Application of human face images on 3D head models using machine learning techniques

Iacopo Erpichini
iacopo.erpichini@stud.unifi.it

Jason Ravagli
jason.ravagli@stud.unifi.it

Università degli Studi di Firenze
Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione
Course of Computer Graphics & 3D

17 February 2021

Table of Contents

Goal of the project

- Given a 3D model of a human head, we want to apply an image of a human face on it using machine learning techniques
- We want to merge the image and the model in a realistic way, to get a convincing rendering of the model with the face applied on it
- We used the **Neural Renderer** framework to have an embedded and trainable rendering tool
- In our experiments we used the BFM model as 3D head model and images of human faces taken from the CelebAMask-HQ dataset

3D Representation and Rendering

- The process of generating a 2D image from a 3D representation is called **rendering**
- Various representation exists for a 3D model: voxels, point clouds, polygon meshes

Polygon meshes

- In this work **polygon meshes** are considered: more compact, memory efficient and more suitable for transformation
- Rendering of a mesh consists of projecting its 3D points onto the 2D coordinates system of the screen and then generating an image through a grid sampling (**rasterization**)

3D Representation and Rendering

- The process of generating a 2D image from a 3D representation is called **rendering**
- Various representation exists for a 3D model: voxels, point clouds, polygon meshes

Polygon meshes

- In this work **polygon meshes** are considered: more compact, memory efficient and more suitable for transformation
- Rendering of a mesh consists of projecting its 3D points onto the 2D coordinates system of the screen and then generating an image through a grid sampling (**rasterization**)

Neural Renderer

- **Neural Renderer objective**: embedding the rendering process into a trainable neural network
- To enable end to end training of such a network it is necessary to backpropagate the gradient through the rendering process

Problem

Rasterization is a discrete (i.e. non differentiable) operation

Solution

The authors defined an approximated gradient

Neural Renderer

- **Neural Renderer objective**: embedding the rendering process into a trainable neural network
- To enable end to end training of such a network it is necessary to backpropagate the gradient through the rendering process

Problem

Rasterization is a discrete (i.e. non differentiable) operation

Solution

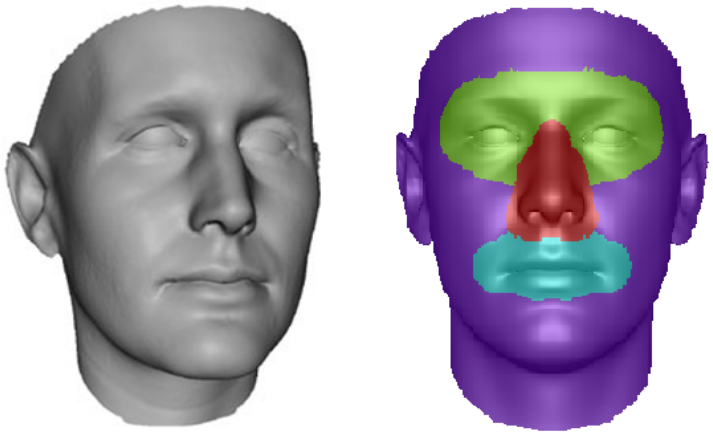
The authors defined an approximated gradient

Neural Renderer: Usage

- With these end-to-end trainable systems it is possible to define a neural network that optimizes some defined 3D model and rendering parameters (e.g. camera position, mesh vertices position, texture of the rendered mesh)
- It is sufficient to embed the neural renderer onto a neural network model and define inputs, parameters to optimize and a **proper custom loss**

