17] Der *ovjDemon* befindet sich trotz ausreichender Funktionalität noch im Entwicklungsstadium, was sich auch nicht mehr ändern wird. Daher existieren sehr viel Debug-Ausgaben, die den aktuell durchgeführten Schritt des *ovjDemons* beschreiben.

- float
- int
- boolean
- string

Die Werteänderungen werden über die *g_object_set_property()* Funktionaltität des *glib2* Modells hinter *gstreamer* gesetzt. Auf diese Weise werden direkt die jeweiligen Werte in den Objekten der Filter und Plugins geändert, was sehr schnell ist. Die für die Videobearbeitung notwendigen 25-30 Bilder pro Sekunde waren auch als Frequenz für Parameteränderungen keine Schwierigkeit.

gshift/openeyed Gshift und OpenEyeD sind zwei zusätzliche Testsysteme, auf die nicht weiter eingegangen werden muss. Beide dienten zur Evaluierung einiger Ideen und wurden verworfen, da sie sich als nicht gut erwiesen. Zum Sammeln von Konzepten taten sie jedoch ihren Dienst mehr als gut und werden deshalb an dieser Stelle erwähnt.

Warsaw Pakt Wie in der Einleitung kurz umrissen wurde, ist der Pakt-Server als Gemeinschaftprojekt zwischen subsignal[18]-Mitglied Daniel Fischer[19] und den Projektteilnehmern in heftigen Diskussionen entstanden. Der Pakt Server ist eine Vereinigung der vorher entstandenen Konzepte und stellt Ende der evolutionären "Nahrungskette" von Media-Control-Servern da. Der Name Pakt bedeutet rückwärts *ToolKit for Art Performances*. Die Namensidee resultiert aus dem auf Windows bekannten *NATO* System und soll in späteren Versionen ein Pendant dazu darstellen.

Im Kern ist der Pakt sehr modular entworfen worden. Es existieren einige Kommunikationsschnittstellen:

- Flash-XML
- HTTP
- XML via TCP

Einige weitere sind in Arbeit:

- osc[20]
- XML via UDP

Kommt eine Steuerungsnachricht beim Pakt an, so wir ein Messagehandler instanziiert. Dieser übersetzt die empfangene Nachricht in ein internes Format und arbeitet diese ab. Um den Paktserver auf um ein weiteres Format zu erweitern, genügt es daher aus, einen neuen Messagehandler für das gewünschte Protokoll zu implementieren.

[18] subsignal.org
[19] huetopic.net
[20] Open Sound Control

Die neuste Version der Pakt Software kann über *cvs.subsignal.org* bezogen werden. Da die Entwicklung am Pakt System auch nach dem Projektende weitergeführt wird, ist es sinnvoll diesem Verweis nachzugehen, da über das CVS auch die für die Installation benötigte Dokumentationen und Protokolldetails bezogen werden können[21]

pakt-Protokoll *Aus der Pakt Dokumentation:* Alle Messagehandler geben entweder *error* oder *warning* messages zurück. Alle Request Messages müssen einen *target* Parameter enthalten.

Anfangs besteht die Handlerhierarchie aus:

- a single, empty, stopped thread at "/",
- a registry handler at registry.

  Beispiele für Zugriffsmethoden:

  * HTTP GET requests directed to port 3900

    ```
    http://<hostname>:3900/<targetnode>?<message>
    &<parameter1=blah>
    ```

    for example:

    ```
    http://localhost:3900/fakesrc0?info&depth=2
    ```

    if you add an empty "xml" parameter, the output will be the raw textxml reply; if not, you will get xslt-transformed html.
  * Flash XML Socket connections connect to port 3901

    ```
    <message target="target" param="blah">
    ```

    for example

    ```
    <info target="fakesrc0" depth="2">
    ```

    the reply will come along on the stream.

