ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
School of Computer and Communication Sciences IC

Digital 3D Geometry Processing

# Face projection mapping

May 2015

*Authors:*
Loubna Rizqi
Ching-Chia Wang
Simon Rodriguez

# 1 Idea

The idea behind our project is to build an installation which enables us to project our animations on a physical head model. This animation would be our own expressions mapped with different textures projected on the model in real time. This process is called projection mapping.

# 2 Realization

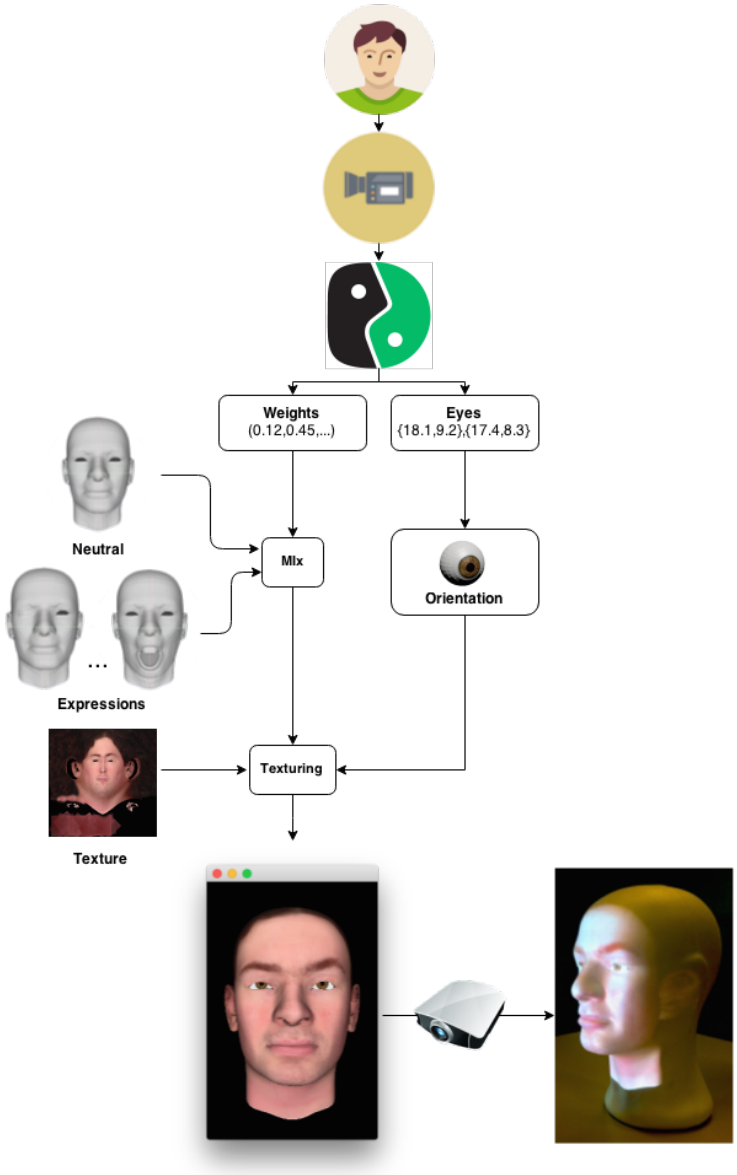


Figure 1: The general process we implemented

## 2.1 Surface Reconstruction

We used PhotoScan to reconstruct the mesh of the styrofoam head model. The original head model was purely white and smooth. PhotoScan failed to detect feature points so the acquired point cloud was in poor quality. Thus We put markers all over the head model on its edges, convex, concave, and plain areas, and the resulting mesh was in reasonable quality as shown in Figure 2. In total we used 52 photos (8 millions pixels each).

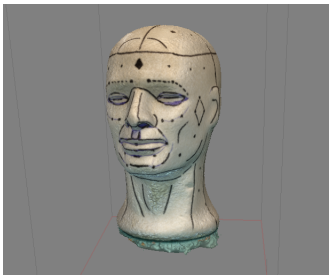

Figure 2: PhotoScan reconstructed mesh with marked Styrofoam head model

## 2.2 Non-rigid Alignment / Deformation Transfer

We used the code from Exercise 7 and 9 to acquire neutral and 48 expressive meshes of the styrofoam head model. Some results are shown in Figure 5 and Figure 4.

Figure 3: Non-rigid alignment fitted head model with neutral expression shown in MeshLab. Render mode from left to right: Flat lines, Flat, and Smooth

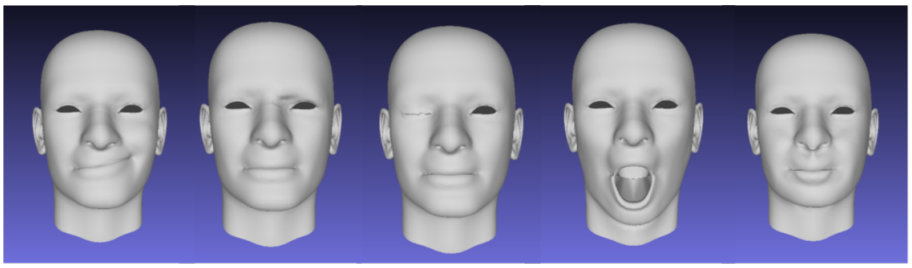

Figure 4: Deformation transfer of the Styrofoam head model mesh

### 2.3 Display and animation

In order to display the animated face, we built a viewer based on the QGLViewer. We first added the FOV control, using F and G keys. In combination with rotation and translation of the head, this allows us to calibrate manually the projected face onto the physical head model.
When the program starts, we load the neutral mesh and the expressions of the foam head. As the deformation transfer used to generate the expressions modifies the centroids of all meshes, we had to reuse the centroids from the Macaw expressions (stored in a text file) to make sure the mesh wouldn't move erratically during the animation.

