

## Kurzfassung

Unsere Semesterarbeit bestand darin, mittels einer Kamera ein Gesicht zu filmen und daraus die Bewegungen von einzelnen Gesichtspunkten (z.B. Mundwinkel, Nase, ...) zu erkennen. Diese Bewegungen werden anschliessend in die MPEG-4 Form gebracht. Mittels einem MPEG-4 Spieler (Greta) werden die erfassten Bewegungen an einem virtuellen Gesicht animiert. Zusätzlich soll der ganze Ablauf in real-time erfolgen. Als MPEG-4 Spieler erhielten wir, zusammen mit der Aufgabenstellung den Greta FAP-Player. Leider ist dieser nicht real-time fähig, womit sich unsere Semesterarbeit auf nicht-real-time abkürzte.

Den grössten Knackpunkt stellte eindeutig die Erkennung von einzelnen Gesichtspunkten dar. Das Einlesen der Bilder via Webcam, das FAP-File erstellen, sowie das Abspielen der FAP-Files auf Greta (FAP-Player) enthalten zwar auch knifflige Tücken, sind aber in akzeptablem Zeitaufwand lösbar. Für die Erkennung von Gesichtspunkten bietet die Theorie interessante Ideen. Die im Buch (MPEG-4 Facial Animation) beschriebenen Lösungsansätze sind leider auf einem zu hohen Niveau und somit im gegebenen Zeitrahmen nicht zu realisieren. Die Erkennung der Gesichtspunkte verschlang dann auch den grossen Teil unserer Zeit. Anfangs versuchten wir das Gesicht aus dem Bild herauszufiltern, was uns noch recht gut gelang. Später entschlossen wir uns dem Gesicht anhand eines Bezugspunktes zu folgen, damit wir immer wissen wo das Gesicht ist. Als Bezugspunkt verwendeten wir die zwei dunklen Nasenlöcher, die relativ einfach zu erkennen sind. Da sich die weiteren Gesichtspunkte (z.B. der Mundwinkel) immer in einem gewissen Abstand zum Bezugspunkt stehen, können wir weitere Gesichtspunkte je in einem kleinen Bereich suchen.

Der Stand unserer Implementation ist soweit, dass sich der erste Gesichtspunkt (rechter Mundwinkel) bewegt. Jedoch nicht fehlerfrei und mit starkem Zittern.

Fazit: Leider haben wir es trotz viel Ehrgeiz und Zeiteinsatz nicht sehr weit geschafft. Aber trotzdem war es eine interessante Semesterarbeit welche uns einen Einblick in die Grundlagen der digitalen Bildverarbeitung gab.

## Inhaltsverzeichnis

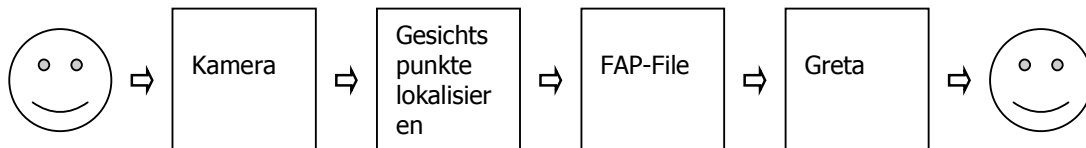
<b>KURZFASSUNG</b> .....	<b>1</b>
<b>VORWORT</b> .....	<b>3</b>
ANFANGSGEGEBENHEIT .....	4
<b>AUFGABENSTELLUNG</b> .....	<b>5</b>
<b>THEORIE</b> .....	<b>6</b>
MPEG-4 .....	6
WAS SIND FAP'S ? .....	6
WAS SIND FP'S ? .....	6
DIE EXTRAKTION DER RELEVANTEN GESICHTSPUNKTE .....	7
VON DEN GESICHTSPUNKTEN ZU DEN FAP'S .....	8
<b>PRAXIS</b> .....	<b>9</b>
KAMERA .....	10
Bedienung der Kamera mittels Matlab.....	10
Bedienung der Kamera mittels VC++ .....	11
GESICHTSPUNKTE LOKALISIEREN .....	11
1. Idee – den Hintergrund loswerden.....	11
2. Idee – fertig vom Internet.....	13
3. Idee – Der Nase Nach.....	13
Herauslesen der Gesichtspunkte .....	14
FAP-FILE .....	16
Protokoll / Schnittstelle .....	16
Erstellen der FAP Files .....	17
GRETA .....	18
<b>ERKENNTNISSE</b> .....	<b>20</b>
<b>CHRONOLOGIE</b> .....	<b>21</b>
<b>ANHANG</b> .....	<b>25</b>
REFERENZEN.....	25
Bücher .....	25
Internet Links.....	25
Kontakte.....	25
MPEG-4 FACIAL ANIMATION PARAMETERS .....	26
FACIAL FEATURE POINTS (FP).....	28
KORRESPONDENZ MIT FRAU BEVAQUA.....	29
C++ IMPLEMENTATION .....	31

## Vorwort

Dieser Schlussbericht soll den Leser durch unsere Semesterarbeit führen. Wir wollen versuchen, die Problematik und den Weg zur Lösung nachvollziehbar zu schildern.

Unsere Hauptschwierigkeit bestand Anfangs darin, dass wir weder Wissen noch Erfahrungen auf diesem Gebiet besaßen. Dies hatte zur Folge, dass es für uns schwierig war, die Problematik abzuschätzen. Anfangs verbrachten wir einige Zeit damit, uns in dieses Thema einzuarbeiten. Dabei erfuhren wir, dass andere das Problem der Gesichtsanimation auch zu lösen versuchten und weiterhin versuchen. Meist wurde auch das Resultat und der grobe Lösungsweg beschrieben. Die guten Resultate anderer ermunterten uns zu glauben, dass auch wir ähnliches erreichen könnten. Von der Website "<http://amp.ece.cmu.edu/projects/FaceTracking/>" probierten wir eine C++ Bibliothek aus, welche Facetraking (Gesichtsverfolgung) ermöglichen sollte. Im Buch (MPEG-4 Facial Animation von Igor S.Pandzic und Robert Forchheimer, ISBN 0-470-84465-5) welches wir zum Beginn der Semesterarbeit erhielten lasen wir von Gittermodellen und normalisierten Gesichtern. Die wichtigsten Punkte davon sind im Theorieteil zusammengefasst.

Damit sich der Leser nicht zwischen Fragmenten wadend durch den Schlussbericht zu kämpfen hat, wollen wir ihm eine kleine Weganweisung mitgeben. Die erste Seite ist eine Zusammenfassung unserer Semesterarbeit (Kurzfassung). Nach dem Vorwort und der Aufgabenstellung beschreiben wir die Theorie. Die wichtigste Aufgabe des Theorieteils ist es den Leser mit Begriffen vertraut zu machen, damit der Kern dieses Berichtes, der Praxisteil, verständlicher wird. Im Abschnitt Praxis sind unsere eigenen Lösungsansätze, Ideen und Ausführungen dokumentiert. Das Kapitel Praxis haben wir in die vier Arbeitspunkte Kamera, Gesichtspunkte lokalisieren, FAP-File und Greta unterteilt. Dabei verfolgen wir den Pfad des Bildes (wie in der Skizze zu sehen).



Im ersten Teil wird die Handhabung der benutzten Kamera beschrieben. Im Kapitel „Gesichtspunkte lokalisieren“ wird gezeigt, wie die Gesichtspunkte extrahiert werden. In den Kapiteln, FAP-File und Greta ist beschrieben wie die berechneten Daten in das richtige Format gebracht und anschliessend an Greta verfüttert werden. Der Zeitliche Ablauf unserer Arbeiten entspricht nicht den vier Arbeitspunkten, sondern ist aus der Chronologie ersichtlich.

Mit Zuversicht und Tatendrang gingen wir unsere erste Semesterarbeit an.

## Anfangsgegebenheit

Zu Beginn der Semesterarbeit erhielten wir das Buch MPEG-4 Facial Animation von Igor S.Pandzic und Robert Forchheimer (ISBN 0-470-84465-5) und eine CD mit dem FAP Spieler namens Greta (FAP : Face Animation Parameters). Wir hatten eine Kamera, MatLab und VC++ zur Verfügung. Besonders nützlich waren MatLab mit seiner Image Processing Toolbox, Vision for Matlab und die Musterprogramme von den Praktikas in Digitalen Medien ([www.medialab.ch](http://www.medialab.ch)).

Unser Arbeitsplatz befand sich im Labor für Digitale Signalverarbeitung (Zimmer 6.004) im Gebäude 6 der HSR.



