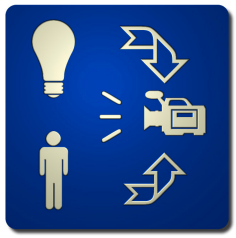
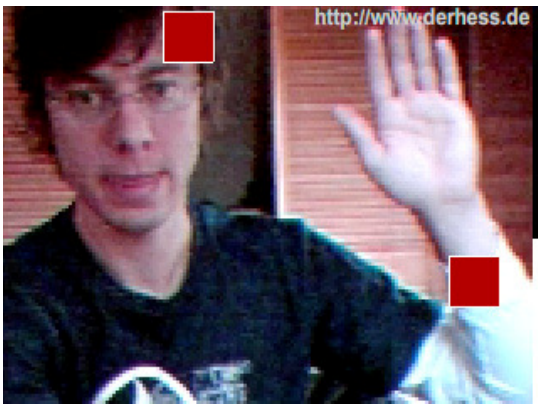
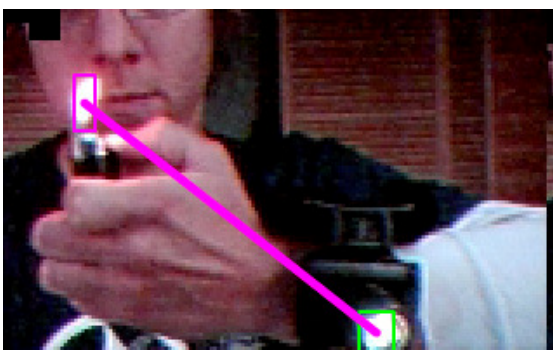
INTERACTIVE WEBCAM PACKAGE 1.2 DOKUMENTATION



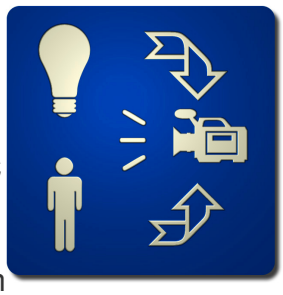
Seit 2006/2007 spiele ich (Florian Weil / der hess) hin und wieder ein bissl mit bildbasierter Interaktion herum. Aus dieser Spielerei heraus ist im Sommer 2007 das hier vorliegende Interactive Webcam Package für (Flash) ActionScript 2 entstanden. Jetzt wo sich ActionScript 3 immer mehr etabliert, wurde auch das Interactive Webcam Package auf ActionScript 3 umgestellt bzw. portiert. Bei dieser Portierung habe ich versucht so gut wie möglich die bestehende Struktur aufrecht zu erhalten. Jedoch unterscheidet sich die ActionScript 2 API sehr von der ActionScript 3 API, so dass sich einige Änderungen beim Eventhandling nicht vermeiden ließen.

Allgemeines zum Interactive Webcam Package:
Das Interactive Webcam Package soll Flash Programmierer bei ihrer Entwicklung von bildbasierter Interaktion unterstützen und so den dazu gehörigen Aufwand hoffentlich auf ein erträgliches Maß senken.
Momentan sind eine devicelose und eine deviceabhängige Interaktion möglich. Bei der devicelosen Interaktion interagiert man mit seinen Körper. Technisch wurde dieses Verfahren mittels MotionDetection und Differenzkeys umgesetzt.
Die deviceabhängige Interaktion erfolgt mittels einer Lichtquelle (z.B. Taschenlampe) Die Lichtquelle wird im Bild getrackt und als Ergebnis erhält man ein Rechteck mit Positions- und Größenangabe der Lichtquelle zurück. Dieses Tracking funktioniert auch mit mehreren Lichtquellen, so dass man z.B. ein Stab visualisieren kann mit 2 Lichtquellen.





Die CamButtonManager – Klasse



Die CamButtonManager Klasse erstellt und verwaltet Buttons bzw. Hotpoints im Kamerabild. Der CamButtonManger überprüft ob der Bereich im Bild aktiv ist oder nicht, und führt dem entsprechend ein Ereignis des CamButtons aus. Die CamButtonManager Klasse arbeitet in zwei verschiedenen Modi.Der CamButtonManager.STATIC Modus arbeitet mit einem Differenzkey, der Modus CamButtonManager.DYNAMIC dagegen arbeitet mit einem Motion Detection Verfahren. Beide Modi haben ihre Vor- und Nachteile, die in der unteren Tabelle aufgelistet sind.

