# Generating 3D Shoe models from a single side view
**COMP 6381 - Fall 2021 - Project Report**

**Nasir Khalid**
ID 40192389

## 1 Motivation

The inverse rendering problem or 3D shape extraction from 2D images is one with a wide variety of applications in commercial environments. A niche such application is in the world of sneakers and shoe design. Brands are looking for ways to allow users to test shoes in AR/VR environments as well as explore designs in interactive ways that build meaningful experiences.

This is only possible by creating 3D models of the shoes, which proves to be an expensive task because of man hours and software costs involved. The goal of this project was therefore to simplify and speed up the creation of 3D shoe models. This was achieved using just a single image of the right shoe

## 2 Objective

The objective of the project can be summarised in the figure below where the objective is to use the single image of the shoe as input and the output would be a 3D model of the shoe.

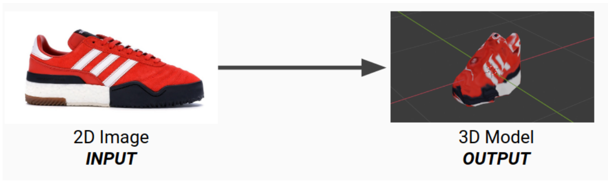

Figure 1: Objective of Project

During the project this was the goal but there were certain criteria that had to be met in obtaining this objective

1. Input to method will be a single 2D image of the right side of the right shoe
2. The method must execute as fast as possible so it can be used in realtime
3. The method should generalize and adapt to any new shoe input
4. The method does not require any 3D data during training or execution

## 3 Data

Since the project is only allowed to use images and the output model needs to be accurate it was important to get data that can help capture intricacies of shoes so the method used can learn details

Using TheSneakerDatabase [1] were able to obtain 360 degree images of around 7500 shoes. This was in the form of 36 images per shoe at an interval of 10 degrees, in total there were 270000 images

Using a pretrained RCNN segmentation model from the torchvision library masks were obtained for all images

Below is an example showing all images for a single shoe

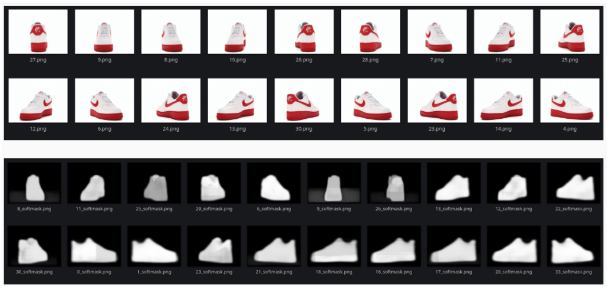

Figure 2: All images for a single shoe

## 4 Method

### 4.1 Options

Going from 2D images to 3D models has been a longstanding problem and multiple approaches have been created over the years. Based on the criteria outlined in the Objective section multiple methods were evaluated

#### 4.1.1 COLMAP

COLMAP is a library that uses Structure from Motion and SIFT feature matching to convert a set of images to a set of pointclouds that can then be converted to a mesh

However this method requires a large set of input images, more than the 36 per shoe from the dataset. It is also slow to run and not realtime enough. Furthermore, the methods are not general and for every new shoe the method would need to be rerun since features cannot be used between different shoes

One benefit is that this technique does not use any 3D data and only images can be used.

Overall COLMAP meets only 1 out of the 4 of the required criteria

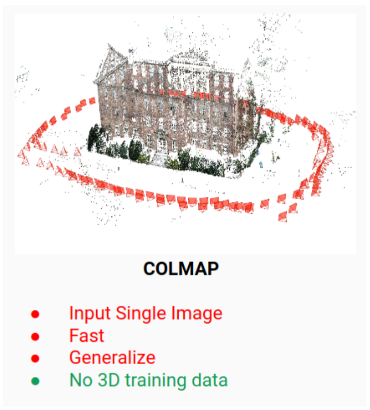

Figure 3: COLMAP

### 4.1.2 Neural Network

Neural networks are incredible at generalizing and there are multiple architectures for image to mesh training

