UNIVERSITY OF
CAMBRIDGE
DEPARTMENT OF
COMPUTER SCIENCE
AND TECHNOLOGY

Auto-Initialising Differentiable Renderers

Group 9 - Machine Visual Perception Project Report

Pranav Talluri, Ke Ding, Yujia Yang

2022

Contents

1 Introduction and Motivation

1.1 Introduction to the Problem: Pranav Talluri

Rendering allows us to produce 2D images from 3D scene data. A common problem in computer vision is inverse rendering, which is the process of estimating physical attributes of a scene (3D) from an image (2D).

Traditional approaches to constructing 3D scenes are typically based on carefully controlled, high quality capture of the scene using specialised equipment and algorithms to construct the scene based on the captured data.

Deep learning approaches learn a mapping between images and scenes. They require very large amounts of data and are not very generalisable.

Inverse rendering approaches aim to infer the scene that an image (or set of images) is based on by rendering scenes, comparing the images and the renders and back-propagating changes to improve the loss between them. The key requirement for inverse rendering is for the rendering operation to be differentiable for the back-propagation to be possible. However, in order to construct an image of visible objects from a scene, most rendering pipelines utilise rasterisation, which is not differentiable.

Differentiable renderers circumvent the rasterisation stage of typical rendering pipelines, allowing us to back-propagate changes, so we can search for an optimal scene estimate by performing stochastic gradient descent over the scene parameters. The idea is shown in Figure 1. We focus on SoftRas (Liu et al. 2019), a rendering pipeline that introduces soft rasterisation, an alternative method of determining the visibility of elements in the scene.

Soft rasterisation utilises a probabilistic interpretation of visibility to maintain the differentiability of the rendering pipeline. Specifically, it identifies the probability of mesh triangles contributing to pixels, rather than the traditional (un-differentiable) discrete sampling approach. The probabilities are aggregated using depth information and colour maps to determine the pixel values of the image.

SoftRas allows us to perform image-based shape deformation. This is where we reconstruct the mesh of an object by providing multiple images of the object from various viewpoints. Specifically, we provide a set of silhouettes of an object, their viewpoints and an initial mesh and SoftRas will attempt to fit the mesh according to the silhouettes.

The key problem we try to address in this project is the initialisation of SoftRas, and the insights we develop will be applicable to differentiable renderers in general. The search space for determining the optimal mesh is large and high-dimensional, so the initialisation can have a significant impact on the quality of the fitted mesh, the amount of iterations required and whether SoftRas is able to converge at all or becomes stuck in local minima.

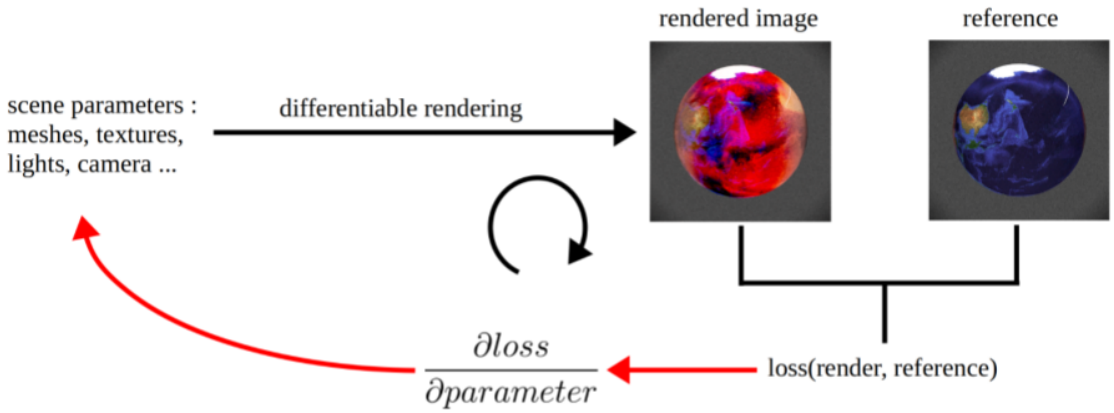
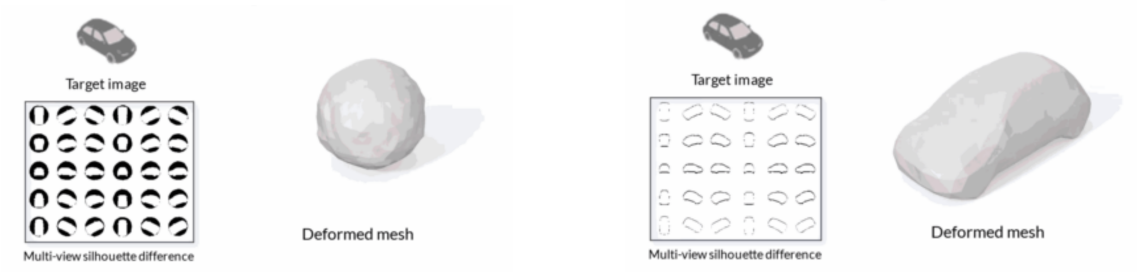


Figure 1: Differentiable Rendering (Robineau 2021)

1.2 Background and Related Work: Pranav Talluri

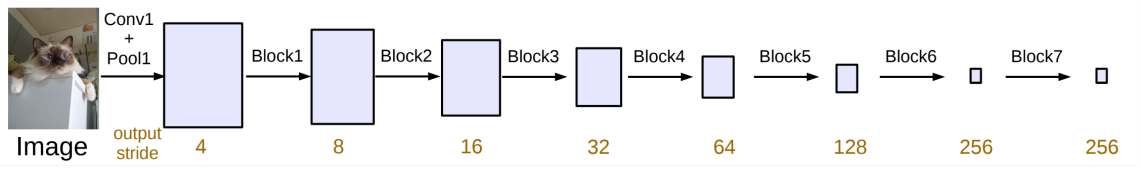
The key literature for this project is *Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning* (Liu et al. 2019). The paper introduces a novel approach to differentiable rendering, replacing the standard discrete rasterisation process with a new soft rasteriser as discussed in 2.1.1. We specifically investigate the usage of the differentiable renderer to perform inverse rendering. Figure 2 shows the fitting procedure, which is discussed further in 2.1.2.



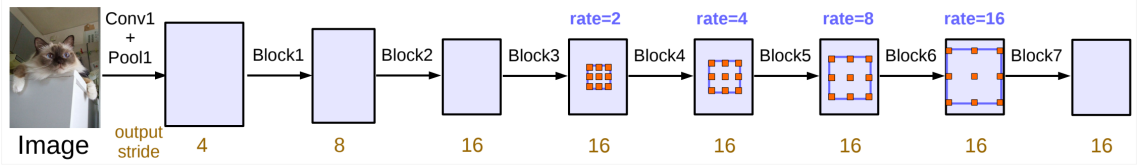
(a) Start (b) End

Figure 2: SoftRas fitting procedure (Liu et al. 2019)

Our project also uses on DeepLabv3 (Chen et al. 2017) for generating image silhouettes. DeepLabv3 utilises modules which employ atrous convolution in cascade or parallel to capture multi-scale context by adopting multiple atrous rates. Atrous convolution enables adding more layers (deeper networks) without having to have larger images, as shown in Figure 3. It utilises pyramid pooling modules to probe convolutional features at multiple scales.



(a) Without atrous convolutions



(b) With atrous convolutions

Figure 3: Cascaded modules and atrous convolutions (Chen et al. 2017)

DeepLab is able to generalise well to new images due to its large and varied training dataset. The authors of the original paper also detail the typical failure modes. The model is able to perform extremely well on images with clear subjects but is weaker when given images without a clear view of the subject or differentiating between similar subject classes. The latter failure mode is not a concern for our usage of DeepLab. Some sample results are shown in Figure 4 where the first two rows show successful results and the last row visualises the failure modes.