## Theorie

Die digitale Bildverarbeitung bietet eine Unmenge von Theorie. Einen Einstieg in den MPEG-4 Standard bietet das Buch „MPEG-4 Facial Animation“. Kapitel 6 beschreibt ein Projekt, welches ebenfalls Gesichtspunkte zu erkennen versucht. Im Anhang des Buches („MPEG-4 Facial Animation“) hat es eine Liste mit wertvollen Links. Diese Websites enthalten Informationen zu Projekten auf dem Gebiet der Gesichtsanimation sowie zum MPEG-4 Standard. Aus dem Buch „Digital Image Processing“ kann man sich die nötigen mathematischen Fähigkeiten aneignen. In dieser Semesterarbeit wurden nur einfache Vergleichsmethoden benutzt um Gesichtspunkte zu finden. Wesentliche Verbesserungen könnten mit komplexeren mathematischen Werkzeugen erreicht werden.

## MPEG-4

MPEG-4 ist ein objektbasierender Kompressionsstandard welcher es erlaubt die in einer Szenerie vorhandenen Objekte, wie in unserem Fall ein Gesicht, unabhängig voneinander zu behandeln. MPEG-4 spezifiziert dabei nur das Decodieren von MPEG-4 konformen Bitstreams. Dies bedeutet, dass es dem Programmierer freigestellt ist, wie er die Information findet und in MPEG-4 Form bringt.

## Was sind FAP's ?

FAP steht für Face Animation Parameter. FAP's sind Bewegungsparameter. Jedes Gesichtsmodell hat definierte FAP Bahnen nach denen sich die Bewegungen richten. So ist es auch möglich die Bewegungen von Menschen auf Tieranimationen zu übertragen. Ein FAP hat seinen definierten Bewegungsverlauf sowie einen Wertebereich, welcher den Bewegungsbereich abdeckt.

Beispiel: FAP 3 open\_jaw 0...360  
FAP 3 bewegt das Kinn vertikal. Null bedeutet geschlossen. 360 bedeutet geöffnet, wobei 360 ein unnatürlicher Extremwert ist. Im Anhang ist eine Liste aller FAP's.

Damit FAP's abgespielt werden können, benötigt man ein Gesichtsmodell mit vordefinierten, modelspezifischen Animationsregeln für jeden FAP. Dies macht Greta für uns. Das Kodieren von FAP's besteht im wesentlichen aus zwei Schritten: Die Extraktion der relevanten Information aus dem Gesicht und die Berechnung der einzelnen FAP-Werte.

## Was sind FP's ?

FP steht für Feature Point was ein Gesichtspunkt ist. MPEG-4 spezifiziert für das neutrale Gesicht (geschlossener Mund, neutraler Gesichtsausdruck) 84 FPs. Der Hauptgrund für die Verwendung von FPs besteht darin, den FAPs

eine Ortsreferenz zu geben. Manche FPs, wie zum Beispiel diejenigen entlang der Haargrenze, werden nicht von FAPs beeinflusst. Diese dienen lediglich zur Erstellung eines Gesichtsmodells. FPs sind in Gruppen wie Nase, Backen, Mund, etc. unterteilt. Um ein MPEG-4 kompatibles Gesichtsmodell zu erstellen, sollten die Orte der FPs bekannt sein. Die FPs sollten gemäss dem MPEG-4 Standard angeordnet sein. FAPs in der Gruppe 2 bis 10 werden low level FAP genannt. Diese spezifizieren um wie viel ein FP bei gegebener Amplitude bewegt werden soll. Übertriebene Amplituden erlauben es auch unnatürliche Bewegungen durchzuführen. Im Anhang ist ein Bild mit allen FPs.

## Die Extraktion der relevanten Gesichtspunkte

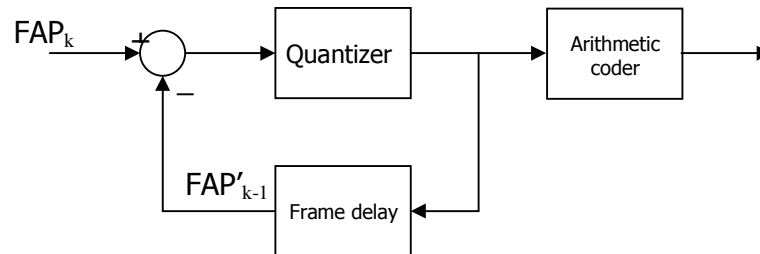
Der optimale Filter sollte auf jedes Gesicht und jeden Hintergrund funktionieren. Bei den Gesichtspunkten handelt es sich um extrahierte Punkte, aus denen wir die FAPs berechnen. Alle Beispiele, welche wir gesehen haben, versuchten als aller erstes das Vorhandensein und die Position eines Gesichtes zu finden. Dies hat den grossen Vorteil, dass die nachfolgenden Filter nicht mehr vom Hintergrund abhängen. Anschliessend wurde dem Gesicht ein Gitter aufgelegt um daraus die Lage und die Form des Gesichtes zu erhalten. Aufgrund der Gitterdifferenzen von zwei aufeinanderfolgenden Bildern erhält man die Bewegung aus welcher man die FAPs generiert. Generell sucht man die charakteristischen Eigenschaften von einem Gesichtspunkt. Solche Eigenschaften können Farbe, Form oder Bewegung sein.

Ein Beispiel das Gesicht über die Farbe zu finden: Man definiert nach einer statistischen Erfassung die Gesichtsfarbe. Durch einen Vergleich mit den Bildpixels erhält man eine Zahlenebene mit den Wahrscheinlichkeiten, dass das Pixel zu einem Gesicht gehört. So erhält man ein Gebiet in dem wahrscheinlich ein Gesicht ist. Bei der Gesichtsfindung und Erkennung wird sehr oft die Wahrscheinlichkeitsrechnung zu Hilfe genommen. Dies ist nicht weiter erstaunlich, denn so kann man die Unterschiede zwischen den Gesichtern ausgleichen. Andere Ideen nutzen die Tatsache aus, dass das menschliche Auge wegen des weissen Augapfels gut erkennbar ist. Je mehr Merkmale man berücksichtigt desto verlässlicher wird der Filter.

Wie aber erhält man nun die Gesichtspunkte? In der Theorie wurden dafür meistens Gitter in Form eines Standardgesichtes auf das gefundene Gesicht angepasst. Aus den resultierenden Gitterpunkten lassen sich FAPs berechnen. Der wesentliche Unterschied zwischen den verschiedenen Lösungsvarianten bestand jeweils in den unterschiedlichen Gitterstrukturen und den Adaptionalgorithmen. Diese Gitter sind vor allem dann hilfreich, wenn man eine Animation eines gefilmten Gesichtes haben will, welches gleich aussieht wie das gefilmte Gesicht. Dabei kann man das Gitter gleich als Rendermodell nutzen. Unsere Methoden sind von weitem nicht so fortgeschritten. Wir verwenden einfache Vergleiche um die gesuchten Koordinaten zu finden.

## Von den Gesichtspunkten zu den FAP's

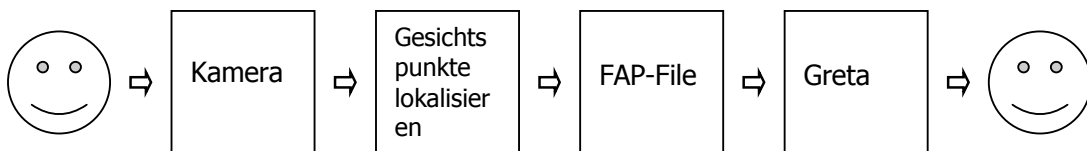
Damit man von den Gesichtspunkten zu den FAP's gelangt ist es notwendig zu wissen, wie die FAP's die Animation verformen. Grundsätzlich versucht man den FAPs geeignete Gesichtspunkte zuzuordnen. Eine einfache Zuordnung kann zu unerwünschten Gesichtsausdrücken führen. Dies versucht man, genau wie in der Regelungstechnik, durch eine Rückführung und einem Vergleich zwischen Erzeugtem und Gewolltem zu beheben. Die Struktur der Regelkreise kann sehr komplex werden. Meist wird dabei ein Tiefpassverhalten gewünscht. Ausreisser werden so unterdrückt.