  Mögliche Messages:

  * info
  * create
  * connect
  * play, pause, stop, delete
  * get
  * set

  Bei größerem Interesse am Pakt und am Protokoll sei nochmals auf die Pakt-Dokumentation verwiesen, zu finden unter:

  ```
  http://cvs.subsignal.org/pakt/doc/protocol.txt
  ```

[21] Bei Problemen sehen sie bitte auf *subsignal.org/cvs* nach den notwendigen Details

Die Dienstagstreffen

Aufgaben und Demonstrationen

* Five Minuter Siehe Plugins.
* *Luftballon* - Aufgabe war es, eine “ungewöhnliches” Eingabegerät an einen Computer “anzuschliessen” und damit Videoeffekte zu steuern. Meine Gruppe hat den Luftballon bekommen.
  Wie bringt man einen Luftballon dazu Videoeffekte zu steuern? Die Grundidee dabei war es, den Luftballon nicht zu verändern, sondern über eine Kamera den Luftballon aufzunehmen, die Bewegungen zu registrieren[22] und damit ein anderes Plugin fernzusteuern. Um die Fernsteuerung zu gewährleisten[23] wurden dem Colorchange direkt Netzwerkfähigkeiten eingebaut. Er schickte so die Änderungsdaten, die aus den Luftballonbewegungen berechnet wurden an einen entfernten ovjDemon. Wir nutzten die Gelegenheit um die Fernsteuerungsfähigkeiten von Gstreamer und die Realisierbarkeit unserer entstandenen Konzepte zu prüfen.
* *Videowall* - Im Rahmen meines Vortrags über “Verteilte Anwendungen” baute ich ein kleines *VideoWall* Setup mit zwei Laptops auf. Für die Realisierung war keine Programmierung nötig, lediglich die passende Gstreamer-Pipeline musste konstruiert werden.

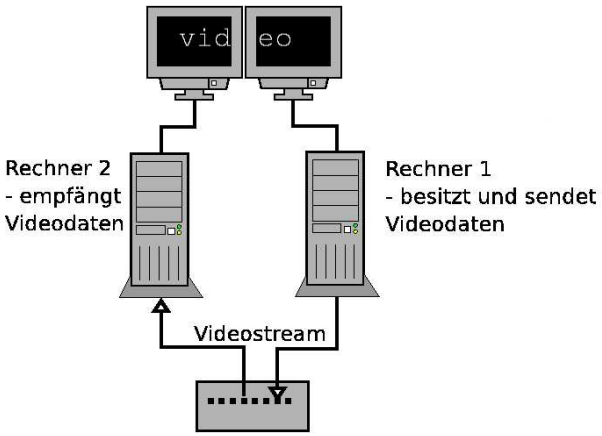


Abbildung 9: Videowall Setup

Ein Laptop spielt dabei das Video aus einer Datei, stellt einen Teil des Videos dar (z.B. linke Hälfte) und spielt das gesamte Video über ein Netzwerk zu einem entfernten Rechner. Dieser wiederum spielt das empfängt das Video aaus dem Netz und stellt einen anderen Teil des Videos (z.B. rechte Hälfte) dar. Auf diese Weise kann schnell und Kostengünstig eine Video-Wand erstellt werden. Bandbreitenproblem sind dabei nicht zu erwarten, da im lokalen

[22] Vgl. Colorchange
[23] Eine Implementation des osc-Dispatchers bestand zu diesem Zeitpunkt noch nicht

Netzwerk ein lokaler Broadcast den gewünschten Effekt erzielt, dass alle Rechner zu möglichst gleicher Zeit das gleiche Video bekommen.

Vortrag Mein Vortrag ging um Verteilte Systeme. Den Mediensystemstudenten im Projekt war der Inhalt schon zum Großteil bekannt, so bemühte ich mich darum, Systeme wie Corba, COM, Agentensysteme etc. auf einfache Bilder des täglichen Lebens zu reduzieren, damit die Nicht-Techniker im Projekt eine Idee davon bekommen, um was es ging.

Die ausgearbeiteten Folien liegen hier zum Download bereit:

Rundgang

CMC Im Rahmen der Projektarbeit ging es darum, mit den Studenten der Gestaltungsfakultät undoder der Mediengestaltung Ideen und Anwendungsscenarien zu realisieren. Eines dieser Szenarien war die CMC. Diese Idee nutzt viele der zuletzt genannten Eigenschaften des VJ-Systems auf eindrucksvolle Weise aus und beweist die Funktionsfähigkeit der Konzepte.

Die Grundidee die Folgende:

* Die Maschine erzeugt Signale
* Die Maschine verarbeitet Signale
* Der Performer erzeugt Signale
* Der Performer verarbeitet über die Reaktion des Systems die Signale und wird in seinen Reaktionen gesteuert.
* Das Bild ist ein Signal, was die neu erzeugten Bilder steuert.

