

# Augmented Reality in 3D Brillen

CS562 Projekt Verteilte Systeme

Maurus Dähler, Mario Weber, Simon Wang

17. Dezember 2013



## Zusammenfassung

Diese Dokumentation beinhaltet den Konstruktionsaufbau einer stereoskopischen 3D Brille und deren Nutzung mit dem Metaio SDK zur Darstellung von 3D Objekten. Augmented Reality wird auf sogenannte "Trackable Images" projiziert. Weiter wird die Logik und Funktionsweise des Programmes anhand einiger relevanten Code Snippets erklärt. Den kompletten Sourcecode, weitere Beispiele und den Vortrag finden Sie unter <https://github.com/cs561/projects>.

## Inhaltsverzeichnis

1	Setup Software Umgebung	3
1.1	Software . . . . .	3
2	Die Brille	3
2.1	Das Prinzip der Stereoskopie . . . . .	3
2.2	Konstruktion . . . . .	4
3	Meatio SDK und AREL - Ein grober Überblick	4
3.1	Dateien . . . . .	4
3.2	AREL - Augmented Reality Experience Language . . . . .	4
4	Projekt - AR Zeitung	5
4.1	Ziel . . . . .	5
4.2	Schritt für Schritt . . . . .	5
4.2.1	Meatio Creator . . . . .	5
4.2.2	AREL Package . . . . .	5
4.2.3	Mergen . . . . .	6
4.2.4	Linking . . . . .	7
4.2.5	GUI und Kameras . . . . .	8
4.3	Die Zeitung . . . . .	9
4.4	Probleme, Frustration, Lichtblicke . . . . .	9
4.4.1	Brille und Blickwinkel . . . . .	9
4.4.2	Auswahl der Library . . . . .	10
4.4.3	Strategie Metaio und Co . . . . .	10
4.4.4	Weiterführung des Projektes . . . . .	10

## 1 Setup Software Umgebung

### 1.1 Software

Zur Realisierung unseres Desktop-Projektes haben wir folgende Software verwendet:

- Microsoft Visual Studio 10 zum Ausführen der Applikation. Ladet das GUI und initialisiert die Kameras. Gibt die Kamerabilder Side-by-Side wieder.



Download für Studenten <http://www.dreamspark.com>

- Metaio Creator zum Erstellen und Experimentieren mit Trackable Images und Augmented Reality. Export der AREL Packages.



Download nach Registrierung <http://dev.metaio.com/sdk/getting-started/>

- Metaio SDK für Templates und Beispiele. Library.



Download nach Registrierung <http://dev.metaio.com/creator/>

- Notepad++ zum Mergen und Anpassen der Javascript und XML Dateien.



Download free <http://notepad-plus-plus.org/>

## 2 Die Brille

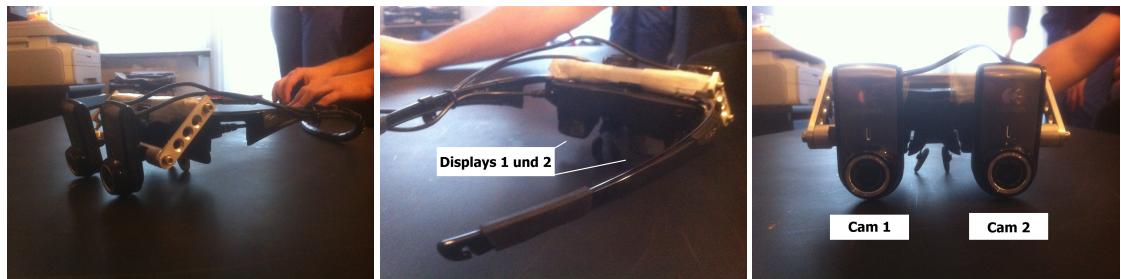
### 2.1 Das Prinzip der Stereoskopie

Bei der Stereoskopie werden zweidimensionale Bilder so übereinander gelegt, dass ein Eindruck von räumlicher Tiefe entsteht, obwohl keine vorhanden ist. Das beste Beispiel hierfür ist das menschliche Sehen. Die Augen erzeugen zwei Bilder aus abweichenden Betrachtungswinkeln, wodurch unser Hirn dann in der Lage ist, Entfernung zu erkennen und ein räumliches Bild der Umgebung zu gewinnen. Genau diesen Trick versuchen wir zu simulieren, indem wir die Bilder von zwei Webcams so manipulieren, dass unser Gehirn den Eindruck räumlicher Tiefe bekommt. Die Bilder werden von der Brille

