Homework 2

Release: 02/01/2021 Due: Fri. 02/19/2021, 11:59 PM

- You are **allowed** to consult any external resources but you must cite them. You are also **allowed** to discuss with each other but you need to acknowledge them. However, your submission must be your own work; specifically, you must **not** share your code or proof.
- Your submission has 3 parts. The first part is a single PDF file, containing proof, code, and results of problem 1, as well as the report for problem 2 and 3. The second part is a zip file containing your code for problem 2 and 3. The third part is a submission to the leaderboard.
- This homework is worth 20/100 of your final grade.
- This homework itself contains 25 points and **10** extra credit!

Problem 1 (Deform a shape). In this problem, we will practice part of what we learned in the image-to-3D lecture for shape deformation.

1. (Laplacian) Given a mesh $M = (V, E, F)$, we assume that the adjacency matrix is $A \in \mathbb{R}^{n\times n}$, $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix where $D[i,i]$ is the degree of the i-th vertex. The **Laplacian** matrix is defined as $L = D - A$.

   Prove that:

   (a) $\sum_{(i,j)\in E} \|x_i - x_j\|^2 = x^T L x$ for $x \in \mathbb{R}^n$. [1pt]

   (b) $L \in \mathbb{S}^n_+$, i.e., L is a symmetric and positive semi-definite matrix. [1pt]

   (c) For the data matrix $P \in \mathbb{R}^{n\times 3}$ where each row corresponds to a point in $\mathbb{R}^3$, denote the columns of P as $P = [x, y, z]$ and rows of P as $P = [p_1^T; p_2^T; \ldots; p_n^T]$, show that $\sum_{(i,j)\in E} \|p_i - p_j\|^2 = x^T L x + y^T L y + z^T L z$. (hint: Use the conclusion from 1(a)) [1pt]

2. **Normalized Laplacian** is defined as the normalized version of the Laplacian matrix above:

$$L_{norm} = D^{-1} L \tag{1}$$

   (a) Prove that the sum of each row of L_{norm} is 1. [1pt]

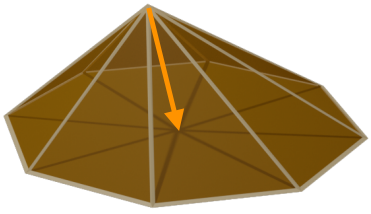

Figure 1: Curvature approximation by discrete Laplacian

   (b) The difference between a vertex x and the average position of its 1-ring neighborhood is a quantity that provides interesting geometric insight of the shape (see Figure 1). It can be shown that,

$$x - \frac{1}{|N(x)|} \sum_{y_i \in N(x)} y_i \approx H\vec{n}\Delta A \tag{2}$$

for a good mesh, where $N(x)$ is the 1-ring neighorhood vertices of x by the mesh topology, $H = \frac{1}{2}(\kappa_{min} + \kappa_{\max})$ is the mean curvature at x (in the sense of the underlying continuous surface being approximated), $\vec{n}$ is the surface normal vector at x, and ΔA is a quantity proportional to the total area of the 1-ring fan (triangles formed by x and vertices along the 1-ring).

Define $\Delta p_i := p_i - \frac{1}{|N(p_i)|}\sum_{p_j \in N(p_i)} p_j$. Prove that $\Delta p_i = [L_{norm}P]_i$, where P and p_i are defined as in 1(c), and $[X]_i$ is to access the i-th row of X. [1pt]

3. (programming, **extra credit**) Please load the `source.obj` and `target.obj` files using the trimesh library of Python, and optimize to deform the vertices of the `source.obj` to match `target.obj`. Plot the source object, target object, and deformed object. [5pt]

   (a) Warm-up by Chamfer-only loss: Use the provided Chamfer distance function as a loss to deform the source towards the target. Optimize the position of the vertices in the source mesh by torch (you can just use the Adam optimizer). Show the result and describe what has happened in the deformation process by language.

