Computer Vision 3 - Assignment 3
2D-to-3D

Yke Rusticus
yke.rusticus@student.uva.nl
11306386

May 30, 2020

1 Introduction

In this work we design a pipeline for constructing a textured model of a face given one or more images. We approach this task by taking into account prior knowledge about human faces, using a multilinear PCA model (Blanz and Vetter (1999)). This model constraints the search space for possible 3D faces, which are defined by a set of latent parameters. Using energy optimization we aim to estimate these parameters, as well as the rotation and translation necessary to model the face. In the following, the components of the pipeline are explained further, and implementation details are given. Step by step, we visualize the effects of the individual components, which together make up our final pipeline. First, we design the pipeline to work for a monocular image, after which we extend it to work for multiple images. In the final section, we conclude and discuss the results.

2 Methods and Implementation

2.1 Morphable model

We used the morphable model Basel Face Model 2017 (BFM)[1]. This model provides means and standard deviations of neutral geometry (facial identity, $\boldsymbol{\mu}_{id}$ and $\boldsymbol{\sigma}_{id}$) and facial expression ($\boldsymbol{\mu}_{exp}$ and $\boldsymbol{\sigma}_{exp}$), as well as their principal components (PC) $\mathbf{E}_{id}$ and $\mathbf{E}_{exp}$. We extracted 30 PC for facial identity and 20 for facial expression. Subsequently, facial geometry is represented as a pointcloud $\mathbf{G}(\boldsymbol{\alpha}, \boldsymbol{\delta})$ as follows:

$$\mathbf{G}(\boldsymbol{\alpha}, \boldsymbol{\delta}) = \boldsymbol{\mu}_{id} + \mathbf{E}_{id}[\boldsymbol{\alpha} \cdot \boldsymbol{\sigma}_{id}] + \boldsymbol{\mu}_{exp} + \mathbf{E}_{exp}[\boldsymbol{\delta} \cdot \boldsymbol{\sigma}_{exp}] \tag{1}$$

where the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\delta}$ are changed to obtain different face geometry.

As a first step in the pipeline, the relevant values are extracted from the model and reshaped to compute Eq. 1 such that $\mathbf{G}(\boldsymbol{\alpha}, \boldsymbol{\delta}) \in \mathbb{R}^{N \times 3}$, where N is the number of vertices in the model. Colors and indices for triangulation of the face are also extracted, as they are necessary to visualize the results.

[1] https://faces.dmi.unibas.ch/bfm/bfm2017.html,model2017-1_face12_nomouth.h5

2.2 Pinhole camera model

After having obtained the 3-dimensional pointcloud $\mathbf{G}(\boldsymbol{\alpha}, \boldsymbol{\delta})$, we can transform and project it. First we add a row of ones to the coordinates in G. Using a transformation matrix $\mathbf{T}$, defined by rotation parameters $\boldsymbol{\omega}$ and translation parameters $\mathbf{t}$, we then transform these coordinates. The result is left-multiplied by the perspective projection matrix $\mathbf{P}$, defined as:

$$\mathbf{P} = \begin{bmatrix} n/r & 0 & 0 & 0 \\ 0 & n/t & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \tag{2}$$

which is a simplified version of the matrix in `https://bit.ly/300gYmf`, as we assume a symmetrical view volume. t is the top of the view, given by $t = n \tan(\text{FOV}/2)$ with FOV the field of view. r is the right edge of the view, given by $r = t \cdot \frac{w}{h}$ where w and h are the width and height of the final view. f and n denote the far and the near cut-off of the view volume. Finally the result is left-multiplied by the viewport matrix $\mathbf{V}$ (simplified from the given source[2], assuming minimum x and y coordinates of 0):

$$\mathbf{V} = \begin{bmatrix} w/2 & 0 & 0 & w/2 \\ 0 & h/2 & 0 & h/2 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3}$$

where w and h denote the width and height of the resulting view. After transformed and projected coordinates ($\hat{x}$, $\hat{y}$, $\hat{z}$, $\hat{d}$) are obtained, we divide them by $\hat{d}$ to get u, v, z coordinates.

