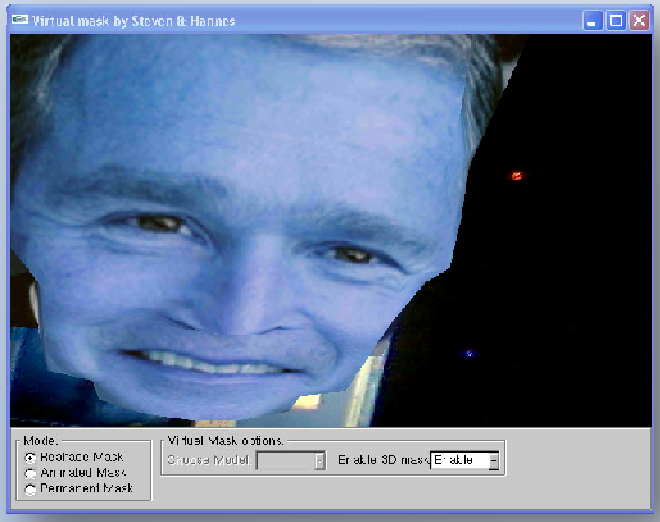
Report assignment 2

Construction of an augmented mask program using AR Toolkit



Course

IN4151 3D Computer Graphics and Virtual Reality (3DCG&VR)

Version

Final (januari 2009)

Group 3

Steven Bos	1319671
Hannes Smit	1228897

Introduction

In this report we present our result for assignment 2. For this assignment we liked to do something with virtual reality. After some investigation we stumbled on AR Toolkit, a computer vision framework. To use the most recent version of the framework (version 4.3) you need to buy an expensive license[**1**]. Fortunately there is also an open source version which is quite complete[**2**]. It has USB camera support and marker detection build-in to give us a head start. And it is build using Open GL, which was mandatory for this assignment. Looking at the project page on the AR Toolkit website, many interesting application have been build. For example ARtag[**3**] has an excellent demo[**4**] of AR Toolkits (open source version) potential on its webpage and on youtube. This inspired us to do something with it. We came up with the idea to place markers on a human face and project a mask over it, including moving features.

In this report we show what we developed in the past weeks. An introduction to our software is described in chapter 1 (Installation) and chapter 2 (User Interface). In chapter 3 we discuss AR toolkit, its basics and discuss a paper on the workings[**8**]. In chapter 4 we discuss building our program using AR Toolkit. We close this report with a short review of the assignment in chapter 5.

Enjoy reading,

Group 3

Hannes & Steven

Table of Contents

1. Installation

S+H Virtual Mask is written in C++ using OpenGL and developed in MS Visual Studio 2005. It is tested on Windows XP only. The software is dependent on several libraries which are not present by default. This section will describe how to setup MS Visual Studio 2005 and play with the code. After compiling the project, run the version by clicking the exe in the bin directory. The included SHvirtualMask executable does not need any installation; just run it.

§1.1 dependencies

This software makes use of:

- OPENGL and its default GL,GLU,GLUT libraries
- GLUI (an extension of GLUT, providing basic UI controls)
- AR Toolkit version 2.72.1

§1.2 setup visual studio 2005

The libraries and header files necessary for this code project are included. The following 7 steps sets-up visual studio 2005, so it can find these libraries and header files.

1 Copy the unzipped folder glut with all its subfolders to the `c:\program files` directory. Result should be `c:\program files\glut\..`

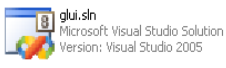
2 Update default search path in windows environment variables by including
`C:\program files\glut\lib`

3 Restart PC to load search path

3 Copy the unzipped folder `virtualMaskProject` with all its subfolder (AR toolkit libraries included here) to any desired location. We installed it at
`my documents\virtualMaskProject`

5 Open visual studio. Go to `Tools > Options > expand Projects and Solutions` > expand `VC++ directories`. In VC++ directories "select show directories for" and select include files. Add the following line:
`C:\Program Files\glut\include`

6 In the same VC++ directory go again to `"select show directories for"` and now select `libraries files`. Add the following line:
`C:\Program Files\glut\lib`

7 Go to the `virtualMaskProject` folder and go to `\src\msvc` subfolder. Double click the Visual Studio solution file to start up the project. 