1.3 Overview of the Idea: Pranav Talluri

The aim of our project is to address the initialisation of SoftRas, in order to improve its performance in constructing a fitted mesh when given images of an object from multiple viewpoints. There are three key initialisations for SoftRas which we aim to address: silhouettes, meshes and camera parameters. In addition to these initialisations, we also investigate the data used. The original

(a) Good results

(b) Failure modes

Figure 4: DeepLab segmentation samples, including failure modes (Chen et al. 2017)

paper uses a subset of the ShapeNet (Chang et al. 2015) dataset, which features renders of artificial 3D models of simple objects that fit into a small number of classes.

1.3.1 Silhouette Initialisation: Pranav Talluri

The first initialisation is the silhouettes of the multi-view images. The original SoftRas pipeline directly takes silhouettes of the object to be reconstructed. This is possible because the authors construct renders from the ShapeNet dataset which already have a transparent background. This is problematic as most real world multi-view datasets feature full images, including a background. To address this problem, we will utilise semantic segmentation (DeepLab) to produce segmentation masks, which can then be used to generate silhouettes. Retrofitting this pipeline to SoftRas will enable us to reconstruct 3D meshes from natural 2D images.

1.3.2 Mesh Initialisation: Pranav Talluri

The next initialisation is the mesh that SoftRas deforms to fit to the input images. By default, the SoftRas pipeline utilises a sphere. The initial mesh has a large impact on the both the number of iterations required by the fitting procedure and the final product. We plan on modifying this initialisation based on the input images to accelerate the fitting procedure and procedure better results. The modifications are: manipulating the polygon count of the mesh, changing the scale of the sphere and translating its position.

1.3.3 Dataset Investigation: Pranav Talluri

The data used in the original paper comes from the ShapeNet dataset. This has some key limitations. The images utilised are relatively low resolution (64x64) and are not natural images, they come from renders of computer generated 3D models. In our project, we aim to investigate the effectiveness of SoftRas on higher resolution, photo-realistic images. We achieve this primarily by experimenting with the Amsterdam Library of Object Images (ALOI) (Geusebroek et al. 2005) dataset, which contains photographic images of a large variety of objects with resolutions of 384x288. The images are also taken from more limited viewpoints (no vertical viewpoint variation), which will test the fitting procedure's ability to extend to less informative input images. This dataset includes the ground truth camera parameters, so we also aim to test the pipeline on images that we capture ourselves where the ground truth for image viewpoints is not known and the backgrounds are more challenging.

1.3.4 Camera Parameter Initialisation: Pranav Talluri and Ke Ding

The final initialisation is the viewpoint labels for each of the multi-view silhouettes passed as input to the fitting procedure. These parameters tell the fitting procedure which angles to generate renders of the predicted mesh from so that these renders can be compared to the silhouettes to compute a loss. The fitting procedure used in the original paper utilises ground truth viewpoint labels that are given by the ShapeNet dataset. We aim to estimate these parameters using SfM in order to enable fitting on natural images where the ground truth viewpoint parameters are unknown.

2 Method

2.1 Baseline Algorithm

2.1.1 Soft Rasteriser: Pranav Talluri

The backbone of our project is the official implementation of SoftRas created by the authors of the paper. We utilise the soft rasteriser from this original project in order to generate renders of the predicted model. Figure 5 compares the rendering pipeline used by SoftRas and a traditional pipeline.

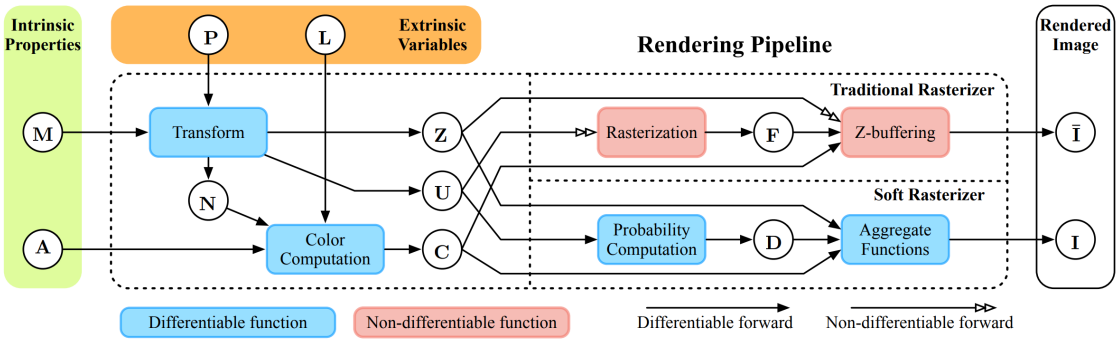


Figure 5: SoftRas vs Traditional Rendering Pipeline (Liu et al. 2019)

The soft rasteriser works by aggregating probabilistic contributions from each triangle in the scene mesh, which makes it truly differentiable. This is a notable improvement over prior works in differentiable renderers, which only approximate the rendering gradient in the back propagation stage, and utilise traditional methods in the forward pass, such as Kato et al. 2017. SoftRas therefore avoids an inconsistency between the forward and backward passes which would lead to uncontrolled optimisation behaviours and a limited ability to generalise to other 3D reasoning tasks.

SoftRas interprets rasterisation as a problem of binary masking determined by the relative positions between pixels and triangles, and z-buffering merges rasterisation results on a per-pixel basis using the relative depths of triangles. SoftRas then aims to solve this problem in a differentiable manner, which it achieves utilising probability maps $\{D_j\}$ and an aggregation function A. D_j models the influence of triangle mesh f_j on the image plane. We need to estimate this value using a function, which needs to take into account the relative position and distance between the pixel p_i and triangle f_j. We define D_j^i using the sigmoid function

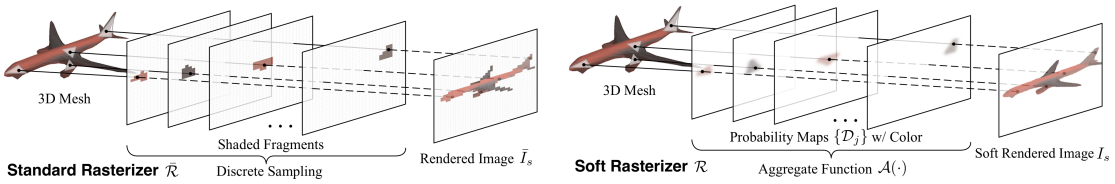
$$D_j^i = sigmoid(\delta_j^i \times \frac{d^2(i,j)}{\sigma}).$$

For each mesh triangle f_j, we define its colour map C_j at pixel p_i on the image plane by interpolating vertex colour using barycentric coordinates.

A is an aggregate function that we use to merge the colour maps $\{C_j\}$ and obtain rendering output I based on probability maps $\{D_j\}$ and the relative depths $\{z_j\}$. Figure 6 shows how this compares with a traditional, discrete rasterisation pipeline.

2.1.2 Fitting Procedure: Pranav Talluri

Given an initial mesh, the procedure makes changes to the faces and vertices of the mesh, produces 2D renders of this mesh using the soft rasteriser and then compares these renders to the input images to compute a loss. This loss is then used to inform back-propagation through the rendering pipeline to modify the mesh. This repeats for some large number of iterations. The initial mesh, therefore, has a large impact on the performance of the process, including the number of iterations required to converge, the mesh that is converged at and whether or not the procedure is able to converge.