Now we can simulate taking a 2D picture of our 3D face by applying the pinhole camera model. Using vertex annotations of landmarks in `Landmarks68_model2017-1_face12_nomouth.an` we can find the landmarks of our modeled face.

2.3 Latent parameters estimation

Given a "real" image of a face, we detect ground-truth landmarks with the provided `detect_landmark` function. We then use our pipeline up till now from front to end to model a 3D face with parameters $\boldsymbol{\alpha}$, $\boldsymbol{\delta}$, $\boldsymbol{\omega}$, and $\mathbf{t}$ and derive "predicted" landmarks. All parameters are initialized appropriately (see Experiments for more details). We iteratively update the parameters to optimize for the loss

$$\mathcal{L}_{fit} = \mathcal{L}_{lan} + \mathcal{L}_{reg}, \tag{4}$$

in order for the model to better fit the given image. The loss is implemented as specified in the assignment. Once the last two loss values differ less than a given threshold, the model is said to have converged, and the iterating stops.

[2] `http://glasnost.itcarlow.ie/~powerk/GeneralGraphicsNotes/projection/viewport_transformation.html`

2.4 Texturing

Our 3D face looks much more realistic if we use corresponding colors from the provided image. For each vertex in our model, we use its uv-coordinates to select the corresponding colors in the given image. A naive method would be to round the coordinates to the nearest integers and select the color of corresponding pixel in the image. However, we can improve upon this method by using bilinear interpolation of colors of the four nearest pixels, given the uv-coordinates of a vertex. In our approach, we follow the example of the final section of the provided source[3]. After obtaining these colors, we use them to replace the earlier used colors of the BFM.

2.5 Energy optimization using multiple frames

As a final step we extend our pipeline to work with multiple images. In this setting, each image i has its own rotation, translation and expression parameters $\boldsymbol{\omega}_i$, $\mathbf{t}_i$ and $\boldsymbol{\delta}_i$, and neutral geometry parameters $\boldsymbol{\alpha}$ are shared. So the only differences with the single image pipeline are that we keep track of the individual parameters per image and we optimize for the adjusted loss $\mathcal{L} = \sum_i^M \mathcal{L}_i$, where each $\mathcal{L}_i$ is calculated according to Eq. (4), with the image specific parameters for $\boldsymbol{\omega}$, $\mathbf{t}$ and $\boldsymbol{\delta}$, and the shared parameters for $\boldsymbol{\alpha}$.

3 Experiments

The following subsections and results are structured in the order as they are presented or asked for in the assignment.

3.1 Morphable model: 3D visualization

We first sample $\boldsymbol{\alpha} \sim \mathrm{U}(-1, 1)$ and $\boldsymbol{\delta} \sim \mathrm{U}(-1, 1)$, and calculate $\mathbf{G}(\boldsymbol{\alpha}, \boldsymbol{\delta})$ to obtain the point-cloud of a face. We use `save_obj` together with the colors and triangles, to obtain 3D meshes. Results of three samples are visualized in Figure 1. We can see that the faces show variation in both facial geometry and expression, which is expected because we vary both $\boldsymbol{\alpha}$ and $\boldsymbol{\delta}$.

3.2 Pinhole model and landmarks

To see the effect of the transformation matrix applied to the pointcloud, we refer to Figure 2. This figure shows meshes as obtained in previous section, but left-multiplied by their corresponding transformation matrices. As can be seen, the faces are placed in the origin by default. However, when modeling a pinhole camera, the camera is also placed at the origin. Therefore we need to either translate the camera or the face, to get a clear view of the face when seen from the camera. In the right-most figure we translate the face over the z-axis by -500, which we can then use to model the pinhole camera. The results

[3] `https://en.wikipedia.org/wiki/Bilinear_interpolation`

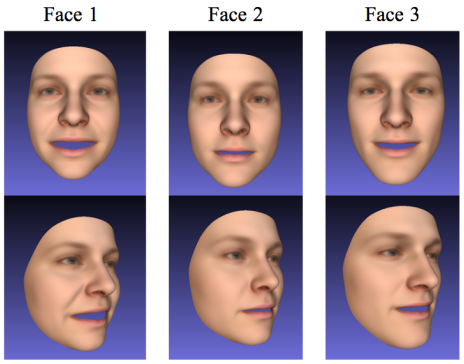

Figure 1: Three different faces defined by different (uniformly sampled) $\boldsymbol{\alpha}$ and $\boldsymbol{\delta}$. The top row shows front view. The bottom row shows the same faces, but slightly turned.