CamButtonManager.STATIC	***CamButtonManager.DYNAMIC***
	
Vorteile: - klare Lokalisierung der Peson im Bild - Gute Basis für Kollisionsanwendungen - Ereignis ON_HAND_OVER funktioniert zuverlässiger	**Vorteile:** - Sehr störungsrobust gegenüber Belichtungsänderung (Blendenautomatik) - Nur bewegungsreiche Bereiche im Bild werden angezeigt
Nachteile: - Sehr anfällig für Licht- und Schärfeänderungen im Bild - Funktioniert bei Kameras mit Automatik nur sehr schlecht - Schlechte Handhabung für Endnutzer, weil ein Snapshot (Bild) ohne Person erstellt werden muss	**Nachteile:** - Perfomancehungriger gegenüber des STATIC-Modus - ON_HAND_OVER und ON_HAND_OFF Ereignis nur bedingt verwendbar, weil wenn der Körper über den Buttonbereich stehen bleibt, verschwindet die Aktivität.

Variablen und Methoden der Klasse CamButtonManager

. static CamButtonManager.STATIC	Statische Variable für den Static-Modus
. static CamButtonManager.DYNAMIC	Statische Variable für Dynamic Modus
.CamButtonManager(webcamSignal:Sprite, camMode:int)	Konstruktor
.createCamButton(xpos:int,ypos:int, width:int,height:int):CamButton	Erzeugt einen CamButton
.addCamButton(button:CamButton):Boolean	Fügt einen schon bestehenden Button der Liste wieder hinzu
.removeCamButton(button:CamButton):Boolean	entfernt einen Button aus der Liste
.doSnapshot():void	Erstellt ein Snapshotbild für den Static-Modus
.startAnalyze():void	Startet den Analyse Modus der Buttons
.getBitmapData():BitmapData	Gibt das Analysebild zurück
.getMode():int	Gibt den aktuellen Modus des CamButtonManagers zurück
.setMode(modus:int)	Mithilfe der statischen Variablen STATIC und DYNAMIC kann man hier den Analysemodus zur Laufzeit verändern
.getAccuracy():int	Gibt den Wert der Button-empfindlichkeit (Zahl zwischen 0 und 100) zurück.
.setAccuracy(value:int):void	Setzen der Buttonempfindlichkeit. Wert muss zwischen 0 und 100 sein
.getQuantityFrames():int	Frameanzahl der Differenzüberlagungen beim DYNAMIC Modus
.setQuantityFrames(number:int):void	Setzen der Frameanzahl der Differenzüberlagungen im DYNAMIC Modus. Werte zwischen 0 und 15 werden empfohlen

Der Konstruktor

Dem Konstruktor wird eine Sprite Instanz und ein CamMode übergeben. Die Sprite Instanz muss das Webcambild enthalten und sollte von der Auflösung ein Vielfaches von 80px X 60px haben (z.B. 320x240px oder 640x480px). Programmiert wurde das System auflösungsunabhängig, daher sollte es eigentlich mit jeder Bildauflösung funktionieren.
Bei dem CamMode Parameter handelt es sich um eine statische Variable der Klasse CamButtonManager. Der CamMode bestimmt in welchen Modus die Interaktion erfolgen soll. Für die jeweiligen Vor- und Nachteile der zwei verschiedenen Modi siehe Tabelle am Anfang dieses Kapitels.

Beispielcode:

```
import de.derhess.iaCam.CamButtonManager;
import de.derhess.iaCam.CamButton;

var manager:CamButtonManager = new
CamButtonManager(webcamBild, CamButtonManager.STATIC)

oder

var manager:CamButtonManager = new
CamButtonManager(webcamBild, CamButtonManager.DYNAMIC)
```

CamButtons erstellen

CamButtons sind eigentlich nicht weiteres als rechteckige Hotpoints im Kamerabild. Diese Hotpoint-Bereiche werden auf aktiv geschaltet, wenn über 80% der innen liegenden Pixel weiss sind. Weisse Pixel im Analysebild sind aktive Pixel, Schwarze Pixel entsprechen unaktiven Bereichen in der Kamera.
Diese rechteckigen Hotpoints werden durch die Klasse CamButton repräsentiert und können nur mit der CamButtonManager Klasse erstellt werden - Die Implementierung orientiert sich an dem Factory-Design Pattern - Als Parameter übergibt man der createCamButton() Funktion eine X und Y Koordinate (der Ankerpunkt liegt immer links oben), sowie eine Breite und Höhe.