Since the architectures are flexible it is easy to adapt them to take single images as input. In addition once a network is trained it can be compiled to execute quite quickly, they will also be able to generalize if the training data is varied and unique

The disadvantage here is that the existing architectures require 3D data for model training and this is not possible since the dataset contains only images

Overall neural networks meets 3 out of the 4 of the required criteria

### 4.1.3 Differentiable Renderer

Differentiable renderers are a new technique that allow backpropogation through rendering pipelines which can be used for shape and texture optimization

Here an initial mesh like a sphere could be rendered from same angles as the shoe images and

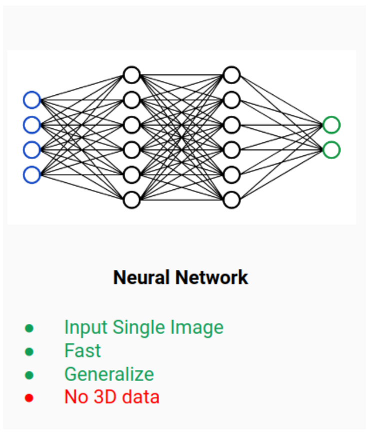

Figure 4: Neural Network

then an L2 loss between rendering and target image could be backpropogated to the sphere and deform it to the desired shape and texture of the shoe

Here the main benefit is that no 3D model is needed and using just the images we can deform an arbritary shape to the desired one but the disadvantages are that we need multiple multiview images so a single image cannot be used. The optimization is also slow and cannot generalize to a new shoe it would need to be rerun

Overall differentiable renderer meets 1 out of the 3 criteria

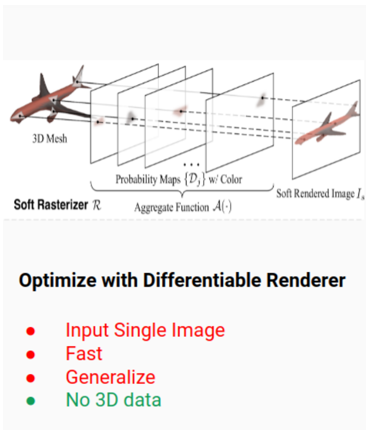
3D Mesh
Probability Maps $\{\mathcal{D}_j\}$ w/ Color
Soft Rendered Image $I_s$
Soft Rasterizer $\mathcal{R}$
Aggregate Function $\mathcal{A}(\cdot)$
Optimize with Differentiable Renderer
Input Single Image
Fast
Generalize
No 3D data

Figure 5: Differentiable Renderer

## 4.2 Selected Method

Since all of the 3 options do not meet all criteria the best approach is to combine 2 of them to meet all criteria of the project.

Combining the neural network with a differentiable renderer is the approach selected in the project and the main reason is that both methods are differentiable which means that losses can be propogated between them both. COLMAP uses non differentiable techniques which make it harder to integrate it with the other techniques

A high level overview of the selected method can be seen below where the neural network is responsible for giving a 3D mesh output and then the differentiable renderer is responsible for helping optimize the mesh using the images as a loss. In this way all the benefits of the neural network can be used and the issue of requiring 3D data can be solved through a differentiable renderer. The black arrows show the forward pass and the green arrows show backpropogation of a loss to update weights

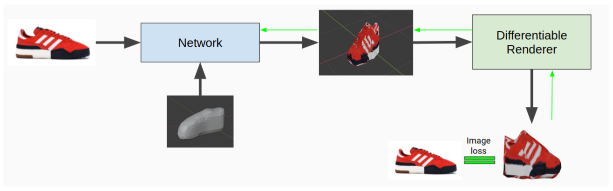

Figure 6: Selected Method

## 5 Approach

Below is a figure showing an indepth look at the whole pipeline with the neural network, renderer, loss functions (in green)

