Human 3D face annotation

Rohit Kumar supervised by Dr.Nils Hasler

A work for The Captury, Saarland, Germany

Overview

- Problem statement
- Understanding dataset
- Setting up development
- Blender Mesh setup
- Solution chart
- Step by Step explanation
- References
- My further ideas and analysis

Problem Statement

- To understand 3D face annotation, it's easy to derive the relation from 2D face annotation.
- 2D face annotation
  - We are given an image
  - We need to detect the face
  - We need to do landmark annotation on detected face
- 3D face annotation
  - We are given a mesh with texture. Consider that person in mesh keeps moving, it is not static.
  - Since it's 3D there can be multiple perspective to view the mesh.
  - **Problem 1 :** How do you find a camera location so that we can detect a face in the camera view (a 2D image) ?
  - Once we detect face in some camera view, it's a 2D landmarks annotation problem.
  - **Problem 2 :** We solve problem 1, than we have landmarks of 2D image, how do you find the 3D correspondances ?

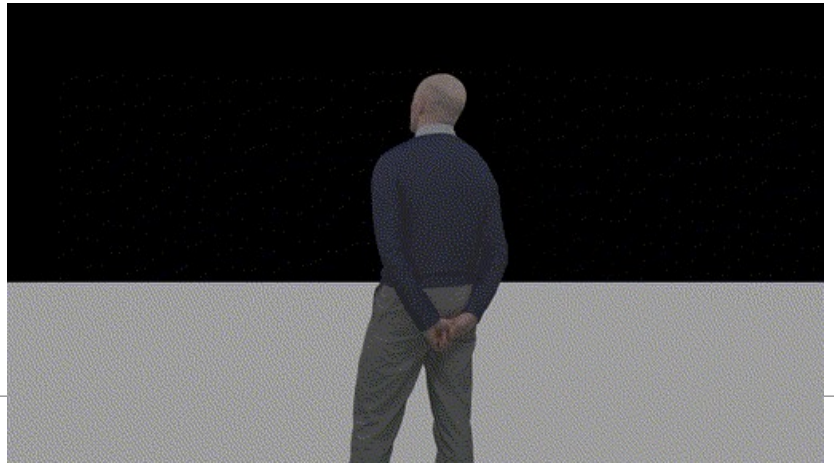
Understanding Dataset

- From Wikipedia

  Three-dimensional (3D) models represent a physical body using a collection of points in 3D space, connected by various geometric entities such as triangles, lines, curved surfaces, etc. Being a collection of data (points and other information), 3D models can be created manually, algorithmically (procedural modeling), or by scanning. Their surfaces may be further defined with texture mapping.

- In our case, the dataset is a human frames of 3D meshes, just like how 2D frames are but in form of 3D meshes at each frame.

- A circular view around the mesh looks like the image attached.



