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

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. These approaches 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 backpropagating changes to improve the loss between them. The key requirement for inverse rendering is for the rendering operation to be differentiable for the backpropagation 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.

Inverse rendering utilises differentiable renderers to circumvent the rasterisation stage of typical rendering pipelines. Differentiability means we can backpropagate changes, so that searching for an optimal scene estimate is as simple as performing stochastic gradient descent over the scene parameters. 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.

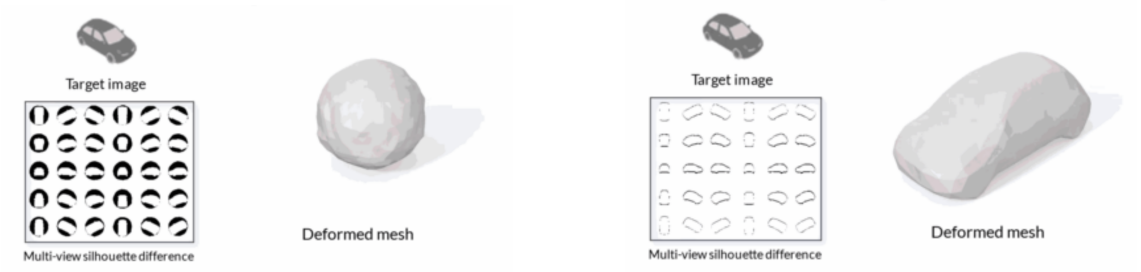
Soft rasterisation utilises a probabilistic interpretation of visibility to maintain the differentiability of the rendering pipeline. Specifically, it identifies the probability of mesh triangles contributing to pixels, rather than the traditional (undifferentiable) discrete sampling approach. The probabilities 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.

## 1.2 Background and Related Work

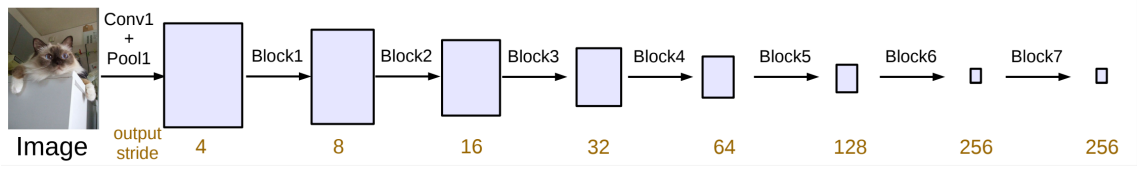
The key literature for this project is *Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning* (Liu et al. 2019). The paper introduces a novel approach to differentiable rendering, replacing the standard discrete rasterisation process with a new soft rasteriser as discussed in 2.1.1. We specifically investigate the usage of the differentiable renderer to perform inverse rendering. Figure 1 shows the fitting procedure, which is discussed further in 2.1.2.



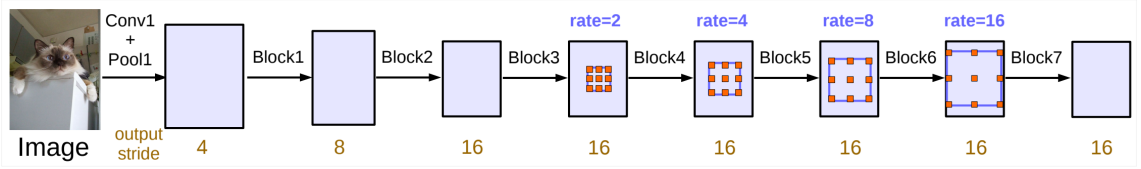

(a) Start (b) End

Figure 1: SoftRas fitting procedure

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 2. It utilises pyramid pooling modules to probe convolutional features at multiple scales.




(a) Without atrous convolutions




(b) With atrous convolutions

Figure 2: Cascaded modules with 2b and without 2a atrous convolution

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 3 where the first two rows show successful results and the last row visualises the failure modes.

## 1.3 Overview of the Idea

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 3: Samples from the original paper, including failure modes

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

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 objects from natural images.

### 1.3.2 Mesh Initialisation

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

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, photorealistic 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 (Extension) Camera Parameter Initialisation

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

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 4 compares the rendering pipeline used by SotfRas and a traditional pipeline.