NOTE: The specific dependencies (in configuration `properties>linker>input> additonal dependencies`) are already specified in the project file. They should be `glui32d.lib, glut32.lib, glu32.lib, opengl32.lib, odbc32.lib, odbccp32.lib`, `libAR.lib, libARgsub.lib, libARgsub_lite.lib, libARvideo.lib`

2. User interface

This section discusses the user interface controls. The UI is actually very simple, but complete. The primary input-device is the mouse. There are no right or middle mousebutton event-handlers and keyboard input is also limited to arrow keys only (to modify menu values). The program closes by clicking on the X button in the top right corner. In Fig. 2-1 the available controls are pictured.

Fig. 2-1 *Available user interface controls*

The controls are grouped in 2 columns: Mode controls and Virtual Mask options

Mode Control

There are three modes to choose from:

Real face Mask — This mode shows a human face (texture) when a single particular marker is detected. It only reacts on markers active in this mode.

These markers are: *Patt.Bush, Patt.Gates, Patt.Hirst*

Animated Mask — This mode shows an animated face or parts of an animated face (based on standard open gl objects) when the program detects particular markers. It only reacts on markers active in this mode and is able to detect multiple markers concurrently.

These markers are: *Patt.body (the face without features), Patt.eyes (just the eyes), Patt.mouth(just the mouth), Patt.eyebrows(just the eyebrows).*

Permanent Mask — This mode shows the same animated face or parts of that face as the previous mode but differs in function. When the program detects a marker it enables the face part and keeps drawing it, even when the markers is not detected anymore. When a delete all marker is detected the program returns to its initial state. This way a face can be constructed and adjusted in a more controlled way.

The active markers are the same as in the previous mode and include one more: *Patt.kanji (the delete all marker).*

Virtual Mask options

Choose model	This drop down box becomes active only in the *animated mask* mode and *permanent mask* mode. It allows the user to choose from four basic emotions/facial expressions which are directly translated on the active markers.
Enable 3D Mask	This drop down box becomes active only in the *real face mask* mode. It allows the user to enable/disable a 3d wireframe model containing very basic facial features behind the texture, to create the illusion of a 3d face. AR Toolkit is able (in a limited way) to detect a marker from multiple angles.

3. Augmented Reality & AR Toolkit basics

This chapter gives a short introduction to augmented reality and AR Toolkit, a framework to augment reality by means of a webcam and markers.

§3.1 Augmented Reality

Augmented Reality is a Computer Vision field which is getting lots of attention nowadays. The use of these applications reach from Head Mounted Displays (HMD's) to mobile phone applications which are suited with camera's and good displays.

Sometimes *3D models* are displayed over real world data, but it is also possible to add information (texts, figures, sounds) to certain contexts.

Using markers, which can be identified by camera's, applications like AR Toolkit create a link between virtual information and real physical objects. Virtual characters can be used in a physical game for example.

In Augmented Reality you have basically two computer vision techniques to combine real world images with artificial 3D models. One is the superimposition of 3D data on live video data, the other is the use of transparent displays were only the 3D data is displayed on it.

With AR Toolkit not only recognition of markers takes place, the position of markers and their orientation are also detected. In this way the coordinates of virtual objects can be aligned to real objects.

§3.2 AR Toolkit internal workings

The Augmented Reality (AR) Toolkit projects 3D Computer graphics over live video frames. By making use of tracking markers coordinates in the real world and coordinates of the models can be related. This is done via the following steps, which are displayed in figure 3-1.