When an array of weights is received, either from TCP streaming or from a .fsb file, we recompute the position of each vertex of the mesh as a weighted sum of the corresponding positions in the neutral mesh and the expressions. We can then push the set of vertices to OpenGL for rendering.

To texture the mesh, we bind the texture contained in a .bmp file (easier to read) before drawing. As the texture is fragmented in multiple parts, some vertices have multiple texture coordinates, stored in the half-edges of the mesh. To handle this multiplicity in OpenGL, we duplicate vertices : we iterate over all faces, and for each vertex of the current face, we output a new set of vertex, texture, normal coordinates for the OpenGL buffer.
Using the 1-8 keys, one can change the active texture. The L key enables/disables the default lighting of the viewer.

For the eyes animation, Faceshift messages contain the pitch and yaw values for each eye. At launch we load a spherical mesh and its associated texture twice. We put each eye at its position inside the head. We then update the model matrix of each eye at drawing time, to reflect the movements of the actor's irises.

### 2.4 Real-time Faceshift Streaming

The connection between our viewer and Faceshift is bridged by FSTCPClient. We implemented the message receiving function in our viewer, and pass the viewer as a callback object into the FSTCPClient. Whenever a "readyread" signal is received by the TCP client, the corresponding message receiving function of our viewer will be called, mesh weights and eye tracking will be updated, leading to an instant update of the expression of the styrofoam head model mesh in the viewer. The whole process is so fast that the face tracking animation is in real time.

## 3 Libraries and Materials

- Styrofoam head model: we used two identical head models. One was covered with marks for surface reconstruction in PhotoScan; the other used as the projection target.
- Code from Exercises 7,8 and 9
- Faceshift + sensor (Kinect)
- libQGLViewer: we started using GLUTViewer to show the mesh, but the display loop was blocking the thread, preventing the Faceshift network updates from being received. We had to switch to libQGLViewer in order to run both the Viewer and the TCP server in parallel.
- Faceshift TCP client: we used the code provided by the TAs to handle the communication with Faceshift and the decryption of the messages.

# 4 Results

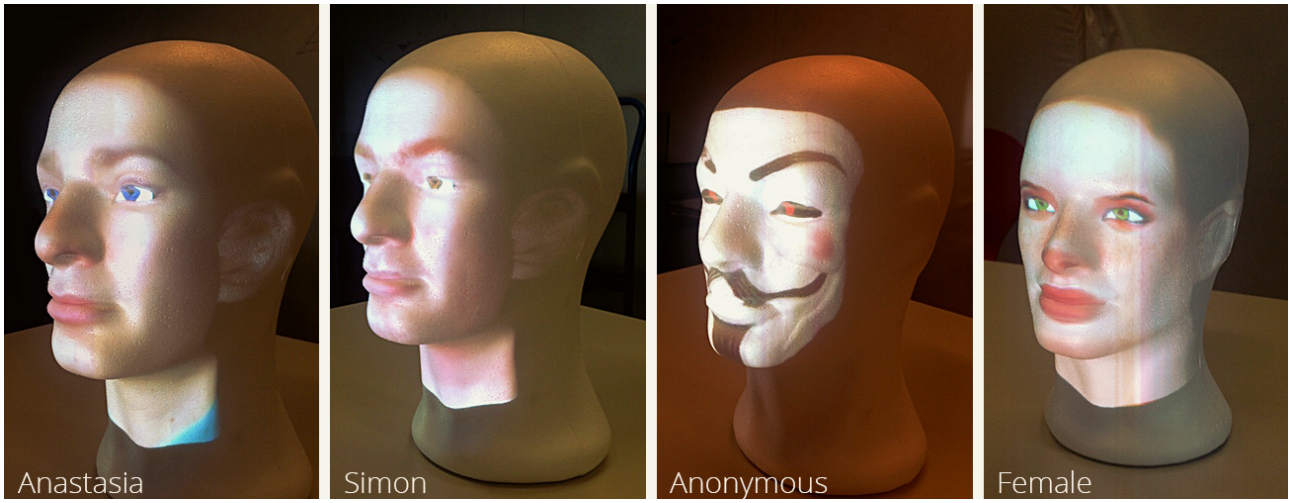


Figure 5: Results of the projection with different textures.

We found the results to be quite convincing, even if there is some kind of "weird effect" when seeing such an animated face. Some limitations arise because of textures quality, and when some wide moves (especially with the mouth) are performed.

# 5 Work Repartition

- Loubna: Textures creation, texture coordinates, loading, Animation of the mesh using expressions meshes.
- Ching-Chia Wang: Surface Reconstruction, Non-rigid Alignment, Deformation Transfer, and TCP Streaming
- Simon: Loading the meshes. OpenGL code (texture handling,...). Loading and moving the eyes. Correction of the centroid bug.