Face Reconstruction

Final Report

Luong, Min Ting Neugebauer, Philip Saribas, Berk Emre

Bentellis, Amine

August 19, 2021

Abstract

Our goal is to reconstruct a 2D face model based on a single RGB image. To accomplish this, we first detect facial landmarks on the RGB image. The Basel Face Model's landmarks are given by the model. After we detected the facial landmarks, we learn the weights of the parameters of the 3D model. We want to find the camera pose and the weights for the expression, shape and the color of the face. For this we use a sparse term based on the facial landmarks, a dense term based on the image color and a statistical regularizer.

At the end our generated 2D model can be used to transfer the expression from a person in one image to a person in another image. The target image will then have the same facial expression of our input source image.

1 Introduction

Face reconstruction has various use cases in the industries, where we want to construct a virtual face on images, a sequence of images (video) or models. In the field of acting it can be used for creating facial expressions for fictional characters in the film production. It can also be used for dubbing, where the mouth motion of the different voice actors is reconstructed on the target actor in the movie to adjust the mouth movements shown in the film to the dubber's mouth movements.
Another area, where face reconstruction is utilized, is in virtual reality. Face reconstruction in virtual reality can be used to generate virtual avatars for online meeting sessions. The user's face, face expression and eye gaze direction will be rendered onto the avatar's face [6].
The rise of the popularity of video filters and image filters on social media form a huge

marketing place for companies. Reconstructed face filters can be used to advertise brands similar to AR filters. Fast algorithms allow for face reconstruction on mobile devices [8].

In the following, we present how we converted a 2D image into a 2D model. At the end we show that we can render the facial expression from one person in an image onto another person in a differing image.

2 Related Work

2.1 Dlib

Dlib is an open source toolkit providing a wide range of methods for C++. These methods include machine learning and numerical algorithms, as well as image processing utilities. [2]

2.2 Basel Face Model

The Basel Face Model is a model that adds principle components for facial expression, shape and color together to form a 3D face model. The principle components can be differently weighted to create a broad variety of face models. [1]

2.3 FLAME

The face model FLAME (Faces Learned with an Articulated Model and Expressions) is trained on 33.000 face scans and includes the head, eyes and the neck. The model considers pose, shape, expression and color as input parameters and works by using pose-dependent corrective blendshapes and additional global expression blendshapes [3]. Indeed, different actuation of the eyeballs, jaw and neck joints can influence the pose blendshapes. FLAME is running in Python and uses Pytorch for the learning part. Overall, FLAME model is more expressive and realistic than BFM or FaceWareHouse. However, we could not use it for our task since most of our optimization code was C++-based and we found no convenient way to transfer the Pytorch model over.

2.4 Face2face

The paper "Face2Face: Real-time Face Capture and Reenactment of RGB Videos" from Thies et al. [7] presents how to create a 3D reconstruction of a face observed in an RGB stream. It uses a principle component analysis model. This model consists of two dimensions for geometric shape and skin reflectance and a dimension that represents facial expression. Furthermore, illumination is approximated by using Spherical Harmonics[5]. The weight parameters for each dimension of this model then get trained by minimizing an energy term. This term consists of a term for photo-consistency for each pixel, feature alignment and statistical regularization. Thies et al. optimize the energy by using a GPU based Iteratively Reweighted Least Squares (IRLS) approach. Using this optimization strategy leads to good performance so that the face reconstruction can be

calculated in realtime. With this reconstruction, Thies et al. perform expression transfer for face reenactment. For this they formulate an energy term based on the expression of the source and target and minimize it in realtime on a GPU.

3 Method

3.1 Landmark Detection

We use DLib to extract facial landmarks from input RGB images. This landmark extraction can be performed either on single images or at runtime on video streams on a CPU. The landmark detection works good for faces looking at the camera, but fails if the face is rotated more than a few degrees. It still works sometimes in low light settings and can keep track of fast head movement, even if the face is blurry. It can also detect small faces and multiple faces in one image. The algorithm detects 68 landmarks per face, at the same positions as the Basel Facel Model detects facial landmarks. Figure 1 shows an example of the landmarks detected on an RGB image.