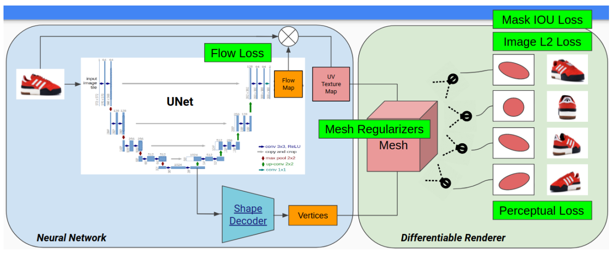

Figure 7: Approach Overview

Below is a figure showing the shape decoder in the neural network

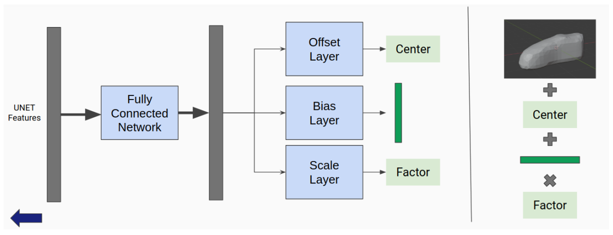

Figure 8: Shape Decoder

The following subsections discuss the neural network, differentiable renderer and the loss functions

### 5.1 Neural Network

The neural network is initialized with a base shape that roughly resembles a shoe and using an input image the network outputs a modified version of the base shape and a UV texture map that can be used to create a textured mesh

The architecture consists of a UNet that takes a single image as input and after passing through the encoder decoder layers the output is a flow map the size of the texture map of base shape

The output of the encoder layers passes through a shape decoder which outputs modified vertices and helps define the shape of the shoe.

The flow map helps get a texture map, each pixel in the flow map corresponds to a location in the original input image, it can essentially be thought of a lookup for color and if the flow map is multiplied with the original image the output is a new image which contains only colors from the original input image

In the network case the UV texture map is obtained by multiplying the flow map with the original image and while it may initially be a random image after training the texture map gives accurate colors. This flowmap method is used because it speeds up optimization as the network does not need to explore the entire colorspace but instead just a 2D grid space the size of the input image

### 5.2 Differentiable Renderer

The differentiable renderer uses the base shape UV and faces along with the neural network output vertices and texture map to create a mesh which it then renders from the same camera angles as the dataset shoes. So while the neural network can only take a single image as input the differentiable renderer uses all the 360 degree images and renders the output shape from all these angles.

The renderer outputs mask images and colored render images which can then be compared with the dataset image masks and images to obtain a loss value. This loss is then backpropogated through the renderer to the neural network weights to adjust them so that it gives us a better mesh and mesh texture

Running this pipeline over 300 epochs ends up training the neural network to start giving accurate meshes from just a single shoe image

### 5.3 Losses

Multiple losses are needed for training the network through the differentiable renderer and they all help speed up optimization

#### 5.3.1 Image L2 Loss

Given an image $I$ from the dataset we render the mesh using the differentiable renderer from the same camera angle as $I$ to obtain a renderered image $\widetilde{I}$

The L2 loss can then be given by the formula below

$$L_{image} = E_1[\|I - \widetilde{I}\|_2]$$

#### 5.3.2 Mask IOU loss

Given an image mask $S$ from the dataset we render the mesh using the differentiable renderer from the same camera angle as $S$ to obtain silhouette mask $\widetilde{S}$

Defining $\otimes$ as element wise multiplication and $\oplus$ as element wise summation. The mask IOU loss can then be given by the formula below

$$L_{IOU} = E_1\left[1 - \frac{\|S \otimes \widetilde{S}\|_1}{\|S \oplus \widetilde{S} - S \otimes \widetilde{S}\|_1}\right]$$

#### 5.3.3 Perceptual Loss

Given an image $I$ from the dataset we render the mesh using the differentiable renderer from the same camera angle as $I$ to obtain a renderered image $\widetilde{I}$

We then obtain a perceptual loss value by measuring the difference in activations of Alexnet model when given $I$ and $\widetilde{I}$

#### 5.3.4 Laplacian Loss

This is a regularization loss applied to the mesh obtained from the output of the neural network. During optimization of vertices the change in vertices is restricted to ensure that the overall mesh deforms smoothly and prevents jagged mesh face intersection [2]

#### 5.3.5 Flat Loss

This is also a regularization loss applied to the mesh obtained from the output of the neural network and it tries to encourage adjacent triangle faces to have similar normal directions [2]

#### 5.3.6 Flow Loss

The flow map output from the UNet selects points from the original input image. However, a large part of the original input image is just the background that is white and contains no useful information for getting the texture.