Data - BFM 2009

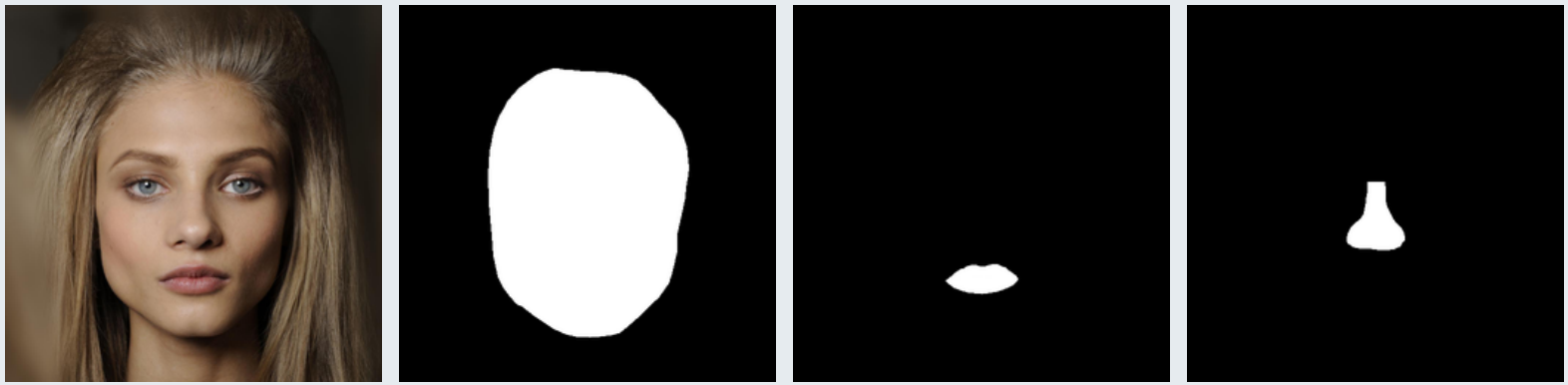
- The **Basel Face Model (BFM)** is a 3D human face model obtained from multiple scans of different real human faces
- We used the 2009 version that includes vertices segmentation for skin, mouth, nose and eyes
- As we will see, vertices segmentation is crucial for the camera optimization
- The geometry of BFM consists of 53,490 3D vertices and 160,470 triangles



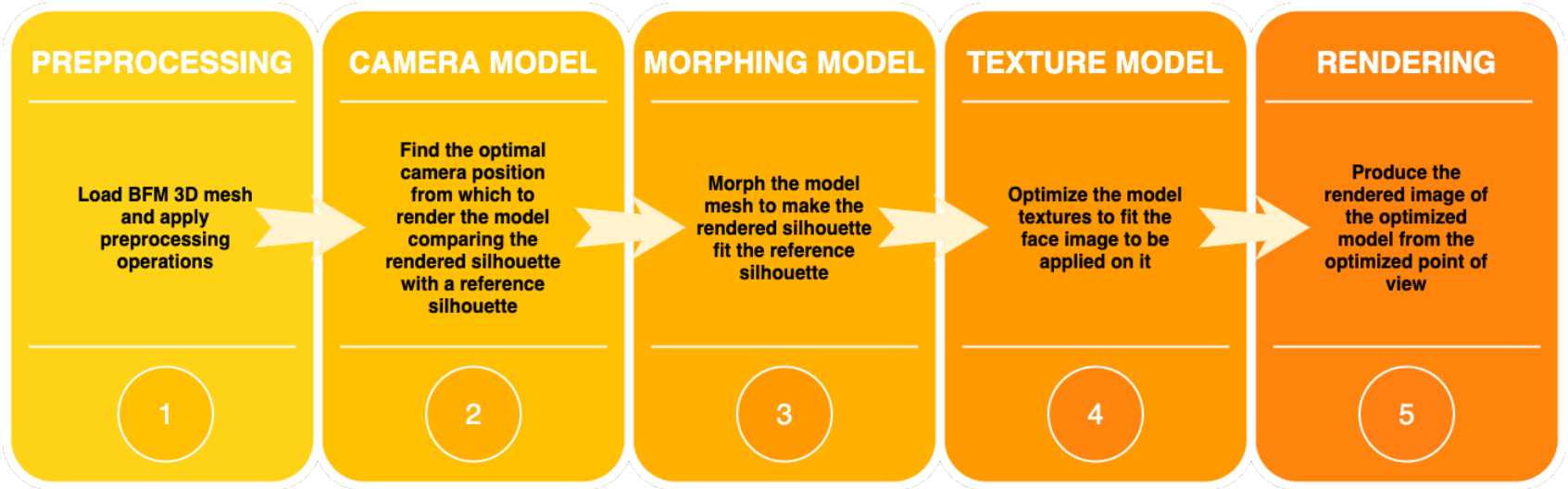
Data - CelebAMask-HQ

- **CelebAMask-HQ** is a large-scale face image dataset that has 30,000 face images
- Each image has segmentation masks of facial attributes
- The images considered in our work are 256x256 pixels