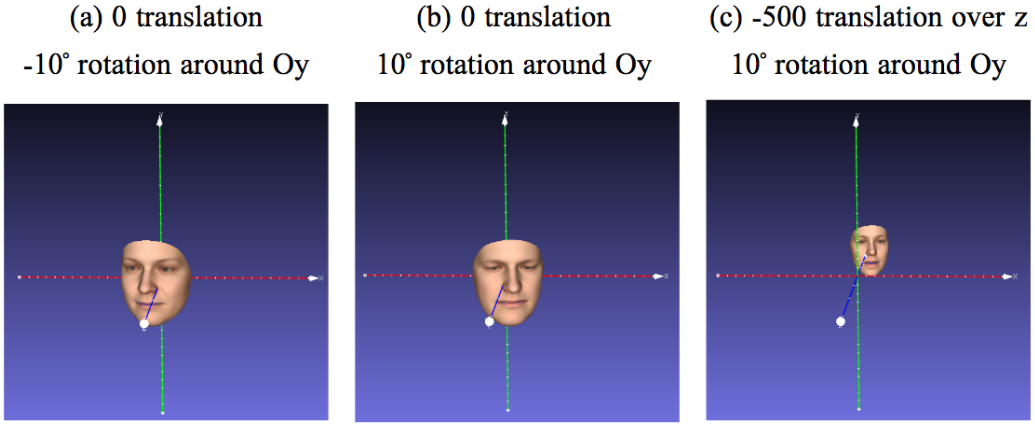

Figure 2: Different transformations applied to different faces. In (a) and (b) we rotate around the y-axis (green) and in (c) we translate over the z-axis (blue).

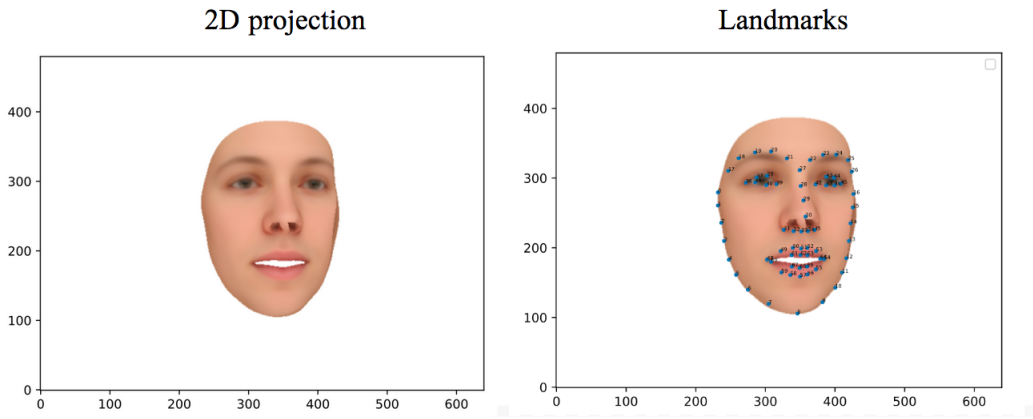

Figure 3: 2D projection of the face in Figure 2c, and overlayed landmarks with indices (0-68).

of subsequently applying the perspective projection matrix and the viewport matrix is visualized in Figure 3. We used a field of view of $40°$ and near n and far f of 10 and 1000 respectively. A width $w = 480$ and height $h = 640$ were used. Using provided landmark indices, we scatter the landmarks of this face over the obtained projection in Figure 3.