(a) Traditional rasteriser (b) Soft Rasteriser

Figure 6: Traditional vs Soft Rasteriser (Liu et al. 2019)

The loss is computed using the intersection over union (IoU) between the renders and the input images. This is a common choice for measuring similarity between images. We are unable to use 3D IoU using voxels as we do not have a ground truth 3D model of the target object, so we approximate with 2D IoU averaged over all the viewpoints. Given a set of n ground truth images i_{gt}, viewpoints v_{gt} and a predicted mesh m_{pred}, we generate projections i_{pred} of m_{pred} at v_{gt} using SoftRas. We specifically compute the loss as a negation of the average IoU over the multiple viewpoints:

$$IoU = 1 - \frac{1}{n}\sum_{j=0}^{n}\frac{intersection(i_{gt,j}, i_{pred,j})}{union(i_{gt,j}, i_{pred,j})}$$

2.1.3 Data: Pranav Talluri

Originally, SoftRas directly takes multi-view silhouettes as an input. The pipeline requires a 4 channel image (R, G, B and Alpha for transparency), so it doesn't support standard RGB images. If given an RGBA image with no transparent regions, the fitting procedure is not able to properly generate a mesh for the subject of the image as the transparency is required for loss computation. i.e., The transparent regions of the images tell SoftRas which regions to treat as the background. Figure 7 shows the data pipeline used by the original implementation of the fitting procedure. An example of an input to the original pipeline taken from ShapeNet renders is shown in Figure 8

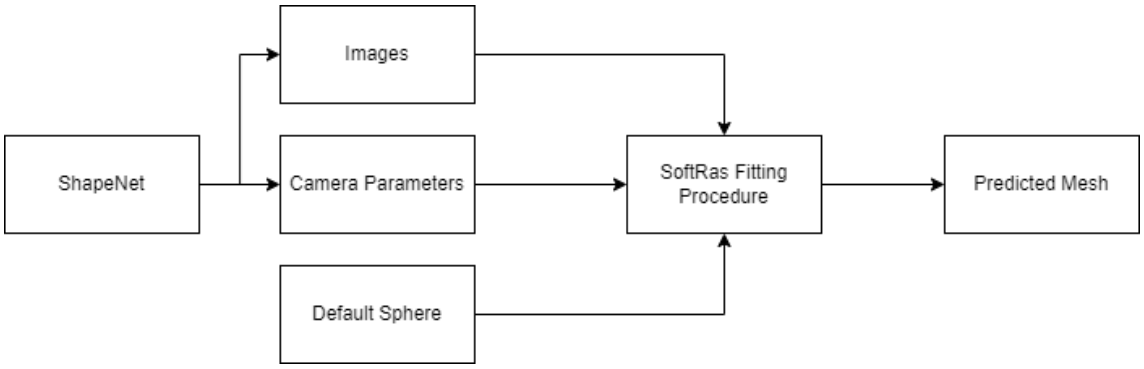


Figure 7: Data pipeline of original project

The pipeline takes images with a 64x64 resolution, but it is extensible to other (square) resolutions. However, attempting to use much higher resolution images produces poor results, qualitatively and quantitatively, but the speed of the fitting procedure is the real limiting factor. The pipeline also takes a camera viewpoint for every image. This determines which directions the fitting procedure should generate renders of the mesh from to compute a loss between the input images and the renders.

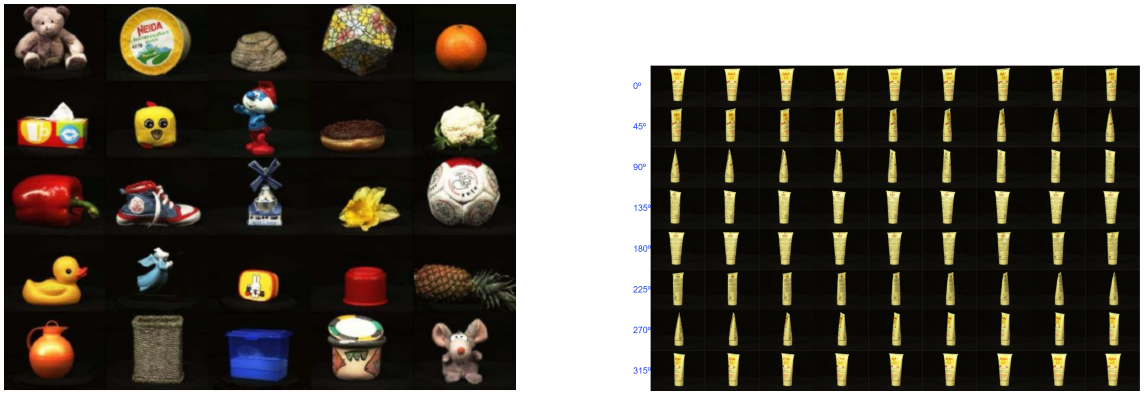
Figure 8: Silhouette Input from ShapeNet renders

2.2 Algorithm Improvements

2.2.1 Data: Pranav Talluri

The original pipeline is built for images from ShapeNet. In order to utilise other data sources, we first build a pipeline for processing input images. We modify SoftRas to operate on 256x256 input images. We then resize all larger input images to this size. Details for how these images are converted to silhouettes are in 2.2.2. We also need to make adjustments to the mesh to deal with the more detailed images and the details for this are in 2.2.3.

To investigate the performance of SoftRas on higher resolution, photo-realistic data, we utilise the Amsterdam Library of Objects (ALOI) dataset (Geusebroek et al. 2005). It contains images of 1000 small objects at a resolution of 384 x 288. We test these changes using both ALOI, where the camera parameters and silhouettes are known, and images that we have captured, where we have no viewpoint or silhouette ground truth. Some sample images from ALOI are shown in Figure 9. Some sample images that we captured are shown in Figure 10.



(a) Sample of different objects

(b) Each object is captured from 72 angles

Figure 9: Samples from ALOI dataset.

Figure 10: Samples of natural images that we captured

2.2.2 Silhouette Initialisation: Pranav Talluri

Synthetic datasets such as ShapeNet allow us to directly produce silhouettes. For carefully produced photo-realistic datasets such as ALOI, where object/background mask labels have been provided, silhouettes can be generated as shown in 2.3.1. However, for natural images, these silhouettes have to be estimated.

We modify the pipeline to enable fitting meshes for natural images with backgrounds. We utilise DeepLabv3, to produce a segmentation map of a set of inputs. We can then convert the semantic classifications into a binary pixel map, with 0 representing the background and 1 representing the subject of the image. We utilise this binary pixel map as the alpha channel of the image and also to set the pixel values of the background of the image to 0. We also convert the active pixel values to grey pixels, as the colour information is not utilised. These new images can be used as silhouette inputs to the standard SoftRas pipeline.

$$I = \begin{bmatrix} I_R & I_G & I_B \end{bmatrix}$$

I_R, I_G, I_B are matrices with values for each pixel in image I.

$$deeplab(i) = C$$

C is a matrix of semantic classifications for each pixel.

$$M = (C > 0)$$

M is a matrix of binary classifications (object or background) for each pixel.

$$\alpha = M \times 255$$

α is a matrix of alpha channel values.

$$I'_{i,j} = I_{i,j} * (M_{i,j} \times 124)$$

I' is a grey-scale version of I with zeroed background pixels.

$$I_{out} = \begin{bmatrix} I'_R & I'_G & I'_B & \alpha \end{bmatrix}$$

2.2.3 Mesh Initialisation: Pranav Talluri

We modify the initial mesh passed to the fitting procedure. The sphere used in the original pipeline, shown in Figure 11a, has some key issues.

