

Georg-Simon-Ohm Hochschule Nürnberg

Fakultät Elektrotechnik Feinwerktechnik Informationstechnik efi

Projekt MatRigX

Prüfungsstudienarbeit von

Johannes Kiesel

2124263

B-ME 6

Sommersemester 2012

Kiesel Johannes

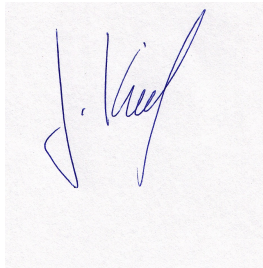
Ich bestätige, dass ich die Prüfungsstudienarbeit mit dem Titel:

„MatRigX“

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 5. Februar, 2012

Unterschrift:

A handwritten signature in blue ink, appearing to read 'J. Kiesel', is written on a light-colored, textured background.

Abstract

Das Projekt MatRigX entstand durch die Idee, einen Bullet-time Effekt zu kreieren. Dieses sehr bekannte Mittel der Medienproduktion dient dem Zweck Aufnahmen im Zeittotenraum zu erstellen wie man sie aus dem Blockbuster Matrix kennt. Hier wird im Prinzip eine Kamerafahrt im Zeittotenraum nachgestellt. Dies wird realisiert, indem eine Anzahl von Kameras (in unserem Fall 8) gleichzeitig auslöst, und somit das Objekt von verschiedenen Winkeln aufnimmt. Diese einzelnen Images müssen dann kalibriert und letztendlich zu einem Clip interpoliert werden.

Das Projekt konnte man in vier Teilgebiete unterscheiden.

Der erste Schritt war das Einlesen und Erlernen des Programmes After Effect und des PlugIn Twixtor, um zu sehen, wie die Kalibrierung der Images und die Interpolation von statuen geht. Man wollte am Ende des Projekts einen Vergleich ziehen können zwischen der professionellen Kalibrierung und der Kalibrierung durch die hauseigenen Software. Auch ein Endergebnis der beiden kalibrierten Clips interpoliert, soll verglichen werden.

Im Zweiten Teil wurde das Kamera Rig gebaut, um eine Hardwarekalibrierung der Kameras vorzunehmen. In unserem Fall war das Rig eine Holzkonstruktion, auf welcher die Kameras mit der Hilfe von Schrauben befestigt werden konnten. Das Rig ist faltbar und somit sehr mobil.

Der dritte Schritt befasst sich mit der Aufgabe, die Kameras gleichzeitig auszulösen und die Images einzeln auf einen Rechner zu laden. Hierbei wurde das CHDK Skript benutzt, da unsere Kameras alle von der Firma Canon waren und dieses Skript frei zur Verfügung steht. Um das Skript auf den Kameras auszulösen, musste man über den USB-Port eine Voltzahl von 3-5 Volt übergeben, was mit dem System Arduino realisiert wurde.

Der vierte Schritt beinhaltete das Codieren unseres hauseigenen Codes, welcher die Images der Kamera lädt, die Marker erkennt (Farberkennung), eine Matrix zur Lösung der Affinen/Perspektivischen Transformation berechnet, die Images mit dieser Matrix Affin/Perspektivisch transformiert und somit kalibriert. Letztendlich werden die Bilder interpoliert (linear / alpha blending), um einen Clip zu erstellen.

Das Resultat wird dann mit der Version aus der professionellen Software verglichen.

Projekt-Roadmap

After Effects

Tätigkeit	Dokument	Beteiligung
Video Rendering der Aufnahmen	.../Bilder/ .../Video/	Kiesel
Tracking und Kalibrierung der Frames	.../Video/	Kiesel
Interpolation der Aufnahme durch Twixtor	.../Video/	Kiesel

Auslösen der Kameras

Tätigkeit	Dokument	Beteiligung
CHDK Skript	.../CHDK/	Siemer/Voit
Auslösen mit Arduino	.../Arduino/	Siemer

Kalibrierung/Software

Tätigkeit	Dokument	Beteiligung
OpenCV/Openframeworks Recherche	.../OpenCV/	Kiesel/Neubauer/Siemer/Voit
Festlegen der Libraries etc.	.../Code/	Kiesel/Neubauer/Voit
Loading der Bilder in OpenCV	.../Code/	Kiesel/Voit
Farberkennung/Markererkennung	.../Code/	Kiesel/Voit/Neubauer
Morphologische Transformation um Marker zu isolieren	.../Code/	Voit
Auslesen der Markerpositionen	.../Code/	Kiesel/Voit
Berechnung der Affinen/Perspektiven Transformationsmatrix	.../Code/	Kiesel
Affine/Perspektivische Transformation der Images	.../Code/	Kiesel
Einbindung des Codes in OpenFrameworks	.../Code/	Neubauer
Interpolation der Einzelbilder zu einer Videoaufnahme in OF	.../Code/	Neubauer

Präsentation des Projekts

Zwischenpräsentation	.../dokumente/	Kiesel/Neubauer/Siemer/ Voit
Abschlusspräsentation	.../dokumente/	Kiesel/Neubauer/Siemer/ Voit
Gesamtdokumentation	.../dokumente/	Siemer
Wiki Dokumentation	.../dokumente/	Voit
Organisation - Projektleitung, Zeitplan	.../dokumente/	Voit
Protokollierung		Kiesel/Neubauer/Siemer/ Voit
Dokumentation des Codes	.../Code/	Kiesel/Neubauer/Voit

Inhaltsverzeichnis:

1. **P**rojekt-Auswahl
2. **V**erteilung der Aufgaben/Kompetenzen
3. **A**fter Effects
4. **T**wixtor
5. **K**alibrierungssoftware
 1. Recherche und Festlegung der Entwicklungsumgebung u. Sprache
 2. Laden der Images
 3. Farberkennung und Markererkennung
 4. Position der Marker auslesen
 5. Berechnung der Affinen/Perspektivischen Transformation's Matrix
6. **P**rotokollierung und Code Dokumentation
7. **I**st/Soll Vergleich
8. **R**eflexion
9. **L**iteratur/Quellen

1. Projekt Auswahl

Angetrieben von der Vorstellung einer perfekten Bullet-time Aufnahme, wie sie in professionellen Studios produziert wird, gingen wir das Projekt an. Die Thematik war umfangreich, und man hatte eine grosse Auswahl an Punkten, auf die man sich spezifizieren konnte. Ein Bullet-Time ist ein perfektes Projekt für einen Media Engineer, da es eine grosse Spanne in dem Technischen wie auch in dem Designbereich bietet, und man sich, je nachdem, auf das eine oder andere mehr einlassen kann.