Beispielcode:

```
import de.derhess.iaCam.CamButtonManager;
import de.derhess.iaCam.CamButton;

...

var button1:CamButton = manager.createCamButton(5,5,30,30);
var button2:CamButton = manager.createCamButton(50,65,50,50);
```

CamButtons entfernen

Wenn man einen Button nicht mehr benötigt entfernt man ihn einfach mit der Funktion removeCamButton(). Als Parameter übergibt man die entsprechende CamButton-Instanz. Ist diese Instanz in der ButtonListe enthalten, wird der Button aus dieser Liste gelöscht und man erhält den Wert *true* zurück.

Beispielcode:

```
import de.derhess.iaCam.CamButtonManager;
import de.derhess.iaCam.CamButton;

...

var button:CamButton = manager.createCamButton(5,5,30,30);

if(manager.removeCamButton(button))
     trace(„Löschung erfolgreich“);
else
     trace(„Löschung nicht erfolgreich“);
```

CamButtons wieder hinzufügen

Aus der CamButtonManager gelöschte Buttons können dem CamButtonManager wieder mit der Methode addCamButton(alterButton:CamButton) zugefügt werden. Wenn dieser Button schon vorhanden ist erhält man den Wert *false* zurück.

Die Methode startAnalyze()

Die Methode startAnalyze() startet die Bildanalyse und muss periodisch ausgeführt werden. Die Funktion kann mit Hilfe der Timer Klasse oder mit dem Ereignis Event.ENTER_FRAME ausgeführt werden.

Beispielcode:

```
import de.derhess.iaCam.CamButtonManager;
import de.derhess.iaCam.CamButton;

var manager:CamButtonManager = new
CamButtonManager(webcamBild, CamButtonManager.DYNAMIC)

var button:CamButton = manager.createCamButton(5,5,30,30);

var timer:Timer = new Timer(100);
timer.addEventListener(TimerEvent.TIMER, analyze);
timer.start();
```

```
function analyze(e:TimerEvent):void {
     manager.startAnalyze
}
```

Die Methode doSnapshot()

Die Methode doSnapshot() muss in Verbindung mit dem STATIC-Modus eingesetzt werden. Die Methode doSnapshot() muss vor dem Starten der Bildanalyse ausgeführt werden, denn mit Hilfe von doSnapshot() wird ein Frame (Bild) zwischengespeichert, welches ein Abbild des Kamerahintergrunds (ohne User)sein muss. Dieses erstellte Abbild wird bei der Bildanalyse immer wieder mit dem neusten Frame verrechnet.

Beispielcode:

```
import de.derhess.iaCam.CamButtonManager;
import de.derhess.iaCam.CamButton;
import flash.events.*;
import flash.utils.*;

var manager:CamButtonManager = new
CamButtonManager(webcamBild, CamButtonManager.STATIC)

var button:CamButton = manager.createCamButton(5,5,30,30);

//Webcambild ohne User
manager.doSnapshot();

var timer:Timer = new Timer(100);
timer.addEventListener(TimerEvent.TIMER, analyze);
timer.start();

function analyze(e:TimerEvent):void {
     manager.startAnalyze
}
```

Die Methode getBitmapData()

Die Methode getBitmapData() gibt das aktuelle BitmapData Objekt zurück, welches einem Bitmap Objekt hinzugefügt werden kann. Bei einem Seitenverhältnis von 4:3 in einer Größe von 80x60px

Alles rund um Accuracy / Buttonempfindlichkeit

Mit der Funktion setAccuracy() könnt ihr die Buttonempflindlichkeit zur Laufzeit verändern. Stanardmäßig ist der Wert auf 80 eingestellt. Bei diesem Wert handelt es sich um einen Prozentwert, wie viele Pixel in der Buttonfläche aktiv sein müssen (weisse Pixel).

Beispielcode:

```
import de.derhess.iaCam.CamButtonManager;
import de.derhess.iaCam.CamButton;

var manager:CamButtonManager = new
CamButtonManager(webcamBild, CamButtonManager.STATIC)

manager.setAccuracy(60);

trace(„Button Empfindlichkeit: „ + manager.getAccuracy());
```

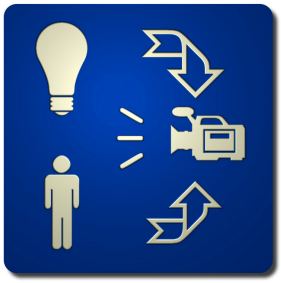
Alles rund um QuantityFrames / Motion Detection

Die Funktionen rund um QuantityFrames können nur im MotionDetection Modus (CamButtonManager.DYNAMIC) eingesetzt werden. Mit der Funktion setQuantityFrames(anzahlFrames) kann eingestellt wie lange auf Bewegung reagiert werden kann. Standardwert sind 5 Frames. Allgemein empfehle ich nicht über 15 Frames zu gehen, da dann die Verzögerung der Bewegung zu lange andauert.