Setting up the development Environment

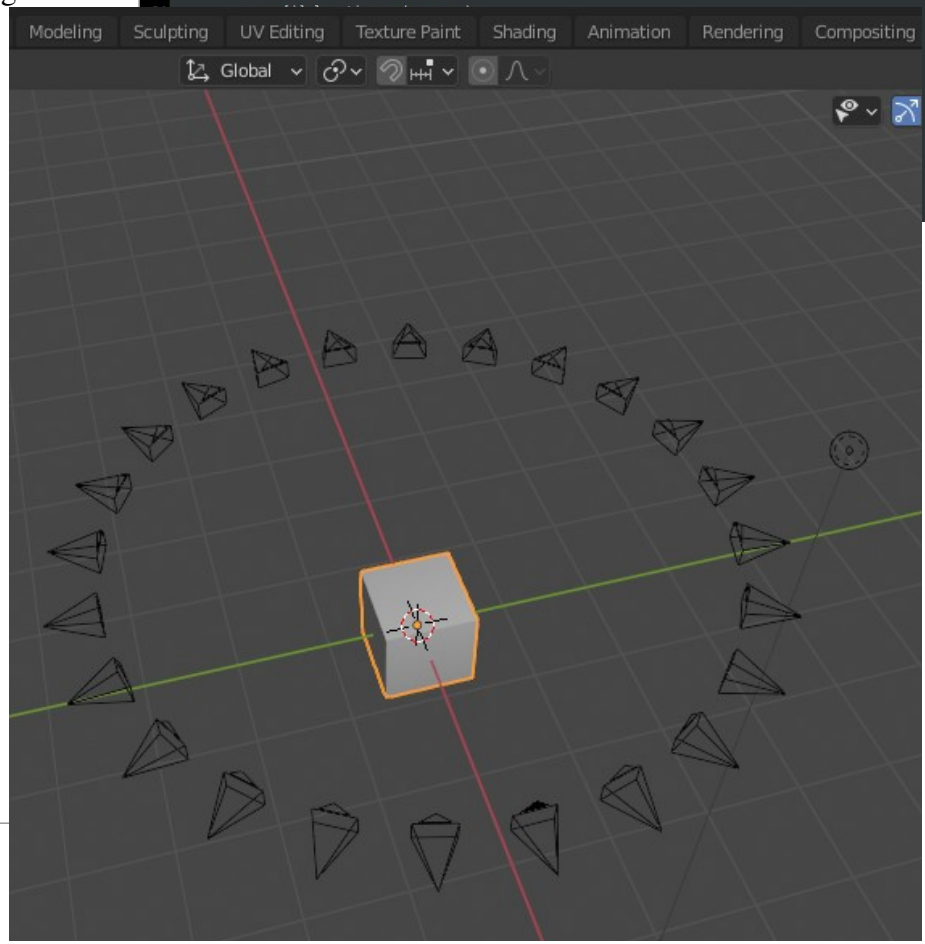
- To interpret a 2D image, we need an image viewer.
- To interpret a 3D mesh, we need a 3D software.
- Our task is to construct a pipeline from human face annotation. A software that supports scripting would be nice.
- Blender provides a support for python scripting, as well as it is open-source and free to use.
- Used OpenCV's implementation for face detection (DNN as well as Haar Cascade).
- Used Dlib's facial landmarks annotation support.