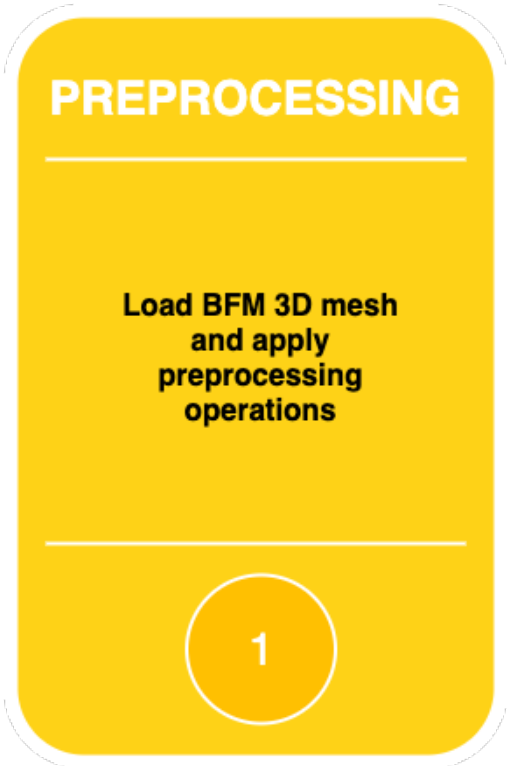
CelebAMask-HQ - Example



Our Pipeline

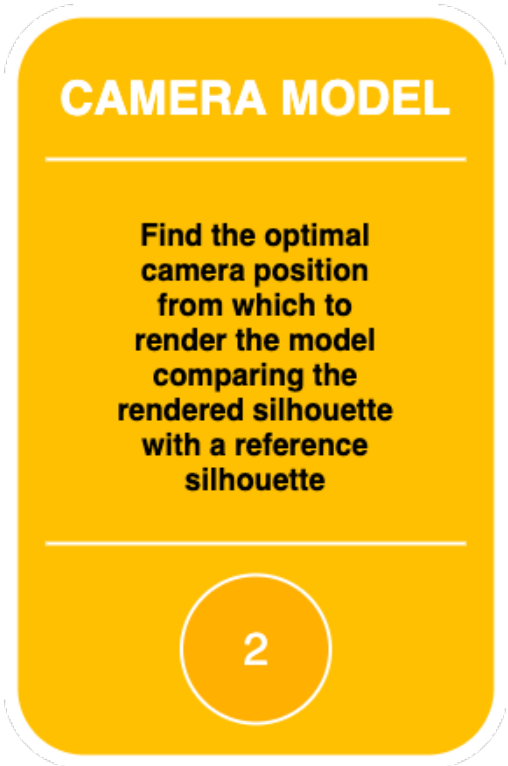


Data loading and preprocessing



- Load BFM model 2009 with vertices, faces, textures and segmentation (mouth, nose)
- Scale vertices coordinates in [0,1] and translate the center of mass in the origin
- Load an image from CelebMask-HQ to be rendered on the 3D model, along with its face, mouth and nose masked regions
- Images are resized to 256x256

Camera Model



- **Input**:
  - 3D mesh model
  - Coordinates of the camera starting point (x,y,z)
  - Face, mouth and nose regions of the image to be rendered on the model
- **Output**: optimized camera position from which to render the model

Camera Model

- The model tries to **optimize the camera position** comparing the silhouette of the rendered model with the reference silhouette. This reference silhouette is the face region of the image that we want to render on the 3D model.
- Besides comparing the global silhouettes, the model also uses the mouth and nose silhouettes as additional **anchor points** to guide the optimization
- We add **constraints** the camera position to avoid undesired divergences

Camera Model

Camera Model - Functioning

Camera Model - Loss

The Camera Model loss has 3 components that sum up together:

- Silhouette image loss
- Anchor points loss
- Constraints loss

Camera Model - Loss

$$L = L_{Img} + L_{Anch} + L_{Constr}$$

Camera Model - Loss

Silhouette image loss

$$L_{Img} = \sum_i \sum_j (I_{ij} - I_{ij}^{Ref})^2$$

Where:

- I_{ij} is ij-th the pixel of the image containing the silhouette of the rendered model image
- I_{ij}^{Ref} is the ij-th pixel of the image face segment that we want to render on the 3D model

Camera Model - Loss

Anchor points loss

$$L_{Anch} = \sum_i \sum_j (I_{ij}^{Nose} - I_{ij}^{NoseRef})^2 + \sum_i \sum_j (I_{ij}^{Mouth} - I_{ij}^{MouthRef})^2$$

Where:

- I_{ij}^{Nose} and I_{ij}^{Mouth} are the ij-th pixels of the images respectively containing the nose and mouth silhouette of the rendered model image
- $I_{ij}^{NoseRef}$ and $I_{ij}^{MouthRef}$ are the ij-th pixels of the images containing the nose and mouth segments of the image that we want to render on the 3D model

