# Assignment 3: Neural Rendering and Shape Processing

Probabilistic Machine Learning for Visual AI
by Helge Rhodin

This assignment is on neural rendering and shape processing—computer graphics. We provide you with a dataset of 2D icons and corresponding vector graphics as shown in the figure below. It stems from a line of work on translating low-resolution icons to visually appealing vector forms and was kindly provided by Sheffer et al. [1] for the purpose of this assignment.

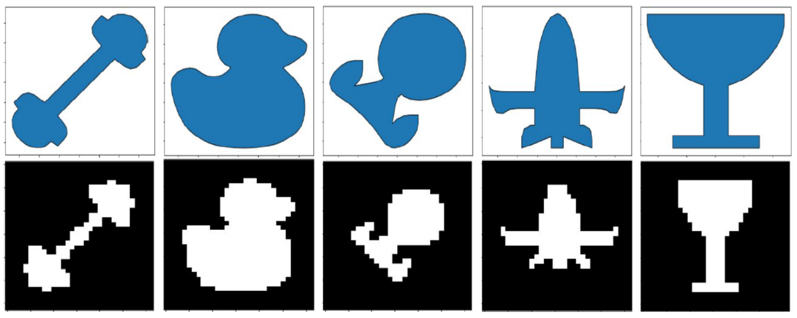

Figure 1: Icons.

**Dependencies.** This project utilizes a Conda environment to manage all necessary dependencies, which are listed in the provided environment.yml file.

**Main tasks.** The overall goal of this assignment is to find transformation between icons. We provide the 'ImagerIcon' dataset as an HDF5 file and dataloading functionality, with training and validation splits, as well as display and training functionality.

Images are 32 x 32 icon bitmaps stored as $1 \times W \times H$ tensors. Vector graphics are represented as polygons with $N = 96$ vertices and are stored as $2 \times N$ tensors, with neighboring points stored sequentially. Entry points for your code are marked with `# TODO`.

- **Task I (1 point): Neural Rendering** Starting from the code in 'icon_generator.py', implement a neural network that takes a batch of vector images as input and outputs corresponding bitmap images, i.e., that can map the top row of the teaser figure to the bottom row. This is an instance of neural rendering via supervised learning. The core of your task is to implement a neural network that uses upscaling layers, such as transposed convolution (aka. deconvolution) with stride 2 or more. On top of that, you can use any other neural network layers, such as 'ReLU', 'linear', 'batch norm', 'max pooling', and 'bilinear interpolation' operations. Choose a suitable loss function as the criterion.
- **Task II (1 point): Training a simple auto encoder** Train the simple autoencoder (AE) provided in 'polygon_ae.py' for polygons. In the subsequent tasks, we will improve this autoencoder to learn a more meaningful space, such that it is suitable for interpolation. Inspect the red dot in the output, which indicates the starting point of the polygon. Note that although the polygon is drawn as a closed loop, we have to store it as a sequence with a start and end. Application of the MSE

loss compares the positions of the polygon vertices in the order they are stored in memory. However, the start changes arbitrarily for each of the vector icons and is difficult to predict for the network. The **Chamfer distance** is designed to handle such mismatch via nearest neighbor comparisons. Implement the ‘chamferdistance‘ function to train the AE a second time and explain why it creates fold-overs in the polygon.

- **Task IIIa (1 point): Beyond Chamfer** To avoid the issues with Chamfer, implement a new distance metric ‘start_invariant_MSE‘ that is inspired by Chamfer but better exploits the polygon structure. It should measure the similarity of prediction and label using the MSE between pairs of points, be invariant to the start and end position. Test your new loss. It should avoid the fold-overs you have observed in the previous task.
- **Task IIIb (1 point):** The output of the loss must not change when the start and end point is changed in either of the polygons while keeping the vertex positions fixed. Your ‘start_invariant_MSE‘ implementation may not fulfill these requirements or have other bugs. To be sure, construct toy examples with the same points but different starting points. Make it simple enough such that you can compute the result by hand and show proper working. Make sure your toy example and loss works correctly not only with individual polygons but batches of polygons, as required during training.
- **Task IV (1 point):** The new training loop contains data augmentation code that randomizes the start point for every iteration for improving generalization to new shapes (the red dot, indicating the starting point, is random). Although the new loss makes the objective invariant, the network remains sensitive to the order of points and does not train well (give it a try). It is your task to improve the provided autoencoder to be invariant to the polygon starting point. You could use transformer layers or 1D convolutions (e.g., with kernel size 1 or 3), MaxPooling, ReLU activations, and global pooling to construct the encoder that is free of linear (fully-connected) layers. In particular, 1D convolution with ‘padding_mode=’circular’’ is useful. For us, 6 to 8 convolutional layers and a couple of max-pooling, ReLU, and BatchNorm layers worked for us, but other architectures and network depths are likely to improve.
- **Task V (1 point):** The representation learned by autoencoders is well-suited for interpolating within the manifold of feasible shapes, a so-called ‘shape space‘. It is your task to test different ways of interpolating between two vector icons. Interpolate two point clouds $M_1$ and $M_2$ from the training dataloader, i.e., $M_{\text{out}} = \lambda M_1 + (1 - \lambda) M_2$. Display the outcome for $\lambda \in (0, 0.2, \ldots, 1)$. Apply your encoder from task IV on $M_1$ and $M_2$ and save the corresponding latent codes $h_1$ and $h_2$. Interpolate these latent codes and reconstruct them using the decoder. Plot these as before. Try a few different icon pairs and describe and explain the difference between interpolating points and interpolating in the shape space. Do you see issues? Try a couple of different input examples and explain your observations.
- **Task VI (4 points):** Turn your Task IV auto-encoder into a generative model, either a Variational Auto Encoder (VAE), a diffusion model, or normalizing flow. You can also choose to build the generative model on the images (Task 1) instead of the polygon data, but the image domain requires larger models and more compute. Attend the upcoming tutorials to receive guidance!

**Hints and comments.**

- You must use a machine with GPU support as training is otherwise too slow.

- Pay attention to your model complexity. How many parameters does the baseline autoencoder have? How many parameters does your model have.
- The provided dataset is relatively small scale. One epoch contains only a handful of images. You will have to run many epochs. Training deeper neural networks takes a while. Don't draw conclusions too early. Particularly, your network from Task IV may take 1000 or 2000 iterations to show reasonable results.
- Due to the small dataset, you have to focus on a compact architecture with relatively few parameters. Still, some amount of overfitting is acceptable/expected for this assignment. It is intentionally kept small to limit the required resources.

**Submission.** After completing your work, please compress your folder containing all Python scripts, graphs, and results into a zip file and upload it to Moodle. Ensure that you do not include downloaded datasets or any temporary files in your submission (it would take too much space).

## References

[1] Shayan Hoshyari, Edoardo A. Dominici, Alla Sheffer, Nathan Carr, Duygu Ceylan, Zhaowen Wang, I-Chao Shen *Perception-Driven Semi-Structured Boundary Vectorization.* SIGGRAPH, 2018