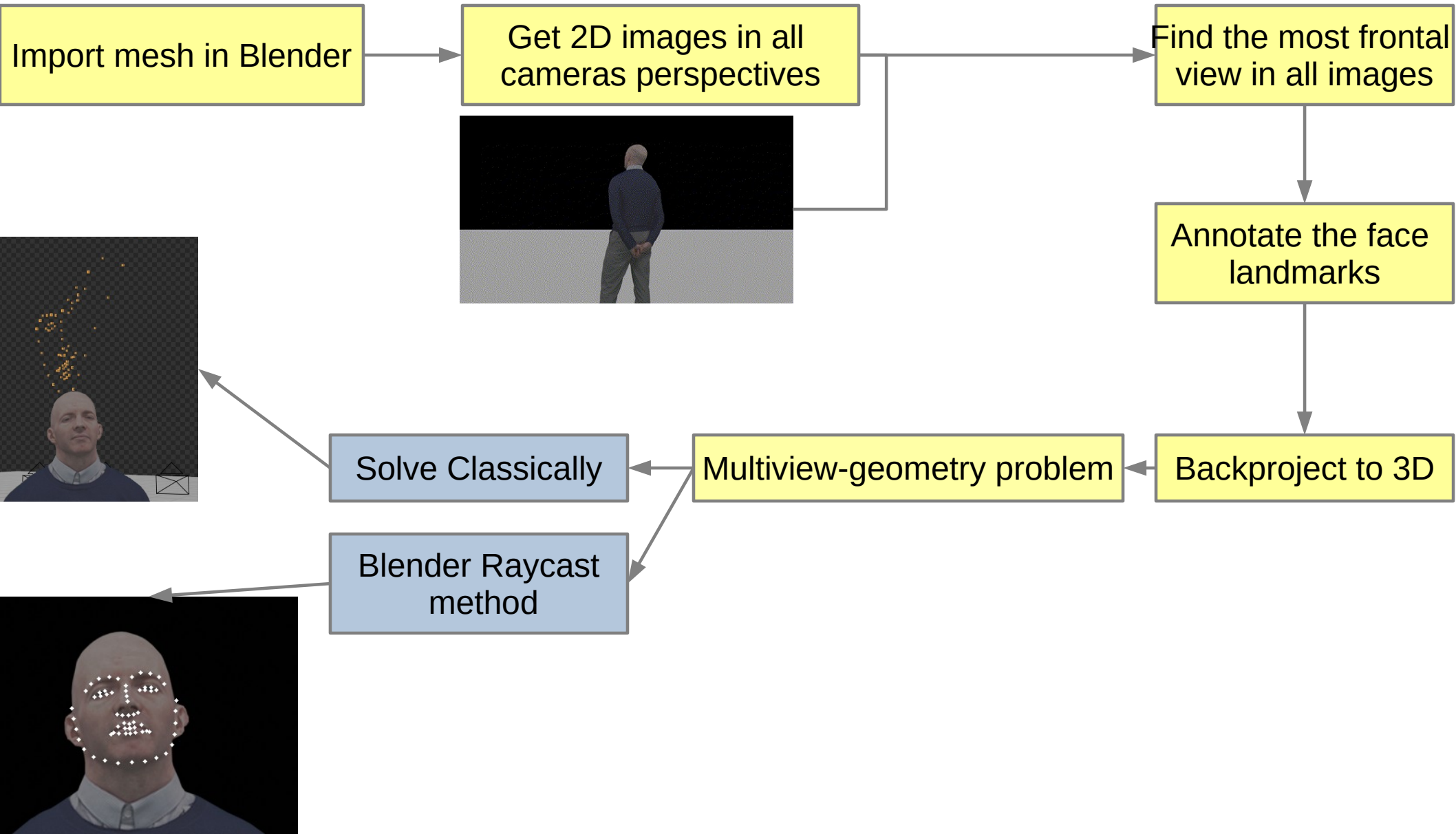
Blender Mesh Setup

- Setup cameras in a circle around the mesh. 15 degree separation would be good.
- A python code similar to this would work.

```python
def cameraSetup(scene,
                context,
                model_position: list = [0, 0, 0],
                model_dimension: list = [1.8, 1.8, 2],
                theta: "degree" = 15,
                sensor_height=32) -> list:
    '''
    model_position = position of model, default at [0,0,0]
    model_dimension = dimension of model in metre, default [1.8,1.8,2]
    theta = theta at which to sorround the object in circle through cameras, defualt 15 degree
    '''

    camera = []
    position_model = [1, 1, 0]

    no_of_cameras = int(360 / theta)  # theta in degree
    print("setting up {0} cameras, at {1} degree each in circle ".format(
        no_of_cameras, theta))
    theta = theta * math.pi / 180  # theta to radians

    for i in range(0, no_of_cameras):

        camera_data = bpy.data.cameras.new("Camera")  # take default camera
        camera.append(bpy.data.objects.new("Camera.000", camera_data))

        x = model_position[0] + (model_dimension[2] +
                                 model_dimension[0]) * math.cos(i * theta)
        y = model_position[1] + (model_dimension[2] +
                                 model_dimension[1]) * math.sin(i * theta)
        z = model_position[2] + 0.25 + model_dimension[2] / 2
```



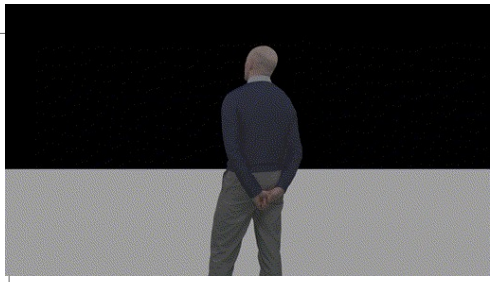
Solution Chart



Step 1 : Import mesh in Blender

- Write a generic python script that
  - Setup scene, camera, lights, parameters, etc.
  - Renders the view in every camera perspective