Quelle: MPEG-4 Facial Animation, Seite 31

## Praxis

Als wir uns für dieses Thema entschieden, hatten wir noch keine Vorkenntnisse über Facial Animation und MPEG-4. Wir fingen also an uns im Buch „MPEG-4 Facial Animation“ einzulesen. Besonders Kapitel 6 schien vielversprechend. Das Kapitel ist eine Zusammenfassung eines sehr ähnlichen Projektes. Es geht um die Extraktion von MPEG-4 FAPs aus einem Video. Die ersten Kapitel sind eine Einführung in den Teil des MPEG-4 Standards, welcher für uns interessant ist. Als besonders nützlich erwiesen sich die Internet-Adressen welche im Anhang des Buches zu finden sind. Von der „www.visagetechnologies.com“ Site aus haben wir mit Igor Pandzic Kontakt aufgenommen. Herr Pandzic Unterstützte uns in Form von nützlichen Dokumenten (Usermanual zum „visage-toolkit“) und zeitbeschränkten Lizenzen für ein MPEG-4 Tool. So erwarben wir uns einen kleinen Überblick über das heutige Umfeld der Gesichtsanimation. Mit dieser Sicht der Dinge erstellten wir einen Zeitplan mit den Meilensteinen Kamera, Filter für Gesicht, Gitter auflegen, Gitter optimieren, Parameter auslesen, FAP-File erstellen und auf Greta abspielen. Herr Schuster schlug uns schon damals Vereinfachungen vor, doch wir waren zu sehr in Fahrt und davon Überzeugt die Probleme in nützlicher Frist zu meistern. Mit der Zeit mussten wir jedoch einsehen, dass sich unser erster Fahrplan zu sehr am Buch 'MPEG-4 Facial Animation' orientierte. Der Lösungsansatz mit dem Gitter auf dem Gesicht ist schön und gut, aber es bleibt auch im Buch nur beim Ansatz. Um so zu zaubern, wie die Meister ihres Fachs, bedürfte es viel Zeit. Die neuen Arbeitspunkte (Kamera, Gesichtspunkte lokalisieren, FAP-File und Greta) sind etwas allgemeiner gefasst. Nach den ersten Anfangsturbulenzen war es nun unser Ziel, nicht einen Arbeitspunkt nach dem andern abzuarbeiten, sonder primär die gesamte Struktur erst einmal zum Laufen zu bringen. Einen einfachen Gesichtspunkt einlesen, zu einem FAP wandeln und auf Greta abspielen. Wie bereits angetönt, ist auch die Dokumentation gemäss diesen vier Arbeitspunkten gegliedert.



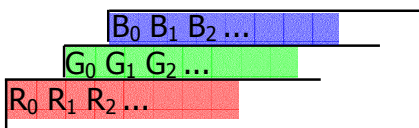
## Kamera

Für unsere Semesterarbeit verwenden wir die Fire-i400 Digital Camera von Unibrain, welche im Labor 6.004 an jedem Arbeitsplatz standardmässig installiert ist. Die Kamera liefert uns ein Bitmap. Das Bild ist 480 Pixel hoch und 640 Pixel breit. Jedes Pixel besteht aus Rot-, Grün- und Blauanteil, wobei der Intensitätsbereich einer Farbe von 0 bis 255 (8 bit) variieren kann. Wir erhalten also ein 24bit – RGB Bitmap (\*.bmp) à  $480 \cdot 640 \cdot 24 \text{bit} = 901 \text{KB}$ . Etwas mühsam war, dass die Kamera oft übersteuerte. Dies führte zu einem schlecht ausgenutzten Intensitätsbereich. Prüft man also die Pixel auf bestimmte Bereichseigenschaften, werden die Tolleranzen sehr klein. Dies bedeutet, dass der Wertebereich in dem sich das gesuchte Pixel befindet schrumpft. Dieses Problem kann durch Kontrastausdehnung gelöst werden (Buch: Digital Image Processing, Kapitel 4). Allgemein, kann mit solchen Methoden viel an Robustheit gewonnen werden.

### Bedienung der Kamera mittels Matlab

Unser erstes Bild haben wir mit der Kameraeigenen Software aufgenommen und abgespeichert. Im Matlab fanden wir uns dank der „Imaging processing Toolbox“ schnell zurecht. Mit der Funktion „Bild = imread('pfadname\dateiname.bmp')“ wird das Bild in Matlab eingelesen und in einem drei dimensional Array abgespeichert, wobei eine Ebene jeweils eine Farbe enthält. Dank dieser übersichtlichen Darstellung ist es relativ einfach ein simples Filter zu schreiben.

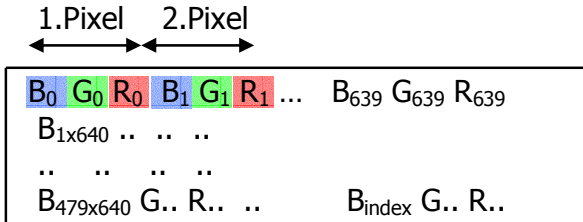
Mit „figure, imshow(Bild)“ wird das Bild in einem Fenster angezeigt. Matlab bietet aber noch mehr. Mit Hilfe der VFM (Vision for Matlab) Bibliothek, geht es noch einiges bequemer. Mit der Funktion „Bild = vfm('grab', 1)“ wird ein einzelner Frame direkt von der Kamera eingelesen. Somit können wir unsere Filter auf Bildfolgen austesten. Matlab ist in dieser Beziehung sehr langsam. Deshalb empfehlen wir das dynamische Verhalten in C++ zu untersuchen. Matlab eignet sich gut zum Tüfteln, denn man erreicht schnell ein funktionierendes Resultat.



## Bedienung der Kamera mittels VC++

Die Ansteuerung der Kamera mittels VC++ wird im Praktikum 5 im Fach 'Digitale Medien' behandelt. Wir haben dieses interessante Praktikum durchgearbeitet und uns die unklaren Punkte von Herrn Schuster erklären lassen. Das Praktikum ist unter [www.medialab.ch](http://www.medialab.ch), Digitale Medien, Praktikum 5 zu finden.

Leider haben wir in C++ nicht ein 3-dim Array wie in Matlab, sondern ein 1-dim Array. Ausserdem haben wir BGR anstelle von RGB.



$$\text{Index} = 3(r * X + c) + \text{Offset}$$

r: rows (y-Achse)

c: columns (x-Achse)

Offset: 0 → B, +1 → G, +2 → R

X: Bildbreite in Pixel

## Gesichtspunkte lokalisieren

Ausgangslage ist ein vorhandenes Bild des Gesichtes. Wie im Arbeitspunkt „Kamera“ behandelt, erhalten wir von der Kamera ein 24bit RGB Array. In diesem Arbeitspunkt beschreiben wir unsere Ideen, wie wir Gesichtspunkte aus dem Gesicht filtern wollen.

### 1. Idee – den Hintergrund loswerden

Das Ziel unseres ersten Gehversuches war es, den Kopf einer Person von einem blauen Hintergrund zu unterscheiden. Der Grundgedanke dafür war, sich später nur noch auf das Gesicht konzentrieren zu können.

Als Hintergrund diente uns ein mehr oder weniger blaues Blatt Papier (Format A3). Da damit nicht der ganze Hintergrund ausgefüllt war, ergänzten wir, mittels Bildbearbeitungsprogramm, den äusseren Bereich von Hand mit blau.

In der Blaubene erschienen grosse Werte im Bereich vom Hintergrund und kleine Werte im Bereich vom Gesicht. In unserem ersten Filter fragten wir lediglich Punkt um Punkt nach seinem Blauwert ab. Alle die einen grösseren Wert als 200 aufwiesen, definierten wir als Hintergrund (Null), alle anderen als Gesicht. Auf der Stirn hatte sich zum Zeitpunkt der Bildaufnahme Licht reflektiert. Diese Reflexion stellt im Bild einen weissen Fleck dar.



Letzter Stand vom Konturfilter



Weiss besteht ja bekanntlich aus hohen Anteilen von rot, grün und blau, was dazu führte, dass ein Teil der Stirn als Hintergrund definiert wurde. Des weiteren enthielt der Schatten zu wenig Blauanteil so dass dieser als Gesicht erkannt wurde. Zur Verbesserung des Filters prüften wir noch auf zusätzliche Bedingungen. Da das Gesicht relativ viel Rotanteil enthält formulierten wir folgende Bedingung: Wenn der Blauanteil hoch ist und der Rotanteil tief, dann gehört das Pixel zum Hintergrund. Somit gelang es uns auch den Schatten zu berücksichtigen. Weiter versuchten wir die Geschichte der vorangegangenen Pixel einzubeziehen.