To encourage the network to select colors from the shoe and ignore the background a flow loss is optimized and it can be given by the formula below where $F$ is the flow map output and $S$ is the mask of the input image with $\otimes$ being element wise multiplication

$$L_{flow} = \sum S - E_1[F \otimes S]$$

## 6 Results

Once the network is well trained the differentiable renderer can be removed and the neural network will output good meshes given an input image. Results can be explored on https://2d-to-3d-shoe-webapp.vercel.app/

## 7 Issues

While the performance of the network is quite good there are 3 main issues that can be mitigated to improve performance

### 7.1 UV Map

The UV map of the base shape consists of islands of the base mesh. While this makes the UV map interpretable for a user it actually degrades performance since the network must learn to ensure that borders of each map have similar colors and this does not happen successfully in practice.

A solution to this problem would be to unwrap the base shape spherically to ensure that colors flow well across the texture map rather than abruptly change. In addition there can be another loss on the flow map to keep colors selected from the original image in a similar vicinity

This issue is demonstrated in the figure below with the UV map on the left and its islands alongside a mesh result on the right

### 7.2 Invariance

During training the input images were all of a similar distribution with the shoes being positioned in the exact same location throughout with no augmentations. This leads to an issue where if the shoe is slightly scaled or positioned differently in the input it leads to a drastic change in the output

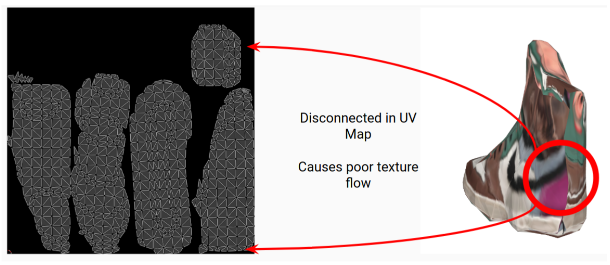

Figure 9: UV map issue

The figure below demonstrates this where the 2 input images on the left are slightly different in scale but their mesh outputs on the right are drastically different

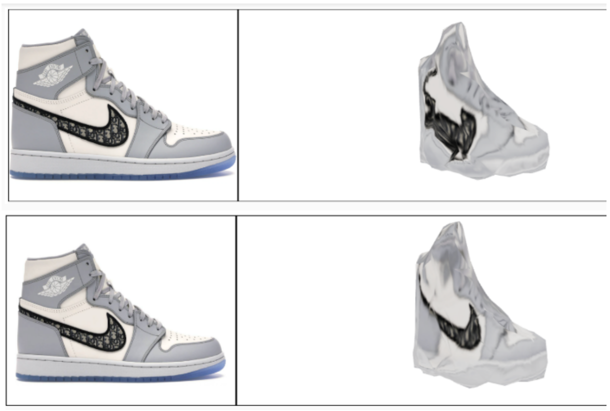

Figure 10: Invariance issue

### 7.3 Geometry

Even with all the regularizers the output geometry leaves a lot to be desired as it is very jagged and twisted. This can be seen in the image below where the left is a visual of the output geometry and the right is the base mesh

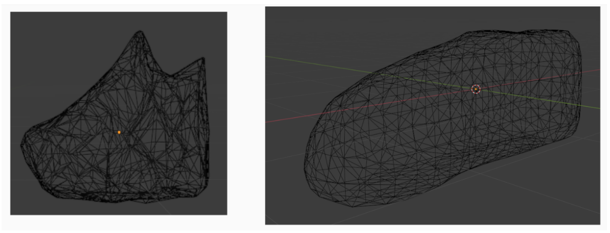

Figure 11: Geometry Issue

The triangles tend to deform and change very aggressively and this could be alleviated by remeshing and changing the base shape at set epochs during training

## 8 Conclusion

The goal of the project was met under the given criteria and the results are quite impressive given that they use a single image

In addition the issues outlined in the previous sections can be solved through existing techniques to further boost performance of the network

(4) (4)

## References

The sneaker database, 2021.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.