A fixed size sphere is used, so if the images provided detail a subject that is much larger or smaller than the sphere, then many iterations are used to adjust the size of the mesh, if it succeeds at all. Figure 11 shows an example where the position and scale of the target (a small apricot at the bottom of the image) leads to failure with the sphere initialisation. To address this, we vary the size of the initial mesh based on the input images as well as scaling the x and y dimensions to match the aspect ration of the target object. We do this by forming a bounding box around the target object and calculating the scale factor to modify the height and width of the sphere by accordingly. The equation below hides the complexity of converting between pixel based widths for the target image and real-world measurement units for the sphere.

$$w_{scale} = \frac{w_{sphere}}{w_{target}};\ h_{scale} = \frac{h_{sphere}}{h_{target}}$$

(a) Sphere-based mesh initialisation

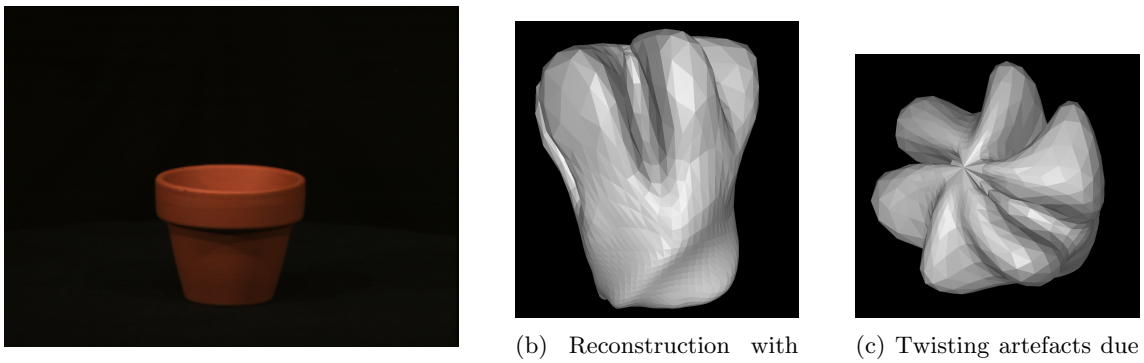
(b) Target object: an apricot

(c) Failure to converge

Figure 11: Failure to converge with sphere initialisation

Another issue is that the sphere is always centred at the origin. This is not an issue for synthetic datasets like SoftRas, where the target object is likely at the centre of the frame. However, for photo-realistic datasets such as ALOI, and especially for natural images captured by a camera/phone, the subject is unlikely to be centred at the origin. This means that the fitting procedure will utilise many iterations to move the mesh to where the subject is within the image. Additionally, this is achieved by moving individual vertices, which leads to stretching, distortion and twisting artefacts as shown in Figure 12. To address this, we translate the initial mesh according to the position of the subject within the input images. We once again achieve this using a bounding box on the target object. This step is performed after the scaling, so aligning the top and left of the sphere with the top and left of the target object is sufficient for aligning the image and mesh.

$$w_{shift} = l_{mesh} - l_{object};\ h_{shift} = t_{mesh} - t_{object}$$



(a) Target object: bucket

(b) Reconstruction with no mesh initialisation

(c) Twisting artefacts due to off-centre target

Figure 12: Artefacts with sphere initialisation

The sphere used in the original pipeline has roughly 1300 polygons. This polygon count does not vary during the fitting procedure, only the positions of the vertices. This is sufficient for the simple 64x64 images used in the original pipeline, but not for higher resolution images with more high frequency information. In Figure 13, the detail in the predicted mesh of the target is limited by the polygon count of the sphere. To address this, we introduce a new sphere with around 1900 polygons that we created in Blender and dynamically choose the initial sphere based on the target image, selecting between 600, 1300 and 1900 polygons.

(a) Target object (b) Lack of detail

Figure 13: Lack of detail when attempting to fit to larger targets with many folds

2.3 Implementation Details

2.3.1 Data Pipeline: Pranav Talluri

The original dataset uses images and viewpoint parameters from ShapeNet stored in NumPy array files as inputs. To make the pipeline extensible, this is replaced with a folder input for images. The camera parameters can then be given according to the ground truth of the dataset or inferred from the images. This enables us to switch between data sources and image samples by simply editing the image source path. All input images are resized to 256x256 as this was found to be ideal for execution time for the fitting procedure.

We can place novel images that we captured in an input folder and then utilise camera parameter estimation or, if we know what angles we took the images from, ground truth data. Figure 14 shows the data pipeline when we utilise novel images.

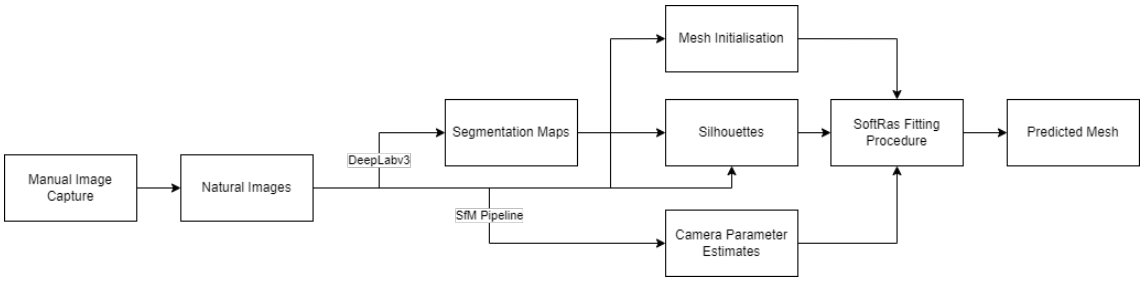


Figure 14: Data pipeline for newly captured images

The ALOI dataset images are taken every 5 degrees in a complete rotation around the object. The camera parameters can be inferred from this information. The code used to perform this initialisation is shown in Figure 15.

```
cameras = []
for i in range(72):
    cameras.append([2.732, 0., i*5.])
```

Figure 15: Viewpoint parameter initialisation for ALOI

The ALOI dataset also comes with ground truth object/background binary masks, which we utilise to generate silhouettes from the raw images. This is shown in Figure 16. We experimented with grey-scale and colour silhouettes and found no difference in results (as colour is not considered

in the loss computation), so we converted the silhouettes to grey scale. Figure 17 shows the new data pipeline for testing using the ALOI dataset.

(a) Original image (384 x 288)

(b) Ground truth mask (384 x 288)

(c) Generated silhouette (256x256)

(d) Generated silhouette grey-scale (256x256)

Figure 16: Image pipeline for ALOI

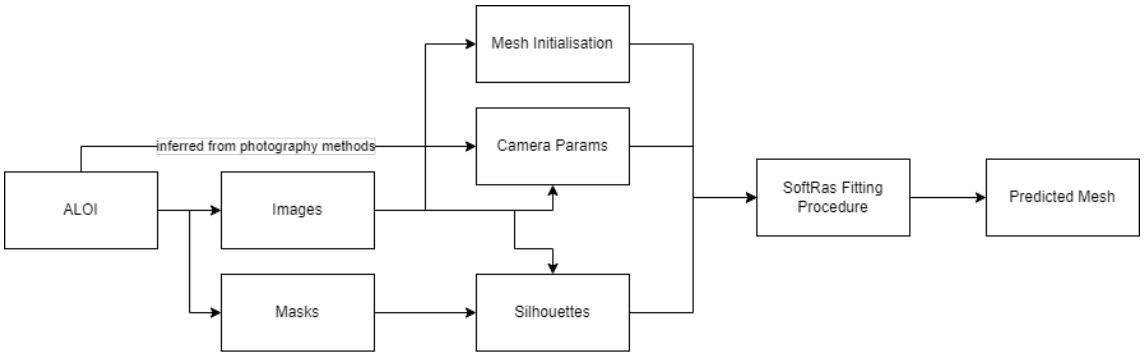


Figure 17: Data pipeline using ALOI dataset

2.3.2 Silhouette Initialisation: Pranav Talluri

As stated in 2.2.2, we estimate silhouettes for natural images using segmentation. We run DeepLab on the input images and use the segmentation masks as binary object/background masks, which we can add to the image as an α channel. This is performed in the segment.py module as shown in Figure 18.

```
def segmentMany(self, images):
    input_batch = []
    for image in images:
        input_batch.append(self.preprocess(image))
    input_batch = torch.stack(input_batch).to(self.device)

    with torch.no_grad():
        output = self.model(input_batch)['out']
    output_predictions = output.argmax(1)

    o = (output_predictions > 0).type(torch.int)
    o = torch.unsqueeze(o, dim=3)

    alpha = o*255

    o = o.expand(-1,-1,-1, 3)

    images = torch.Tensor(images).to(self.device)
    mod = torch.mul(images, o)

    out = []
    for image, alpha_c in zip(mod, alpha):
        image = (image > 0).int() * 124
        out.append(torch.dstack([image,alpha_c]))

    out = torch.stack(out).to(self.device)

    return out
```

Figure 18: Code used to perform segmentation using DeepLabv3

2.3.3 Mesh Initialisation: Pranav Talluri

We create higher polygon count sphere initialisations in Blender using a simple UV sphere and the subdivide functionality. The straightforward process is shown if Figure 19. We also modify the SoftRas pipeline to deal with 256x256 images, which is a simple parameter change when initialising the soft rasteriser during the fitting procedure.