   (b) Curvature and normal-based loss: We observe that Chamfer loss alone is not able to deform the source mesh properly, and additional loss needs to be added to regularize the process. Here we introduce a simple idea that would work for the provided instances by matching $\{\Delta p_i\}$:

   - First, we compute the Δp_i for each vertex of the source and the target meshes using 2(b).
   - Then, when we compute Chamfer-loss as in 3(a), we actually know about the correspondences across the source and target mesh vertex sets. For each pair of correspondences found in computing the Chamfer loss, we can use the L_2-norm square difference between the corresponding Δp_is' as the loss. Note that this loss computation can be bidirectional (from source to target and from target to source).

   Implement this loss in pytorch and combine it with the Chamfer loss to deform the source shape. Show your final results and deformation process (*e.g.*, the deformed mesh at every 100 steps).

   Note: `source.obj` and `target.obj` have similar amount of vertices, so the ΔA for vertices in the two meshes do not differ much, and matching Δp_is' corresponds to match mean curvatures and normal direction. If there is significant difference in vertex numbers, we need to compensate for the effect caused by significantly different ΔA in the two meshes.

Problem 2 & 3. In the next problems, you will use 2 methods, ICP and PointNet, to predict object poses in a dataset. The training data and testing data are provided at
https://drive.google.com/drive/folders/11TPw-XOypMLEgZLXUwFdBEHBPaMi0VCs?usp=sharing
You can download the data or directly use it in Google Colab by mounting the folder.
Read the following carefully and ask on Piazza immediately if you have questions.

1. The task is to predict 6D poses for all the objects of interest in the scene given the image and the depth map. There are 79 different object classes in total. In each scene, there is only one instance for each object class. Across different scenes, instances of an object class might be scaled differently (in fact, only a discrete range, e.g. 0.5 or 1.0). This scale is provided in both training and testing datasets.

2. Here are some notations you might need to know before reading the rest.

   - NUM_OBJECTS: total number of objects we have. it is 79 in this assignment.

- object_id: each object class is assigned with a unique id, decided by its order in objects.csv.

3. Data description. In training data, there are 2 folders: v2.2 and splits/v2, and there is a file object_v1.csv. The v2.2 folder contains the training data and splits/v2 is a pre-made train/validation split. You are allowed to create your own train/validate split and ignore splits/v2 completely. In splits/v2/train.txt and splits/v2/val.txt, each line is in the format "{level}-{scene}-{variant}". You can use it to find corresponding data files in the v2.2 data folder. Each variant with the same {level}-{scene} uses the same set of object classes with different poses. Here we give a detailed description of all files.

   - **objects.csv**. It provides metadata for the object meshes used to generate the scenes. The important fields are:
     - location: described where to find the meshes in models.zip.
     - geometric_symmetry: describe what symmetrical properties this object has. For example "z2|x2" means this object has a 2 fold symmetry around z axis and a 2 fold symmetry around x axis (which also implies a 2 fold symmetry around y axis). You can see Wikipedia entry Rotational_symmetry for details. "no" means this object has no symmetry; "zinf" means this object has an infinite-fold symmetry around z (e.g. cylinder). The symmetry properties are considered in our evaluation metric (e.g., for a cube, there are 24 rotations that will result in 0 error in evaluation).
     - visual_symmetry: similar to geometric symmetry, but considers object texture. Our evaluation metric will NOT consider visual symmetry, but it is an interesting research topic.
   - **{level}-{scene}-{variant}_color_kinect.png**: an RGB image captured from a camera containing the target objects.
   - **{level}-{scene}-{variant}_depth_kinect.png**: a depth image captured from a camera containing the target objects. The depth is in the unit of mm. You need to convert it into m.
   - **{level}-{scene}-{variant}_label_kinect.png**: a segmentation image captured from a camera containing the target objects. The segmentation id for objects are from 0 to 78.
   - **{level}-{scene}-{variant}_meta.pkl**: camera parameters, object names, gorund-truth poses, etc.
     - poses_world (list): The length is NUM_OBJECTS; a pose is a 4x4 transformation matrix (rotation and translation) for each object in the world frame, or None for non-existing objects.
     - extents (list): The length is NUM_OBJECTS; an extent is a (3,) array, representing the size of each object in its canonical frame (without scaling), or None for non-existing objects. The order is xyz.
     - scales (list): The length is NUM_OBJECTS; a scale is a (3,) array, representing the scale of each object, or None for non-existing objects.
     - object_ids (list): the object ids of interest.
     - object_names (list): the object names of interest.
     - extrinsic: 4x4 transformation matrix, world $\rightarrow$ viewer(opencv)
     - intrinsic: 3x3 matrix, viewer (opencv) $\rightarrow$ image
   - We provide a Jupyter notebook to show how to use the data and also some handy scripts for visualization.
   - In testing_data, The only difference from training data is that there are no splits and the ground truth poses are not provided in the metadata.