Beispielcode:

```
import de.derhess.iaCam.CamButtonManager;
import de.derhess.iaCam.CamButton;

var manager:CamButtonManager = new
CamButtonManager(webcamBild, CamButtonManager.DYNAMIC)

manager.setQuantityFrames(3);

trace(„Anzahl QuantityFrams: “ +manager.getQuantityFrames());
```

Die CamButton - Klasse



Die CamButton Klasse beinhaltet die Position und Größe des CamButtons bzw. die Position und Größe der HotArea im Webcam Bild. Je nach Zustand des CamButtons werden die Ereignisse CamButtonEvent.ON_HAND, CamButtonEvent.ON_HAND_ OVER und CamButtonEvent.ON_HAND_OFF ausgeführt. Der CamButton kann nur über den CamButtonManager.createCamButton() erstellt werden.

Variablen und Methoden der Klasse CamButton

.CamButton(xpos:Number,ypos:Number, breite:Number, hoehe:Number,scale:Number)	Konstruktor
.move(xpos:Number,ypos:Number):Void	Verschieben des CamButton
.changeSize(breite:Number,hoehe:Number):Void	Verändern der Größe
. getPosition():Point	Koordinate des CamButtons
. getCamButtonRect():Rectangle	liefert alle geometrischen Information des CamButton
. getAnalyzeRect():Rectangle	Geometrische Information über den CamButton im Analysebild des CamButtonManagers

Mit Ereignissen der CamButtons arbeiten

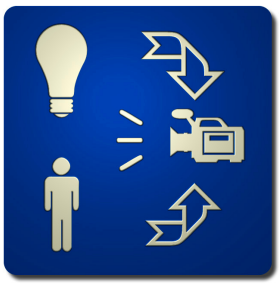
Ein CamButton kann 3 verschiedene Ereignisse werfen. Das Ereignis CamButtonEvent.ON_HAND wird ausgeführt wenn der Button vorher unaktiv war und jetzt aktiv wird. Z.B. durch das erste Berühren der Hand des CamButtons.
Das zweite Ereignis CamButtonEvent.ON_HAND_ OVER wird ausgeführt, wenn der Button bereits aktiv war und noch ist. Dieses Ereignis tritt meisten auf wenn sich die Hand weiterhin über den Button bewegt.
Das dritte Ereignis CamButtonEvent.ON_HAND_OFF tritt ein wenn der CamButton vorher aktiv war und jetzt nicht mehr aktiv ist. Diese Situation triit ein wenn z.B. die Hand über den Button verschwindet.

Beispielcode:

```
import de.derhess.iaCam.CamButtonManager;
import de.derhess.iaCam.CamButton;
import flash.events.*;
import flash.utils.*;


var manager:CamButtonManager = new
CamButtonManager(webcamBild, CamButtonManager.STATIC);

var button1:CamButton = manager.createCamButton(5,5,30,30);
var button2:CamButton = manager.createCamButton(50,65,50,50);

// Listener für Cambutton1 registrieren
button1.addEventListener(CamButtonEvent.ON_HAND,onHand);
button1.addEventListener(CamButtonEvent.ON_HAND_OVER,
                         onHandOver);
button1.addEventListener(CamButtonEvent.ON_HAND_OFF,
                         onHandOff);
```

```
function onHand (e:CamButtonEvent):void {
     trace(„Button1 onHand“);
}

function onHandOver (e:CamButtonEvent):void {
     trace(„Button1 onHandOver“);
}


function onHandOff (e:CamButtonEvent):void {
     trace(„Button1 onHandOff“);
}

manager.doSnapshot();

var timer:Timer = new Timer(100);
timer.addEventListener(TimerEvent.TIMER, analyze);
timer.start();

function analyze(e:TimerEvent):void {
     manager.startAnalyze();
}
```

Die LightTracking - Klasse



Die LightTracking Klasse ist eigentlich nur eine Tracking Klasse. Es wird die Position und Größe einer oder mehrerer Lichtquellen ermittelt. Das Tracking erfolgt mittels eines Helligkeitsschwellenwert und dieser kann zur Laufzeit verändert werden.
Beim Tracking von mehreren Lichtern wird versucht das Trackingergebnis so aufzubereiten, dass alle Lichtquellen immer den gleichen Index im Ergebnis Array haben. Diese Sortierung erfolgt mittels Distanzberechnung zum vorherigen Trackingergebnis. In der Praxis hat sich das Verfahren als sehr brauchbar erwiesen.