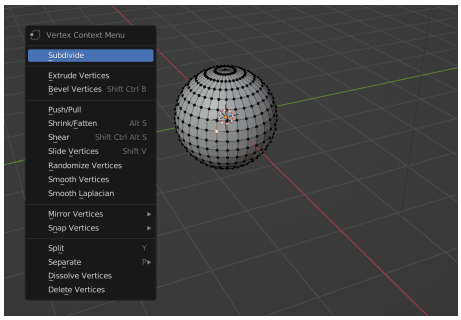


Figure 19: Creating higher polygon count spheres in Blender

In order to modify this mesh according to the target object, we first examine an image of the target object as shown in Figure 20. This gives us the height and width of the object in pixels, as well as the top and left of the object in terms of the image dimensions. The code for these mesh modifications is found in init_mesh.py.

```python
class Initialiser:
    def __init__(self, image, im_dim=256):
        self.visible = (image[0] > 0).astype(int)
        self.objheight, self.top = self.vert()
        self.objwidth, self.left = self.horiz()
        self.im_dim = im_dim

    def vert(self):
        rows = []
        for i, row in enumerate(self.visible):
            if np.isin(1, row):
                rows.append(i)
        return rows[len(rows)-1] - rows[0], rows[0]

    def horiz(self):
        cols = []
        for i, col in enumerate(np.transpose(self.visible)):
            if np.isin(1, col):
                cols.append(i)
        return cols[len(cols)-1] - cols[0], cols[0]
```

Figure 20: Inferring information about the target object from a target image

The first step is to scale the x and y dimensions of the sphere according the the aspect ratio and size of the target object. The code for this is shown in Figure 21.

```python
width_scale = orig_rad / (self.objwidth/2)
height_scale = orig_rad / (self.objheight/2)

self.horizontal_scale(mesh, width_scale)
self.vertical_scale(mesh, height_scale)
```

(a) Calculate the scale factors

```python
def horizontal_scale(self, mesh, scale):
    x = torch.transpose(mesh.vertices[0], 0, 1)[0]
    x = torch.unsqueeze((x / scale), 0)
    i = torch.arange(mesh.vertices[0].size(0)).long()
    mesh.vertices[0,i,0] = x

def vertical_scale(self, mesh, scale):
    y = torch.transpose(mesh.vertices[0], 0, 1)[1]
    y = torch.unsqueeze((y / scale), 0)
    i = torch.arange(mesh.vertices[0].size(0)).long()
    mesh.vertices[0,i,1] = y
```

(b) Scale the mesh accordingly

Figure 21: Scaling the initial mesh according to the target object

We can then shift the initial mesh according to where in the 256x256 image the target object

is. The code for this is shown in Figure 22.

```
orig_top = (self.im_dim/2) - orig_rad/height_scale
orig_left = (self.im_dim/2) - orig_rad/width_scale
shift_per_pixel = 1/orig_rad
horiz_shift = (orig_left - self.left) * shift_per_pixel
vert_shift = (orig_top - self.top) * shift_per_pixel

self.horizontal_shift(mesh, horiz_shift)
self.vertical_shift(mesh, vert_shift)
```

(a) Calculate the shift amounts

```
def horizontal_shift(self, mesh, shift):
    x = torch.transpose(mesh.vertices[0], 0, 1)[0]
    x = torch.unsqueeze((x - shift), 0)
    i = torch.arange(mesh.vertices[0].size(0)).long()
    mesh.vertices[0,i,0] = x

def vertical_shift(self, mesh, shift):
    y = torch.transpose(mesh.vertices[0], 0, 1)[1]
    y = torch.unsqueeze((y + shift), 0)
    i = torch.arange(mesh.vertices[0].size(0)).long()
    mesh.vertices[0,i,1] = y
```

(b) Shift the mesh accordingly

Figure 22: Translating the initial mesh according to the target object

2.4 Fitting Procedure: Pranav Talluri

The fitting procedure is relatively unchanged. We still use the Adam optimiser to optimise the negative IoU loss between renders of the target image and the ground truth target image. The function used to compute the loss is shown in Figure 23b. The original paper uses $\beta = (0.9, 0.999)$, however, we found that $\beta = (0.5, 0.99)$ produces better results, qualitatively and quantitatively, in practice. Figure 23a shows this and the other initialisations made. The code for the fitting procedure is given in build_mesh.py. The code iterated in each loop to improve the mesh is shown in Figure 23c.

```
model = Model(self.mesh_init).to(self.device)
transform = sr.LookAt(viewing_angle=15)
lighting = sr.Lighting()
rasterizer = sr.SoftRasterizer(image_size=256, sigma_val=1e-4, aggr_func_rgb='hard')
optimizer = torch.optim.Adam(model.parameters(), 0.01, betas=(0.5, 0.99))
```

(a) Set up model and initialisation

```
def neg_iou_loss(predict, target):
    dims = tuple(range(predict.ndimension())[1:])
    intersect = (predict * target).sum(dims)
    union = (predict + target - predict * target).sum(dims) + 1e-6
    return 1. - (intersect / union).sum() / intersect.nelement()
```