Anfangs noch sehr motiviert und angetrieben von der Ausarbeitung des Designs des Bullet-time, dem Rig-bau und den technischen Anforderungen des Auslösens von mehreren Kameras, haben wir uns in die Ausarbeitung des Lastenhefts gestürzt. Da unsere Vorstellungen des Projekts nicht ganz im Einklang waren mit den unserer Klienten, musste man die einzelnen Spezifikationen ein wenig ändern und ausarbeiten.

Die Ausarbeitung des Projekt-Konzepts nahm daher ein wenig mehr Zeit in Anspruch wie Anfangs vermutet. Letztendlich hatte man sich darauf geeinigt, das Projekt in 4 verschiedene Gebiete zu unterteilen. Es wurde sich darauf verständigt, das man eine professionelle Software für die Ausarbeitung solcher Medienproduktionen erlernt, und eine ausgearbeitete Version als Vergleich zur hauseigenen Software nutzt.

Der Rig-bau nahm auch seine Zeit in Anspruch, da finanzielle Mittel wie auch technische Anforderungen definiert werden mussten, die in das Rig einfließen.

Ein Gebiet, was schon anfangs sehr vielversprechend aussah, war das Auslösen der Kameras per CHDK und Arduino. Da die Kameras nicht standardgemäss auf solche Aufgaben ausgelegt sind, musste man auch an dieser Stelle sehr kreativ werden.

Der grösste Teil unseres Projekts wurde nun die Codierung einer Kalibrierungssoftware, welche die einzelnen Images so transformiert, dass sie angepasst sind, und das Endprodukt nicht verwackelt erscheint.

2. Verteilung der Aufgaben und Kompetenzen

Die grobe Strukturierung der Aufgaben und deren Spezifikationen wurde von allen anfangs ausgearbeitet. Wir haben uns zusammen überlegt, was für technische requirements jeder Aufgabenbereich haben müsste, um es realisieren zu können. Danach haben wir eine Schätzung zu deren Arbeitsaufwand unternommen. Währenddessen wurde auch beschlossen was für Details man gerne einbauen möchte, und ob sie unter den gegebenen Umständen auch realisierbar seien.

Als dann eine grobe Struktur zu erkennen war und die Anforderungsabschätzung beendet war, wurden die Aufgaben verteilt, und je nachdem mit welchen Gebieten man sich befassen mochte, konnte man sich zu diesen äußern. Allen war relativ schnell klar, das die Erarbeitung einer Software zur Kalibrierung der Bilder die meiste Zeit in Anspruch nehmen würde, da man sich auch wieder in die Sprachen C und C++ einlesen musste, welche an diesem Punkt am relevantesten erschienen. Auch die Plattform OpenFrameworks war uns noch völlig unbekannt bis zu diesem Zeitpunkt.

Da ich persönlich grosses Interesse an der Animationssoftware After Effects hatte, und Sie auch in meinem späteren Berufsleben ein Teil meiner Arbeit ausmachen sollte, war es für mich schnell klar, das ich diesen Teil sehr gerne übernehmen möchte.

Christian Neubauer war sehr affin darauf, das Rig zu bauen, und hatte auch die benötigten Werkzeuge, wie auch den Platz, um das relativ weitreichende Rig zu konstruieren.

Christian Siemer sah seine Stärken in der Auslösung der Kameras mit CHDK und Arduino, was er gerne verfolgen wollte. Armin Voit unterstützte ihn noch mit der Einarbeitung und Findung des richtigen CHDK Protokolls für die automatische Auslösung der Kameras.

Als die Einarbeitung in After Effects und Twixtor beendet war, das Rig stand und ein passendes CHDK Protokoll gefunden war, machten Armin, Christian und ich uns daran, den Code für die Kalibrierung zu erstellen. Auch hier hat es eine Einarbeitungsphase gegeben, in der man sich erkundigen

musste, wie man so etwas realisieren könne. Dies fing damit an, welche Sprache, libraries und desgleichen man benötige. Es wurden auch diverse Artikel und Web Foren nach Informationen Anderer durchforstet, die sich auch schon mit der Thematik befassten. Nachdem man sich dann ein allgemeines Bild darüber machen konnte, was alles anfiel, und eine grobe Software Architektur hatte, fing man das eigentliche Codieren an. Auch während des coding wurde wieder aufgeteilt wer was programmieren würde.

Hier haben Armin und ich die Programmierung des Images Loaders und das Auslesen der Markerpositionen, wie auch die Farberkennung/Markererkennung übernommen.

Persönlich habe ich mich um die Berechnung der affinen/perspektivischen Transformationsmatrix gekümmert und der affinen/perspektivischen Transformierung der Bilder.

Armin hatte die morphologische Transformation übernommen, um die Marker im Bild von anderen Störpixel zu isolieren.

Christian Neubauer importierte das Programm in OpenFrameworks und interpolierte die angepassten Bilder linear, um eine Vergleichssequenz (Clip) zu erhalten, mit der wir schlussendlich die professionelle und hauseigene Software verglichen.