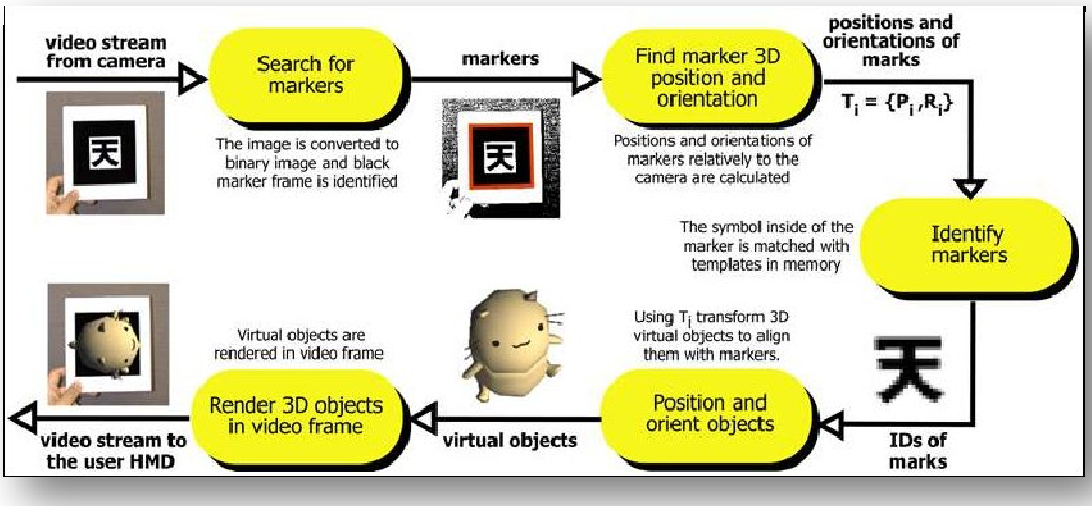


Fig. 3-1 *AR Toolkit general flow diagram (image courtesy of [**2**])*

1. **Video capture**
   Live video frames from i.e. a regular webcam are captured as still images.

2. **Search for square shapes**
   By making use of binary images containing distinctive squares the software can track markers. In this way the marker frames are identified.

3. **Calculate position of the camera**
   When the position and form of the marker is detected, the position of the camera can be calculated.

4. **Identify marker**
   Every marker has a certain symbol inside its frame. An AR Toolkit project has a collection of symbols preloaded in its memory. These symbols are matched to the captured symbol from the video frame.

5. **Model drawing**
   A three dimensional object, defined in the software application, is projected. The position and 3D orientation of the objects is determined by the marker orientation in the real world.

6. **Overlay video with model view**
   The model is rendered on top of the video stream. Now it looks as if the object is attached on the real world marker.

Note: It is possible to track/project multiple markers/objects at the same time in this scheme.

AR Toolkit has a straight-forward architecture as seen in figure 3-2. The main pipeline is shown in figure 3-3.

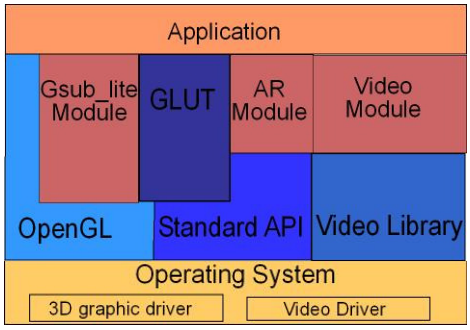


Fig. 3-2 *AR Toolkit architecture*

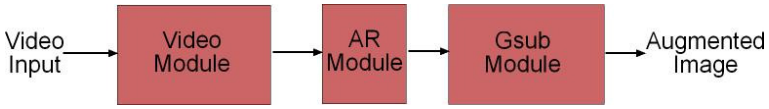


Fig. 3-3 *AR Toolkit main pipeline*

§3.3 Tracking markers

To accurately track markers, image processing techniques are used. Before we explain how images are processed, first, we will discuss how the coordinate systems are organized (figure 3-4)