Side-by-Side wiedergegeben. Das heisst, sie entzerrt die Bilder in das von der Brille unterstützte Format und verteilt die Bilder gleichzeitig auf die Displays des linken und rechten Auges.

## 2.2 Konstruktion

Das Grundmodell der Brille ist eine WRAP 920. Diese haben wir auseinander gebaut und verwenden nur das Gestell und die Displays. Auf das Gestell haben wir nun zwei Logitech Webcams 2MP montiert, welche uns die zwei Bilder in leicht unterschiedlichen Blickwinkeln liefern.



Brille, Webcams und Side-by-Side Apdapter sind alle per USB angeschlossen.

Brille: [http://www.vuzix.com/consumer/products\\_wrap920.html](http://www.vuzix.com/consumer/products_wrap920.html)  
Cams: <http://www.logitech.com/de-ch/support/portable-webcam-c905>

## 3 Meatio SDK und AREL - Ein grober Überblick

### 3.1 Dateien

Das SDK wird für die jeweilige Plattform heruntergeladen. In unserem Fall ist dies Windows-Desktop. Darin enthalten sind Dokumentation, die Library dlls, Javascript Library, implementierte Beispiele und ein leeres App-Template.

### 3.2 AREL - Augmented Reality Experience Language

Ein AREL Projekt besteht aus einem statischen und einem dynamischen Teil. Der statische Teil ist eine XML Datei, welche den gesamten Inhalt, also den Link zu den Trackable-Images und dazugehörige Augmented Reality enthält. Dieses statische XML wird mit dem dynamischen Teil verbunden. Dies geschieht in Form von Javascript, welches die Interaktionen und das Verhalten von einzelnen Objekten in der gesammten Szene definiert. Javascript verbindet also die Logik der SDK API mit dem statischen Inhalt der XML Datei. Mit AREL ist es möglich eine Plattform unabhängige Applikation zu schreiben, da nur XML und Javascript verwendet werden.