(b) Loss computation

```
images_pred = rasterizer(mesh)
loss = neg_iou_loss(images_pred[:, 3], images_gt[:, 3]) + \
    0.03 * laplacian_loss + \
    0.0003 * flatten_loss
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

(c) Iterated code

Figure 23: Code for fitting procedure

3 Experiments and Evaluation

3.1 Datasets

3.1.1 ALOI and ShapeNet: Pranav Talluri

ALOI is the main dataset that we use to test and evaluate the performance of the fitting procedure. It consists of photo-realistic images of small objects taken from 72 angles. This is in contrast to the dataset used by the original paper's authors, ShapeNet, which consists of rendered images of very simple computer generated models from multiple angles and multiple elevations. We still perform some testing on ShapeNet, specifically to verify the benefits of mesh initialisation. Samples from ALOI are shown in Figure 9 and samples from ShapeNet are shown in Figure 8.

The reasons we chose to use ALOI in our investigations align with the differences between it and ShapeNet. We wanted to explore whether SoftRas was capable of fitting objects when given high-detail, photo-realistic images. The lack of multiple elevations was also an interesting test to see whether SoftRas could reconstruct appropriate meshes given variation in only two axes. In reality, the lack of multiple elevations was causes issues for some of the objects. Specifically, when an object has internal folds (i.e., a concave structure), without vertical angles, there is no way for SoftRas to identify the structure.

Most importantly, the added challenge in the ALOI dataset further highlights the need for auto-initialising the other parameters in order to achieve good reconstruction results.

As well as testing on ALOI, we perform some testing on natural images that we captured ourselves. This highlights the improvement in functionality that our initialisations enable. Specifically that the silhouettes and camera parameters of natural images can be estimated before the fitting procedure, improving the compatibility of the fitting procedure with far more data.

3.1.2 Natural Images: Pranav Talluri

This data is the most challenging for the pipeline for multiple reasons.

Firstly, the success of the pipeline depends on successful segmentation of images from every viewpoint of the object. This is possible in our current pipeline but testing has highlighted the need for retraining DeepLabv3 on a more suitable dataset in order to segment backgrounds from small objects with simple backgrounds.

Next, the estimation of camera parameters introduces another source of inaccuracy in the reconstruction, based on how well the viewpoints can be inferred using the SfM pipeline.

Finally, compared to large-scale datasets like ALOI, the natural images we capture are likely to have more inaccuracy in alignment, as well as more variability in lighting and viewpoints.

3.2 Silhouette Initialisation: Pranav Talluri

We test whether our silhouette generation pipeline works on novel, natural images that we capture. We perform two tests: on images of a bottle, and images of a potted plant.

The segmentation on the bottle performs quite well. With the vast majority of the 72 viewpoints getting good silhouettes. Occasionally, a silhouette will pick up elements from the table as shown in Figure 24. Also, the boundaries could be a bit tighter to the bottle.

Figure 24: Segmentation performance on natural images of a water bottle

The segmentation on the pot also performs well as shown in Figure 25. It also appears that having a more even table below the object, unlike the with the water bottle where there was a piece of paper, makes it less likely that the table is picked up as the foreground.

(a) Plant pot

(b) Silhouette

Figure 25: Silhouette estimation for natural image of plant pot

3.3 Mesh Initialisation: Pranav Talluri

We explore results for various objects from ShapeNet, ALOI and natural images with and without the initialisation detailed in Section 2.3.3. Quantitatively, we can compare the losses of the various experiments. Qualitatively, we can compare the quality of the meshes produced.

3.3.1 ALOI: Pranav Talluri

3.3.1.1 Pot

The first result we explore is a pot from ALOI. We examine the results of the fitting procedure without any mesh initialisation, with initialisation and also with a higher polygon count sphere.

Quantitatively, the loss over iterations suggests that the mesh initialisations we make are successful. Specifically, the initialised mesh converges to the same loss much faster and more stably than an uninitialised mesh. The performance between the high polygon count and low polygon count meshes appears the same.

Qualitatively, we see results in line with the losses. Initialising the mesh removes the artefacts we have at the top of the pot from the uninitialised mesh. This is likely due to the fact that the pot is in the bottom half of the frame and the fitting procedure no longer has to move and scale the mesh as much. Furthermore, using the higher polygon count mesh reduces the rotational artefacts as well. We hypothesis that with an even higher polygon count mesh, we would obtain a result without any twisting artefacts.

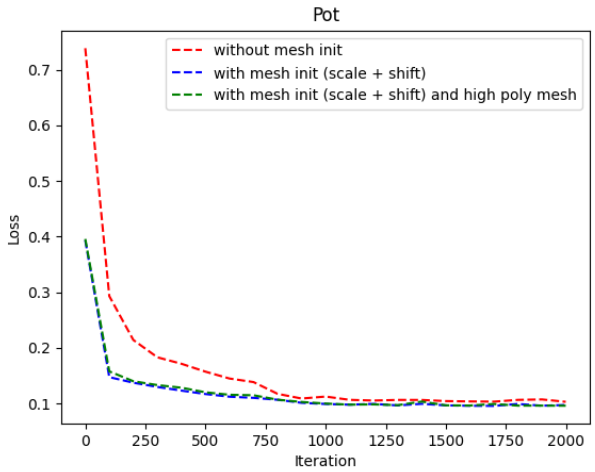
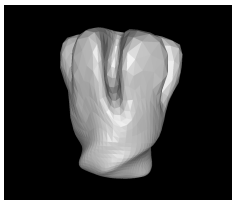
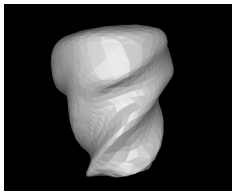


Figure 26: Quantitative comparison between mesh initialisations for a pot

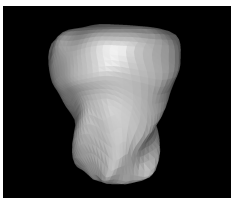
(a) Target



(b) No mesh initialisation



(c) Mesh initialisation (scale and shift)



(d) Mesh init and high poly mesh

Figure 27: Qualitative comparison between mesh initialisations for a pot

3.3.1.2 Moisturiser

Next, we examine the results for a moisturiser bottle in ALOI under the same three initialisations as the pot.

This time, the initialised mesh appears to perform worse quantitatively. i.e., the loss converges slower than with the uninitialised mesh. Although this is somewhat mitigated with the higher polygon count initial mesh. We hypothesise that this is due to the scaling of the sphere not coinciding with the shape of the bottle. Specifically, scaling the sphere to the aspect ratio of the bottle produces a vertically stretched sphere, which thins at the top and bottom. However, the bottle is just as thick at the bottom as it is in the centre and even thicker at the top. This means extra iterations are required to shift the volume of the bottle towards these edges.