Step 2 : Find the most frontal image

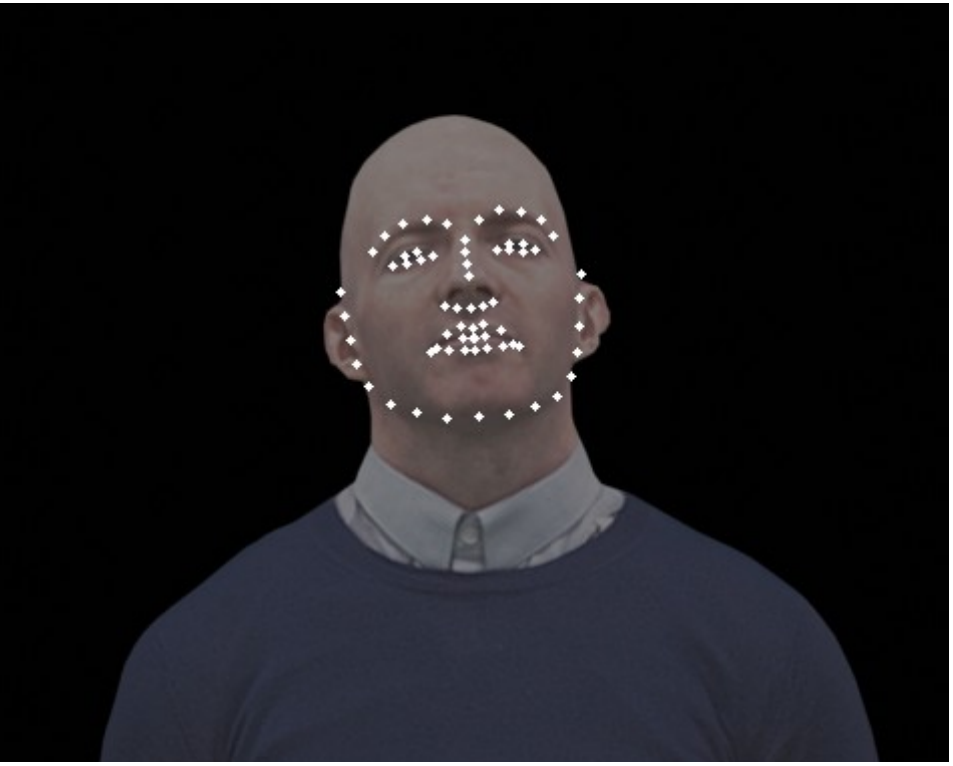


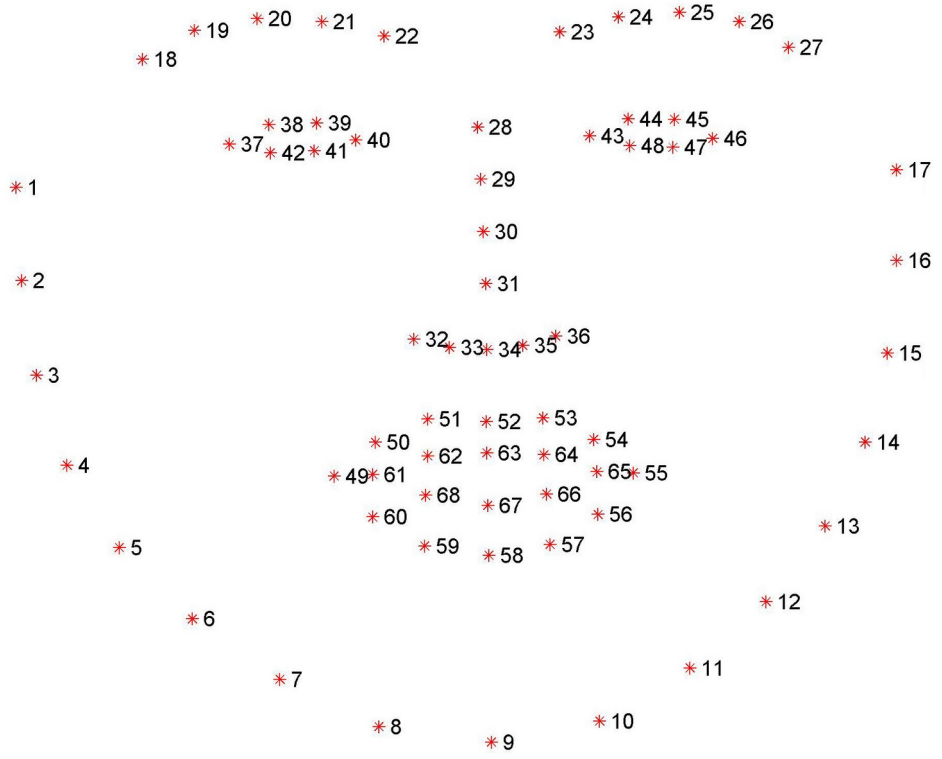
- Given : 24 images (if camera at 15 degree separations)
- Detect face in the images using either **openCV's haar Cascade detector or DNN detector.
- Prepare the list of images that have face detected.
- Now choose the central image in the sequence. It is the most central view (or 2nd most central view which also works fine for us due to quite less separation b/w cameras. We can reduce cameras separation from 15degrees to 10 degrees)