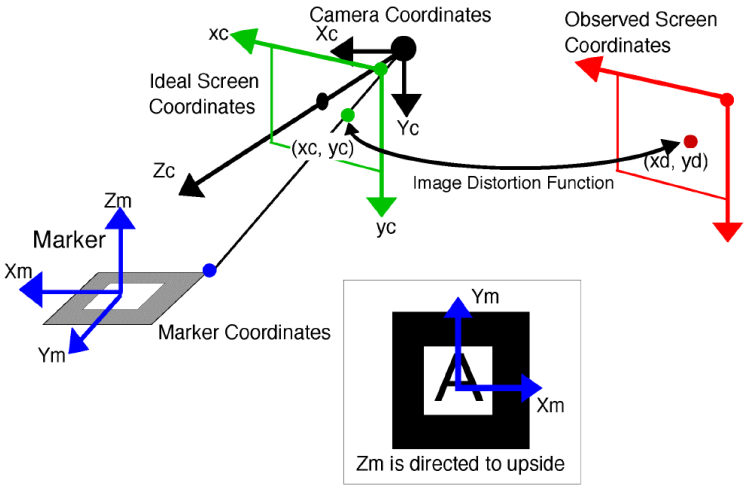


Fig. 3-4 *Coordinate systems & distortion*

The marker coordinates (Xm, Ym and Zm) are related to the camera coordinates (Xc, Yc and Zc) via the ideal screen coordinates (xc and yc). The following formula in figure 3-5, describes the transformation between marker coordinates and camera coordinates.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & V_{13} & W_x \\ V_{21} & V_{22} & V_{23} & W_y \\ V_{31} & V_{32} & V_{33} & W_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_m \\ Y_m \\ Z_m \\ 1 \end{bmatrix}$$

Fig. 3-5 *Transformation formula between marker and camera coordinates*

Here the V components define the rotation, the W components define the translation.

Due to camera distortions and the perspective projection matrix the observed screen coordinates (xd and yd) are not exactly the same as the ideal (see figure 3-6). Therefore a distortion formula was created in which the distortion factor can be set (and can be tuned via camera calibration).

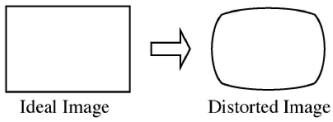
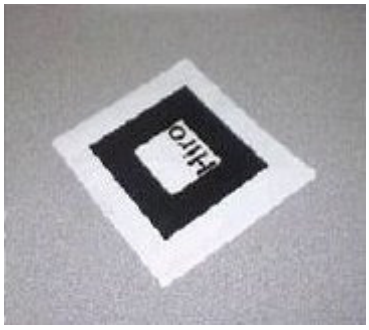
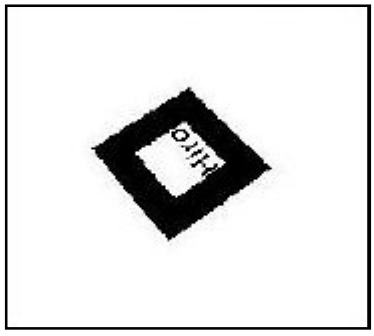


Fig. 3-6 *Distortion*

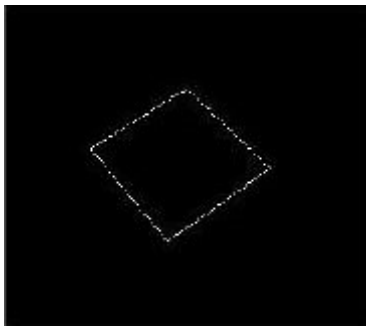
For every video frame AR toolkit searches for a marker. The square frame of the marker makes it possible to recognise features. This is done by thresholding, labelling and feature extraction (area and position). Next the contours are extracted and four straight lines are fitted. The images in 3-7 below show how the *Hiro* pattern is processed according to this procedure.



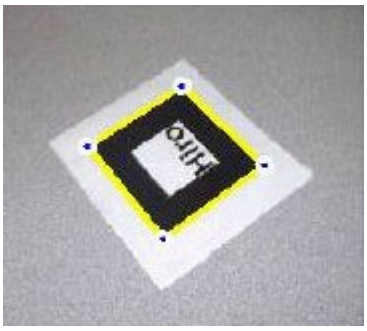
1. The original image is captured



2. The image is thresholded to a binary image



3. The contours are found



4. The marker edges and corners are extracted

Fig. 3-7 *AR Toolkit marker detection mechanism*

This process is fast and will be done for every video frame. Unfortunately features will have bad quality when jitter occurs for example. This is improved by making use of previous frame information. Although this cannot be used in the first frame, it gets more stable results. We disabled the use of previous frame information in our program because of the trade off against accuracy, which is important for real time and accurate face features.