Variablen und Methoden der Klasse LightTracking

. static LightTracking.ONE_LIGHT	Statische Variable für das Tracken einer Lichtquelle
. static LightTracking.MORE_LIGHTS	Statische Variable für das Tracken mehrer Lichtquellen
.LightTracking(webcamSignal:Sprite, schwellenwertHelligkeit:int)	Konstruktor
. startTracking(modus:int):void	Starten der Bildanalyse und als Übergabeparameter muss eine der statischen LightTracking Variablen übergeben werden
. getBitmapData():BitmapData	Gibt das Analysebild zurück
. getThreshold():int	Gibt den Schwellenwert des Helligkeitstracking zurück
. setThreshold(schwellenwert:int):void	Ändern des Helligkeitsschwellenwertes. Muss zwischen 0 und 255 sein
. addEventListener(ArtEreignis:String, Listener:Obeject):Void	Anhängen eines Listeners
. removeEventListener(ArtEreignis:String, Listener:Obeject):Void	Entfernen eines Listeners

Der Konstruktor

Der Konstruktor der LightTracking Klasse benötigt ein Sprite Objekt in dem das Webcamsignal enthalten ist. Die Auflösung des Webcam Bildes sollte ein Vielfaches von 80x60px (z.B. 320x240, 640x320) sein. Andere Auflösungsverhältnisse sind möglich, können jedoch ungewollte Fehler verursachen.

Der zweite Parameter ist der Helligkeitsschwellenwert sein. Der Wert muss zwischen 0 und 255 sein. Von diesem Wert ist es abhängig ab welchen Helligkeitswert die Lichtquelle erkannt wird. Werte zwischen 230 und 240 lieferten bisher gute Ergebnisse.

Beispielcode:

```
import de.derhess.iaCam.LightTracking;

// eine Lichtquelle tracken
var track:LightTracking = new LightTracking(webcamBild,
LightTracking.230);
```

Starten des Trackings

Mit der Funktion startTracking() startet man die Analyse des Webcambildes. Diese Funktion muss periodisch ausgeführt werden. Das heißt die Funktion kann per Timer oder durch das Ereignis Event.ENTER_FRAME periodisch ausgeführt werden. Als Übergabeparameter wird einer der zwei statischen LightTracking Variablen benötigt. Das Tracken mehrerer Lichtquellen ist performancehungriger und sollte bei der Verwendung der LightTracking Klasse beachtet werden

Beispielcode:

```
import de.derhess.iaCam.LightTracking;

// eine Lichtquelle tracken
var track:LightTracking = new LightTracking(webcamBild,230);

var timer:Timer = new Timer(100);
timer.addEventListener(TimerEvent.TIMER, analyze);
timer.start();

function analyze(e:TimerEvent):void {
     // Ein Licht tracken
     track.startTracking(LightTracking.ONE_LIGHT);
     //oder  mehrere Lichter tracken
     track.startTracking(LightTracking.MORE_LIGHTS);
}
```

Mit Ereignissen arbeiten

Die LightTracking Klasse erzeugt so genannte LightEvent. Um einen Überblick der zur Verfügung gestellten Daten und der Ereignisse zu bekommen, liefert die folgende Tabelle