Camera Model - Loss

Constraints loss

- For modeling the constraints loss we needed a more natural coordinates system than the cartesian one (x,y,z) to express the camera position
- Considering the model face as laying on the xy plane and the z axis as exiting from front of the face, we defined:
  - d (distance): it corresponds to the z coordinate of the camera
  - θ_{lr} (left-right angle): rotation angle around the y axis
  - θ_{ud} (up-down angle): rotation angle around the x axis

Camera Model - Loss

Constraints loss

$$L_{Constr} = e^{w * \frac{1}{(d^{min} - d)^2}} + e^{w * \frac{1}{(\theta_{lr}^{max} - (\theta_{lr}^{start} - \theta_{lr}))^2}} + e^{w * \frac{1}{(\theta_{ud}^{max} - (\theta_{ud}^{start} - \theta_{ud}))^2}}$$

Where:

- θ_{lr}^{max} is the maximum allowed left-right angle variation
- θ_{lr}^{start} is the starting left-right angle
- θ_{ud}^{max} is the maximum allowed up-down angle variation
- θ_{ud}^{start} is the starting up-down angle
- w is a weight coefficient empirically chosen

Morphing Model

- **Input**:
  - Mesh model
  - Optimized camera position
  - Face region of the image to be rendered on the model (reference silhouette)
- **Output**: mesh vertices optimized to fit the reference silhouette

Morphing Model

- The model modify the position of the mesh vertices to make the rendered image silhouette fitting the reference silhouette
- The silhouette of the rendered image is taken from the camera position optimized in the previous step
- Optimization is run for few iterations: after a while the model tends to create unnatural spikes on the 3D mesh

Morphing Model

Camera Model - Functioning

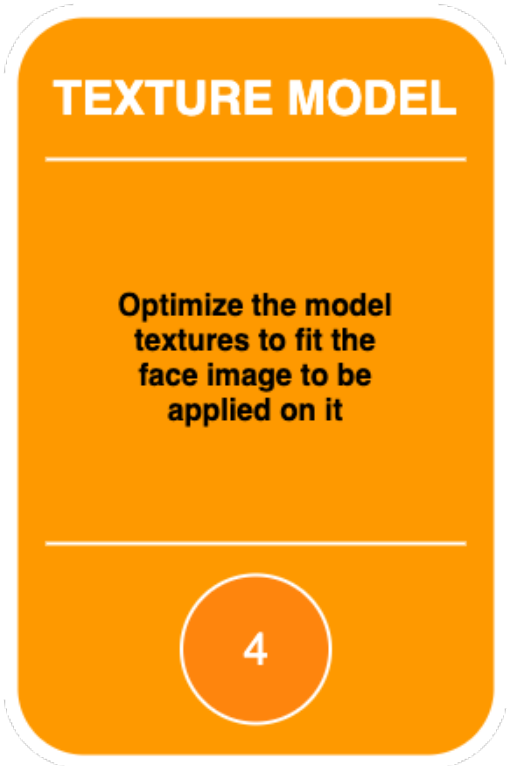
Morphing Model - Loss

Morphing Model - Loss

$$L = \sum_i \sum_j (I_{ij} - I_{ij}^{Ref})^2$$

- I_{ij} is the ij-th pixel of the image containing the silhouette of the rendered model image
- I_{ij}^{Ref} is the ij-th pixel of the reference silhouette

Texture Model



- **Input**:
  - 3D mesh model with base textures
  - Optimized camera position
  - Image to attach on the 3D model (texture image)
- **Output**: mesh textures optimized to fit the texture image

Textures Model

- Starting from the BFM base textures, the optimization process modifies the model textures in order to obtain a rendered image as similar as possible to the texture image
- The model image is rendered from the optimized camera position