§3.4 Recognition performance

The following conditions influence the recognition performance:

- **Visibility.** Markers must be in sight; covering up markers with hands or other markers lets it disappear.
- **Range.** The size of the marker and the distance between the marker and the camera has a big influence on the performance of the marker detection.
- **Pattern Complexity.** The more complex a pattern is, the worse the tracking performance gets.
- **Marker orientation.** The more perpendicular the marker is to the camera, the better recognition performance is. When markers become more horizontal (and parallel to our cameras viewing direction) also the patterns become indistinctive.
- **Lighting conditions.** Too much light or reflective patterns may influence the results of the threshold step and thereby influence the recognition of frames or patterns.

4. Implementation

Just installing the AR Toolkit and extending one of its examples is (logically) not considered as sufficient result for this assignment. This chapter elaborately discusses the creation of our program (the process), what code we reused and what code we developed ourselves.

§4.1 Step 1: Constructing the initial framework

AR Toolkit provides a useful set of examples out-of-the-box. It contains the "*simpleTest example"* demonstrating the detection mechanism of a single marker and the display mechanism when a single marker is detected (and draws a basic OpenGL cube). Another useful example is the "*loadMultiple example*" explaining the detection and display mechanism of multiple markers concurrently (on non pre-programmed locations). All these examples use the gsub rendering library of AR Toolkit which is layer on top of OpenGL handling amongst others the window events. We wanted to implement a basic user interface (using GLUI), which isn't present in AR Toolkit. GLUI is an excellent library for rapid UI design but it needs control over the window events. Luckily another example ("*simpleLite example"*) also present out-of-the-box, demonstrates the usage of the gsub_lite library which provides OpenGL window handling. This example however uses the detection mechanism of single marker.

To wrap up, we replaced the multi marker mechanism of *loadMultiple* with the single marker mechanism present in *simpleLite*. We did just that and after some tweaking and including GLUI, we created our initial framework. With the framework ready we could begin calibrating the Logitech Quickcam Messenger webcam (a tedious process).

§4.2 Step 2: Develop features

Initially we wanted to track a human face with markers attached to it and project a virtual mask on it with moving face features. When we developed a mask with only two markers (one on the forehead and one on the chin) we noticed that the projected talking mouth was not in sync with the original human talking and was sometimes displaced incorrectly compared to the nose and other face features. Assuming AR Toolkit handles its projections consistently, the marker was not recognized correctly because of:

- Changing light conditions
- Curved marker surface when attached smoothly to skin
- Similar looking markers when using a low-resolution camera

Although the extension on AR Toolkit, AR Toolkit+ (now named Studierstube[**5**]) is able to track far tinier markers under worse light condition it still wouldn't solve the curved marker surface detection problem. Also, both AR Toolkit and AR Toolkit+ as they are now, are just too sensitive for lighting changes. Commercial solutions, who use specialized software and markers just for the purpose of tracking facial features have nothing to fear (yet) from this open source alternative.

Our original plan failed, so we came with the idea to add some marked based interaction on an animated face (we stuck to the mask theme). First we developed an animated face, constructed of four face parts (using only OpenGLdrawing primitives) and gave the face four basic expressions. We then added a function that permanently displays any detected marker and a delete-all marker that disables all detected markers. The emotions can be changed via the GUI, although implementing a marker based change is easily constructed.