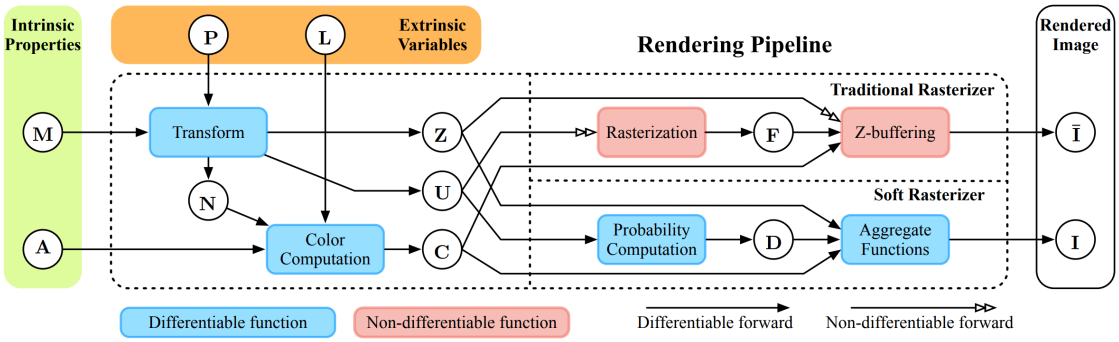



Figure 4: SoftRas vs Traditional Rendering Pipeline

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 5 shows how this compares with a traditional, discrete rasterisation pipeline.

### 2.1.2 Fitting Procedure

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 backpropagation through the rendering pipeline to modify the mesh. In fact, the original procedure takes pairs of viewpoints for every batch to compute a single loss. 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

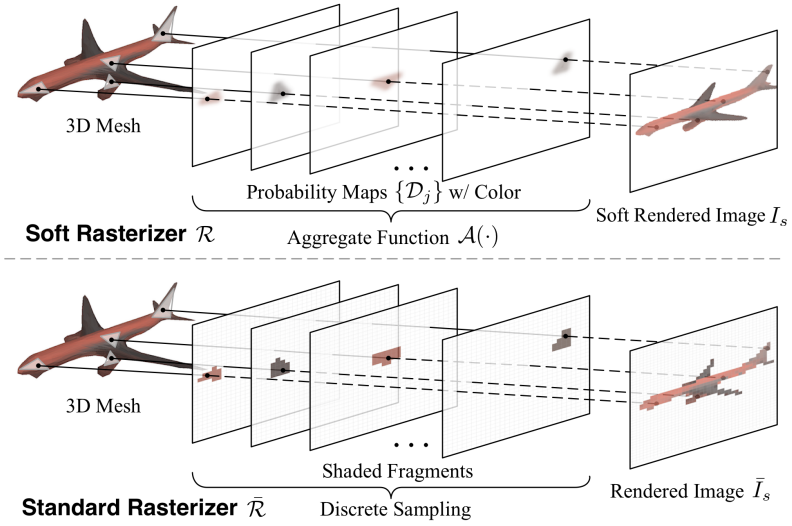



Figure 5: SoftRas vs Traditional Rasteriser

required to converge, the mesh that is converged at and whether or not the procedure is able to converge.

The loss is computed using the intersection over union (IoU). This is a common loss used when fitting shapes. 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

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. An example of an input to the original pipeline taken from ShapeNet renders is shown in Figure 6. Figure 7 shows the data pipeline used by the original implementation of the fitting procedure.

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.