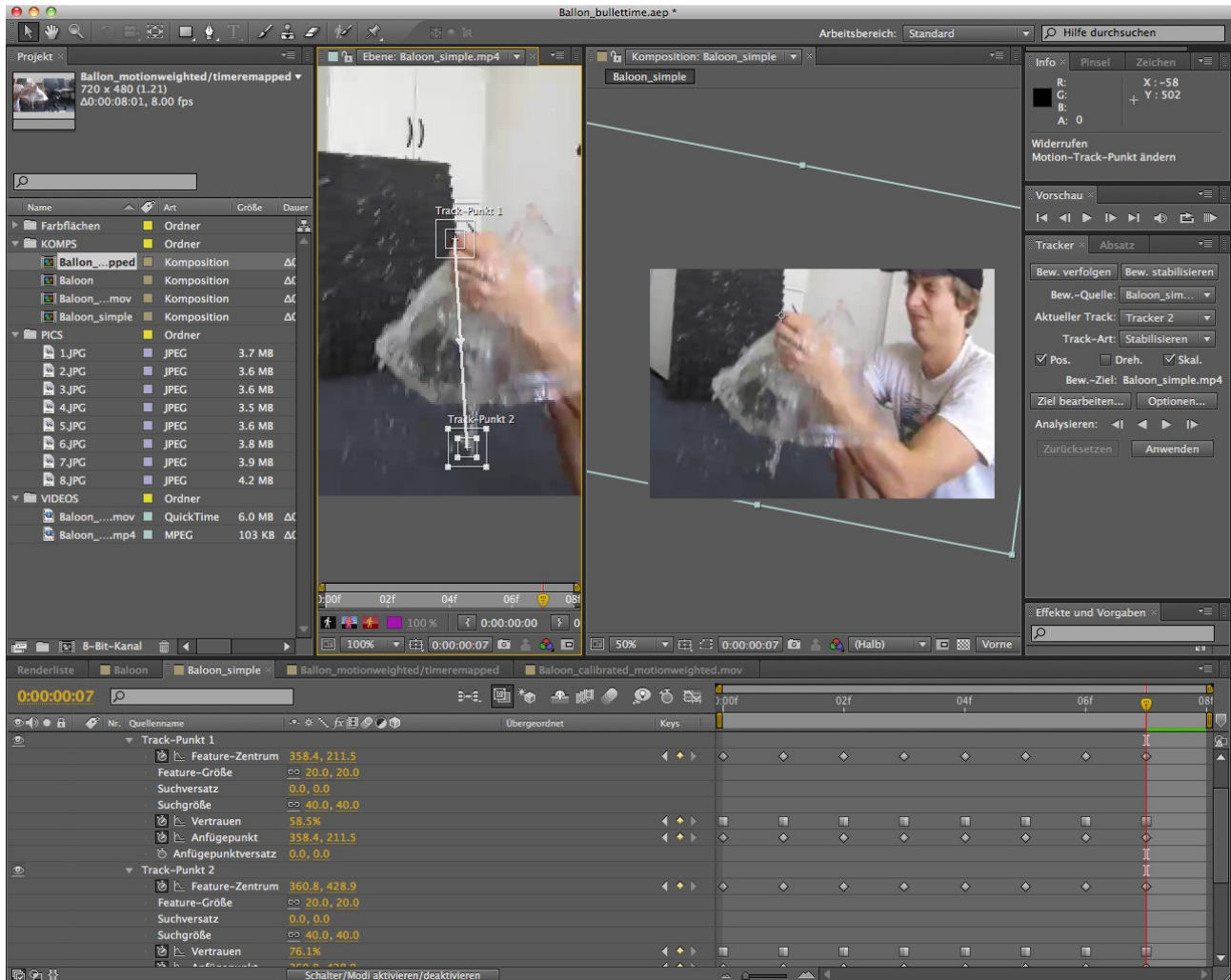
3. After Effects

After Effects ist eine von Adobe entwickelte Film und Media Production Software, die speziell für Filmschnitt und Animation oft verwendet wird. Wie auch Photoshop, Illustrator und alle anderen Adobe Programme ist sie sehr mächtig und überwältigend. Die genauen Prozesse rauszufinden, die man benötigt um spezielle Effekte oder gewünschte Aktionen auszuführen, bedingt viel Zeit und Geduld. Wenn man dann jedoch die Prinzipien verstanden hat, sind einem fast keine kreativen Grenzen gesetzt.

Speziell auch in unserem Fall war dieses Programm perfekt. Das eigentlich Programm After Effects übernimmt schon per Tracking Aktion das Kalibrieren der Frames. Hier wird ein Punkt von hohem Kontrast gewählt und dann angegeben, in was für einem Radius er im nächsten Frame nach diesem Punkt suchen soll. Die Frames werden dann aufeinander abgeglichen und angepasst, so dass das Verwackeln der Bilder minimal ist oder gänzlich wegfällt. Um die Affine Transformation 'Rotation' mit einzubeziehen, werden zwei Punkte im Bild definiert, so dass AE den Winkel der Verdrehung raus-rechnen kann. Wenn man schon im vornherein eine gute Hardwarekalibrierung vornimmt, erreicht man ein so gut wie perfektes Ergebnis.

Im Bild auf der nächsten Seite kann man diese Tracking Aktion sehr gut erkennen.

Auf diesem Bild sind zwei verschiedene Bühnen dargestellt. Auf der linken Seite, die Bühne mit der Tracker Position und auf der rechten Seite die Bühne, welche die Transformation der Frames auf dem Clip anzeigt.

