BTech Project(COD492)
Surveying use of Neural Surface Maps for inter-surface mapping

Sanyam Ahuja (2018CS10380)

Supervisor: Prof. Rahul Narain
Co-supervisor: Prof. Subodh Kumar

August 2022

1 Introduction

Surface maps are useful objects and are found throughout graphic processing. They are used for applying textures to a mesh, applying surface features such as normals, and parameterisation of surface, just to name a few. As such, we also wish to create a correspondence between surface maps of two surfaces. This process of transferring surface maps between two surfaces is termed inter-surface mapping. What we wish to do is to develop a representation for surface that allows for ease in calculation of inter-surface map from one mesh to another.

2 Current State of Art

Currently, there exists several kinds of techniques for constructing these inter-surface maps.
Some of them parametrize the surface [3] and then compose those parametrizations to create an intersurface map from one surface to another.
There is also literature in which they construct transport plan [1] between 2 surfaces and then optimise it to construct a conformal mapping between 2 surfaces.
An approach that is particularly intriguing is using neural nets to encode those surface maps [2]. And this is the approach that we explore in this project.

3 Neural Surface Maps

The neural surface map implementation that is used is available at this link(switch to the branch sanyama for updated code).
This is forked from the original code provided by the author of the paper, Luca Morreale.

3.1 Benefits

The use of neural surface maps has several benefits

- **Differentiability**: Since the neural nets are differentiable, it is easy to optimise over some cost function such as conformal energy.
- **Composable**: Since the neural nets are composable, several surface and intersurface maps may be composed to get a mapping from one surface to another surface
- **Optimised libraries**: There are third party open source libraries such as pytorch available which allow us to easily create and optimise neural nets.

3.2 Pre-processing

The scripts used for pre-processing are available at the given link.
There were some caveats and unspecified requirements for pre-processing the 3D models. Thus, we list down the steps concisely along with the problems faced while doing that. This will give a brief idea about the kinds of pitfalls that may be faced while creating neural surface maps.

1. The input mesh must not have any non-manifold vertices or edges(except boundary). Further, it must be homeomorphic to a disc.
   For cutting a surface, code is provided in the link above.
   Further, after cutting the surface, it should be scaled such that the size of the bounding box of the object is around 1 units. This is done because the trainer provided has fixed learning rate. This leads to erroneous surface maps for objects that are too small.

2. The next step is to recalculate a parametrization for it. For this, the code provided implements the slim parametrization. This maps the texture points in the UV space in the shape of a unit circle to points on the mesh. This is essential as the surface map takes input points in UV space and maps them to the points in 3D space.
   After parametrizing the surface, it is **essential** to calculate the normals again as they are needed during the creation of surface map and are not calculated by the slim program. To do this, meshlab can be used to recompute the vertex and face normals.

3. Next, we have to subdivide the surface until there are about 500k vertices on the mesh. This can be done using meshlab's filter to subdivide surface by midpoint.
   Again, as before, we have to recalculate the normals.

4. With the meshes prepared, the next step is to convert the sample to a form that can be read by pytorch. For this, the main repository contains a script `preprocessing.convert_sample` that takes as input the slim parametrised mesh along with its oversampled version and stores the useful information from the mesh(such as the coordinates, normals, UV parameters, etc.) in a pytorch file.

At this point, we are done with the pre-processing and can proceed to the next step.

3.3 Creating surface maps

For creating a surface map, use the script `train_surface_map` provided in the main repository.
The input to this script is just one argument - the preprocessed sample pytorch file. This can either be provided as command line argument or specified in the config file.
There are some more parameters that can be tweaked in the config file itself(such as the learning rate and the number of iterations that you want to run the code for).
After the training, run the script `show_surface_map` while also specifying the sample pytorch file in the script to get ply models.

3.4 Creating intersurface maps

For creating a surface map, use the script `train_intersurface_map` provided in the main repository.
The input to this script is three main arguments - the trained surface models of the source and destination, and the landmark points on them.
This can either be provided as command line argument or specified in the config file.
There are some more parameters that can be tweaked in the config file itself(such as the learning rate and the number of iterations that you want to run the code for).
The landmark points are the vertex numbers on the slim parametrised object file.
After the training, run the script `show_intersurface_map` while also specifying the source and destination surface maps along with landmarks in the script to get ply models.

4 Experiments

There are several parameters to consider while creating the surface maps.