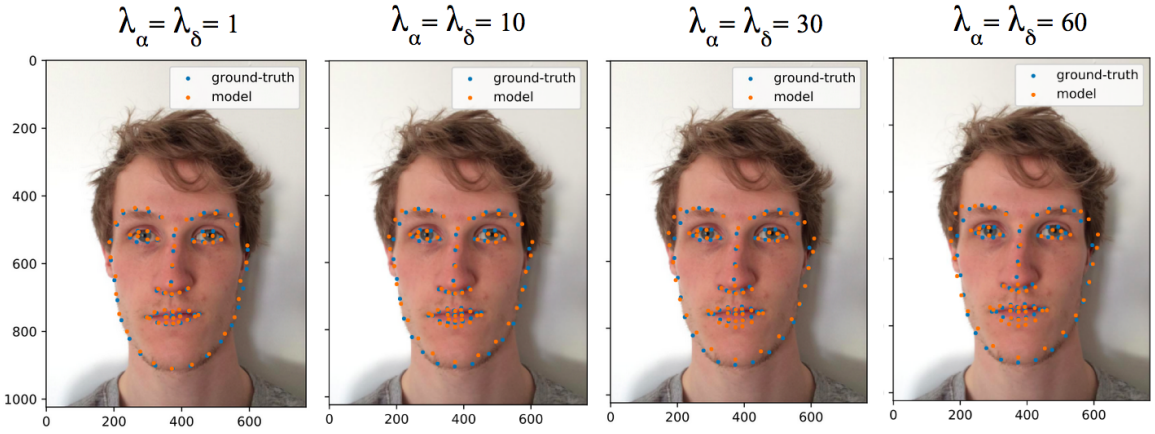

Figure 4: Ground-truth (detected) and predicted (modeled) landmarks for different parameter settings.

Setting	μ_α	σ_α	α range	μ_δ	σ_δ	δ range
$\lambda_\alpha = \lambda_\delta = 1$	0.61	1.6	[-2.6, 4.6]	-0.94	2.2	[-5.6, 4.6]
$\lambda_\alpha = \lambda_\delta = 10$	0.064	0.64	[-1.5, 1.4]	-0.26	0.68	[-1.5, 1.2]
$\lambda_\alpha = \lambda_\delta = 30$	0.0071	0.34	[-1.2, 0.65]	-0.13	0.32	[-0.79, 0.49]
$\lambda_\alpha = \lambda_\delta = 60$	-0.0031	0.22	[-0.82, 0.37]	-0.076	0.19	[-0.51, 0.26]

Table 1: Different hyperparameter settings and their effect on resulting latent parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\delta}$, based on a single run.

3.3 Latent parameter estimation

Using `detect_landmark`, ground-truth landmarks were detected from a face close to frontal and neutral, shown in Figure 4. These ground-truth landmarks were used to optimize latent parameters $\boldsymbol{\alpha}$, $\boldsymbol{\delta}$, $\mathbf{t}$ and $\boldsymbol{\omega}$, with the loss as provided in the assignment. All parameters were initialized uniformly between -1 and 1, except for t_z, which was initialized to -500. We used the Adam optimizer in PyTorch with learning rate 0.1. Several settings with different values for regularization parameters λ_α and λ_δ were experimented with. Figure 4 shows detected and predicted landmarks for the different settings after convergence (convergence threshold is set to 1). Although differences are small, we can see some points diverge from the ground-truth more and more as the regularization parameters are increased. To select the hyperparameters, we rely on the statistics presented in Table 1. For realistic results, we aim the parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\delta}$ to have a range between -1 and 1. The logical choice seems therefore to choose either $\lambda_\alpha = \lambda_\delta = 10$ or $\lambda_\alpha = \lambda_\delta = 30$. For further experiments we use $\lambda_\alpha = \lambda_\delta = 30$, because this setting showed more stable results in different situations. Different settings could be considered, e.g. where $\lambda_\alpha \neq \lambda_\delta$ or where the parameters have more fine-tuned values between 10 and 30, but we leave that for future work.