Figure 1: Detected Facial Landmarks

3.2 Face Imagery Synthesis

We used Basel Face Model [1] for our project. BFM model is a PCA model providing us mean shape, expression and color data with basis functions and their corresponding standard deviations to morph the face. We denote for n vertices, average shape $a_{shape} \in R^{3n}$, average expression $a_{expression} \in R^{3n}$, average color $a_{color} \in R^{3n}$ and their basis functions $E_{shape} \in R^{3n \times 199}$, $E_{expression} \in R^{3n \times 100}$, $E_{color} \in R^{3n \times 199}$ with corresponding standard deviations $\sigma_{shape} \in R^{199}$, $\sigma_{expression} \in R^{100}$, $\sigma_{color} \in R^{199}$. We parametrize a face as:

$$Face_{geometry} = a_{shape} + a_{expression} + E_{shape} \times \alpha + E_{expression} \times \beta \tag{1}$$

$$Face_{color} = a_{color} + E_{color} \times \theta \tag{2}$$

We then define a pinhole camera model with translation matrix t, rotation matrix R and a projection matrix. We use perspective projection constructed with fov (field of view) angle. We transform face vertices with R and t then project it to a 2d plane using the projection matrix and viewport matrix (defined by input image width and height):

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = [V] \times [P] \times \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3}$$

In this project, we will be optimizing for face weights α, β, θ and camera parameters R, t and fov.

3.3 Energy Formulation

We introduce two energy terms for optimization, namely sparse and dense terms based on the method introduced in [7]. We use sparse term to do point to point optimization between landmarks detected in the input image (see figure 2) and BFM landmarks. Using sparse term solver, we optimize face weights α, β and camera parameters R, t and fov. In the result, we get the face without colors.
Next, we render the face with model and camera parameters that were optimized. We then use dense term to do point to point color optimization between the input image and the rendered face image. We optimize color weights θ for every pixel in both the input image and the rendered face image.

Figure 2: Input image we use throughout the method section

3.3.1 Sparse Energy Function

For the sparse term energy cost function, we set residuals for x and y coordinates. We calculate the norm between projected coordinates of landmarks from BFM model and coordinates of the corresponding landmark in the input image.

$$\sum_{j=1}^{68} \|P_j - I_j\|_2^2 \tag{4}$$

We first optimize the camera parameters using sparse term cost function. This is done to align our face model with the input image, see Figure 3a. We could have use Procrustes alignment to do that, but we could not make it work. Then we optimize for both camera parameters and face weights using the same cost function. In the results, we fit our face model to the input image (Figure 3b).

(a) Camera parameters optimized

(b) Camera parameters and face weights optimized

3.3.2 Dense Energy Function

Dense term energy function is applied for every pixel in the input image and the rendered image. We optimize color weights for each pixel in the both input image and rendered result of the morphed model.

$$\sum_{y=0}^{height} \sum_{x=0}^{width} |P_{x,y} - I_{x,y}| \tag{5}$$

We need to know which primitive triangle was used to render a specific pixel. Thus, in addition to normal rendering, where we render vertex colors, we also render triangle ids. To do that, we separate triangle id (16-bit integer) as 8-bit values. We then use this values as vertex colors. We then use the render result as a lookup texture to find corresponding triangle for each pixel (figure 4).

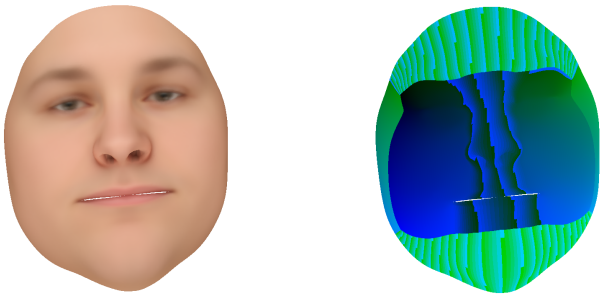

Figure 4: Albedo render on the left, Triangle ID render on the right

With a triangle id, we find the corresponding three vertices for that triangle. In the cost function, we transform and project vertices to their corresponding 2D location. We then calculate barycentric weights for each one of the 2D vertex locations according to the center of the pixel. We then set our residuals for red, green and blue channels. For each channel, we calculate the average triangle color for the current pixel using barycentric weights and get the norm with the input pixel's corresponding channel. After the optimization we get the results in Figure 5.