- **Size**: The size should be normalised such that the bounding box size is around 1 units in size as mentioned before.
- **Number of vertices**: The number of vertices is dependent on the hardware. Running the program on cpu is not possible due to how slow pytorch is on such hardware. On the other hand, GPUs are memory limited. If the VRAM cannot hold all the data required by pytorch, then the program won't run. Thus, this code might not run for some meshes if they have too much vertices and the gpu provided to you cannot handle it.
  In such a case, consider simplifying the mesh to smaller number of vertices.
  NOTE that here only the number of vertices in the original mesh matter, not the one in the oversampled mesh.
- **Nature of cuts**: Small cuts give a skewed boundary, but take shorter time to overfit. On the other hand, longer boundary objects take longer to train(higher number of iterations).
- **Learning rate**: Higher learning rate(like 10e-3) doesn't work properly in some cases. On the other hand, slower learning rate takes way longer to run and converge.

The code was also edited to allow for tuning of the loss parameters. This hasn't been tested out satisfactorily and hence has been left out of this report. But it can be edited in the config file to see what works and what doesn't.

In the following sections, we compare the mean of the top 20 curvatures of the objects. These have been calculated in blender using the code given in the following link.
The experiments were run on a server with a GTX1080Ti having 12GB VRAM.

4.1 Surface Maps

In the following figures, we have the original model followed by the trained surface map.

Table 1: Mean of top 20 curvatures for surface maps

Model	Original Model	Surface Model
Bull	129.70	142.32
Spot	38.12	35.40
Horse	156.98	148.30
Nefertiti	206.88	225.78

Figure 1: Bull Surface Map

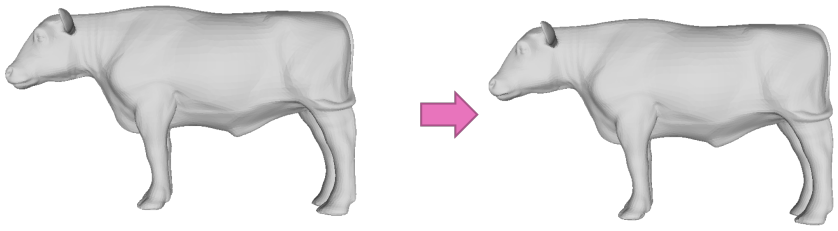


Figure 2: Spot Surface Map

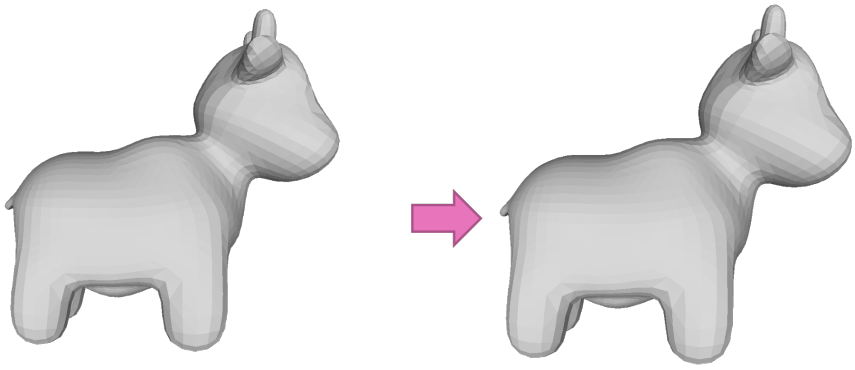


Figure 3: Horse Surface Map

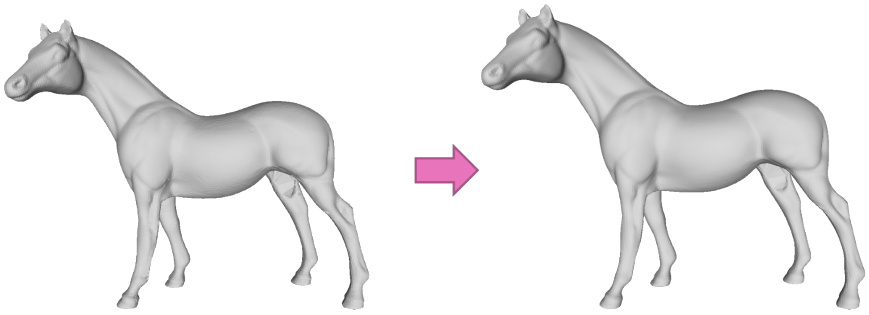
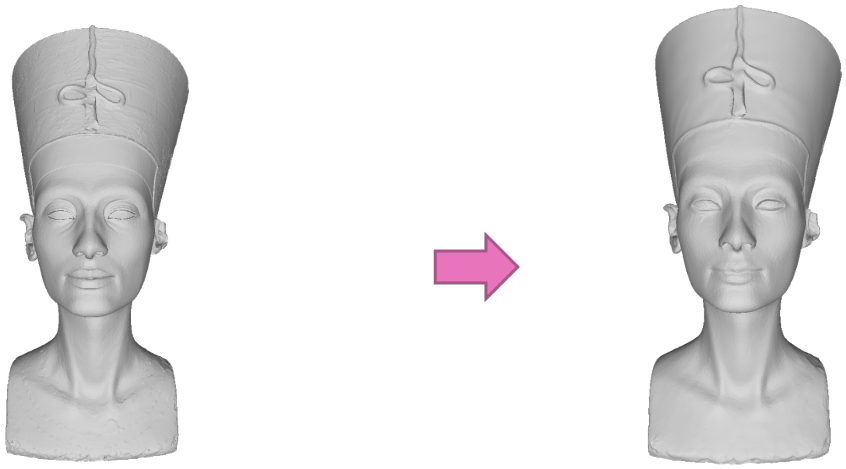


