Overview

In this assignment you will implement four implicit surface reconstruction algorithms to approximate a surface represented by scattered point data. The problem can be stated as follows:
Given a set of points $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ in a point cloud, we will define an implicit function f(x,y,z) that measures the signed distance to the surface approximated by these points. The surface is extracted at f(x,y,z) = 0 using the marching cubes algorithm. All you need to implement are four implicit functions that measure distance in the following ways: (a) signed distance to tangent plane of the surface point nearest to each point (x,y,z) of the grid storing the implicit function (b) Moving Least Squares (MLS) distance to tangent planes of the K nearest surface points to to each point (x,y,z) of the grid storing the implicit function, (c) Radial Basis Function (RBF) interpolation for approximating the signed distance function, (d) a DeepSDF variant for approximating the signed distance function. The scikit-image package already provides implementations of marching cubes. Thus, all you need to do is to fill the code in the specified scripts to implement the above implicit functions. The implicit functions rely on surface normal information per input surface point. In the provided test data files, surface normals are included (the format of the point cloud file is: 'point_x_coordinate' 'point_y_coordinate' 'point_z_coordinate' 'point_normal_x' 'point_normal_y' 'point_normal_z' [newline]).
There are three test point cloud (bunny-500.pts, bunny-1000.pts, and sphere.pts) that you will experiment with.
Download the starter code and data below. You need to install the packages 'plotly' and 'skimage' (version >= 0.13) to run the code.

This assignment counts for **23 points** towards your final grade.

What You Need To Do (23 points in total)

[3 points] One way to estimate the signed distance of any point $\boldsymbol{p}=\{x,y,z\}$ of the 3D grid to the sampled surface points $\boldsymbol{p}_i$is to compute the distance of $\boldsymbol{p}$ to the tangent plane of the surface point $\boldsymbol{p}_i$ that is nearest to $\boldsymbol{p}$. In this case, your signed distance functon is:
$f(\mathbf{p}) = \boldsymbol{n}_j \cdot (\boldsymbol{p} - \boldsymbol{p}_j)$ *with* $j = \text{argmin}_i \{ \|\boldsymbol{p} - \boldsymbol{p}_i\| \}$
Your task: Implement this distance function in the 'naiveReconstruction' script. Show screenshots of the reconstructed bunny (500 and 1000 points) and sphere in your report.
[4 points] The above scheme results in a C^0 surface (i.e., the derivatives of the implicit function are not continuous). To get a smoother result, the Moving Least Squares (MLS) distance from tangent planes is more preferred. The MLS distance is defined as the weighted sum of the signed distance functions to all points $\boldsymbol{p}_i$:
$f(\mathbf{p}) = \boldsymbol{\Sigma}_i d_i(\mathbf{p}) \varphi(\|\boldsymbol{p} - \boldsymbol{p}_i\|) \; / \; \boldsymbol{\Sigma}_i \varphi(\|\boldsymbol{p} - \boldsymbol{p}_i\|)$
where:

$d_i(\mathbf{p}) = \boldsymbol{n}_i \cdot (\boldsymbol{p} - \boldsymbol{p}_i)$

$\varphi(\|\boldsymbol{p} - \boldsymbol{p}_i\|) = \exp(-\|\boldsymbol{p} - \boldsymbol{p}_i\|^2 / \beta^2)$

Practically, computing the signed distance function to all points $\boldsymbol{p}_i$ is computationally expensive. Since the weights $\varphi(\|\boldsymbol{p} - \boldsymbol{p}_i\|)$ become very small for surface sample points that are distant to points **p** of the grid, in your implementation you will compute the MLS distance to the K=20 nearest surface points for each grid point.

Your task: Implement this distance function in the 'mlsReconstruction' script. You will also need to compute an estimate of $(1/\beta^2)$. Set β to be twice the average of the distances between each surface point and its closest neighboring surface point. Show screenshots of the reconstructed bunny (500 and 1000 points) and sphere in your report.