So weit so gut, das Resultat war nicht schlecht. Dieser Filter ist aber noch viel zu empfindlich. Wir mussten für jeden von uns Parametersets definieren. Trotzdem haben wir einiges daraus gelernt. Sobald Probleme wie Bart, andere Hautfarbe, Brille, Lichtreflexion, Schatten,... auftauchen, stehen wir vor erheblichen Schwierigkeiten.

Kurzer Beschrieb der Funktion:

Die Funktion nennen wir Konturfilter, weil wir alle Pixel ausserhalb des Gesichtes auf Null setzen wollen. Wie findet unser Filter die Gesichtsgrenze? Wir übergeben der Funktion die Blaugrenze die Rotgrenze und die Suchtiefe.

function [Y] = konturfilter(B, tiefe, bg, rg)

% B = Bild[höhe, breite, 3]

% tiefe = n geschichte Pixel      tiefe = 3      Wenn die letzten drei Pixel als Gesicht erkannt wurden, dann beende die Reihe.

% bg = Blau-Grenze      bg = 165

% rg = Rot\_Grenze      rg = 95

Alle Punkte, welche auf der Blaubene einen höheren Wert aufweisen als die Blaugrenze und einen tieferen Wert auf der Rotebene als die Rotgrenze, werde in allen Ebenen auf Null gesetzt.

## 2. Idee – fertig vom Internet

Durch die Internetsuche nach vorhandener Theorie und Projekten sind wir auf eine frei zugängliche Funktionsbibliothek für VC++ gestossen.

<http://amp.ece.cmu.edu/projects/FaceTracking/>

Beschreib der Bibliothek:

Die Bibliothek heisst „FT Library“ und dient der Gesichtsverfolgung. Sie bietet Funktionen an, welche uns erlaubt hätten, die Position eines sich bewegenden Gesichtes zu verfolgen. Gemäss Beschreibung errechnet die Funktion FtTraining mit Hilfe der Wahrscheinlichkeitsrechnung die Farbverteilung der Gesichtspixels. Diese wird später in der Funktion FtTrackNextFrame benötigt um dem Gesicht zu folgen. Diese Funktion nutzt die Tatsache aus, dass sich die Position des Gesichtes von Frame zu Frame nicht wesentlich ändert.

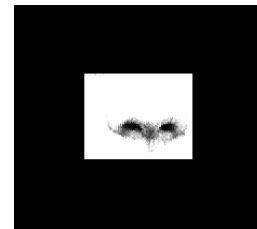
Leider mussten wir nach der Implementierung feststellen, dass eine Bibliotheksfunktion bei uns einen Speicherzugriffsfehler erzeugt. Da die cpp Files nicht zugänglich sind konnten wir diesen Fehler nicht beheben und mussten den Versuch mit viel Zeitverlust beenden.

## 3. Idee – Der Nase Nach

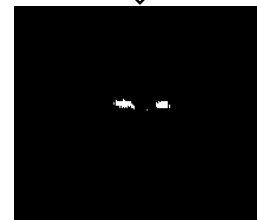
Anstatt bei jedem Bild das Gesicht herauszufiltern, genügt es uns lediglich zu wissen, wo sich das Gesicht befindet. Als Bezugspunkt entschieden wir uns für den Nasenspitz. Genauer für die Mitte der beiden Nasenlöcher. Dieser Punkt ist unabhängig von den Gesichtsbewegungen (ausgenommen Rotation, Heben und Neigen) und relativ einfach zu implementieren (finden/detektieren/...) da jede Person zwei dunkle Nasenlöcher besitzt. Im diesem Filter geben wir die ungefähren- und berechneten daraus die genauen-Koordinaten des Bezugspunktes.

Anhand den ungefähren Koordinaten bestimmen wir ein Fenster, welches den Bereich der Nase enthält. Alle anderen Werte ausserhalb vom Fenster werden auf null gesetzt. Nun werden die drei Ebenen von R,G und B elementweise addiert, womit wir den Effekt der dunklen Nasenlöcher hervorheben (mit multiplizieren gelingt dies erstaunlicherweise nicht so gut). Das Resultat sind zwei mehr oder weniger runde Flächen, welche die Nasenlöcher darstellen. Wie finden wir die Nasenmitte? Die Höhe erhalten wir indem wir nach einer bestimmten Anzahl Nasenpixel pro Reihe suchen. Die zwei äussersten Pixel dieser Reihe werden gemittelt. So erhalten wir in etwa die Nasenmitte, welche uns als Bezugspunkt dient. Ob die Genauigkeit dieses Verfahrens ausreicht wird sich noch zeigen. Vorerst begnügen wir uns mit diesem Filter.

Als weiterer Schritt dieser Idee wollen wir diesen Filter in einer Schleife einbauen. Diese Schleife soll nichts weiteres tun als pro Zyklus ein Bild schiessen und den Filter anwenden. Dabei ist das vorangegangene Resultat



Nach dem addieren der Farbebenen



Die gefundenen Nasenlöcher

der Übergabewert für den aktuellen Funktionsaufruf des neu gesuchten Bezugspunktes. Somit haben wir ein Fadenkreuz das immer dem Bezugspunkt folgt.

In Matlab implementier benötigt die Funktion rund 0.7sec für einen Schleifendurchgang (Bild holen, bearbeiten und mit Fadenkreuz ausgeben). Dies haben wir mit den Matlabfunktionen tic und toc gemessen. 0.7sec sind viel zu lange. Der Ablauf wird äusserst ruckartig und instabil.

In C++ wird die Bearbeitungszeit zweitrangig, denn die Geschwindigkeit wird nun von der Kamera begrenzt, die maximal 15 Bilder pro Sekunde liefern kann. Dies ist jedoch nur eine Abschätzung, da wir die Zeit in C++ nicht gemessen haben. Mit den 15 Bildern pro Sekunde ist die Funktion zwar stabiler, aber eine höhere Framerate wäre wünschenswert. Gerade bei einem Gespräch bewegen sich die Gesichtspunkte im Mundbereich schneller.

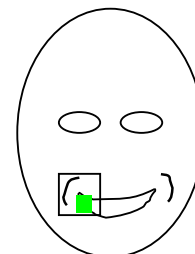
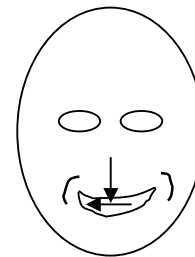
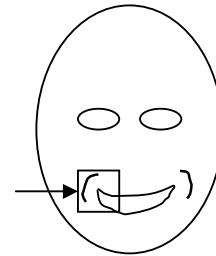
Beschränkung: Der Kopf darf weder gedreht noch geneigt werden und die Startposition muss bekannt sein.

### Herauslesen der Gesichtspunkte

Die Basis zum Herauslesen der Gesichtspunkte, bei der aktuellen Idee, ist der Bezugspunkt von vorhin.

Als ersten Gesichtspunkt nehmen wir uns den rechten Mundwinkel vor. Der Mundwinkel befindet sich, relativ zum Bezugspunkt, immer in der selben Region. Ebenso wie bei den Nasenlöchern nutzen wir hier die Tatsache aus, dass die Mundwinkel durch einen dunklen Schatten und Farbkontrast auffallen. Am einfachsten wäre es in einer gewissen Distanz zum Bezugspunkt ein Fenster zu definieren und darin von aussen her die ersten 2 bis 4 dunkelsten Punkte als Mundwinkel zu definieren. In einem entspannten Gesicht funktioniert dies. Beim Lachen bilden sich Lachfalten. Diese Falten werfen Schatten, welcher als Mundwinkel erkannt würde. Deshalb funktioniert diese Idee in unserem Fall nicht.