Qualitatively, the initialised mesh performs the same as, if not worse than the uninitialised mesh, with both of the results suffering from twisting artefacts. However, the higher polygon count mesh rectifies this and produces a very high quality reproduction of the target object. This is likely due to the fact that introducing more vertices makes it easier for the fitting procedure to create the required cylindrical shape at the bottom of the mesh, where the initialisation simply ends in a point.

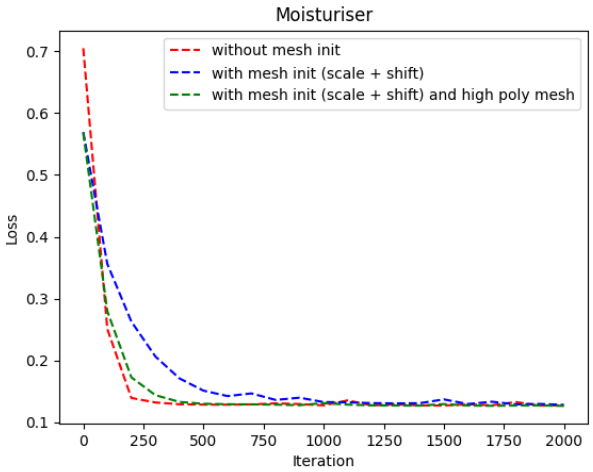
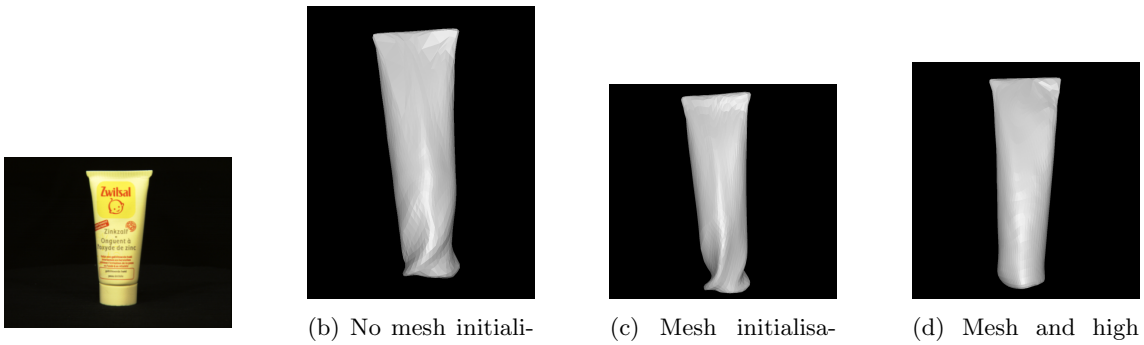


Figure 28: Quantitative comparison between mesh initialisations for a moisturiser bottle



(a) (b) No mesh initialisation (c) Mesh initialisation (scale and shift) (d) Mesh and high poly mesh

Figure 29: Qualitative comparison between mesh initialisations for a moisturiser bottle

3.3.1.3 Apricot

Next we examine an apricot from the ALOI dataset. This example was one of the motivating factors for the initialisation method, given the failure modes exhibited without any initialisation due to the size and location of the object in the frame.

Quantitatively, the mesh initialisation not only improves the speed of convergence, but also results in a better loss. We also see that using a lower polygon count initial mesh leads to further improvements in convergence time and the sample quality.

This is likely due to the fact that, as previously mentioned, when we have a small off-centre object, it is difficult for the fitting procedure to move and scale the mesh quickly and without introducing artefacts. We can see in the images of the meshes, that without an initialisation, the fitting procedure completely fails, resulting in a separation of the mesh. We hypothesise that this occurs due to the distance between the top of the mesh and the object, resulting in a lack of gradient to improve upon. This is rectified by the scaling and shifting of the mesh. Not that the shift is required when we scale here because otherwise, due to the size of the object, there would no longer be any overlap between the mesh and the image, resulting in a lack of gradient to optimise over. We also hypothesised that a lower polygon mesh would work better due to the size of the image and the artefacts found on the standard initialisation. Specifically, the excess material on the size of the mesh. In practice, we found that allowing the procedure to run for more iterations eventually results in this extra material being absorbed into the main shape, giving better qualitative results.

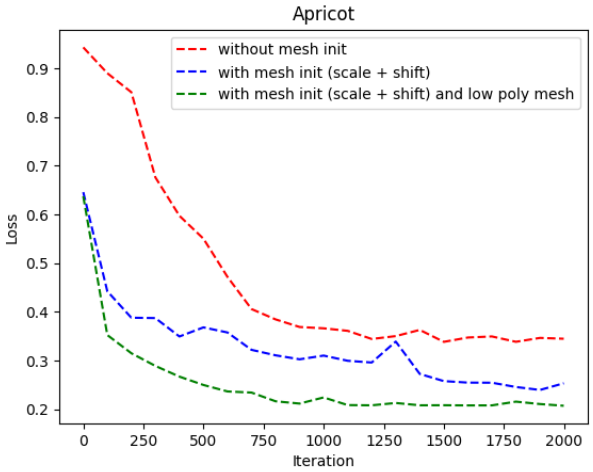
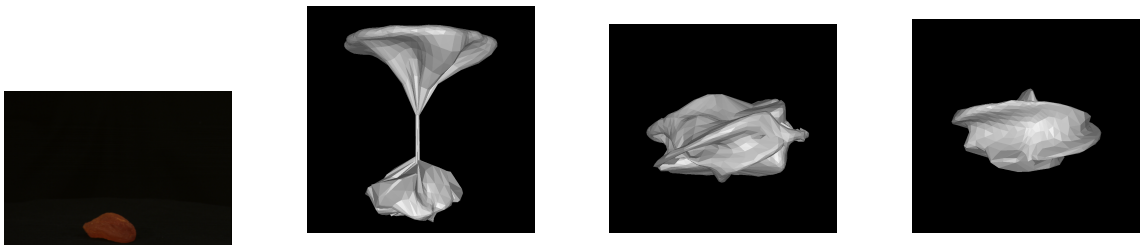


Figure 30: Quantitative comparison between mesh initialisations for an apricot



(a) Target

(b) No mesh initialisation

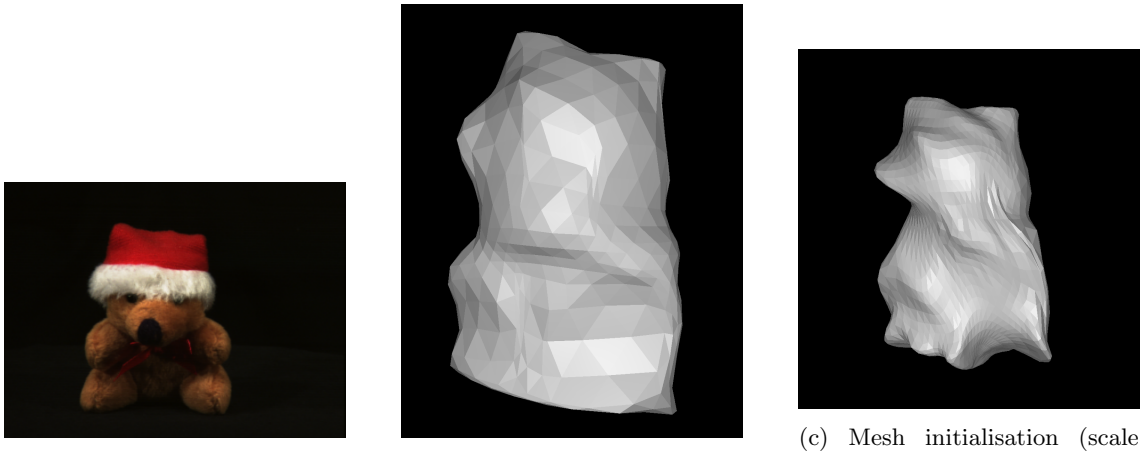
(c) Mesh initialisation (scale and shift)

(d) Mesh init and low poly mesh

Figure 31: Qualitative comparison between mesh initialisations for an apricot

3.3.1.4 Teddy Bear

Finally from the ALOI dataset, we examine a teddy bear. The key difficulty in this reconstruction, which highlights a drawback of the ALOI dataset is that SoftRas will never be able to construct the concave area between the teddy's legs without images taken from higher or lower elevations as the structure is not visible with a silhouette. This means that, although reconstructions are better with the changes to the mesh initialisation that we have made, they still do not perfectly resemble the original image.



(a) Target (b) No mesh initialisation (c) Mesh initialisation (scale and shift)

Figure 32: Qualitative comparison between mesh initialisations for a teddy bear

3.3.2 ShapeNet: Pranav Talluri

We now examine examples from ShapeNet. We start with a recreation of the original pipeline, and then add the changes we made to the mesh initialisation and image scale.

3.3.2.1 Plane

We examine the results on the airplane images. The original project performed very well on this example, producing a tight reconstruction without artefacts. The goal here is not to improve on these original results, but simply match the performance when the initialisation is used, while using less iterations. However, we found that increasing the resolution of the images and the polygon count of the mesh counteracts the effects of the mesh initialisation, which resulted in slower convergence in practice, as shown in Figure 34. However, under a fair comparison, equal polygon counts and resolutions, we found our initialisation to lead to faster convergences in our experiments. We hypothesise that the render of the plane with the higher resolution, that uses the mesh initialisation, would eventually converge to the same result.

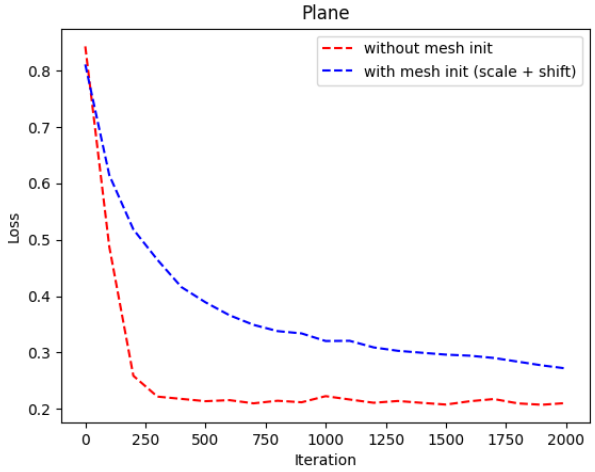
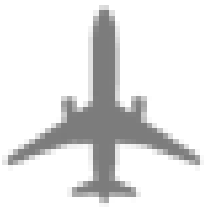
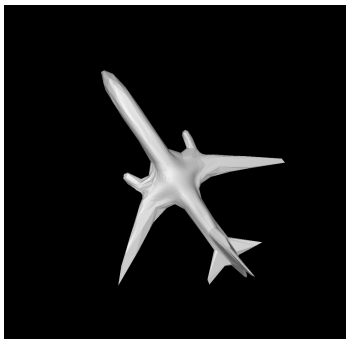


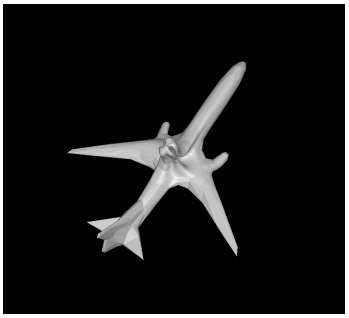
Figure 33: Quantitative comparison between mesh initialisations for a plane



(a) Target



(b) No mesh initialisation - 64x64 input



(c) Mesh initialisation - 256 x 256 input

Figure 34: Qualitative comparison between mesh initialisations for a plane

3.3.3 Natural Images: Pranav Talluri and Yujia Yang

3.3.3.1 Bottle

We test the mesh initialisation pipeline on natural images we captured of a water bottle. In this case, the mesh initialisation slows down the convergence to the final mesh. This is likely for the same reason as the moisturiser example from ALOI, where using a mesh initialisation with a taper at the top and the bottom is detrimental to convergence time for cylindrical objects. The qualitative results are impressive considering that they are a new feature of the pipeline, and hence have no previous results to compare with. It is notable that the twisting artefacts are present.