## 4 Projekt - AR Zeitung

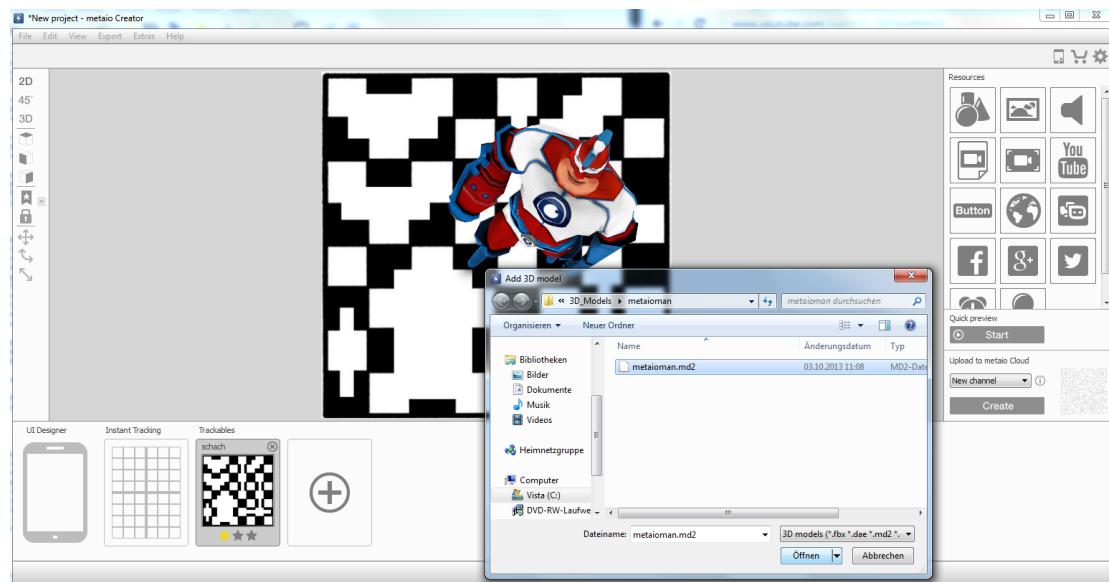
### 4.1 Ziel

Ziel unseres Projektes war es, ein Beispiel für eine AR Anwendung zu implementieren und diese mit einer eigenen Brille in 3D darzustellen. Da die Goolge Glasses zur Zeit ein sehr aktuelles Thema sind, wollten wir in unserer Arbeit selbst die Probleme erfahren, welche die Realisierung einer Augmented Reality Brille mitsichbringt. Augmented Reality befindet sich noch in den Kinderschuehen. Deshalb soll das Projekt soll auch die Komplexität der verschiedenen Programmiermöglichkeiten wiederspiegeln.

### 4.2 Schritt für Schritt

#### 4.2.1 Meatio Creator

Im Metao Creator wird eine Szene erstellt. Dazu wird das Trackable-Image und die dazugehörige Augmentation ausgewählt. Diese kann natürlich ein 3D Objekt, ein Video oder weiteres sein. In der freien Version dürfen nur jeweils zwei Trackable-Images verwenden.



#### 4.2.2 AREL Package

Die Szene wird nun als AREL Package exportiert. Dadurch werden automatisch html, js und xml Dateien erzeugt. Wenn wir jetzt zwei verschiedene Szenen erstellen, können wir die Dateien miteinander mergen, so dass wir mehr als nur zwei Trackables in einer Szene haben.

#### 4.2.3 Mergen

Im index.xml hat jede Augmentation eine eigene <coordinatesystemid>. Dies darum, weil natürlich jede Augmentation seine eigenen Koordinaten bezüglich des Tracking-Images hat. Haben zwei die gleichen Koordinaten, können diese natürlich auch auf die ID gleichzeitig zugreifen. Die übrigen statischen Werte im XML kann man hier auch von Hand ändern. Eine ausführliche Dokumentation zum AREL XML gibt es unter <http://dev.metaio.com/arel/xml-reference/>

Beispiel Snippet eines **index.xml**:

```
<object id="image5">
    <title><! [CDATA[4]]></title>
    <assets3d>
        <model><! [CDATA[html/resources/123456789.zip]]></model>
        <transform>
            <translation>
                <x>0.0000000000</x>
                <y>0.0000000000</y>
                <z>5.0043873787</z>
            </translation>
            <rotation type="eulerdeg">
                <x>0.0000000000</x>
                <y>0.0000000000</y>
                <z>0.0000000000</z>
            </rotation>
            <scale>
                <x>0.8774999976</x>
                <y>0.8774999976</y>
                <z>0.8774999976</z>
            </scale>
        </transform>
        <properties>
            <coordinatesystemid>2</coordinatesystemid>
        </properties>
    </assets3d>
    <viewparameters/>
</object>
```

Im **resources** Ordner müssen alle Dateien gemäss Ordnerstruktur zusammengefügt werden. Im **Tracking.xml** müssen dann <SensorCos> und <Cos> (Abk. für CoordinateSys) angepasst oder ergänzt werden. Im Package findet man weiter die **logic.js** Datei. Darin muss lediglich der Konstruktor und die Patterns der Objekte reinkopiert werden.