Unser nächster Versuch basiert darauf, dass wir im Mund entweder dunkle Schatten oder weisse Zähne haben (wie weiss die auch immer sein mögen..). Vom Bezugspunkt, plus einen Versatz direkt in die Mundmitte, soll die Funktion nach Aussen wandern, bis sie am Mundwinkel angekommen ist. Realisiert haben wir es mit einer einfachen Abfrage der angrenzenden Pixel. Diese Funktion stellte sich leider als unbefriedigend heraus, da der resultierende Punkt einerseits nervös herumsprang und andererseits oft an den Zähnen hängen blieb. Der Grund für das Herumspringen ist einfach zu erklären. Bei jedem Frame wird wieder in der Mundmitte begonnen. Die Farbumgebung hat sich leicht verändert. Deshalb wird ein anderer Wanderweg begangen. Dies führt wegen der enormen Licht- und Farbempfindlichkeit der Funktion zu grossen Resultatunterschieden was sich als



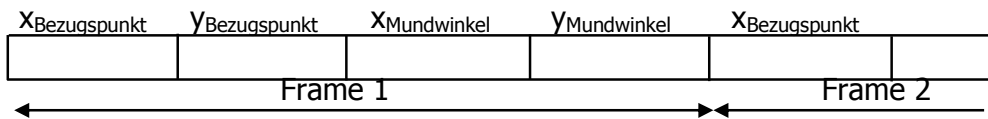
wildes Herumspringen äussert. Nach eigenen Einschätzungen wäre diese Funktion mit einigem Zeitaufwand wahrscheinlich realisierbar. Einfachheitshalber haben wir uns entschieden, den Mundwinkel mit einem grünen Punkt zu markieren. Dazu verwenden wir wieder das Fenster mit einer gewissen Distanz zum Bezugspunkt und suchen darin nach hohem Wertanteil in der Grünebene und tiefen Werten in den anderen Farbebenen. Als Resultat liefert uns die Funktion die x und y Koordinaten vom Mundwinkel. Unter Arbeitspunkt „FAP-File“ ist erklärt, wie wir von diesem Gesichtspunkt zum FAP gelangen.

Wie die Zunge, die Augenlieder oder andere schwierige Gesichtspunkte herausgelesen werden können wissen wir nicht. Bei mehreren FAP's stossen unsere Ideen auf Ihre Grenze. Da müsste man komplexere Methoden anwenden. Unsere Funktionen sind viel zu empfindlich. Ein anderer markanter Nachteil ist unsere uneindeutige Suche. Beispielsweise suchen wir nach dunklen Werten um die Nasenlöcher zu finden. Dunkle Werte sind aber überall auf dem Bild zu finden. Deshalb ist es auch nicht erstaunlich wenn ab und zu Augenhöhlen mit Nasenlöcher verwechselt werden. Einen Vorteil haben unsere Funktionen dennoch! Sie sind einfach, schnell und leicht verständlich. Schnell bedeutet hierbei: Viele Resultate pro Zeiteinheit. Besonders das „schnell“ gibt Lösungsideen welche statistisch vorgehen Aufwind. Es ist vorstellbar, viele Punkte schnell zu finden und diese clever auszuwerten. Die Entwicklung auf diesem Gebiet weist jedoch in eine andere Richtung. Will man bessere Erfolge erzielen muss man spezifischer suchen. Spezifischer suchen heisst, nicht erwünschte Einflüsse zu filtern. Die Grenzen hängen von der Rechenleistung ab.



## Erstellen der FAP Files

Momentan verwenden wir die Version 2.1. Der Filename ist „FapFile.fap“. Die Framerate stellen wir auf den selben wert wie bei der Webcam (15 Frames pro Sekunde). Diese Variante ist zwar nicht so elegant, aber da uns die Zeit ausgeht, nehmen wir einfach an, die Verarbeitungszeit sei kürzer als die Zeit pro Frame. Unter Arbeitsschritt „Gesichtspunkte lokalisieren“ ist erklärt, wie wir die Gesichtspunkte aus dem Bild herauslesen. Die Koordinaten des Bezugspunktes und des Mundwinkels werden in einem 1-dim Array gesammelt (siehe Skizze). Ursprünglich sollten die Parameter real-time übergeben werden, da dies aber nicht möglich ist, haben wir uns für diese Methode entschieden. Einfachheitshalber haben wir dem Array eine fixe Länge zugeteilt, wodurch die Lände des FAP-Files natürlich auch gegeben ist.

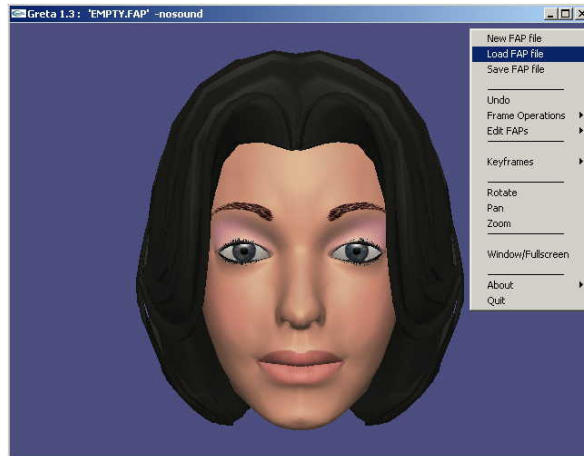
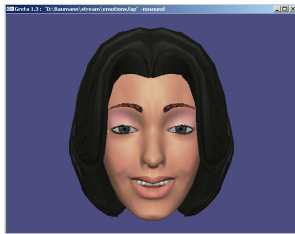


Die Maske wird wir am Anfang einmal definiert.

Wenn wir unseren Bezugspunkt (Nasenspitz) und den rechten Mundwinkel haben (Koordinaten in Pixel), können wir die zwei FAP's 7(stretch\_r\_cornerlip) und 13(raise\_r\_cornerlip) berechnen. Im neutralen Gesicht, das heisst alle Muskeln entspannt und Blick geradeaus, sind in Greta die FAP Werte gleich Null. Der FAP Wert sagt aus, wo die Bewegung hingehet, nicht um wie viel, im Bezug auf den letzten Punkt, bewegt werden muss. Den rechten Mundwinkel definieren wir im neutralen Gesicht mit einer bestimmten Distanz zum Bezugspunkt. Dies ist jedoch problematisch. Wenn das Gesicht nicht immer dieselbe Distanz von der Kamera (Grösse im Bild) hat, variieren die neutralen Koordinaten. Vorerst verlangen wir, dass die gefilmte Person immer denselben abstand zur Kamera hat. Raise\_r\_cornerlip ist im wesentlichen die Differenz von Bezugspunkt zu Mundwinkel auf der Ordinate ( $[Y_{munwR} - Y_{bezugspkt} - Y_{mundwRneutral}] * \text{Skalierungsfaktor}$ ) und stretch\_r\_cornerli ist im wesentlichen die Differenz von Bezugspunkt zu Mundwinkel auf der Abszisse ( $[X_{MunwR} - X_{Bezugspkt} - X_{MundwRneutral}] * \text{Skalierungsfaktor}$ ). Die genaue Umsetzung muss mit Versuchen ermittelt werden. In unserer Implementation haben wir uns für einen linearen Skalierungsfaktor entschieden. Derselbe Weg kann für einige andere Gesichtspunktepunkte ebenfalls angewendet werden. Der rechte Mundwinkel ist noch relativ einfach. Bei schwierigen Gesichtspunkten, wie der Zunge, muss man sich was anderes einfallen lassen.

## Greta

Greta ist der Name unseres FAP Players, welchen wir zusammen mit der Aufgabenstellung erhielten. Im Bild nebenan ist die liebliche Greta abgebildet.



Greta lässt sich in 3 verschiedene Modi starten.

→ Normal Mode / Pipe Mode / Server Mode

Um Greta kennen zu lernen, benutzten wir sie anfangs im normal Mode. Im Internet fanden wir ein FAP-File (emotions.fap von visagetechologies), welches uns als Testfile diente. Vom GUI aus luden und starteten wir das File emotions.fap. Greta lächelte uns das erste mal entgegen. Die Bewegungen sehen zwar künstlich aus, aber für einen animierten Kopf, der lediglich von ein paar Parametern gesteuert wird ist's ganz akzeptabel.

Zu unserer Freude konnte man Greta auch in Servermode Starten. Dank dem Praktikum 2 (TCP Verbindung, [www.medialab.ch](http://www.medialab.ch)) vom Fach Digitale Medien hatten wir bereits einen für uns brauchbaren Client. Schnell ersichtlich war die Portnummer 1333. Einem Mail von Frau Bevaqua an Herrn Schuster konnten wir entnehmen, dass es sich um einen TCP Server handelt. Frau Bevaqua ist bei Greta als Kontaktperson angegeben. Wir mussten also nur noch den Client vom Praktikum 2 verstehen und ergänzen, damit dieser die FAP-Files an Greta senden würde. Die Idee war, die FAP Mask und Werte in einem Stream zu senden. Leider wollte sich Greta auch mit Herrn Schuster's Hilfe nicht bewegen, worauf wir mit Frau Bevaqua Kontakt aufnahmen (Korrespondenz im Anhang). Anhand unseren Fragen konnten wir folgendes aus der Korrespondenz mit Frau Bevaqua erfahren:

Greta ist ein multithreaded Server und kann somit das einte File abspielen, während das nächste schon empfangen wird. In Servermode will Greta ein FAP-File und ein wave File laden. Wobei das wave File nach dem FAP-File folgt, sofern es überhaupt gesendet wird (wave File ist nicht zwingend). Greta kann mit einem Stream von FAP Masks und Werten nichts anfangen, sondern lädt eine Serie von FAP Masks und FAP Werten immer in einem File. Greta wurde nicht für real time Anwendungen programmiert.