Figure 5: Color weights optimized

3.3.3 Regularization

We also implemented regularization cost functions to enforce the parameters to stay statistically close to the mean. The regularized parameters are the expression, shape and color weights. We set different weights for each regularization term. It prevents overfitting to the input image and can solve degeneration of facial geometry caused by noisy depth maps. We also planned to implement a Laplacian term to prevent vertex displacement and act as a smoothing prior. However we were not able to correctly

implement it and we will talk about it in the Future Work section of this article.

3.4 Improved Color Results and Blending

The results we get after dense term optimization is not particularly great. The resulting image is too noisy even though it resembles the original face. To further improve, we implemented another depth term. This time, instead of optimizing color weights, we directly optimized vertex colors, a_{color}. The resulting image is directly the same with the input image. Even though it looks great, since no weights were trained in this method the colors of invisible sides of the face cannot be determined. So, we blended the results from this method with the results we get from dense term. Blend results can be seen in Figure 6. We used this method to improve our results from previous step.

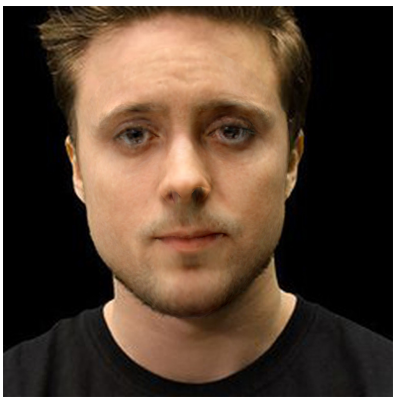

Figure 6: Color weight results blended with pixel colors

4 Results

We represent results from different images in Figure 7. We get visually similar results with our reconstruction method. However, we get some artifacts due to our color method, if landmark detection has issues. In Figure 7, at the bottom we see actor Tom Cruise's smile is not perfectly translated in the fitted model. Because of that, when we apply our color method, we get some of his teeth appear on the lips.
We also tested if we could transfer the expression we learn from one face to another. We optimized a face with a sad expression. We did not optimize colors for that image since we only need the expression weights. We then optimized another face and then rendered it with the expression weights from the previous image. Results can be seen in Figure 8.

Figure 7: In order, input image, fitted image and result with color

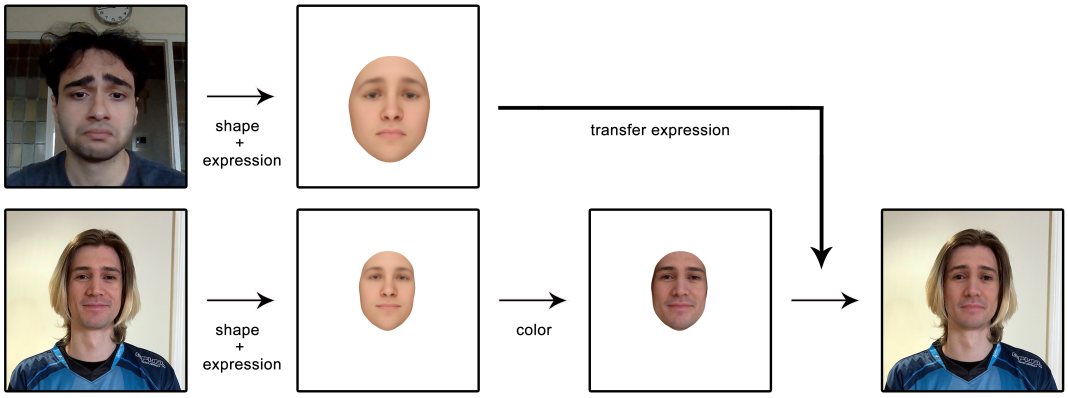

Figure 8: Method for expression transfer

5 Future Work

The result of our approach can be refined with Procrustes for the initial alignment of the facial landmarks, by using RGB-D images we make use of the depth information for 3D reconstruction and by introducing the Laplacian term we will get a smoother model as a result. In the future, we could use a sequence of images (video) as an input for reconstructing the face models in real-time.