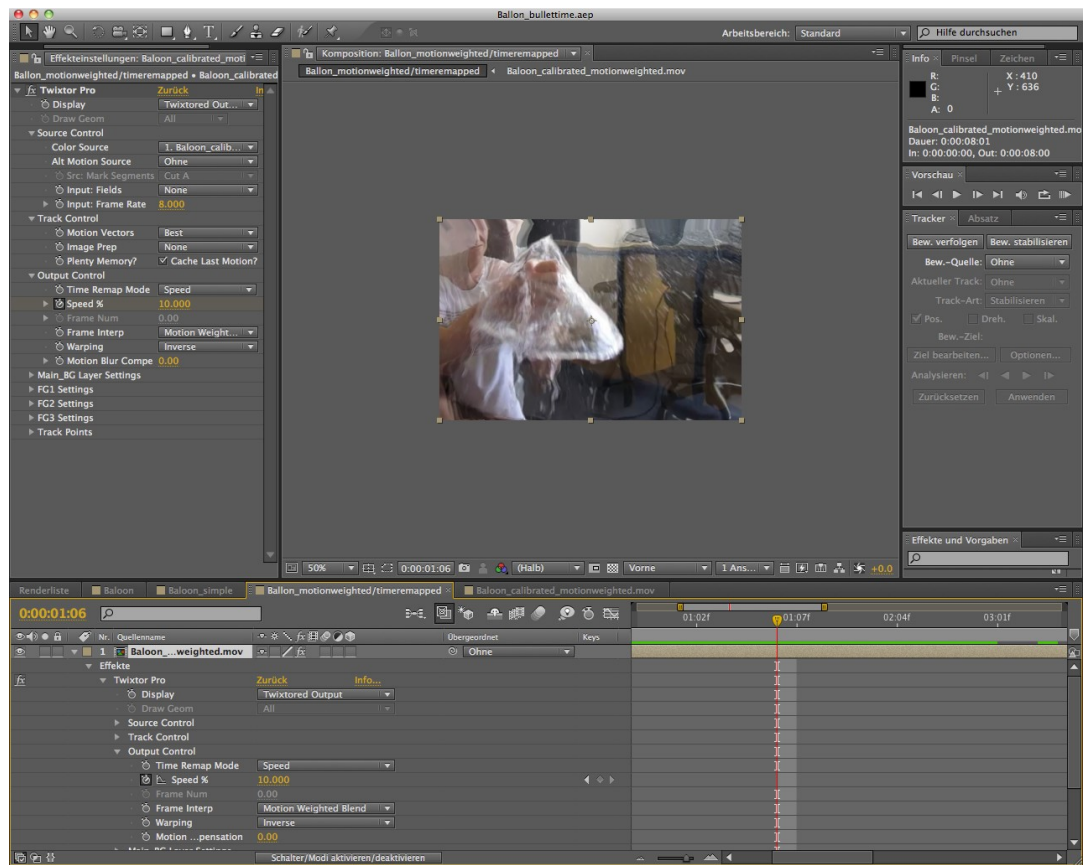


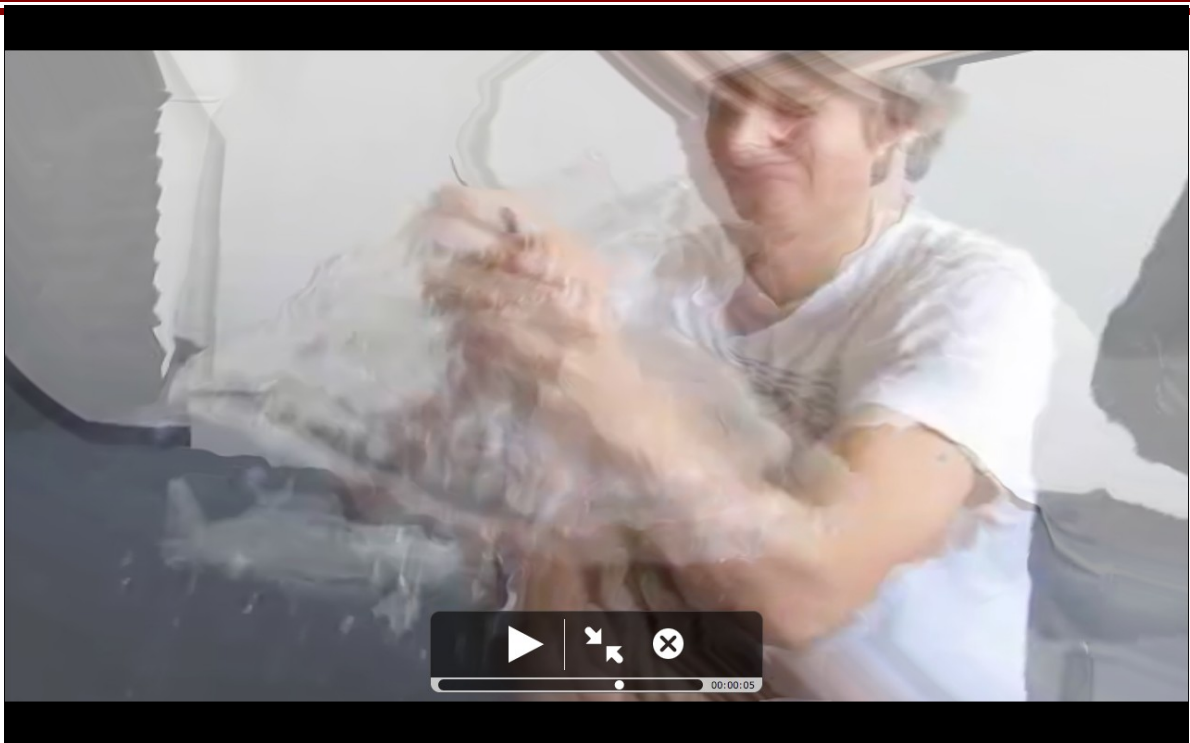
4. Twixtor

Twixtor ist ein von RE:Vision Effects, Inc entwickeltes PlugIn für After Effects und ein paar andere, ähnliche, Software Programme. Mit diesem unheimlich mächtigen PlugIn lässt sich die Motion Interpolation fast spielerisch auf ein Clip anwenden. Der herkömmliche Gebrauch dieses PlugIns liegt in dem Slowmotion Effekt, den diese Interpolation ermöglicht. Ausgelegt auf mindestens 59.96 fps, kann es bis auf mehrere hundert Frames interpolieren und einen sehr sauberen Slowmotion Effekt erzeugen. Slowmotion ist in unserem Fall unter einer anderen Anwendung bekannt. Normalerweise verbindet man Slowmotion mit einem Video Clip, der dann zeitlich langsamer abläuft. Aber wie es das Wort schon verrät, muss Zeit in diesem Fall keine Rolle spielen. Motion ist hier das Stichwort. Das einzige was Slowmotion beinhaltet ist das Verlangsamen der Bewegung, ob im Zeittotenraum oder nicht. Diese motion-weighted Interpolation kann man sich natürlich auch zu nutzen machen, wenn man eine Kamerafahrt im Zeittotenraum nachahmen möchte. Hierbei werden zwischen den schon existierenden Frames weitere 'interpolierte' Frames hineingerechnet und man kann sich mal schneller und mal langsamer um das stillstehende Objekt drehen, soweit ein Kreisförmiges Rig gewählt wurde.