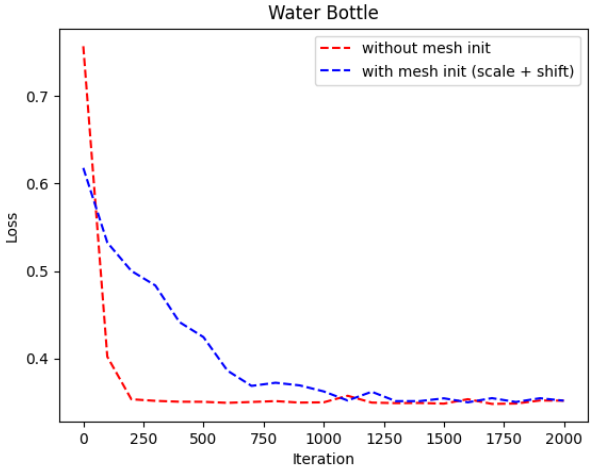
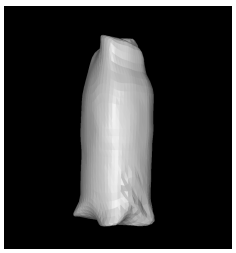
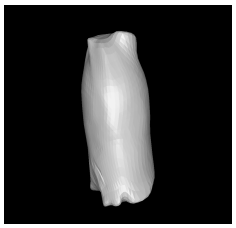


Figure 35: Quantitative comparison between mesh initialisations for a bottle

(a) Target



(b) No mesh initialisation



(c) Mesh initialisation (scale and shift)

Figure 36: Qualitative comparison between mesh initialisations for a bottle

4 Conclusions and Future Directions

4.1 Conclusions: Pranav Talluri

The main aims of this project were to:

1. Improve the speed of convergence of shape-fitting of SoftRas using mesh auto-initialisations
2. Improve the results of shape-fitting using mesh auto-initialisations
3. Extend the functionality of shape-fitting using silhouette and camera parameter initialisation

By this criteria, the project has been very successful.

The mesh initialisation performed identifies and addresses the weaknesses in the default initialisation used in the original code. This often improves the amount of time to converge taking the responsibility of position and scale of the mesh away from the fitting procedure and performing it faster and with lower likelihood of introducing artefacts.

Additionally, these changes often result in better outputs from the fitting procedure, either by mitigating the artefacts that would have otherwise been introduced in the shape fitting procedure, by utilising the extra iterations available due to the faster convergence, or because a successful fit was not originally possible (as in the case of the apricot 31).

Finally, the functionality and usability of the pipeline has been vastly improved. Originally, the fitting procedure would only accept inputs in the form of silhouettes and of a specific size. Now, due to the addition of the silhouette prediction layer and the auto-image resizing, the pipeline can accept images of any size and format and attempt to produce a mesh for a recognised object. This is vital in allowing the pipeline to be used in the real world where the ground truth silhouettes and masks are not known.

4.2 Discussion of Limitations: Pranav Talluri

4.2.1 DeepLab for Silhouettes

The first limitation we came across was utilising DeepLab for performing segmentation on natural images. Although DeepLab is very flexible in terms of its inputs, without proper training on the desired data types, it is not able to perform accurate segmentation. For example, attempts to perform segmentation on the data from ALOI rather than using the ground truth masks failed as DeepLab was not able to generalise to such a large variety of images and often would not detect any classes in the image.

4.2.2 Mesh Initialisation Methods

The second limitation was the methods we used for mesh initialisation. Although the methods used were very successful for most inputs and achieved our aims, they also highlighted some issues for some didactic examples. As previously discussed, utilising a stretched sphere is not ideal for cylindrical or cubical objects as the fitting procedure has to then shift volume from into the poles.

4.2.3 Polygon Count and Iterations

The final limitation was the limitation of computing power. We performed all testing on a single device, which meant we were limited by its performance. This meant we were not able to conduct extensive testing with even higher polygon counts or more iterations. From some initial testing, it seemed that adding more iterations could be used to give higher quality results when there were still unnecessary extrusions from the surface of the mesh.

4.3 Future Directions: Pranav Talluri

The future directions address the limitations of the current project. Especially for the mesh initialisations, there were many ideas that we hypothesise would have been useful, but we were not able to implement given the time constraints.

- Retrain DeepLab on a new dataset that better reflects the objects that we are trying to reconstruct
- Detecting cylindrical target objects and using a cylinder mesh initialisation
- Currently, the sphere scaling only applies to two dimensions, considering another image orthogonal to the first would allow us to scale the third axis as well
- Detecting cubical target objects and using a cube mesh initialisation
- Using a more powerful machine such as the HPC to test with higher polygon count initialisations and with more iterations
- Use the silhouette estimations of images to generate a 3D mesh, i.e., convert the 2D silhouette into a 3D prism to act as the initialisation

5 Contributions

5.1 Pranav

- Pranav was responsible for bulding the fitting procedure pipeline. This is the main code from which other modules are called and is built to be extensible.
- Pranav was responsible for adapting the pipeline to new datasets, i.e., dealing with higher resolution inputs and image based inputs rather than NumPy inputs.
- Pranav implemented the silhouette generation for ALOI using the given masks by generating alpha channels for the standard RGB inputs.
- Pranav implemented DeepLabv3 for novel image silhouette estimation.
- Pranav built the mesh initialisation pipeline for selecting the initial mesh, and then scaling and transforming it according to the input image.
- Pranav generated figures and renders for inclusion in the paper and the presentation.

5.2 Yujia

- Yujia captured the novel images to test in the pipeline.

5.3 Ke

- Ke worked on camera parameter estiamtion using SfM in ColMap and MeshRoom.

References

Geusebroek, Jan-Mark et al. (2005). *The Amsterdam Library of Object Images*. URL: https://doi.org/10.1023/B:VISI.0000042993.50813.60.

Chang, Angel X. et al. (2015). *ShapeNet: An Information-Rich 3D Model Repository*.

Chen, Liang-Chieh et al. (2017). *Rethinking Atrous Convolution for Semantic Image Segmentation*. URL: http://arxiv.org/abs/1706.05587.

Kato, Hiroharu et al. (2017). *Neural 3D Mesh Renderer*. URL: https://arxiv.org/abs/1711.07566.

Liu, Shichen et al. (2019). *Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning*. URL: https://arxiv.org/abs/1904.01786.

Robineau, Ariane (2021). *An overview of Differentiable Rendering*. URL: https://blog.qarnot.com/an-overview-of-differentiable-rendering/.