Figure 4: Nefertiti Surface Map

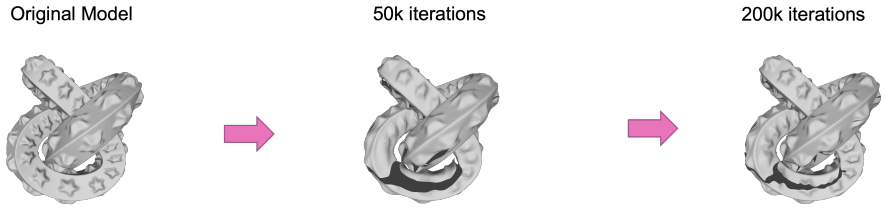


A peculiar case was training the surface map for a knot. In this case, the original model has high curvature along with a long boundary cut that covers the entire surface. We observe that the model doesn't converge visually even after 200k iterations which took about 34h to complete on our reference machine.

Table 2: Mean of top 20 curvatures for knot surface map

Model	Original Model	Surface Model with 50k iterations	Surface Model with 200k iterations
Curvature	215.82	1152.92	960.41

Figure 5: Knot Surface Map



4.2 Inter-surface Maps

In the following figures, we have the surface model followed by the trained intersurface map.
We present 2 inter-surface maps here. From camel to bull with the landmark points set to be the base of their feet. And from bimba to Nefertiti with the nose and the 2 shoulders set as the landmark points.
In case of camel to bull, we move from a higher curvature model to a lower curvature model and we see that the curvature on the inter-surface map is also significantly higher. On the other hand, in the case of bimba to Nefertiti, we move from a lower curvature model to a higher curvature model and see that the curvature of the inter-surface map also reduces significantly, albeit at the loss of quality and some jagged edges.

Table 3: Mean of top 20 curvatures for inter-surface maps

Model	Original Model	Surface Model	Inter-surface model
Camel	3362.05	2096.67	NA
Bull	129.70	142.32	434.14
Bimba	74.67	74.32	NA
Nefertiti	206.88	225.78	116.26

Figure 6: Camel to Bull Inter-surface Map

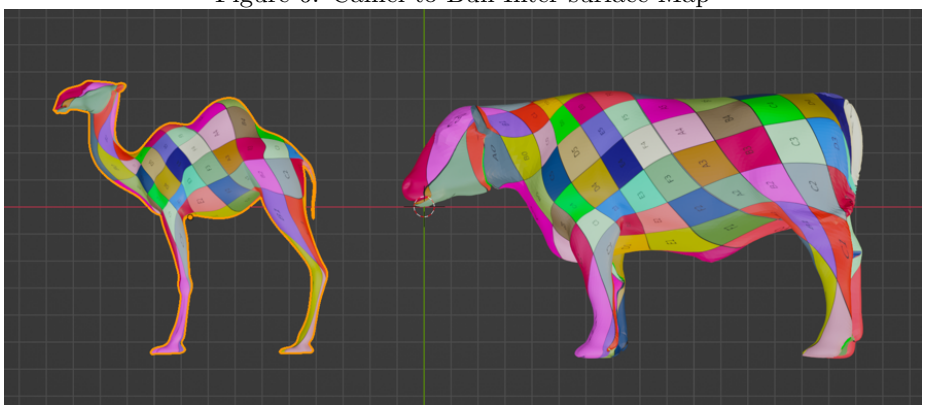
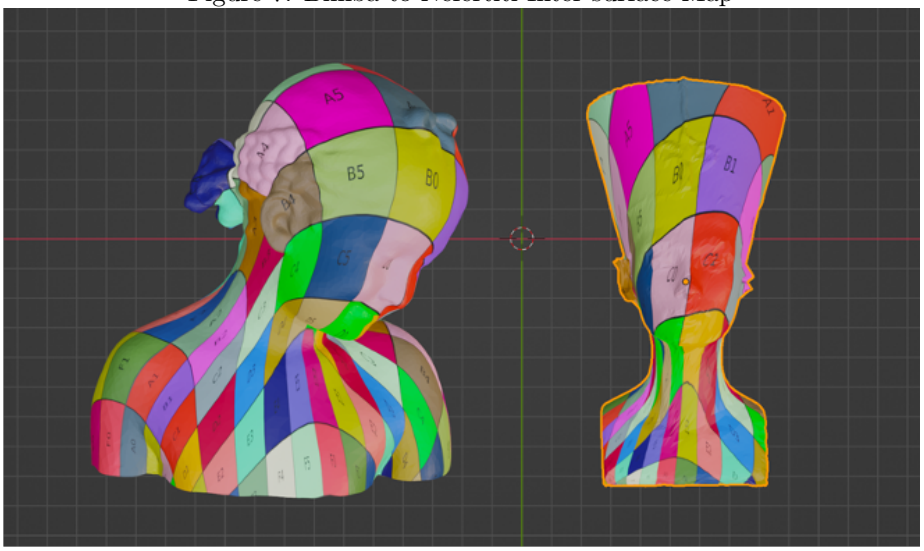