Wie auch After Effects, hat Twixtor hunderte von fein Einstellungen, deren Beschreibung alleine den Umfang eines Projekts ergeben würde. Die einfachste Variante, die auch wir letztendlich gewählt hatten, war das Reduzieren der Speedramp unter motion-weighted Blend. Hier kann man den Clip stark verlangsamen, und das motion-weighted Blend übernimmt die Interpolation. Nach der Interpolation muss man den Clip timeremappen, um die volle Länge des neuen Clips darstellen zu können. Da wir jedoch nur mit 8 fps arbeiten konnten, war das Resultat relativ enttäuschend da auch Twixtor mit nur so wenig fps keine ordentliche Interpolation zu Stande bekommt. In den unteren

Bildern kann man zunächst das PlugIn sehen und dann einen Screenshot von dem gerenderten Video.





5. Kalibrierungssoftware

1. Recherche und Festlegung der Entwicklungsumgebung u. Sprache:

Zu Anfang des Projekts hatte man sich verschiedenste Optionen offen gehalten, welche Entwicklungsumgebung und Sprache man in diesem Projekt verwenden wolle. Wir einigten uns darauf in X-Code zu programmieren und die Sprachen C/C++ zu verwenden.

Die Libraries von OpenCV machten eigentlich den Grossteil des Projekts möglich, womit zum einen die Markererkennung programmiert, und zum anderen die Affine/Perspektivische Transformationmatrix berechnet werden konnte. Auch die Affine/Perspektivische Transformation der Images war in OpenCV möglich. Zum ende des Projekts hatte man sich gemeinsam nochmals darauf geeinigt den Code von OpenCV in die OpenFrameworks Umgebung zu integrieren, da dort die Interpolation eines Videos mit den berechneten Transformations Matrizen einfacher zu realisieren wäre und die Arbeit weiterer Generationen erleichtert werden würde. Nach der Bedienung und dem Einarbeiten in After Effects, war das Laden der Images, Die Markererkennung

und das berechnen der Transformationsmatrizen meine weiteren Aufgaben.

1.Laden der Images

Unsere Software wurde so programmiert das sie aus einem Folder auf dem Rechner die Bilder nacheinander in eine Matrize lädt, auf welche die weiteren Funktionen unseres Programmes dann Zugriff haben.

```
//Load destination Image and define srcimage
IplImage* desimage = cvLoadImage("./MarkerPics/des.jpg");

IplImage* srcimage;

// Couldn't get a image? Throw an error and quit
if(!desimage)
{
    printf("Could not initialize desimage...\n");
    return -1;
}
```

In diesem Schritt werden zwei Zeiger vom Typ IplImage definiert wobei nur desimage auf ein unserer Images zeigt, welches unsere Referenz Bild ist. Die anderen folgenden Bilder werden in einem späteren Schritt von srcimage aus dem Array Images[] gelesen. Die if Anweisung ist hauptsächlich zum debuggen gedacht, in dem sie im log die Fehlermeldung „Could not initialize desimage“ ausgibt. Falls dies auftritt kann man sich relativ sicher sein das etwas an dem Pfad oder file des Images nicht stimmt.

```
for (int i=1, j=1; i<=counter; i++, j++) {

    char buffer_mrk[33];

    sprintf(buffer_mrk,
"/Users/arminvoit/Documents/openframeworks/apps/Stand_17.07/Matrigx/bin/MarkerPics/src
%d.jpg", j);

    markerImages[i-1] = cvLoadImage(buffer_mrk);

}
```

```
for (int i=1, j=1; i<=counter; i++, j++){  
  
    char buffer_src[33];  
  
    sprintf(buffer_src,  
"/Users/arminvoit/Documents/openframeworks/apps/Stand_17.07/Matrigx/bin/Pics/%ds.jpg", j);  
  
    images[i-1]= cvLoadImage(buffer_src);  
  
}
```

Hier wird ein Array nacheinander mit den den Markerbildern geladen, und ein weiteres Array mit den Bildern des eigentlichen Objekts. Hier ist die variable counter genau die Anzahl der Bilder welche bearbeitet werden müssen. Durch sprintf wird in jeder Iteration der Pfad zum neuen Bild in buffer gespeichert, welcher dann von cvLoadImages() verwendet wird.

Spaeter im Code wird dann mit

```
//Load Images for mapping  
srcimage = markerImages[i];  
  
// Couldn't get a image? Throw an error and quit  
if(!srcimage)  
{  
    printf("Could not initialize srcimage...\n");  
    return -1;  
}
```

Die einzelnen Bilder im array markerImages[i] geladen und im verlauf des Algorithmus bearbeitet.

1. Farberkennung und Markererkennung

Die Farberkennung wird in verschiedenen Schritten durchgeführt. Das zu bearbeitende Image wird an eine Funktionen übergeben, welche zuerst das Image in ein HSV wandelt und in einem neuen (generierten) Image abspeichert. Danach wird auf die Farbe im HSV Raum gefiltert und in ein binaeres Schwarz/Weis Bild gewandelt. Hier wird im Prinzip das HSV Bild einfach in ein generiertes Image mit nur einem Farbkanal gespeichert. Fuer 4 verschiedene Marker/Farben existieren 4 verschiedene Funktionen die jeweils das Image auf eine Farbe filtern und auswerten. Im unteren Code wird

beispielhaft die Funktion zum filtern für den gelben Marker veranschaulicht.

```
IpImage* GetThresholdedImageYellow(IpImage* img)
IpImage* Calibration::GetThresholdedImageYellow(IpImage* img)
{
    //-----
    // Convert the image into an HSV image
    //-----

    IpImage* imgHSV = cvCreateImage(cvGetSize(img), 8, 3);
    cvCvtColor(img, imgHSV, CV_BGR2HSV);

    IpImage* imgThreshed = cvCreateImage(cvGetSize(img), 8, 1);
    cvInRangeS(imgHSV, cvScalar(0, 20, 230), cvScalar(35, 380, 300), imgThreshed);

    //-----
    //erode and dilate imgThreshed
    //-----

    cvErode(imgThreshed, imgThreshed, NULL, 2);
    cvDilate(imgThreshed, imgThreshed, NULL, 1);

    cvReleaseImage(&imgHSV);
    return imgThreshed;
}
```

Letztendlich übergibt die Funktion ein Schwarz/Weiss Image das nur den Marker (in diesem Fall gelb) anzeigt.