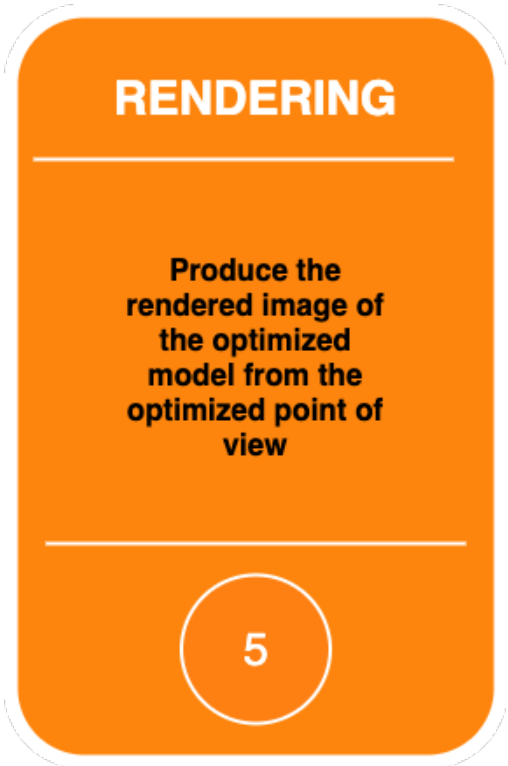
Textures Model - Loss

Textures Model - Loss

$$L = \sum_i \sum_j (I_{ij} - I_{ij}^{Tex})^2$$

- I_{ij} is the ij-th pixel of the rendered model image
- I_{ij}^{Tex} is the ij-th pixel of the texture image

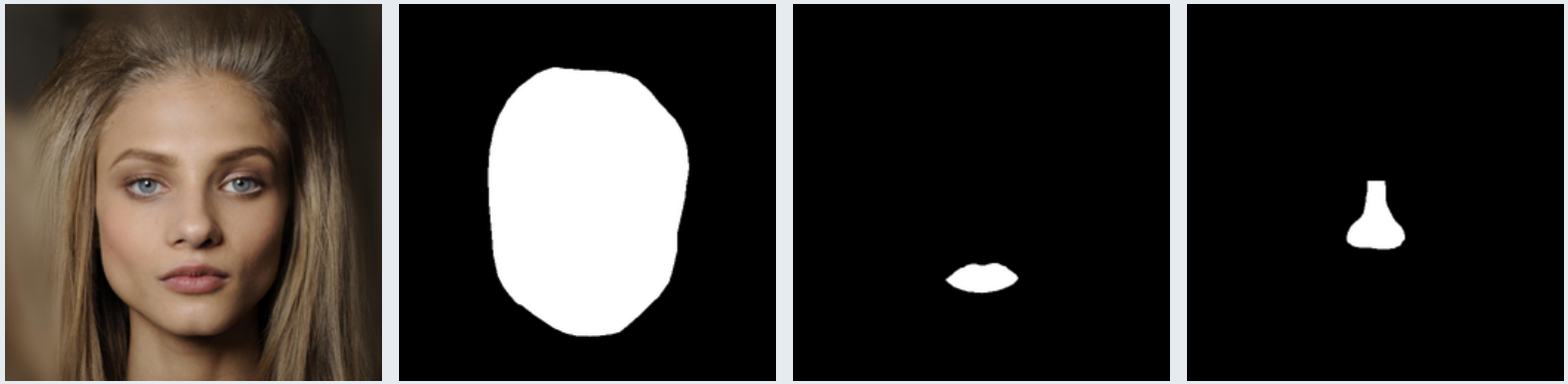
Rendering



- As a final step, we render the 3D model with the optimized mesh and textures
- The rendering is made from multiple point of views starting from the optimized camera position to generate a GIF with an overview of the 3D model
- The rendering process is made with the Neural Renderer functions