## 2.2 Algorithm Improvements

### 2.2.1 Data

The original pipeline is built for images from ShapeNet. In order to utilise other data sources, we first build a pipeline for processing the input images. We modify SoftRas to operate on 256x256 input images. We then process input images, resizing them 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 more detailed images and the details for this are in 2.2.3. We test these changes using both ALOI, where the camera parameters and silhouettes are known, and images that we have

Figure 6: Silhouette Input from ShapeNet renders

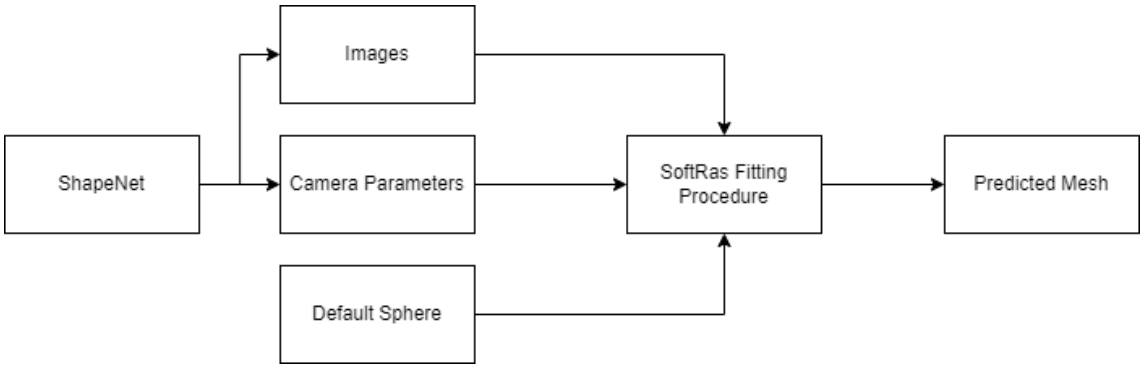
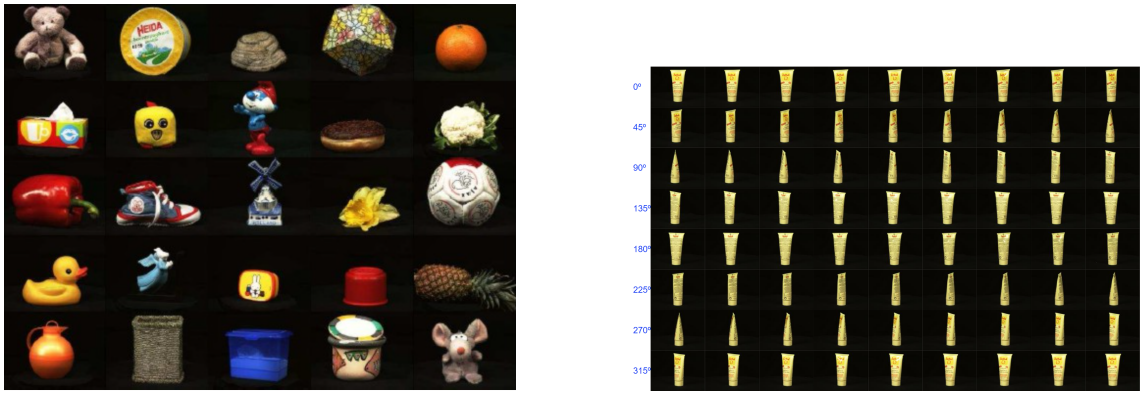



Figure 7: Data pipeline of original project

captured, where we have no viewpoint or segmentation ground truth. Some sample images from ALOI are shown in Figure 8.



(a) Sample of different objects

(b) Each object is captured from 72 angles

Figure 8: Samples from ALOI dataset.

### 2.2.2 Silhouette Initialisation

Synthetic datasets such as ShapeNet allow us to directly produced silhouettes. For carefully produced photorealistic 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 estiamted.

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.
$I = \begin{bmatrix} I_R & I_G & I_B \end{bmatrix}$, where $I_R, I_G, I_B$ are matrices with values for each pixel in image $I$.
$deeplab(i) = C$, where $C$ is a matrix of semantic classifications for each pixel.
$M = (C > 0)$, where $M$ is a matrix of binary classifications (object or background) for each pixel.
$\alpha = M \times 255$, where $\alpha$ is a matrix of alpha channel values.
$I'_{i,j} = I_{i,j} * (M_{i,j} \times 124)$, where $I'$ is a greyscale 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

We modify the initial mesh passed to the fitting procedure. The sphere used in the original pipeline, shown in Figure 9, has some key issues.