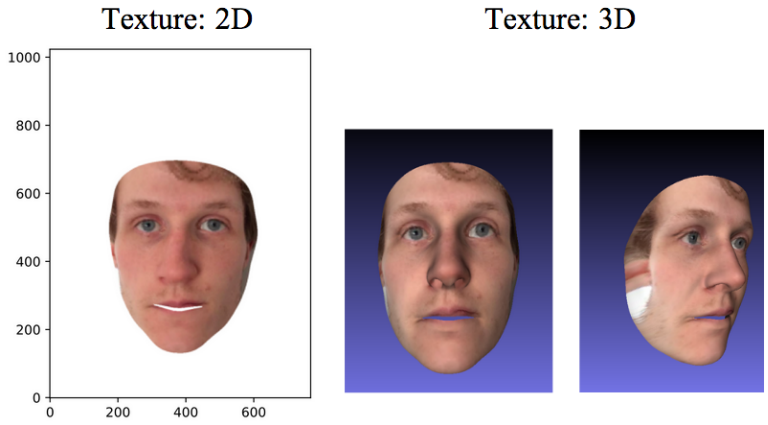

Figure 5: Modeled faces with added texture. The left image shows the 2D rendered image using `render`. The right two images show the 3D visualization in Meshlab.

3.4 Adding texture

For each vertex, an interpolated color is found using the provided image from Figure 4. Using the provided `render` function, the result is visualized in Figure 5, as well as the 3D visualizations of the textured mesh in MeshLab. We see that the model has done a reasonable job, as the modeled face quite accurately follows the contours and the colors of the face in the provided image.

3.5 Multiple images

The pipeline is complete after making it compatible for multiple images at once. With multiple images, facial geometry is better constraint, while the other parameters can still vary across images. Figure 6 shows the results after optimization for 4 frames simultaneously. Two things can be noticed from these results. Firstly, the model struggles to capture contours of the mouth correctly from the first frame, possibly because the smile is slightly asymmetrical or because the expression deviates too much from the mean facial expression. Secondly, the modeled face of the fourth frame exceeds the bounds of the actual face, resulting in background colors projected onto the cheek in the final result. Apart from these minor flaws, the model does produce realistically looking 3D face models based on 2D images, which allow for inspection from different angles than in the provided images.

4 Conclusion and Discussion

In previous work we experimented with depth based 3D reconstruction (i.e. ICP) and texture based 3D reconstruction (i.e. SFM). The ICP algorithm is relatively simple and straight forward, but suffers from the fact that it could get stuck in a poor local minimum. SFM allows for more tuning of the algorithm, which could lead to better results. However, this method (and also ICP) requires multiple frames and does not take into account prior knowledge about the object. In order for ICP to work well, it needs reasonable initial-

Figure 6: Results after optimizing for 4 frames simultaneously. Each frame shows slightly different facial expression, rotation and translation, while facial geometry is modeled identically for each frame. The first row shows the provided images, which are overlayed with landmarks in the second row. The center row showes textured 2D projection of the modeled faces using `render`. The last two rows show textured 3D faces in MeshLab, respectively unchanged and slightly tilted to the left.

ization. This is where ICP and SFM can be combined to possibly produce better results. Given a set of frames we can use SFM to derive pointclouds of the object, which can subsequently be used to initialize the ICP algorithm. The problems remain that we would need multiple frames to produce good results, and prior knowledge is still not taken into account. The advantage of our recently introduced 2D-to-3D reconstruction method, is that it works quite well for only a few images (even a single one) and it takes into account prior knowledge, which constraints the search space of our 3D object. The advantage of this method is that it is very data efficient: a reasonable 3D construction is obtained even from a single image. However, this method only works well for objects with well defined geometries. We had access to a detailed model describing facial geometries and expression, but this might not be available for other objects (e.g. a tree). Nevertheless, for objects which are well described it is worth combining ICP, SFM and this method to attempt to improve results even further, as the latter method provides exactly what the former two were lacking: prior knowledge and data efficiency. Originally, we did not constrain the 3D constructions produced by ICP or SFM the way we did in this work, but by doing so we could obtain more realistic results. Future work will have to investigate the possible new advantages and disadvantages of combining these methods.

References

Blanz, V., & Vetter, T. (1999). A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on computer graphics and interactive techniques* (pp. 187–194).