** I found that choosing weak detector is better for us I.e openCV's Haar Cascade method is better from the dnn's one. It is because the dnn model has a strong affinity to detect face even if only partial view is present. We want to reject those cases.

Step 3: Annotate the face landmarks

- Annotate the central face with 68 points landmarks detector implemented in Dlib here
- Those 68 have correspondances to eyes, nose, lips, etc. as shown in image.

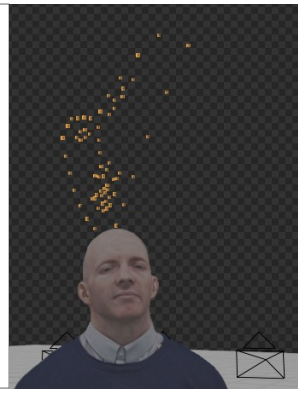


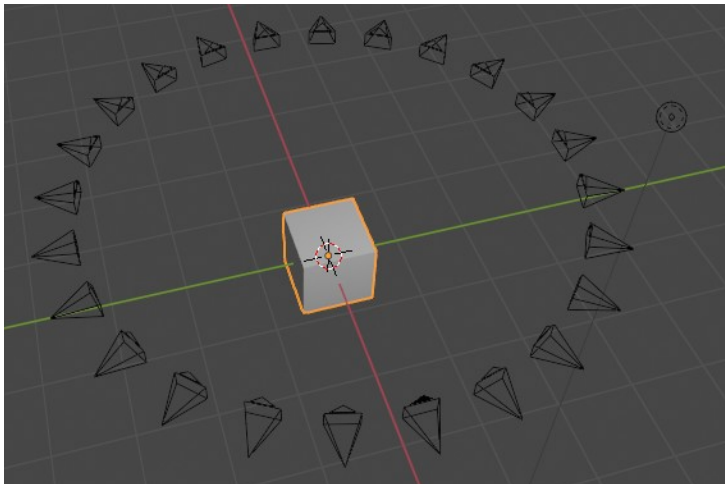


Back-project to 3D

Multi-View geometry Method

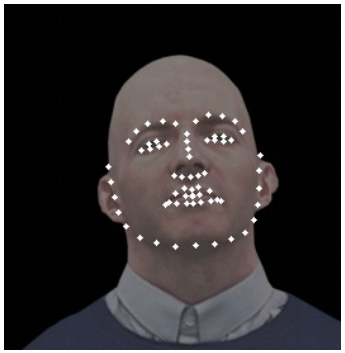
- We need to project the landmarks we got to the 3D mesh.
- One way to think over it is by multi-view geometry.
- However reconstructing by this method will have noise.





Blender Ray-Cast Method

- Blender supports ray-cast method and has it's implementation built in it's API.
- See references to read more about it.



References

- A big thanks to Blender communitites for their discussions.
  - Blender Docs, Blender Stack Exchange, Blender Artists
- Learn openCV blog on face detection and landmark annotation
- towards data science blog
- Semantic Scholar, a review of current methods
- Head pose estimation using opencv and dlib
- Retrieve the depth from the renderer:Retrieve the depth from the renderer
- Given the camera matrix and the depth you can compute the 3d coordinate
- Raycast
- Discussion on raycast

Further Ideas

- The problem of finding the most frontal view can also be viewed as a differential rendering problem.
- We need a camera pose that minimizes the difference b/w the current view through it and a general representation of how frontal human view looks like.
- Pytorch3DPytorch3D supports this type of rendering. I worked with it. However as it is in it's initial phase, there is a bug reported by me which causes memory overflow due to which I was not able to continue with the experiment.
- Pytorch3D tutorials