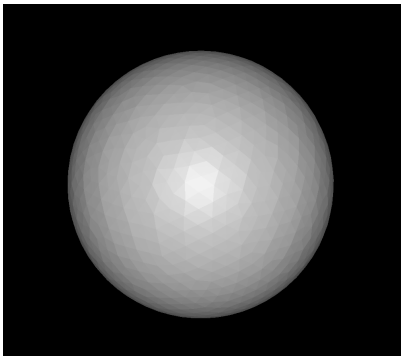


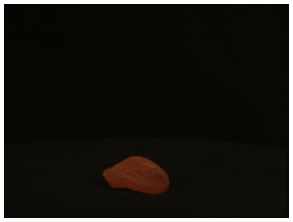
Figure 9: Sphere initialisation

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 10 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.

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

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

Another issue is that the sphere is always centred at the origin. This is not a large issue for synthetic datasets like SoftRas, but for photorealistic datasets such as ALOI or even natural images captured by a smartphone, the subject is unlikely to be centred at the origin. This means that



(a) Target object

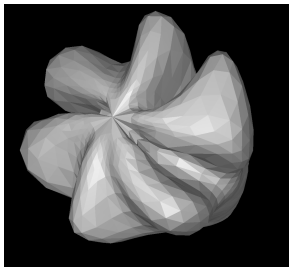
(b) Failure to converge

Figure 10: Failure to converge with sphere initialisation

the fitting procedure will utilise many iterations to move the mesh to where the subject is within the image. Additionally, this is actually achieved by moving individual vertices, which leads to stretching, distortion and twisting artefacts as shown in Figure 11. 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) Twisting artefacts due to off-centre target

Figure 11: Artefacts with sphere initialisation

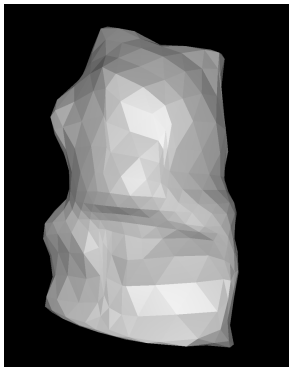
The sphere used in the original pipeline has roughly 1300 polygons. This polygon count does not vary during the fitting procedure, only the positions of the vertices. This is sufficient for the simple 64x64 images used in the original pipeline, but not for higher resolution images with more high frequency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

## 2.3 Implementation Details

### 2.3.1 Data Pipeline

The original dataset uses images and viewpoint parameters from ShapeNet stored as 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

(a) Target object



(b) Lack of detail in reconstruction

Figure 12: Lack of detail when attempting to fit to larger targets with many folds

editing the image source path. All input images are resized to 256x256 as this was found to be a sweetspot for execution time for the fitting procedure.

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 13.

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

Figure 13: 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 14. We experimented with greyscale 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.

(a) Original image (384 x 288)

(b) Ground truth mask (384 x 288)

(c) Generated silhouette (256x256)

(d) Generated silhouette greyscale (256x256)

Figure 14: Image pipeline for ALOI

### 2.3.2 Silhouette Initialisation

As stated in 2.2.2, we estimate silhouettes for natural images using segmentation. This is performed in the segment.py module as shown in Figure 15.

### 2.3.3 Mesh Initialisation

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 16. We also modify the

```python
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 15: Code used to perform segmentation using DeepLabv3

SoftRas pipeline to deal with 256x256 images, which is a simple parameter change when initialsing the soft rasteriser during the fitting procedure.

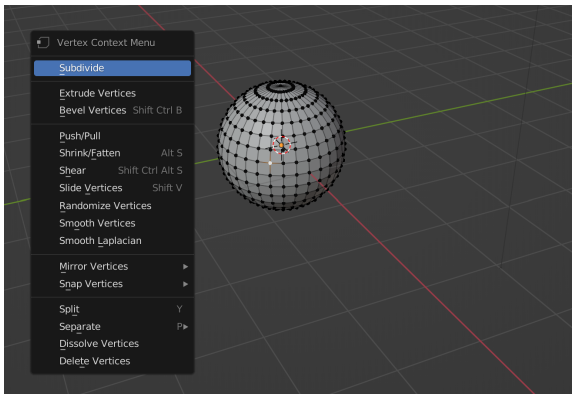



Figure 16: Creating higher polygon count spheres in Blender

In order to modify this mesh according to the target object, we first examine an image of the target object as shown in Figure 17. 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 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 18.

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 19.

## 2.4 Data Pipelines

Figure 20 shows the new data pipeline for testing using the ALOI dataset.

Figure 21 shows the data pipeline when we utilise novel images.