Da wir nun keinen Stream senden konnten, wollten wir einfach viele kurze FAP-Files senden, um so trotzdem eine nahezu real time Animation zu erreichen.

Am Ende einer Animation kehrt Greta nicht auf eine neutrale- oder ausgangs-Position zurück, sondern bleibt auf der letzten Einstellung der Animation stehen. Womit die Idee mit den vielen kurzen FAP-Files eigentlich klappen müsste. Theoretisch wäre es also möglich gewesen, viele kurze FAP-Files hintereinander zu senden. Als wir versuchten Greta ein FAP File zu senden bestätigte Greta den Empfang, spielte aber das FAP-File nicht ab. Leider brachten wir es nicht zu Stande, Greta zur Ausführung der Animation zu bringen.

Die letzte Möglichkeit war Greta in Pipemode zu starten. Die Idee zu diesem Zeitpunkt war, die FAP-Files an einen UDP Server zu senden, welcher Greta die FAP-Files über Pipemode hätte speisen sollen. Da die Animation im Pipemode nicht fehlerfrei ablief, verwarfen wir die Idee.

Weil Greta weder in Servermode noch in Pipemode richtig funktionierte, entschieden wir, zusammen mit Herrn Schuster, das Projekt vorerst nicht in real time zu realisieren. Wir werden also ein FAP-File erstellen und anschliessen im normal Mode laden und starten.

Im Internet haben wir noch einen weiteren FAP Player (von Igor Pandzic, [www.visagetchnologis.com](http://www.visagetchnologis.com)) gefunden. Die Zeit reichte leider nicht mehr diesen genauer unter die Lupe zu nehmen.

## Erkenntnisse

Was können wir in Zukunft besser machen?

Der Hauptzeitverlust ist unserem Optimismus zu verdanken. Die Frage der Realisierbarkeit hat bei uns stark an Gewicht gewonnen. In unserem Fall bedeutet dies, zuerst eine funktionierende Minimallösung zu realisieren welche man in einem zweiten Schritt noch erweitern kann. Dies hat den Vorteil, dass man schneller Erfahrungen sammeln kann und einen grösseren Überblick erhält. Sich verbissen auf bestimmte Zwischenschritte zu stürzen lohnt sich nicht! Möglicherweise gibt es leichtere Wege zum Ziel. Wir haben festgestellt, dass uns eine etwas genauere Führung unseres Laborheftes ein stärkeres Bewusstsein über unsere Arbeit gegeben hätte. Dies steigert die Effizienz.

Unsere ersten Gehversuche haben uns ein Gefühl und eine Vorstellung für die Bildverarbeitung, insbesondere für das Suchen nach speziellen Eigenschaften, gegeben. Es gibt sehr viele Möglichkeiten ein Problem zu lösen, man weiss jedoch nicht im Vorherein welche Konsequenzen sich wie stark auswirken werden. Deshalb ist es wichtig die wesentlichen Eigenschaften des Gesichtspunktes, welchen man sucht, zu erörtern. Beispielsweise erkennt man den Mund eines Gesichtes auch, wenn dieser grün wäre. Farbe ist also nicht eine wesentliche Mundeigenschaft. Vielleicht ist die Form und Position charakteristischer. Das Ziel solcher Überlegungen ist eine höhere Stabilität zu erreichen. Stabilität bedeutet, dass Störeinflüsse sich nicht oder nur gering auswirken. Störeinflüsse sind vor allem Lichtveränderungen, Schatten, sich ändernde Farbintensitäten. Denkbar wären sogar Objekte, welche kurzzeitig die Sicht auf das Gesicht verdecken. Komplexere Lösungen berücksichtigen solche Überlegungen. Wie so oft verschlingen Komplexere Lösungen auch mehr Zeit. Entwicklungszeit und meist auch Rechenzeit was wiederum eine Stabilitätsminderung bedeutet. Für uns war vor allem die Entwicklungszeit der Grund für die Einfachheit.

An dieser Stelle möchten wir unserem Dozenten, Prof. Dr. G. Schuster herzlich danken, für die interessante Semesterarbeit und seine ermutigenden Worte zwischendurch.

## Chronologie

- Do 30.10      1h            12 – 13
- Vorstellung der Semesterarbeiten durch Herr Schuster. Wir erhielten das Buch „MPEG-4 Facial animation“.
- Mi 05.11       8h            8 – 12 ; 13 – 17
- Zeitplan aufgestellt
  - Vorgehen besprochen
  - Fragen für Herr Schuster notiert (siehe MPEG4\_FA\_5-11.doc)
  - Erste Gehversuche in MatLab: Bilder einlesen, Pixel ansprechen, Ideen ausprobieren.
  - Buch lesen (dies geschah in der Freizeit )
  - Beim Mittagessen erfahren, dass Herr Leuenberger mit Bildverarbeitung zu tun hat.
  - 14:20 Besprechung mit Herr Schuster (siehe Protokoll)
  - Überlegungen bezüglich der Dokumentation -> LaTeX oder Word ?
- Di 11.11       3h            17 – 20
- Matlab versuche mit Bildern
  - Erste Filteransätze zur Extraktion des Gesichtes  
Internetsuche nach ähnlichen Projekten mit Hilfe der angegebenen Links vom Buch  
<http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm>  
(MPEG-4 Homepage)  
<http://amp.ece.cmu.edu/projects/FaceTracking/>  
<http://www.iso.ch/iso/en/ISOOnline.frontpage> (ISO/IEC IS 14496-2 Visual (Definition of FAP's by motion of FP's)  
<http://www.visagetechnologies.com/> (Igor S.Pandzic arbeitet bei visagetechnologies. Alternativer FAP Player zu Greta für Schulungszwecke und professionellen Einsatz.)

- Mi 12.11      9h      8 – 12 ; 13 – 18
- Filterprogrammierung
  - Vereinfachungen aufgrund der Filtererfahrung und Hinweisen von Herr Schuster getroffen: Blauer Hintergrund, ausgerichtetes Gesicht. Fürs erste nur Bilder, nicht Film.  
Ideen:
    1. Filter welcher auf bestimmte Intensität auf der Blauenebene prüft.  
Problem: Lichtreflexionen im Gesicht ergeben ungewöhnlich hohen Blauanteil im Gesicht.
    2. Filter welcher ebenfalls auf Blauintensität prüft aber diesmal im Zusammenhang mit einem selbstgemachten Floodfill.  
Problem Schatten: Das Resultat ist schon viel besser wie bei der ersten Idee.
  - Download einer C++ Library namens „FT Library“ zur Gesichtserkennung.  
<http://amp.ece.cmu.edu/projects/FaceTracking/>
  - Lizenzanfrage an Igor Pandzic für FAtoolkit
- Sa 15.11      5h      8 – 13
- Lizenz von Pandzic für einen Monat erhalten.  
Praktikum 5 durchgearbeitet. Darin ging es um das Versenden von Video über IP. Uns interessiert vor allem wie die Kameradaten eingelesen werden.  
Problem: Der logische Vorgang wurde uns dabei nicht klar. -> Fragen über die Funktion „FrameCallbackProc“.
- Di 18.11      2h      17 – 19
- Beginn der Implementierung der „FT Library“.
- Mi 19.11      8h      8 – 12 ; 13 – 17
- Elimination der Programmierfehler.  
Problem: Die Funktion FtTraining() verursacht einen unerlaubten Speicherzugriff.
  - Download von einer MPEG-4 Toolbox. Source: [www.iti.gr](http://www.iti.gr)
  - Download vom visage toolkit , visage MPEG-4 SDK (software development kit) und Sample Files. Source: <http://www.visagetechologies.com/> (Codes von Igor Pandzic erhalten)
- Mi 26.11      8h      8:30 – 12 ; 13 – 17:30
- Versuch die „FT Library“ zum laufen zu bringen ist gescheitert. Wir werden diese Bibliothek nicht verwenden.
  - Greta bewegt sich ! Dank der guten Dokumentation des visage toolkit und den Sample-Files.
  - Probleme mit der TCP Verbindung zu Greta. Sie akzeptiert die Verbindung aber bewegt sich nicht.