Beispiel eines Image **Pattern**:

```
var pattern8 = {};
scenario.trackables.push(pattern8);
pattern8.objectName = "pattern8";
pattern8.cosName = "Werbung_1";
pattern8.cosID = "3";
pattern8.isCurrentlyTracking = false;
pattern8.currentTrackingValues = null;
pattern8.onUnloaded = function () {
    arel.Debug.log(this.objectName + ".onUnloaded()");
    model7.unload();
};
```

Beispiel für ein Video **Objekt**:

```
var image3 = arel.Scene.getObject("image3");
image3.objectName = "image3";
image3.sceneID = "1";
image3.isModel3D = true;
image3.isExplicitlyInvisible = false;
image3.isShownOnTheUserDevice = false;
image3.isShownOnTheInstantTracker = false;
scenario.contents.push(image3);

image3.setSceneID = function (sceneID) {
    this.sceneID = sceneID;
    scenario.switchScene(scenario.currentSceneID);
};

.
```

Die germeigte Szene kann nun als WebApp, MobileApp oder DesktopApp weiterentwickelt werden.

#### 4.2.4 Linking

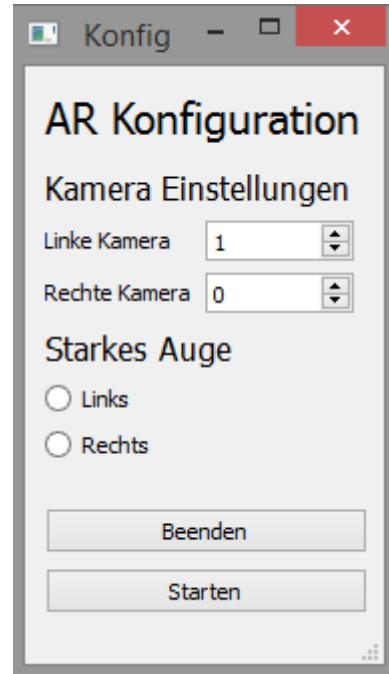
Für unsere Brille wollen wir ja ein DesktopApp mit zwei Videostreams. Dazu haben wir unser Package mit dem Rohtemplate aus der Meatio SDK verbunden. Innerhalb des SDK ..\metaio SDK 5.0.1\\_Windows\Examples\_SDK\Example\_Qt wird die Example\_Qt.mvs Datei geöffnet. Unter void TutorialBaseAREL::loadContent() setzt man den Pfad nun auf die html-Datei des im Creator erstellten Package.

```
void TutorialBaseAREL::loadContent()
{
    // AREL filename, e.g. "../../../../assets/Assets1/arelConfig1.xml"
    std::stringstream filename;
    filename << "../../../../tutorialContent_crossplatform/Tutorial" << m_tutorialNumber << "/Demo/index.xml";
    m_pArelInterpreter->loadARELFile(filename.str());
}
```

#### 4.2.5 GUI und Kameras

Die Tutorial GUI wurde zu unseren Zwecken umprogrammiert. Beim Starten des Programms wird zuerst ein Konfigurationsfenster geöffnet, in dem man die Inputs der Kameras einstellt und sein starkes Auge angibt. Die Inputs können durch Try and Error herausgefunden werden und bleiben anschliessend die selben für die Kameras, sofern sie immer in die gleichen USB-Ports gesteckt werden. Das starke Auge kann mit folgender Übung bestimmt werden. Man formt mit Daumen und Zeigefinger einen Kreis, anschliessend sucht man sich einen Punkt an der Wand und nimmt diesen in den Kreis. Nun schliesst man abwechslungsweise die Augen. Das Auge, mit dem man den Punkt immer noch im Kreis hat, ist das starke Auge. Dies ist wichtig da wir nur von einer Kamera eine Augmentation kriegen können. Würden beide Kameras ein Bild liefern, kann es sein dass sie perspektivisch nicht gleich sind und ein merkwürdiges Bild entsteht, oder das die Kamera des schwachen Auges ein Bild hat das starke aber nicht und ein sehr starker Geistereffekt entsteht. Dieser ist mit dieser Methode zwar auch hier aber weniger schlimm. Es ist nun vergleichbar mit dem Blick durch ein Mikroskop während man beide Auge geöffnet hat.

Anschliessend wird das ARWindow geöffnet. Hier erscheinen die beiden Kamerabilder Side-by-Side, damit räumliche Tiefe erhalten bleibt. Wie oben erwähnt erscheint die Augmentation nur auf der Seite des starken Auges. Ausserdem wurde die Auflösung so gewählt, dass es genau dem Fullscreen der Brille, 1024x768, entspricht.





