Homework 1

1 Surface Computing [4pt]

Problem 1. In this problem, you will do a bit of calculus to see how the operators we introduced in the class work on a simple ellipsoid. Consider the map $f : \mathbb{R}^2 \to \mathbb{R}^3$ defined by

$$f(u,v) = \begin{bmatrix} a\cos u \sin v \\ b \sin u \sin v \\ c \cos v \end{bmatrix}, \text{where } -\pi \le u \le \pi, 0 \le v \le \pi.$$

The function f maps the 2D domain to an ellipsoid.

Let $a = 1, b = 1, c = \frac{1}{2}$. Let $\mathbf{p}= (u, v)$ be a point in the domain of f, and let $\gamma : (-1, 1) \to \mathbb{R}^2$ be a curve with $\gamma(0) =\mathbf{p}$ and $\gamma'(t) =\mathbf{v}$.

1. [Programming Assignment] Let $\mathbf{p}= (\frac{\pi}{4}, \frac{\pi}{6})$ and $\mathbf{v}= (1, 0)$. Draw the curve of $f(\gamma(t))$ on the ellipsoid's surface.

   Hint: You can use open3d.geometry.TriangleMesh to create various simple geometries. To visualize the curve in 3D space, you can show sampled 3D points from the curve, or use piece-wise cylinders to approximate the curve.

2. **Differential map:** The differential of a function f at a point $\mathbf{p}$ is a linear map. Equivalent to how we define the differential in class, we can also define by the gradient of the curve w.r.t. the curve parameter: $Df_{\mathbf{p}}(\mathbf{v}) = f(\gamma(t))'|_{t=0}$.

   (a) What is $Df_{\mathbf{p}}$? Express it as a matrix.

   (b) Describe the geometric meaning of $Df_{\mathbf{p}}$.

   (c) [Programming Assignment] Draw $Df_{\mathbf{p}}(\mathbf{v})$ on the ellipsoid when $\mathbf{p}= (\frac{\pi}{4}, \frac{\pi}{6})$ and $\mathbf{v} = (1, 0)$.

   (d) What is the normal vector of the tangent plane at $\mathbf{p}$?

   (e) [Programming Assignment] Give a group of orthonormal bases of the tangent space at $f(\mathbf{p})$ when $\mathbf{p}= (\frac{\pi}{4}, \frac{\pi}{6})$, and draw it on the ellipsoid.

3. **Normal:** Given $\mathbf{p}= (\frac{\pi}{4}, \frac{\pi}{6})$ and $\mathbf{v}= (1, 0)$. For simplicity, let $g_{\mathbf{v}}(t) = f(\gamma(t))$ denote the curve which passes through $\mathbf{p}$ at $t = 0$.

   (a) What is the arc length $s(t)$ as the point moves from $g_{\mathbf{v}}(0)$ to $g_{\mathbf{v}}(t)$?

(b) Give the arc-length parameterization $h_{\mathbf{v}}(s)$ of the curve.

(c) What is the normal vector of the curve at a point $h_{\mathbf{v}}(s)$? Hint: Use $h_{\mathbf{v}}(s)$ to derive the normal.

Note: Note that $h_{\mathbf{v}}(0)$ is the point $\mathbf{p}$. If you compare the curve normal you get in 3(c) with the surface normal in 2(d) at $s = 0$, you can find that they are different.

4. **Curvature:** In 2(d), you have computed the normal at $\mathbf{p}$. Denote this normal as $N_{\mathbf{p}}$.

   (a) Compute the differential of the normal $DN_{\mathbf{p}}$, and express it as a matrix. Hint: You can use WolframAlpha to compute complicated derivatives that you don't want to compute by hand.

   (b) Find the eigenvectors of the shape operator at $\mathbf{p}$. Hint: You can show the shape operator is diagonal(What does it tell you about the eigenvectors?).

   (c) [Programming Assignment] Draw the two principal curvature directions in the tangent plane of the ellipsoid at $\mathbf{p}= (\frac{\pi}{4}, \frac{\pi}{6})$.

   (d) Compute the Gaussian curvature of the surface f at $\mathbf{p}= (\frac{\pi}{4}, \frac{\pi}{6})$, and demonstrate that ellipsoid f doesn't show isometric invariance with any spherical surface.

2 3D Geometry Processing [5pt]

Problem 2. In this exercise, you will practice common processing routines of point cloud and 3D mesh. All questions are programming assignments.

1. Given mesh **saddle.obj**, sample *100K* points uniformly on the surface. You may use a library(such as Trimesh or Open3d) to do it.

2. Use the iterative farthest point sampling method to sample *4K* points from the *100K* uniform samples. This algorithm is required to be implemented by yourself. You may only use computation libraries such as *Numpy* and *Scipy*. Hint: To accelerate computation, use Numpy matrix operations whenever possible.

3. **Normal estimation:** At each point of the *4K* points, estimate the normal vector by Principal Component Analysis using *50* nearest neighbors from the *100K* uniform points(You can use sklearn.decomposition.PCA). Since making the direction of normals consistent is a non-trivial task, for this assignment, you can orient the normals so that they roughly points in the Y direction.

   Note 1: You can use sklearn.neighbors.KDTree to efficiently extract nearest neighbors.

Note 2: We also provide a saddle point cloud **saddle.ply** with point outliers. You can try integrating denoising algorithms (e.g. RANSAC) to improve the robustness of your method.