Abbildung 10: Max in der Performance

Die Tragweite dieser Ideen kann und soll an dieser Stelle nicht erschöpfend behandelt werden. Der Autor verweist an dieser Stelle auf die Dokumentation von Max und auf einen entstandenen Film, der auf der Pakt-Release Party aufgenommen wurde[24]. Eine weitere Ausserprokjektliche Zusammenarbeit an dieser Stelle ist mit besagtem Max und Dan[25] geplant.

Die Pipelinekonfiguration sieht folgendermassen aus:

```
'gst-launch dffplay location=/video.avi !
colorspace ! motionblur on=true strength=0.8 !
```

[24] www.uni-weimar.de/%7eheymann/Public/Max_Performance.mpg
[25] huetopic.net

```
colorspace ! motionmatrix ! colorspace ! colormix !
colorspace ! loopbuffer ! i420switch mode=2 !
videosink loopbuffer0.thru!i420switch0.sink_2
motionmatrix0.oscsrc!sink oscdispatcher out1=2
range1=i,0.0,400.0 drange1=f,0.0,100.0 out4=2
range4=f,0.0,700.0 drange4=f,0.0,1.0 src1!sink
ovjsink host=localhost node=loopbuffer0
param=speed port=8000 oscdispatcher0.src4!sink
ovjsink host=localhost node=i420switch0
param=strength port=8000'
```

Diese Konfiguration ist im Verzeichnis des ovjDemon unter dem Namen *working* enthalten. Zu starten mit *./ovjDemon -l working*.

Funktionsweise:

* Auf den Performer ist eine Kamera gerichtet.
* Hinter den Performer steht eine Leinwand.
* Bewegungen im Bild (Rückkopplungen/Bewegungen des Performers) werden registriert und dazu benutzt um andere Filter zu steuern/beeinflussen
* Bewegungsintensität steuert den Parameter *strength* des *i420switch* und den Parameter *speed* des *loopbuffer*.

Im Hintergrund versorge ein DJ die Szenerie mit Musik. Der Performer, in diesem Fall Max, begann mit dem Aufbau zu spielen. Er selbst hatte vorher keine Performanceerfahrungen mit dem System und lerne den Umgang während der Performance spielend und mit viel Spass.

Visual Jumper

Theresa und Carmen realisierten im Verlauf des Projekts eine bequeme Liegefläche mit verschiedenen Auslösern, die unterschiedliche Filmquellen hereinzoomen und zwar aus den Standorten, welche auf der gemalten Karte ausgelöst wurden. Um eine tiefere Erklärung zu erlangen, sei auf die Dokumentationen von Carmen und Theresa verwiesen und ausserdem auf den entstandenen Film[26].

Als Engine-Kernsystem wurde bei diesem Projekt der ovjDemon verwendet. Es war ein sehr erfolgreicher Test des ovjDemons. Das System lieft drei Abende durch, ohne Probleme. Der ovjDemon hat sich darüberhinaus, da er so einfach zu handhaben ist, als schnell anpass- und benutzbar herausgestellt. Das Zusammenbauen des Systems mit allen Unterprogrammen (VRIB-IN Software von Christian Lessig/ OpenGL Sink von Przemislaw Musialski) ging überraschend problemlos und schnell vonstatten, was alle Beteiligten sehr begeisterte. Nach vorheriger Absprache der Schnittstellen wurde das Visual Jumper System am Abend der ersten Performance erstmals zusammengebaut.

Neben dem ovjDemon steuerte ich viel Vorarbeit zur OpenGL Senke bei.

[26] www.uni-weimar.de/%7ePublic/VisualJumper.mpeg

Abbildung 11: Der Visual Jumper

Laser-Frame

Gegen Ende des Semesters trat Stephan Unholtz vom Handgriff Projekt an mich. Er baute einen stehenden Holzrahmen, der mit einer "Laserfläche" ausgeleuchtet war. Steckt ein Benutzer seinen Finger in diese Fläche, so leuchtet die Fingerkuppe rot auf. Da dieser Rahmen als berührungsloses Touchpad geplant war und Stephan die handwerkliche Seite übernahm, fehlte noch ein Programm, mit dem das Gerät benutzt und die Funktion dargestellt werden konnte.