### 4.3 Die Zeitung

Wir haben die 20min Zeitung vom 13.12.2013 mit Augmented Reality aufgewertet. Zum "Proof of Konzept" haben wir drei Augmentations implementiert. Ein Video, ein 3D Objekt, sowie eine Slideshow. Damit haben wir drei unterschiedliche Visualisierungen und zeigen auch, dass man sehr leicht in der Logik eigene Ideen (Slideshow) programmieren kann. Schlussendlich haben wir sechs Augmentations auf drei Tracking-Images.

### 4.4 Probleme, Frustration, Lichtblicke

#### 4.4.1 Brille und Blickwinkel

Obwohl die Technologie unserer Brille sehr bescheiden ist, werden die Augmentations meistens sehr schön in 3D dargestellt. Es ist vorgekommen, dass eine Kamera später auf das Tracking-Image reagiert als die andere oder garnicht. Es entsteht ein Geistereffekt, da die Augmentation nur auf einem Auge sichtbar ist. Wir haben herausgefunden, dass wenn das Bild aber trotzdem auf dem starken Auge sichtbar ist, dieser Effekt minimal ist. Deshalb zeigt das Programm nun nur noch auf einer Seite die Augmentation an.

#### 4.4.2 Auswahl der Library

Es gibt unzählige Möglichkeiten eine AR Szene zu implementieren. Jede hat auf ihre Weise Vor- und Nachteile. Wichtige Punkte zur Wahl sind eindeutig die Installation der Programmierumgebung, Handhabung der Augmentations und die Logik, welche alles miteinander Verbindet. Vieles ist kostenpflichtig, steckt noch in den Kinderschuhen oder ist verbuggt. Unsere Wahl viel relativ schnell auf die Metaio Library weil sie einfach extrem gut funktionierte. Selbst bei schlechtem Licht oder Unschärfe bei Kamerabewegung wurde die Augmentation sehr schön angezeigt. Dies im Gegensatz zu vielen Konkurrenzprodukten. Weiter ist eine ausführliche Dokumentation sehr von Vorteil. Damit wir überhaupt erst beginnen konnten, mussten wir erst sehr viel Zeit fürs Einlesen investieren.

#### 4.4.3 Strategie Metaio und Co

Die Tracking-Images sind im freien Metaio Creator auf zwei begrenzt. Wie wir gezeigt haben, ist das Mergen eine doch sehr aufwändige Methode, besonders für grosse Projekte. Wenn man den Aufbau und die Logik der Implementierung verstanden hat, könnte man theoretisch alles selber programmieren. Soll die Szene aber zum Schluss Plattform unabhängig verwendet werden, so kann diese sehr schnell ziemlich kompliziert und unübersichtlich werden. Es ist also möglich, unter riesigem Aufwand alles selber zu schreiben, oder das selbe mit einer Software Lizenz in 15 Minuten per Drag and Draw generieren zu lassen.

#### 4.4.4 Weiterführung des Projektes

Ein weiterer Schritt wäre nun ein Programm zu schreiben, welches die einzelnen Pakete automatisch merged. Man könnte auch die Augmentations in einer Datenbank ablegen und dann verlinken. Aber auch für diese beiden Optionen muss man sich fragen, ob es nicht sinnvoller ist, einfach eine Lizenz zu kaufen. Die Software bietet nämlich alle diese Features, und es ist sehr einfach, diese auch zu Nutzen. Verwendet man eine sogenannte "World", kann diese nämlich auf sämtlichen Endgeräten angesprochen werden. Außerdem könnte man sich noch eine geschickte Eingabemethode für solche handsfree Geräte überlegen. Ideen während dem Projekt waren 3D Tracking der Hand, ähnlich wie es die XBox Kinect mit dem ganzen Körper macht. Ein Datenhandschuh mit einem internen Accelerometer, mit welchem man die Maus durch bewegen der freien Hand steuern könnte und zuletzt noch ein Hirnwellenmesser, welche laufend in Entwicklung sind und Jahr für Jahr kleiner, günstiger und präziser werden.