5.1 Procrustes Alignment

We tried to apply the Procrustes Alignment as described in the lecture/ exercise to the problem. The resulting poses do not match the input image, when rendered with our renderer, and the output values do not match the one our Sparse Energy Term computes. One of our ideas on what causes the algorithm to fail, is that the renderer (which is used by the sparse term) needs the camera position as pose input. Whereas we calculated the pose (transformation matrix) of the model that shifts the model to match the RGB image.

5.2 Depth (4D Images)

We experimented with 4D images which contained depth data. However, we could not bring the depth in our synthesized image to the same space with RGBD images. We tried to normalize both depth values between 0 and 1 but again our results were not good. This could be either because depth terms does not correspond to each other, because of our normalization, or because the dataset we used might not have been the most efficient [4].

5.3 Laplacian Term

Laplacian surface editing is a method to enforce smoothness on deformations. We first look at the transformation for a single vertex and its neighbours. The goal is to minimize differences of those deformations so that we can change the shape while respecting the structural geometric detail.

5.4 Sequence of Images

Instead of using just one input frame, we could reconstruct a face on a sequence of input frames.

6 Conclusion

Our approach with the color term produces valuable results on 2D images. Unfortunately, it is far from being real-time, because currently it takes about multiple minutes to train a face on the CPU. The calculations need to be done on GPU to make it faster (i.e. use CUDA).
Our measurements in Table 1 show that the most time is needed for optimizing the dense color term. Without it, the calculation only needs few seconds. These few seconds include the setup, e.g. a second for the landmark detection, so it might run almost in real time if implemented for a stream of images. When running on a stream, the parameters from the last frame can be used as initialization for the parameters of the current frame, which might reduce the time for optimization. The time for optimizing position, rotation and FoV could probably be optimized by using the Procrustes algorithm instead.

Landmark detection	0.084s
Optimization of position, rotation and FoV	0.3s
Optimization of position, rotation, shape weights and expression weights	0.1s
Reconstruction without color	2.9s
Reconstruction with color	254.2s
Total time for transfer	257.2s

Table 1: Performance Measurments on a Laptop with a Ryzen 5800h CPU (8 cores, 16 threads, 3.2 GHz base clock, 4.4 GHz boost clock, Turbo mode active)

References

[1] IEEE. *A 3D Face Model for Pose and Illumination Invariant Face Recognition*, Genova, Italy, 2009.

[2] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[3] Tianye Li, Timo Bolkart, Michael. J. Black, Hao Li, and Javier Romero. Learning a model of facial shape and expression from 4D scans. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia)*, 36(6):194:1–194:17, 2017.

[4] Giorgia Pitteri, Matteo Munaro, and Emanuele Menegatti. Depth-based frontal view generation for pose invariant face recognition with consumer rgb-d sensors. volume 531, 07 2016.

[5] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 117–128, New York, NY, USA, 2001. Association for Computing Machinery.

[6] J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, and M. Nießner. Facevr: Real-time gaze-aware facial reenactment in virtual reality. *ACM Transactions on Graphics 2018 (TOG)*, 2018.

[7] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Niessner. Face2face: Real-time face capture and reenactment of rgb videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[8] Phuc Truong, C.-W Park, Minsik Lee, S.-I Choi, Sanghoon Ji, and G.-M Jeong. Rapid implementation of 3d facial reconstruction from a single image on an android mobile device. *KSII Transactions on Internet and Information Systems*, 8:1690–1710, 05 2014.