[6 points] Another approach is to determine a signed distance function based on a RBF network. This is done by defining the distance function as a weighted sum of RBF basis functions with centers $\boldsymbol{c}_k$:

$f(\mathbf{p}) = \boldsymbol{\Sigma}_k w_k \varphi(\|\boldsymbol{p} - \boldsymbol{c}_k\|)$

where w_k are unknown coefficients of the thin plate spline basis functions:

$\varphi(r) = r^2 \log(r)$, where $r = \|\boldsymbol{p} - \boldsymbol{c}_k\|$

For our purposes, the centers of the basis functions will be the sampled points in addition to offset points:

$\{\boldsymbol{p}_i, \boldsymbol{p}_i + \varepsilon \boldsymbol{n}_i, \boldsymbol{p}_i - \varepsilon \boldsymbol{n}_i\}$

(total 3N centers where N is the number of input points). The solution for the weights comes from solving a linear system incorporating the on-surface and off-surface constraints.

Your task: Implement the above implicit function in the 'rbfReconstruction' script. Include in your report the value of ε yielding the best reconstruction results and show screenshots of the reconstructed bunny (500 and 1000 points) and sphere for that ε value in your report.

[10 points] Another more modern approach is to determine a signed distance function based on the single-shape DeepSDF variant (DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation, CVPR 2019). This is done by regressing the SDF from point samples using a deep network. More specifically, given a set of 3D point samples **P** and their SDF values $\mathbf{S} = \{s_1, s_2, \ldots, s_n\}$, we train the parameters θ of a multi-layer fully-connected neural network f_θ on (**P**, **S**) to make f_θ a good approximator of the given SDF.

$s_i = f_\theta(\boldsymbol{p}_i), \ \boldsymbol{p}_i \in \boldsymbol{P}, \ s_i \in \boldsymbol{S}$

The network architecture is shown in the following figure. In detail, it takes as input the point coordinate $\mathbf{p}_i = (x,y,z)$ and passes it through 8 fully-connected layers (FC in figure). The size of the vector representaton in each hidden layer has size 512 as shown in the figure. Note that after the fourth FC layer, the layer outputs the hidden vector with 509-dim, which is concatenated with the original 3-dim point coordinate (x,y,z) to form the final 512-dim vector (see blue dashed line in the figure). In the first seven FC layers (i.e., except the last), we append a weight normalization layer (nn.utils.weight_norm), a ReLu non-linear activation layer and a dropout layer (dropout rate=0.2) after it. Finally, the predicted SDF value is obtained with the last FC layer which transforms the vector dimension from 512 to 1 and another tanh activation layer (denoted as TH in the figure).

The training is done by minimizing the sum over losses between the predicted and real SDF values of points under the clamped L_1 loss function:
$L(f_\theta(\boldsymbol{p}_i), s_i) = |clamp(f_\theta(\boldsymbol{p}_i), \sigma) - clamp(s_i, \sigma)|$
where *clamp(x, σ)* clamps *x* value within the boundary σ, i.e. min(σ, max(-σ, x)).
For our purposes, the 3D points are sampled around the given surface points along their normal (or inverse-normal) directions: $\{\boldsymbol{p}_i+\varepsilon\ \boldsymbol{n}_i\}$ where ε is randomly sampled from a Gaussian distribution $N(0, 0.05^2)$. For each point $\mathbf{p}_i$, n=80 samples are used for training and validation set.
The neural network weights are optimized by the Adam optimizer. The default learning rate is 0.0001, weight decay is 0.01. We train the network for 80 epochs with batch size=1024. The loss for training and validation set are reported per epoch. We save the best model with the lowest loss on the validation set. Detailed training hyper-parameters can be found in the training code argument parser.
Your task: Implement the above DeepRBF network in the 'train.py', ''model.py' and 'util.py' scripts. Show screenshots of the reconstructed bunny (500 and 1000 points) and sphere based on the best model you trained in your report.

Submission

Please follow the **Submission** instructions in the course policy to upload your zip file to Moodle. The zip file should contain all your Python code **plus a short PDF report including the results**.