- Di 02.12      1h      17 – 18
- Suchen FAP Liste nach gut erkennbaren Gesichtspunkten ab. Am einfachsten erschienen uns die Mundwinkel (FAP7: stretch\_l\_cornerlip FAP13: raise\_r\_cornerlip)
- Mi 03.12      8h      8:30 – 12 ; 13 – 17:30
- Matlab Funktion ( mittelPkt() ) schreiben zur Findung der Nasenlöcher um daraus den Bezugspunkt zu bestimmen.
  - Matlab Problem: wie definiere ich standardwerte in einem m-File? Indem ich die Anzahl Übergabeargumente abfrage.
- Mi 10.12      8.5h      8:30 – 12 ; 13 – 18
- mittelPkt() ausprogrammieren. Versuche die Funktion auf Bildfolgen anzuwenden haben noch nicht geklappt.
  - Erste Ansätze für die Mundwinkelerkennung.
- Di 16.12      1h      17:30 – 18:30
- Fehlerbehebung der Matlab Funktion mittelPkt().
- Mi 17.12      8h      8 – 12 ; 13 – 17
- Beginnen am Zwischenbericht zu schreiben.
  - Aufgrund der Schwierigkeiten mit Greta entschieden wir uns die Animation nicht mehr in real time zu realisieren. Eventuell können wir einen anderen FAP Spieler einsetzen.
- Sa 20.12      9:30 – 12 ; 13 – 17
- Schreiben am Zwischenbericht.
- So 21.12      10 – 12 ; 12:30 - 20
- Zwischenbericht fertig stellen.
- Mi 24.12.03      8:30 – 12
- Matlabcode zur Ermittlung der Nasenlöcher in C++ implementiert.
  - Erste Versuche Bilder mit C++ zu manipulieren. Punkte setzen, Arrayformat bestimmen, Pointerarithmetik repetieren.
- Fr 2.1.04      9 – 14
- Matlabcode implementieren, Debuging
- Sa 3.1.04      9 – 16
- Die Arrays überlaufen! Array Bereichsgrenzen prüfen.
  - Rückkopplung programmieren. Erst das Speichern des vorangegangenen Punktes ermöglicht es die Bewegung zu erfassen.
  - Die Funktion welche FAP Files erstellt programmieren.
- Mi 7.1.04      8-12 ; 13 – 18
- Debuging der Funktion „Mittelpunkt“ (Nasenlöcher finden).
  - Die Funktion welche FAP Files erstellt programmieren.

- Mi 14.1.04                      8 – 12 ; 13 – 17:30
- Bereichsgrenzen in der Funktion „Mittelpunkt“ abfragen und entsprechend setzen. Einfügen von „Border securities“.
  - Abmachungen bezüglich der FAP Schnittstelle zwischen den Funktionen welche Gesichtspunkte suchen und der Funktion welche daraus das FAP File erstellt.
- Mi 21.1.04                      8 – 12 ; 13 – 19
- Die Nasenerkennung funktioniert! Parameterwerte verbessern.
  - Ideen sammeln zur Mundwinklerkennung.
    - Fensterung
    - Koturverfolgung
    - Farverfolgung: Ist einfach, denn der Mund ist immer sehr dunkel bis man zur Lippe gelangt. Erreicht man die am weitesten rechts bzw. links liegenden Lippengrenzen befindet man sich auf dem Munwinkel.
- Mi 28.1.04                      8 – 12 ; 12:30 – 18
- Debuging der Mundwinkelfunktion. Der Punkt hängt sich an den Zähnen auf.
  - Programmierung einer einfachen Fensterungslösung. Die Vereinfachung geschieht mit Hilfe eines Farbpunktes im Gesicht. Dies funktionierte in Matlab sehr gut.
  - Erste Erfolge im erstellen von FAP-Files welche mit selbsterfassten Mundwinkel Koordinaten gefüttert wurden.
- Mi 4.2.04
- Schlussbericht schreiben
  - Fensterlösung verbessern.
- Mi 11.2.04
- Schlussbericht schreiben.

## Anhang

### Referenzen

#### Bücher

Titel: MPEG-4 Facial Animation  
Autoren: Igor S. Pandzic, Robert Forchheimer  
Verlag: John Wiley & Sons, LTD  
ISBN: 0-470-84465-5

Titel: Digital Image Processing  
Autoren: Rafael C. Gonzalez, Richard E. Woods  
Verlag: Addison-Wesley Publishing Company  
ISBN: 0-201-50803-6

#### Internet Links

URL: [www.medialab.ch](http://www.medialab.ch)  
Beschreibung: Praktikum 5 vom Fach Digitale Medien.

URL: [www.visagetechnologies.com/](http://www.visagetechnologies.com/)  
Beschreibung: Diese Firma bietet eine Reihe von MPEG-4 Gesichts-Animationsprodukten an. Igor Padzic ist dort zu erreichen.

URL: <http://amp.ece.cmu.edu/projects/FaceTracking/>  
Beschreibung: Auf dieser Seite ist ein Projekt zur Gesichtserkennung in Video zu finden.

#### Kontakte

Person: Guido M. Schuster, unser betreuender Professor.  
E-Mail: [gschuste@hsr.ch](mailto:gschuste@hsr.ch)

Person: Elisabetta Bevaqua, Kontaktperson für Greta.  
E-Mail: [elisabetta.bevacqua@libero.it](mailto:elisabetta.bevacqua@libero.it)

Person: Igor S. Pandzic, Kontaktperson bei Visagetechnologies.  
E-Mail: [igor@visagetechnologies.com](mailto:igor@visagetechnologies.com)

## MPEG-4 Facial Animation Parameters

Hier ist eine Liste der ersten 13 FAP's aus 68. Die ersten beiden FAP's sind highlevel FAP's welche wir nicht verwenden werden.

MPEG-4 FAP definitions, group assignments, and step sizes. FAP names may contain letters with the following meaning: l = left, r = right, t = top, b = bottom, i = inner, o = outer, m = middle. The quantizer step-size is a scaling factor for coding.

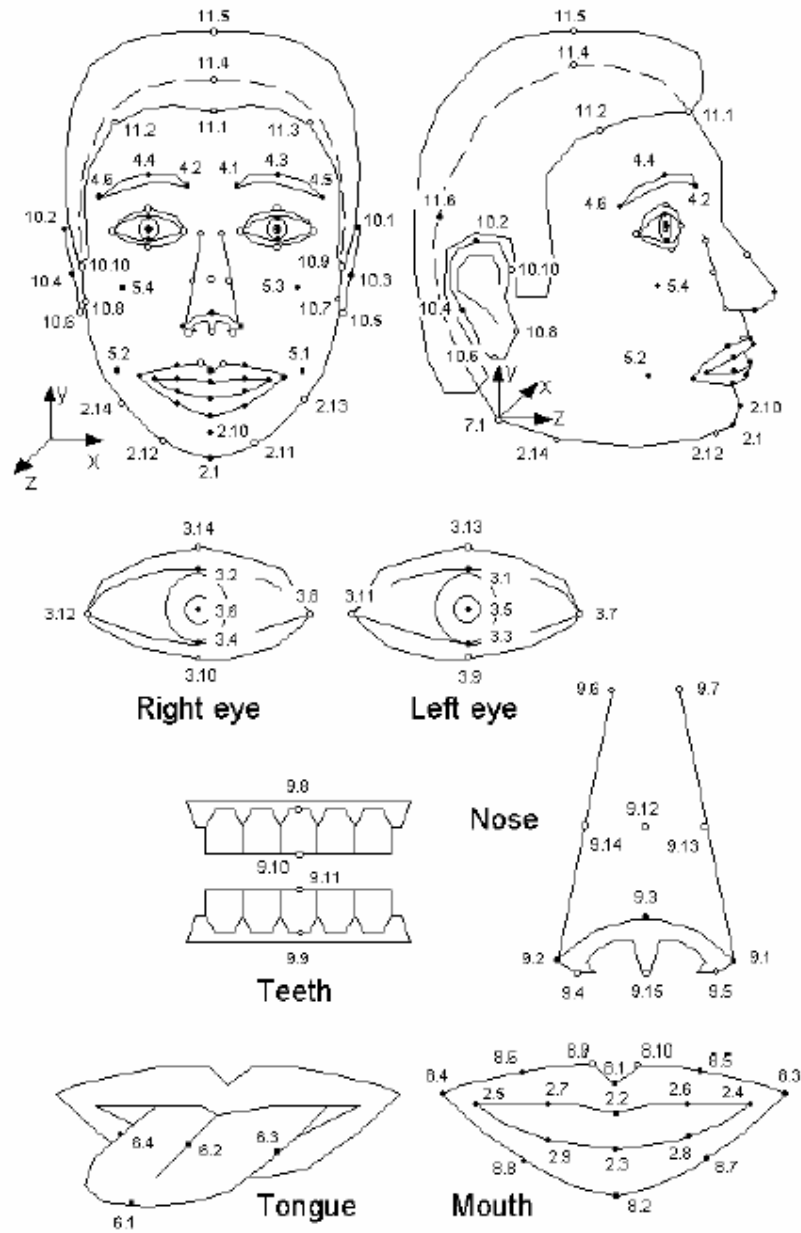
#	FAP name	FAP description	units	Position	Group	FDP subgroup num	Quant step size QP	Min/Max I-Frame quantized values	Min/Max P-Frame quantized values
3	open_jaw	Vertical jaw displacement (does not affect mouth opening)	MNS	down	2	1	4	+1080	+360
4	lower_t_midlip	Vertical top middle inner lip displacement	MNS	down	2	2	2	+600	+180
5	raise_b_midlip	Vertical bottom middle inner lip displacement	MNS	up	2	3	2	+1860	+600
6	stretch_l_cornerlip	Horizontal displacement of left inner lip corner	MW	left	2	4	2	+600	+180
7	stretch_r_cornerlip	Horizontal displacement of right inner lip corner	MW	right	2	5	2	+600	+180