Dies ist wichtig, da im nächsten Schritt mit Momenten die genaue Position des Markers im Bild bestimmt wird. Hierbei wird das Bild abgetastet und der Übergang von Schwarz auf Weiss und von Weiss auf Schwarz ausgewertet, gewichtet, und somit der Mittelpunkt des Markers bestimmt.

```
//-----
// Holds the thresholded images (Marker = white, rest = black)
//-----

imgYellowThreshDes = GetThresholdedImageYellow(desimage);
imgBlueThreshDes = GetThresholdedImageBlue(desimage);
imgRedThreshDes = GetThresholdedImageRed(desimage);
imgGreenThreshDes = GetThresholdedImageGreen(desimage);
```

```
imgYellowThreshSrc = GetThresholdedImageYellow(srcimage);
imgBlueThreshSrc = GetThresholdedImageBlue(srcimage);
imgRedThreshSrc = GetThresholdedImageRed(srcimage);
imgGreenThreshSrc = GetThresholdedImageGreen(srcimage);

cvMoments(imgYellowThreshDes, momentsyellowdes, 1);
cvMoments(imgBlueThreshDes, momentsbluedes, 1);
cvMoments(imgRedThreshDes, momentsreddes, 1);
cvMoments(imgGreenThreshDes, momentsgreendes, 1);

cvMoments(imgYellowThreshSrc, momentsyellowsrc, 1);
cvMoments(imgBlueThreshSrc, momentsbluesrc, 1);
cvMoments(imgRedThreshSrc, momentsredsrc, 1);
cvMoments(imgGreenThreshSrc, momentsgreensrc, 1);

//-----
// The actual moment values
//-----

moment10yellowdes = cvGetSpatialMoment(momentsyellowdes, 1, 0);
moment01yellowdes = cvGetSpatialMoment(momentsyellowdes, 0, 1);
areayellowdes = cvGetCentralMoment(momentsyellowdes, 0, 0);

moment10bluedes = cvGetSpatialMoment(momentsbluedes, 1, 0);
moment01bluedes = cvGetSpatialMoment(momentsbluedes, 0, 1);
areabluedes = cvGetCentralMoment(momentsbluedes, 0, 0);

moment10reddes = cvGetSpatialMoment(momentsreddes, 1, 0);
moment01reddes = cvGetSpatialMoment(momentsreddes, 0, 1);
areareddes = cvGetCentralMoment(momentsreddes, 0, 0);

moment10greendes = cvGetSpatialMoment(momentsgreendes, 1, 0);
moment01greendes = cvGetSpatialMoment(momentsgreendes, 0, 1);
areagreendes = cvGetCentralMoment(momentsgreendes, 0, 0);

moment10yellowsrc = cvGetSpatialMoment(momentsyellowsrc, 1, 0);
moment01yellowsrc = cvGetSpatialMoment(momentsyellowsrc, 0, 1);
areayellowsrc = cvGetCentralMoment(momentsyellowsrc, 0, 0);

moment10bluesrc = cvGetSpatialMoment(momentsbluesrc, 1, 0);
moment01bluesrc = cvGetSpatialMoment(momentsbluesrc, 0, 1);
areabluesrc = cvGetCentralMoment(momentsbluesrc, 0, 0);

moment10redsrc = cvGetSpatialMoment(momentsredsrc, 1, 0);
moment01redsrc = cvGetSpatialMoment(momentsredsrc, 0, 1);
arearedsrc = cvGetCentralMoment(momentsredsrc, 0, 0);
```

```
moment10greensrc = cvGetSpatialMoment(momentsgreensrc, 1, 0);
moment01greensrc = cvGetSpatialMoment(momentsgreensrc, 0, 1);
areagreensrc = cvGetCentralMoment(momentsgreensrc, 0, 0);
```

```
//-----
// Holding the last and current marker positions and placing them into a CvPoint Vector
//-----
```

```
posXyellowDes = moment10yellowdes/areayellowdes;
posYyellowDes = moment01yellowdes/areayellowdes;
```

```
posXblueDes = moment10bluedes/areabluedes;
posYblueDes = moment01bluedes/areabluedes;
```

```
posXredDes = moment10reddes/areareddes;
posYredDes = moment01reddes/areareddes;
```

```
posXgreenDes = moment10greendes/areagreendes;
posYgreenDes = moment01greendes/areagreendes;
```

```
posXyellowSrc = moment10yellowsrc/areayellowsrc;
posYyellowSrc = moment01yellowsrc/areayellowsrc;
```

```
posXblueSrc = moment10bluesrc/areabluesrc;
posYblueSrc = moment01bluesrc/areabluesrc;
```

```
posXredSrc = moment10redsrc/arearedsrc;
posYredSrc = moment01redsrc/arearedsrc;
```

```
posXgreenSrc = moment10greensrc/areagreensrc;
posYgreenSrc = moment01greensrc/areagreensrc;
```

```
//-----
// Des-Points with marker-values
// Uncomment for using markers
//-----
```

```
setDesPoints(posXyellowDes, posYyellowDes, posXblueDes, posYblueDes, posXredDes,
posYredDes, posXgreenDes, posYgreenDes);
```

```
//-----
// Src-Points with marker-values
// Uncomment for using markers
//-----
```

```
setSrcPoints(posXyellowSrc, posYyellowSrc, posXblueSrc, posYblueSrc, posXredSrc,
posYredSrc, posXgreenSrc, posYgreenSrc);
```