Results - 1

Input

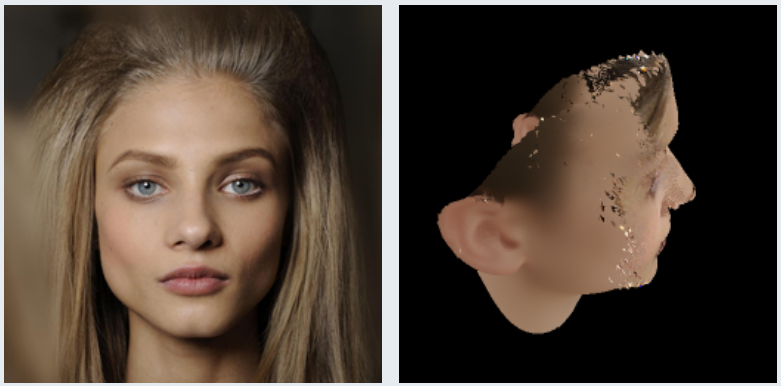


Camera Optimization

Results - 1

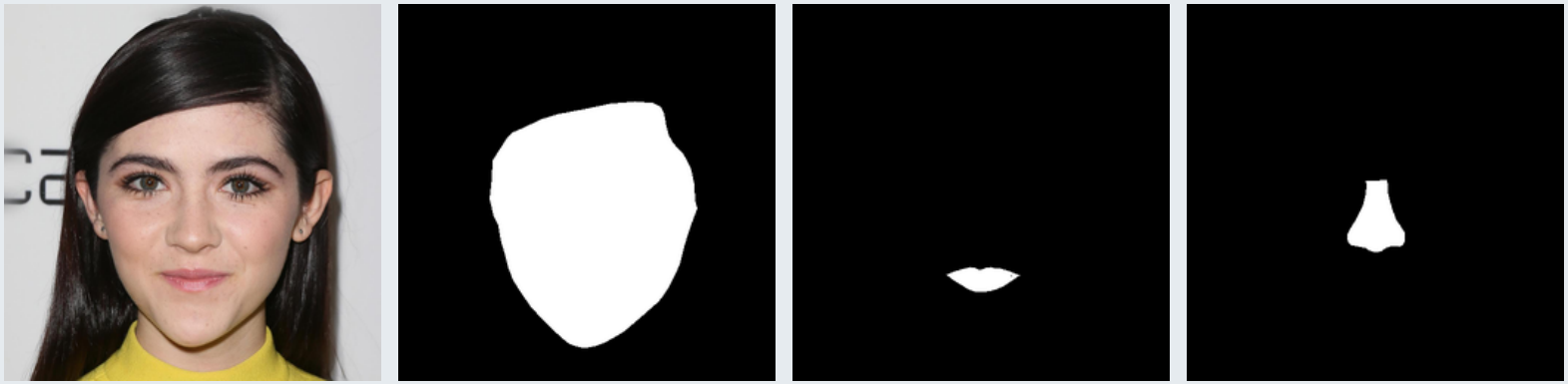
Mesh Vertices Optimization (Morphing)

Textures Optimization



Results - 2

Input

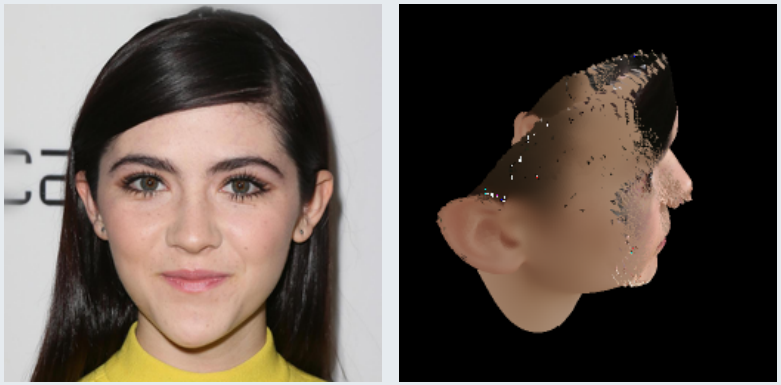


Camera Optimization

Results - 2

Mesh Vertices Optimization (Morphing)

Textures Optimization



Problems faced

- The cartesian coordinates system was not the best option for our optimization pipeline. Spherical coordinates are more natural for rendering a static object, however Neural Rendering showed problems in using them.
- Camera optimization is the most important step of the pipeline. Naive optimizations like the one proposed by the Neural Renderer authors gave very poor results. The introduction of anchor points and mostly contraints was crucial for obtaining sufficient results
- BFM has a very coarse grained vertices segmentation compared to the image segmentation offered by CelebAMask-HQ

Problems faced

- The cartesian coordinates system was not the best option for our optimization pipeline. Spherical coordinates are more natural for rendering a static object, however Neural Rendering showed problems in using them.
- Camera optimization is the most important step of the pipeline. Naive optimizations like the one proposed by the Neural Renderer authors gave very poor results. The introduction of anchor points and mostly contraints was crucial for obtaining sufficient results
- BFM has a very coarse grained vertices segmentation compared to the image segmentation offered by CelebAMask-HQ

Problems faced

- The cartesian coordinates system was not the best option for our optimization pipeline. Spherical coordinates are more natural for rendering a static object, however Neural Rendering showed problems in using them.
- Camera optimization is the most important step of the pipeline. Naive optimizations like the one proposed by the Neural Renderer authors gave very poor results. The introduction of anchor points and mostly contraints was crucial for obtaining sufficient results
- BFM has a very coarse grained vertices segmentation compared to the image segmentation offered by CelebAMask-HQ

Thank you for your attention