Abbildung 12: 2D-Tracking. Der hellste Punkt wird getrackt.

Die Lösung des Problems bildete ein Gstreamer-Plugin, der *track2d*, mit dem der hellste Rotwert im Bild gefunden wird. Zur Visualisierung wurden ein horizontaler und ein vertikaler Balken im Bild eingezeichnet. Das Plugin tat hervorragende Dienste auf der Präsentation.

B11-Installationen

Zum Rundgang der Medienfakultät gab es einige Probleme mit Implementationsdetails. Das erreichte Ziel überzeugte am Ende jedoch und zeigte viele Aspekte des geschaffenen Systems:

* Steuerung über Netzwerk
* Interaktivität der Personen

* Selbststeuerung des Systems

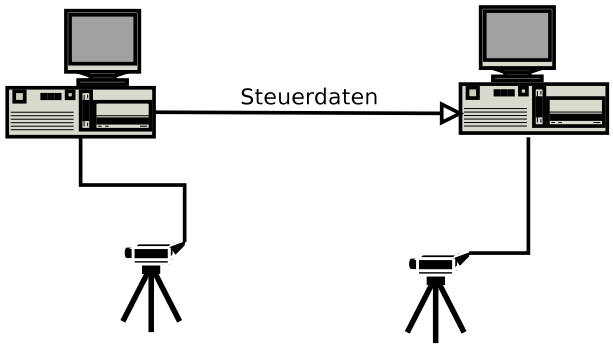


Abbildung 13: Aufbau des Präsentationssystems

Es gab zwei Rechner, die jeweils mit einer Kamera ausgestattet waren. Die eine Kamera filmte die besichtigende Menschenmenge. Die andere Kamera filmte einen bestimmmten Durchgangsbereich. Der Bewegungsänderungen der zweiten Kamera wurden dabei mit dem *motionmatrix* Plugin registriert. Die gesammelten Daten wurden an den Rechner mit der ersten Kamera geschickt und steuerten dort den *loopbuffer*, durch den das Kamerabild gegangen ist. Der *loopbuffer* steuerte die Abspielgeschwindigkeit einer vorher aufgenommenen Folge von Kamerabildern.

Wenn nun also keine Bewegung vor Kamera Zwei passierte, war nur ein Standbild auf der Ausgabe von Kamera Eins zu sehen. Ging jedoch jemand an Kamera Zwei vorbei oder tanzte davor herum, so sah man eine beschleunigte Ansicht auf der Ausgabe von Kamera Eins. Die maximale Abspielgeschwindigkeit ist dabei frei einstellbar.

Schlusswort

Zu Anfang des Projekts war meine Motivation so gross wie nie zuvor in einem Projekt an dieser Universität. Das Thema interessierte mich so, dass ich schon in den Ferien anfing mich mit Node-Based-Video-Processing auseinanderzusetzen und mich in die Gstreamer Doku hereinzulesen.

Die Resultate sprechen für sich und ich denke, dass wir am Ende ein ganzes Stück von dem geschafft haben, was wir uns am Anfang vorgenommen haben.

Die Zusammenarbeit war an einigen Stellen nicht so erfolgreich und die Inter-Projekt Kommunikation ist an einigen Stellen etwas mehr ins persönliche abgeglitten, als es sein sollte. Trotz allem ist die Arbeit größtenteils sehr gut vonstatten gegangen und sehr unerwartete großartige Zusammenarbeiten (Arbeit mit Martin Kleppe und Dan Fischer) haben vieles wettgemacht.

An dieser Stelle sollten implementierte Konzepte zum Großteil verworfen werden und als Erfahrungen in ein neues Konzept einfliessen. Dieser evolutionäre Prozess hat schon beim Pakt-Server sehr gut funktioniert und sollte auch bei Fragen der Inter-Pipeline Kommunikation und Verarbeitung externer Werte[27] aufgegriffen werden, um ein geschlossenes Konzept zu erlangen, was auf ganzer Linie einer Grundstruktur folgt.

Das erzeugte System ist somit als konzeptioneller Prototyp zu verstehen. Ein weiterführendes Projekt könnte aus den gemachten Fehlern und Erfahrungen von vornherein schneller Lernen und noch überzeugendere Ergebnisse hervorbringen.

Sebastian Heymann

[27] siehe Dokumentation Christian Lessig