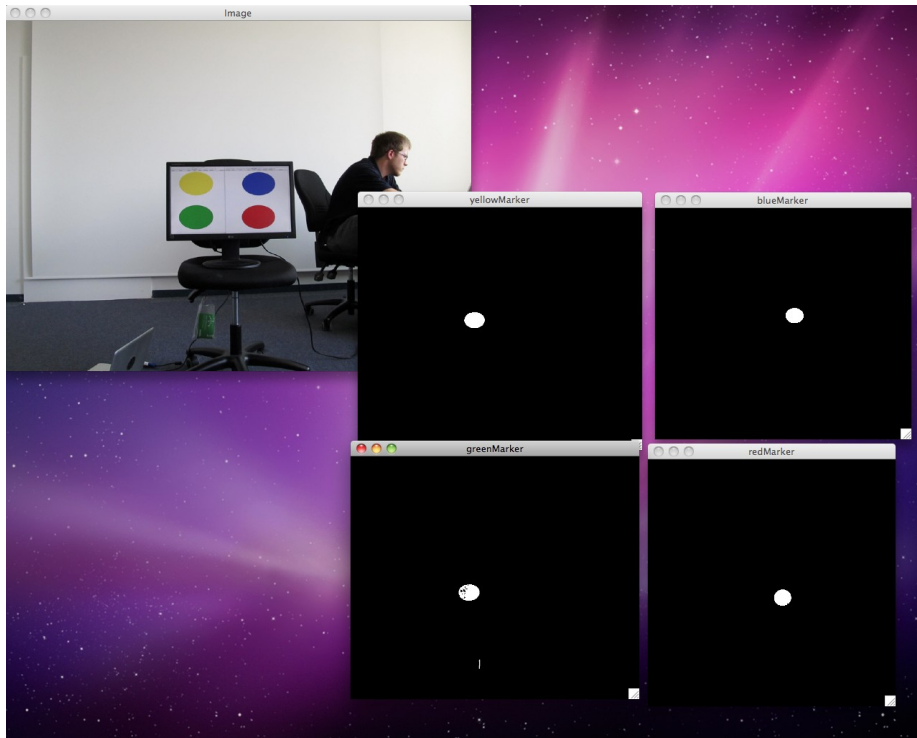
Im obigen Code wird nacheinander das Bild an die einzelnen Farberkennungsfunktionen übergeben, die nach Rot, Gelb, Blau und Grün filtern und ein Binärbild zurückliefern. Danach werden die Momente für die einzelnen Binärbilder berechnet und zwar für das desimage und das jeweilige srcimage. Aus diesen Momenten ergeben sich die X und Y Positionen des Markerschwerpunkts die dann in float variablen übertragen werden. Die einzelnen X und Y Positionen werden dann sequenziell in Arrays des typ Points übertragen, was die Funktionen setDesPoints und setSrcPoints übernimmt.

```
void Calibration::setSrcPoints(float posXyellowSrc, float posYyellowSrc, float posXblueSrc, float
posYblueSrc, float posXredSrc, float posYredSrc, float posXgreenSrc, float posYgreenSrc){

    srcPoints[0].x = posXyellowSrc;
    srcPoints[0].y = posYyellowSrc;
    srcPoints[1].x = posXblueSrc;
    srcPoints[1].y = posYblueSrc;
    srcPoints[2].x = posXredSrc;
    srcPoints[2].y = posYredSrc;
    srcPoints[3].x = posXgreenSrc;
    srcPoints[3].y = posYgreenSrc;

}
```

Durch diesen Schritt wurden die Marker erkannt, deren Position berechnet und die Information für den nächsten Schritt, die Berechnung der Transformationsmatrizen, aufbereitet. Im unteren Bild ein Screenshot der die Markererkennung im Binärbild mit vergleich zum tatsächlichen Image darstellt.



1. Berechnung der Affinen/Perspektivischen Transformation's Matrix

Für die Berechnung der Transformationsmatrizen der Bilder sind im allgemeinen nur zwei Schritte notwendig. Zum einen, muss eine passende Matrize erstellt werden, welche die berechneten Transformationswerte enthalten kann, und im zweiten Schritt, muss diese der Funktion zum berechnen der Transformationsmatrize übergeben werden.

```
//map_matrix1 = cvCreateMat(3,3, CV_32FC1); //<-- USE FOR PERSPECTIVE  
TRANSFORM
```

```
map_matrix1 = cvCreateMat(2,3, CV_32FC1);
```

Mann benötigt für eine affine Transformation eine 3x3 Matrix (Rotation, Skalierung, Scherung und Translation), jedoch wird bei OpenCV eine 2x3 Matrix definiert, da die letzte Zeile (001) im Prinzip 'gedacht' wird. Das gleiche

Prinzip erhält man für die perspektivische Matrix. Hier eigentlich eine 4x4 Matrix, jedoch wird die erweiterte Ebene im Prozess der Funktion `cvGetAffineTransform()/cvGetPerspectiveTransform()` dazugerechnet.

Nach diesem Schritt kann man durch eine fertige Funktion der OpenCV library, durch Übergabe der Point Arrays und der richtigen Matrize, die Transformationsmatrize berechnen.

```
//cvGetPerspectiveTransform(srcPoints, desPoints, map_matrix1); //<-- USE FOR PERSPECTIVE  
TRANSFORM  
cvGetAffineTransform(srcPoints, desPoints, map_matrix1);
```

6. Protokollierung und Code Dokumentation

Während unserem Projekt wurde nach jedem offiziellen Projekttreffen ein Protokoll erstellt, welches hauptsächlich zur Übersicht der Projektverlaufs diente. Hier wurden Fortschritte, setbacks, Änderungen und Planungen festgehalten. Ein jeweiliges Protokoll wurde abwechselnd von den Projektmitgliedern geführt und auf die von uns benutzte Dropbox, gespeichert. Auf dieser Dropbox hatten alle Mitglieder und Projektleiter/Auftraggeber zu jederzeit Zugang. Hier wurden alle Materialien, Dokumente und Code abgelegt. Der Code wurde jedoch zusätzlich in ein Github depository geladen, wo auch jedermann Zugriff hatte und alle simultan daran arbeiten konnten. Letzteres, das github, wurde leider erst im späteren Verlauf des Projekts angelegt. Ein Blog wurde auch erstellt, um eine öffentliche Plattform mit news zum Projekt zu

pflegen.