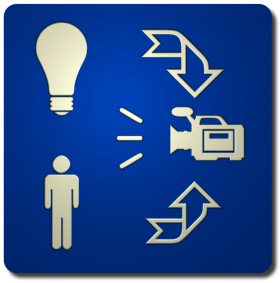
.Ereignisse:	
.static LightEvent.No_LIGHT_FOUND	Wenn kein Licht im Webcambild gefunden wird
.static LightEvent.RESULT_ONE_LIGHT	Wenn ein Licht gefunden wurde im LightTracking.ONE_LIGHT Modus. Das Trackingergebnis liegt im Attribut result als Rectangle Objekt zur Verfügung. Das Attribut resultArray hat in diesem Ereignis den Wert *null*
.static LightEvent.RESULT_MORE_LIGHTS	Dieses Ereignis wird nur im LightTracking.MORE_LIGHTS Modus ausgeführt. Die Tracking Ergebnisse werden im Attribut resultArray abgespeichert. Dieses Array ist mit Rectangle Objekten befüllt. Das result Attribut ist mit dem ersten Rectangle Objekt aus dem resultArray versehen.
.static LightEvent.ADD_LIGHT	Ereignis wird ausgelöst, wenn ein neues Licht erkannt wird. Nur in diesem Modus erhält das Attribut difference einen Wert. Der difference Wert gibt an wie viel Lichter neu dazugekommen sind.
.static LightEvent.REMOVE_LIGHT	Ereignis wird ausgelöst, wenn vorherige erkannte Lichter wieder verschwinden. Nur in diesem Modus erhält das Attribut difference einen Wert. Der difference Wert gibt an wie viel Lichter entfernt wurden.
Attribute	
.result:Rectangle	Trackingergebnis in Form eines Rechtecks
.resultArray:Array	Trackingergebnisse in Form von Rechtecken, abgespeichert in einem Array.
.difference:int	Differenzanzahl der vorherigen ermittelten Lichter zu der neuen ermittelten Anzahl von Lichtern

Beispielcode:

```
import de.derhess.iaCam.LightTracking;
import de.derhess.iaCam.LightEvent;


function onNoLightFound(event:LightEvent) {
     trace(„Kein Licht im Bild vorhanden“);
}



function oneResult(event:LightEvent) {
     trace(„Ein Licht gefunden: “ + event.result.toString());
}

function moreResult(event:LightEvent) {
     trace(„Lichter gefunden: “ + event.resultArray.length);

     for(var i:int = 0; i < event.resultArray.length; i++) {
          trace(i + „: Trackingposition: „+
     event.resultArray[i].x + „/“ + event.resultArray[i].y);

          trace(i + „: Trackingergebnis
          Size:„+event.resultArray[i].width + „/“ +
          event.resultArray[i].height);

     }
}


function onAddLight(event:LightEvent) {
     trace(„Anzahl neuer Lichtquellen: “ + event.difference);

}

function onRemoveLight(event:LightEvent) {
     trace(„Anzahl entfernter Lichtquellen:“+event.difference);

}

// LightTacking Instanz erstellen
var track:LightTracking = new LightTracking(webcamBild,230);

track.addEventListener(LightEvent.NO_LIGHT_FOUND,
                         onNoLightFound);

track.addEventListener(LightEvent.RESULT_ONE_LIGHT,
                         oneResult);
```

```
track.addEventListener(LightEvent.RESULT_MORE_LIGHTS,
                        moreResult);
track.addEventListener(LightEvent.ADD_LIGHT, onAddLight);

track.addEventListener(LightEvent.REMOVE_LIGHT,
                        onRemoveLight);

// Interval starten
var timer:Timer = new Timer(100);
timer.addEventListener(TimerEvent.TIMER, analyze);
timer.start();

function analyze(e:TimerEvent):void {

// Ein Licht tracken
     track.startTracking(LightTracking.ONE_LIGHT);

     //oder  mehrere Lichter tracken
     track.startTracking(LightTracking.MORE_LIGHTS);
}
```

Planung Für Die Zukunft



Wie schon in Version 1.0 angekündigt, möchte ich noch den Algorithmus des LightTrackings in Sachen Perfomance verbessern. Zusätzlich wäre ich sehr glücklich darüber, wenn ich noch in Zukunft ein ColorTracking implementieren könnte, wie es GSkinner bei seinem CamWriter programmiert hat.
(siehe http://incomplet.gskinner.com/index2.html#camwriter)

Weitere Infos zum Farbtracking

- http://www-ra.informatik.uni-tuebingen.de/lehre/uebungen/ws02/ComputerVision/kapitel9.pdf
- http://www.ces.clemson.edu/~stb/klt/

Ein wohl nicht umsetzbarer Wunsch mit ActionScript wäre eine Gestenerkennung mit Fingerzeichen.

Siehe:
http://www.uni-koblenz.de/~cg/Diplomarbeiten/DABreuer.pdf
http://www.techfak.uni-bielefeld.de/ags/ai/publications/master-theses/Gaertner2005-DIP.pdf
http://www.pt-it.pt-dlr.de/_media/01_rogalla.pdf

Wenn jemand da draußen Interesse hat, das Interactive Webcam Package mit mir weiter zu entwickeln, freue ich mich über jede Mail (info@derhess.de). Des weiteren bin ich auch über jeden Tipp oder Idee sehr dankbar!!! ;-)

Vielen Dank!

Florian Weil

www.derhess.de