§4.3 Step 3: Develop new markers

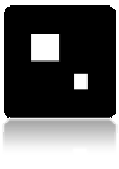
There is few theory on marker design. The following "rules/guidelines" are trivial, found in AR Tookit documentation or result from §3.4 (detection performance):

- The marker must be asymmetric (otherwise correct marker orientation fails)
- The marker must have a small border (detection rate decreases significantly when border is 0.0 because markers can blend in surrounding)
- Markers may have different shapes then the square markers provided with the examples (marker configuration needs to be adjusted for the desired shape)
- Markers may contain color, because AR Toolkit translates the marker to grayscale internally
- Markers must not be complex, since it influences recognition (although no theory on what is considered complex)

But these rules didn't help us figure out which markers design were best. We assumed that high contrast markers (with only pure black and pure white colors) would do best because it is trivially not complex (rule 5). But humans like colors and familiar symbols and not b/w squares. So we experimented a bit with different designs. After every design we trained AR Toolkit to recognize them with a small program called *mk_patt,* which finds the marker in a video frame and then translates it to a binary file.

Our small marker design experiment gave the following result:

- We compared the default "Hiro" pattern with a new simple marker (see Fig. 4-1) containing a high contrast pattern of 2 squares. We noticed no differences in detection of the markers.
- We made a marker which symbol was a jpeg of Bill Gates in grayscale(complex, see Fig 4-2a) and compared it with a simple design (Fig 4-2b). The difference was slightly noticeable, the simple marker could be recognized up to about 0.9 meters distance (surprisingly, because the marker was unrecognizable for human vision and the systems still detected it), while the jpeg marker could only be recognized up to about 0.6m.
- The last experiment we did was a hybrid design where a small grayscale jpeg was centered in the middle. This gave the marker a very recognizable artifact for the human, while at the same time have ¾th of the marker free for high contrast, low complex symbols. We implemented this for our animated mask feature, because its detection was acceptable for our program (distance requirements between 0.2m and 1m)



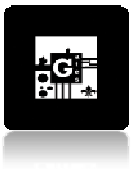


Fig 4-1. *Simple high contrast marker* **Fig 4-2ab** *Complex and simple Gates marker*

Unfortunately, because of the deadline we could not do more and better measurements and therefore cannot draw real conclusions. Though, the mini experiment did not present any counter examples against the design guidelines we presented and listed above. The found results are very dependent on the used webcam and current lighting conditions. Result will be better (as seen on the internet) when using a better webcam and stable lighting conditions.

§4.4 Implementation notes

The following code is all developed by us:

- All code in mask.cpp
- the code in routines: control_cb(), initLights(), draw_object(), GUImode(), a large part in Display() and a large part in main()

On a side note: the initial framework (enabling multiple markers detection with the default OpenGL render routines (gsub_lite) instead of the default AR Toolkit render library) was also developed by us!

5. Review and future work

This chapter presents our review on this assignment in the first paragraph and our recommendation for future work in the second.

§5.1 Review

We enjoyed working on this assignment, although we couldnt complete our initial plan due to the reasons presented in chapter 4. The timeframe and our relative poor programming skills gave us a hard time to produce a nice(r) result. We started this assignment with no AR Toolkit experience, so getting familiar with the library, documentation and related theory took some time. Other time consumers were building the initial framework with GLUI and creating a mask (with correctly placed face parts), based on a hierarchical structure of OpenGL objects.

§5.2 Future work

As mentioned in the introduction a lot of really cool applications have been build with AR Toolkit. Thanks to this assignment we acquired a bit of experience with it and definitely like to play with it some more in the future. Ideas we would like to explore:

- Create a simple game with collision detection of two markers, without losing augmented projection.
- Improve marker detection by adding an infrared camera and RFID on every marker for accurate tracking under any light condition (no need for adjustments to environment and make it camera independent without recallibration)

References

Webreferences

1) http://www.artoolworks.com/ARToolKit_Professional.html

2) http://www.hitl.washington.edu/artoolkit/download/

Note: some images in chapter 3 are used from the website of [2]

3) http://www.artag.net

4) http://www.youtube.com/watch?v=ItOtTdhDoto

5) http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php

Books

6) Hearn,D., Baker, M., (2004). Computer Graphics with Open GL. *3th edition international version, Pearson Education.*

Reader

7) Nieuwenhuizen, P.R, et al. (2007). Computer Graphics Lecture notes. TU Delft reader course in2905-I.

Papers

8) Kato, H., Billinghurst, M. (1999) Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. In Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99). October, San Francisco, USA.