Figure 7: Bimba to Nefertiti Inter-surface Map

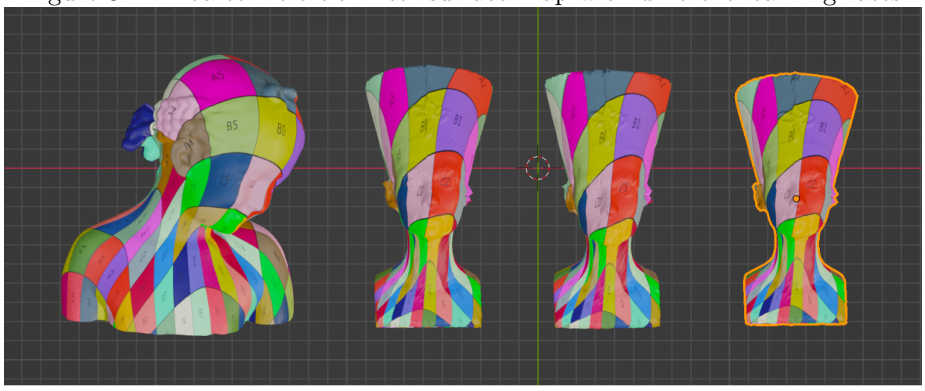


We next test how well the inter-surface maps produced are as we make the learning rate smaller. With smaller learning rate, the time taken for convergence is higher, with almost 24h required for the 1e-6 learning rate compared to 8h or so for the 1e-4 model. But as a result of the lower learning rate, the surfaces are much smoother with lower mean of top 20 curvatures.

Table 4: Mean of top 20 curvatures for Bimba to Nefertiti inter-surface map

Learning rate	1e-4	1e-5	1e-6
Curvature	116.26	96.43	85.09

Figure 8: Bimba to Nefertiti Inter-surface Map with different learning rates



5 Conclusions

- The pre-processing took a long time to get used to because the pipeline wasn't detailed enough. We have created a smoother pipeline now in this repository as mentioned before.
- While longer cuts taken longer time to train on, they are preferrable over normal models to ensure more uniform density of texture points. For smaller cuts, the boundaries look skewed.
- Lower learning rate works better in general while taking much more time.
- With higher number of vertices, more VRAM is used. Sometimes, this leads to pytorch running out of memory even before the whole model is loaded into VRAM. Since VRAM is separate from RAM, this can lead to bigger models even failing to run without the use of specialized hardware with more VRAM.
- While this model performs well in some areas while requiring way less complex code due to the use of open source libraries, the time taken for each model is very long and thus making it unsuitable for on the fly operation in large scale applications

6 Future Works

- One possible step is to do more testing with different loss energies and learning rates and determine what kind of parameters work for what kind of meshes.
- Optimisations are also required to reduce the amount of data loaded into the VRAM. In pursuit of this, testing can be done by using a neural net model different from the one used in the paper referenced.

- Can explore the use of CPU to train the model. This eliminates the need for VRAM and is thus beneficial for bigger models.
  While CPU doesn't perform well when training ML models, there exist languages like Halide by MIT which optimise the workflow for CPU architecture and have significant gains in performance.

References

[1] Manish Mandad, David Cohen-Steiner, Leif Kobbelt, Pierre Alliez, and Mathieu Desbrun. Variance-minimizing transport plans for inter-surface mapping. *ACM Trans. Graph.*, 36(4), jul 2017.

[2] Luca Morreale, Noam Aigerman, Vladimir Kim, and Niloy J. Mitra. Neural surface maps, 2021.

[3] John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. Inter-surface mapping. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, page 870–877, New York, NY, USA, 2004. Association for Computing Machinery.