Der Code wurde säuberlichst dokumentiert, so das weitere Generationen die an dem Projekt weiterarbeiten möchten, sich möglichst einfach in den Code selbst einlesen können. Es wurde jeder einzelne Schritt beschrieben, und die einzelnen Funktionen erklärt. Auch die Variablen, Arrays, Matrizen wurden beschriftet, so das man ihre Zugehörigkeit im Code, schon Anfangs erkennen kann. Eine schnelle Einarbeit in das Projekt sollte somit geschaffen sein.

7. Ist/Soll Vergleich

Die von uns Anfangs gesetzten Ziele haben wir erreicht und einen lauffähigen Prototypen entwickelt. Es ist uns gelungen ein Rig zu erstellen das die für uns gestellten 8 Kameras unterbrachte, ein Konzept für die Hardwarekalibrierung zu entwickeln, eine automatische synchrone Auslösung aller Kameras, und die Messung der zeitlichen Genauigkeit auszuarbeiten und mit der Plattform OpenFrameworks, eine Feinkalibrierung zu entwickeln.

Diese Feinkalibrierung benutzt Farberkennung/Markererkennung zur Berechnung der Marker im Bild, wertet deren Position aus und berechnet eine Transformationsmatrix welche auf die Objektbilder angewendet wird, und diese per alpha blending anzeigt. Im Vergleich zu unserer getesteten professionellen Software, Adobe After Effects, ist sie noch sehr rudimentär, aber das war zu erwarten und dient dazu weitere Generationen zu motivieren dieses Ziel einer Vergleichbaren Software zu realisieren. Die in C++ programmierte Software, ist durchgehen dokumentiert und kommentiert, was eine leichte Einarbeit für weitere Projektgruppen erleichtert. Da auf der Plattform OpenFrameworks programmiert wurde, und für diese eine Vielzahl von libraries der Bildverarbeitung zur Verfügung stehen ist das Ausarbeiten von anderen Lösungsansätzen durchaus denkbar.

Wir hatten eine sehr gute qualitative Beurteilung der Software durchführen können, in dem wir ein kalibriertes Video mit beiden, der hauseigen wie auch professionellen Software, erstellten. Das mit After Effects

erzeugte Video benötigte eine erhebliche geringere Bearbeitungszeit und konnte gleich mit den Images des Objektes kalibrieren während unsere hauseigene Software zwei verschiedene Sets an Images brauchte, ein Set mit Marker und ein Set ohne. In der professionellen Software After Effects ist es zusätzlich möglich einzelne Tracking Positionen im Nachhinein per Hand zu definieren oder auszubessern.

Leider war es uns nicht mehr möglich eine gute Quantitative Messung der Pixelverschiebungen von Frame zu Frame zu berechnen.

Das erzielte Ergebnis ist wie erwartet noch ausbaubar, aber das Forschungsprojekt hat definitiv den Anstoss für mehr ermöglicht. Die von uns im Lastenheft definierten Ziele haben wir jedoch erreicht, und das Projekt somit mit Erfolg abgeschlossen.

Damit ist ein Grundbaustein gesetzt für weitere Generationen, die auf unserem Ergebnis aufbauen können und mit den bisher erzielten Erfolgen noch weiter in die Tiefe, dieses relativ komplexen Themas eindringen mögen. Die ganze Gruppe hat viel gelernt über die Do's and Dents einer Projektarbeit mit diesem Umfang, und können dies als wertvolle Erfahrung in ihrem professionellen Werdegang einfließen lassen.

8. Reflexion

Auch nachdem die Spezifikationsphase des Projekts relativ lange andauerte, und Unklarheiten auf Seiten der Projektmitglieder und Auftraggeber immer wieder auftraten, war das Projekt jedoch ein Erfolg. Eine hauseigene Farberkennungssoftware, wie auch Kalibrierungssoftware, für eine Image Sequenz im Zeittotenraum wurde erstellt und war ausführbar. Das letztendlich Resultat war auf Grund der wenigen Kameras ein wenig enttäuschend, aber der Ausblick für weitere Projekte in diese Richtung ist vielversprechend. Die Protokollierung und Dokumentation des Projekts, war im großen und ganzen gut. Durch den Zugang auf die Dropbox hatte jeder zu jederzeit Zugriff auf alle Projekt relevanten Dokumente. Die Kommunikation dieser Inhalte

hätte noch ein wenig ausgebaut werden können, um Missverständnisse zwischen den Projektmitgliedern und den Kunden zu vermeiden, und ein besseres Projektklima zu gestalten.

Weitere Generationen die mit diesem Projekt als Vorlage weiterarbeiten wollen, habe eine breite Dokumentation der Inhalte und eine detailreiche Code Dokumentation zur Verfügung.

Projektziele wie das testen mit mehreren Kameras 60+, motion-weighted interpolation mit hauseigener Software, wie auch das generieren eines UI für die Software, wären in meiner Sicht, interessant für die nächste Generation. Im Grossen und ganze bin ich zufrieden mit dem was wir als Projektgruppe erzielten, und wie die Gruppe miteinander arbeitete und umging.

9. Literatur/Quellen

- (1) <http://opencv.willowgarage.com/wiki/>
- (2) <http://www.shervinemami.info/colorConversion.html>
- (3) http://wiki.openframeworks.cc/index.php?title=Main_Page
- (4) <http://www.openframeworks.cc/>
- (5) <http://www.creativecow.net/>
- (6) <http://www.videocopilot.net/>
- (7) Gary Bradski & Adrian Kaehler 2008, „Learning OpenCV – Computer Vision with the OpenCV library“, O'REILLY