- There is also the models.zip, which provides the meshes and textures for the objects. You will need to sample canonical-space point cloud from the meshes for ICP.

4. Goal. Your goal is to predict the poses of objects in the testing data. The segmentation masks for objects are given so you do not need to solve the detection problem. You can simply segment out the point cloud for each object and use PointNet or ICP to solve the problem. You need to do it first using ICP and then using neural network.

5. Output format. We provide a sample output file "result.json". You need to read and understand the submission format.

6. Evaluation metrics. Your assignment is graded based on the pose accuracy score: success if rotation error < 5 degree and translation error < 1 cm.

   Your score is computed as the higher of absolute and relative score.

   Update: to make sure the time of computation is reasonable and the difficulty of this homework is fair, we will only use **level 1 and 2** for this assignment. So you only need to train and test on levels starting with **1-** and **2-**. You should ignore all other levels for now, however, they will be part of the final project.

   Absolute score

   - 5 points: if you achieve higher than 90% on the overall scoreboard.
   - 4 points: 70% accuracy on the overall scoreboard.
   - 3 points: 60% accuracy on the overall scoreboard.
   - 2 points: 50% accuracy on the overall scoreboard.
   - 1 points: Beats trivial baseline.

   Relative score

   - 5 points: rank 1-5.
   - 4 points: rank 6-15.
   - 3 points: rank 16-20.
   - 2 points: rank 21-25.
   - 1 points: the rest.

7. Submission. The submission has 3 parts

   - A short report describing your method and result. (A short version of "our method" and "experiments" section of academic publications. 1-2 pages should be enough, but you may write more if you have interesting findings.) This report should be combined with the solution of problem 1 into a single PDF.
   - A .zip archive containing your code.
   - You need to submit your results on testing data to our internal benchmark.
     https://storage1.ucsd.edu/cse291ibenchmark/benchmark
     The benchmark will be online this week.
     A benchmark submission will take about 60s, please be patient and please avoid trying to submit multiple times.
     The benchmark follows the following format

- The json is a dict whose keys are {level}-{scene}-{variant} for each level. The length of this dict must be the same as the number of test cases.
- The value of each dict is another dict with a single key "poses_world". The value for "poses_world" is a list of 79 elements. Each element is either a None or a 2D list of size (4×4), representing the pose matrix (transformation from object frame to world frame) of the object.
- For example, if the data contains object with ID 4 and 5, then the list should have 77 None values and 2 matrices at position 4 and 5.
- A trivial baseline "baseline.json" is provided. You can submit it to have fun.
- You are free to choose your user name, however, you cannot use another name after the initial submission.

8. You should always assume each neural network experiment will take 1 whole day on a GPU. So you should start solving the problems **now**. If you are using the free version of Colab, make sure you save and load the network weights often as there is a running time limit.

Problem 2 (ICP-based pose estimation). Solve the problem above with an ICP-based method. You will need to sample canonical-space point cloud from the provided object meshes. You do not need training for this part.
Scoring. report: 5pt, benchmark ranking: 5pt.
Problem 3 (Learning-based pose estimation). Solve the problem above with a learning-based method. We recommend using PointNet.
Scoring. report: 5pt, benchmark ranking: 5pt.
Extra credit (Combine 2 and 3). Find a way to combine problem 2 and 3 and improve your score. Describe your method in report and show increase of performance in the benchmark system: up to 5pt.