8	lower_t_lip_lm	Vertical displacement of midpoint between left corner and	MNS	down	2	6	2	+600	+180
---	----------------	---	-----	------	---	---	---	------	------

visage|toolkit user manual 119 Introduction to MPEG-4 Face Animation Annex 1: MPEG-4 Facial Animation Parameters

9	lower_t_lip_rm	Vertical displacement of midpoint between right corner and middle of top inner lip	MNS	down	2	7	2	+600	+180
10	raise_b_lip_lm	Vertical displacement of midpoint between left corner and middle of bottom inner lip	MNS	up	2	8	2	+1860	+600
11	raise_b_lip_rm	Vertical displacement of midpoint between right corner and middle of bottom inner lip	MNS	up	2	9	2	+1860	+600
12	raise_l_cornerlip	Vertical displacement of left inner lip corner	MNS	up	2	4	2	+600	+180
13	raise_r_cornerlip	Vertical displacement of right inner lip corner	MNS	up	2	5	2	+600	+180

## Facial Feature Points (FP)



- \* Feature points affected by FAPs
- Other feature points

## Korrespondenz mit Frau Bevaqua

- dear Ms. Bevacqua  
I am a student by Mr. Schuster from the HSR. You have allready corresponded with him.  
My colleague and I would like to thank you for your support and allow ourselves a further question.  
We would like to use Greta in server mode.  
We know from your previous e-mail that we have TCP communication and the FAP file format is also clear.  
We were able to successfully play a FAP file on the player in normal mode.  
I tried to build up a TCP connection. Greta replied with 'accepted connection from:127.0.0.1'. So far every thing is fine. Unfortunately, the stream didn't have any effekt on Gretas face.  
This has thrown up a bunch of questions:  
How big is Gretas Buffer ? Are we sending too fast? What does Greta do, if we send slower than the frame rate? Does Greta give a response ?  
What can you tell us about the use of Greta in server mode? What important aspects are there to consider ?  
We are anxiously awaiting your enlightening answer :-).
- Dear Peter,  
we have several versions of Greta and, in order to answer to your questions properly, I need to know which one you are using and where you got it.  
Best wishes,  
Elisabetta Bevacqua
- hello Ms. Bevacqua  
Thank you for your prompt answer.  
We use the Greta version 1.3. Our Professor Dr. Schuster gave Greta to us.  
We wish you a nice weekend  
Peter Baumann & Tomas Kropaci
- Dear Peter,  
in server mode Greta uploads a file fap and a file wav. The server is a multithreaded one and can receive another couple of files while animating the first file fap. Therefore the stream is not actually in real time, Greta uploads a series of fapmask and fap values altogheter in a file fap and cannot receive a stream of single fapmasks and values.  
I thought you asked me if it is possible to animate a 3D head compliant with the MPEG-4 standard in real time, it is possible, indeed, but we didn't implement this kind of server for Greta, not as yet at least.  
If you send a file fap and a file wav to the server, Greta should works.  
I hope my answer has been enlightening! :)  
Can you tell me when did your professor get Greta? We have more than one version 1.3 of Greta and I don't undestand which

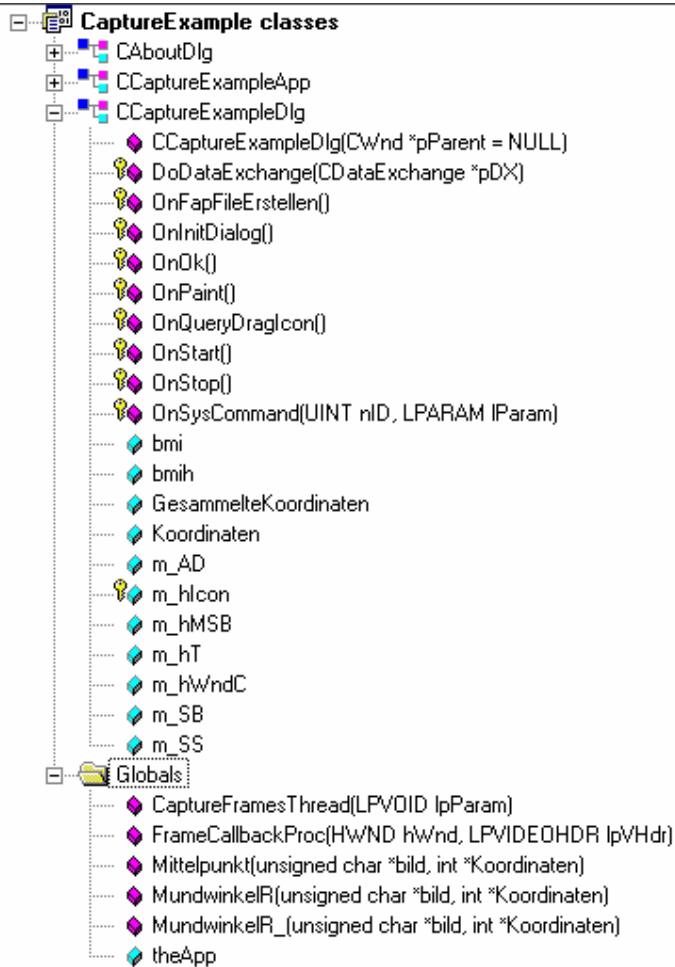
one of them you have. Has Greta a body in the final animation?  
Best wishes,  
Elisabetta

- dear Ms. Bevacqua  
Unfortunately, we were not able to find out the exact version of our Greta player. Dr.Schuster didn't get a reply from his colleague that supplied him with Greta. We added a screen shot of Greta, that will hopefully tell you enough to evaluate the exact version. Many of the previous questions were answered in you last e-mail. Since we do not send a stream, the timing aspect is solved. But new problems evolved. Is the order of the wave and FAP files relevant ? How does Great know which wave and FAP file belong together (same file name)? If we make the FAP files very short to simulate a stream would it work ? Does the wave file need to have the same time expanse or is it posible to send a empty wave file? We only need the animation without the sound. What does Greta do when goin over to the next fap file ? Does she reset all parameters or does she hold the last position?  
We are very grateful for your support.  
Kind regards Tomas Kropaci and Peter Baumann
- Dear Peter and Tomas,  
when you send files to Greta the order is very important: before FAP and then WAV; but the file wav is not absolutely necessary. Greta will work also without the audio file.  
At the end of an animation, Greta does not return to her neutral position, but she remains on the last set of faps animated.

Regards,  
Elsiabetta Bevacqua

## C++ Implementation

Dieser Dokumentation angehängt ist der C++ Code der bildverarbeitenden Funktionen. Die Funktionen wurden im Praktikum 5 eingefügt. Die Einfügestellen sind aus dem abgezeichneten Klassenbaum ersichtlich. Nicht zu vergessen sind die Konstruktoren am Anfang des CPP Files.



Eingefügt wurden: OnFapFileErstellen()

Mittelpunkt()

MundwinkelR()

MundwinkelR\_()

Sowie die Variablen: GesammelteKoordinaten[]

Koordinaten[]