```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 17: Inferring information about the target object from a target image

```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 18: Scaling the initial mesh according to the target object

```python
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

```python
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 19: Translating the initial mesh according to the target object

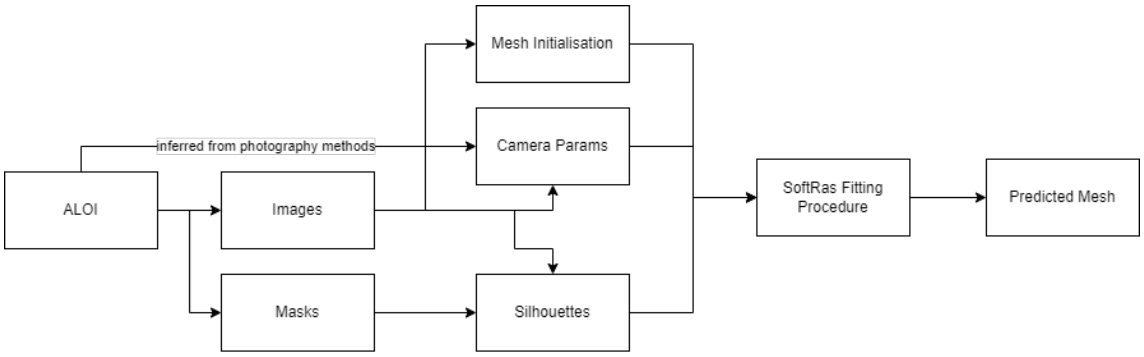



Figure 20: Data pipeline using ALOI dataset

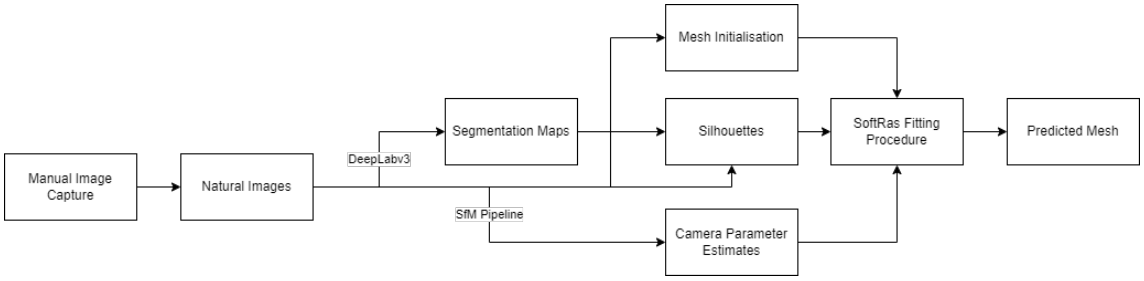



Figure 21: Data pipeline for newly captured images

## 2.5 Fitting Procedure

Modified fitting procedure from softras demo. Uses Adam optimiser.

# 3 Experiments and Evaluation

## 3.1 Datasets

A key focus of our project is to investigate the performance of SoftRas on higher resolution, photo-realistic data. For this, 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. Shown below are some samples from the dataset.

COMPARE SHAPENET AND ALOI RESULTS

### 3.1.1 Quantitive

### 3.1.2 Qualitative

## 3.2 Silhouette Initialisation

COMPARE SEGMENTED IMAGES WITH MASKS

### 3.2.1 Quantitive

### 3.2.2 Qualitative

## 3.3 Mesh Initialisation

COMPARE DIFFERENT INITIALISATIONS

### 3.3.1 Quantitive

### 3.3.2 Qualitative

## 3.4 Camera Initialisation

### 3.4.1 Quantitive

### 3.4.2 Qualitative

# 4 Conclusions and Future Directions

## 4.1 Conclusions

## 4.2 Discussion of Limitations

## 4.3 Future Directions

# 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.

# Appendicies