4. **Mesh curvature estimation:** Rusinkiewicz's method is a an effective method for face curvature estimation. Given a 3D triangular mesh, we assume the surface function can be parameterized as $f(u, v) \in \mathbb{R}^3$ in a local small triangle, where $[u, v] \in \mathbb{U}$ is a 2D coordinate. Since the triangle is small, we also assume that the tangent planes at each of the three vertices are approximately parallel, and $\mathbb{U}$ is roughly aligned with these tangent planes (i.e. $D_f \approx [\xi_u, \xi_v]$, where D_f denotes the first-order derivative of f). Since f can be arbitrarily defined, ξ_u and ξ_v can be defined as orthogonal vectors for simplicity.

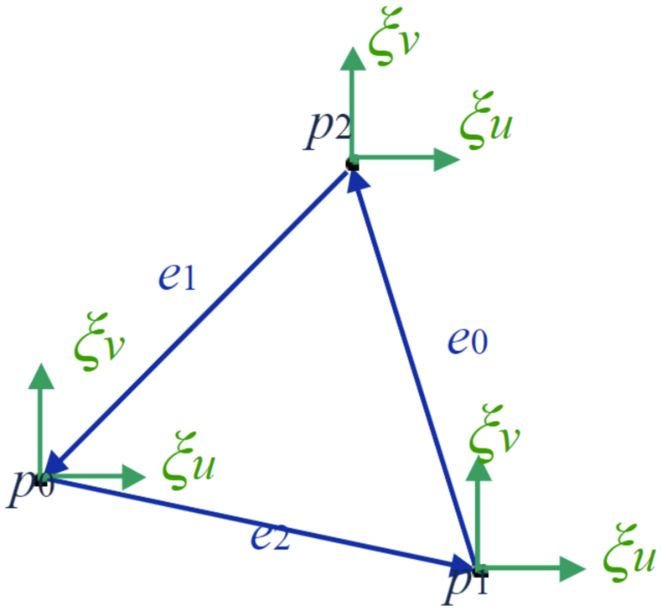

Figure 1: Assumptions in Rusinkiewicz's method.

Let $D_N = [\frac{\partial \vec{n}}{\partial u}, \frac{\partial \vec{n}}{\partial v}] \in \mathbb{R}^{3\times2}$ denotes the first-order derivative of the normal. For each point on the surface, we define $S \in \mathbb{R}^{2\times2}$ that satisfies $D_N = D_f \cdot S$. S describes the normal change when the point moves along any direction. The principle curvatures at the point can be defined as the two eigenvalues of S.

When we consider a small triangle on a mesh, $S = D_f^T D_N$ since $D_f = [\xi_u, \xi_v]$. According to the approximation that $D_N \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} \approx \Delta \vec{n}$ and

$D_f \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} \approx \vec{e} = \Delta p$, we have

$$\begin{cases} SD_f^T(p_2 - p_1) = D_f^T(\vec{n_2} - \vec{n_1}), \\ SD_f^T(p_0 - p_2) = D_f^T(\vec{n_0} - \vec{n_2}), \\ SD_f^T(p_1 - p_0) = D_f^T(\vec{n_1} - \vec{n_0}), \end{cases} \tag{1}$$

where we use subscripts $\{0, 1, 2\}$ to distinguish three different vertices of the triangle. We can directly compute S by the least-square method (4 variables and 6 equations).

We can build up the following routine to implement Rusinkiewicz's method:

- Step 1: Load p_i, $\vec{n_i}$, and the face normal $\vec{n}$
- Step 2: Select two orthogonal vectors ξ_u and ξ_v on the tangent plane of the face
- Step 3: Define $D_f = [\xi_u, \xi_v]$, design the optimization problem by Equation 1 and compute S by least-square
- Step 4: Compute the eigenvalues of S as the principal curvatures

Use Rusinkiewicz's method to compute the principle curvatures for each triangular face in **icosphere.obj** and **sievert.obj**. Show your results by histograms. You may use *trimesh* or *open3d* to load the mesh and compute normals. However, you should not use any function that computes curvatures (the function may not compute them correctly).

In fact, the surface (**Sievert's surface**) in **sievert.obj** is locally isometric to a region on a sphere! If you compute Gaussian curvature (the product of principle curvatures), then you can find the Gaussian curvatures of most points equal to a constant 1.

3 Rotation [4pt]

Problem 3. Let $p = (1 + j)/\sqrt{2}$ and $q = (1 + k)/\sqrt{2}$, denote the unit-norm quaternions. The rotation $M(p)$ is a 90-degree rotation about the Y axis, while $M(q)$ is a 90-degree rotation about the Z axis. Here we composed the two rotations $M(p)$ and $M(q)$. Here, we instead investigate the rotation that lies halfway between $M(p)$ and $M(q)$.

The quaternion that lies halfway between p and q is simply

$$\frac{p+q}{2} = \frac{1}{\sqrt{2}} + \frac{j}{2\sqrt{2}} + \frac{k}{2\sqrt{2}}$$

1. Calculate the norm $|(p + q)/2|$ of that quaternion, and note that it is not 1. Find a quaternion r that is a scalar multiple of $(p + q)/2$ and that has unit norm, $|r| = 1$, and calculate the rotation matrix $M(r)$. Around what axis does $M(r)$ rotate, and through what angle(to the nearest tenth of a degree)?

2. What are the exponential coordinates of p and q?

3. **Skew-symmetric representation of rotation:** In this problem, we use $[\omega]$ to represent a skew-symmetric matrix constructed from $\omega \in \mathbb{R}^3$ as instructed in class:

   (a) Build the skew-symmetric matrix $[\omega_p]$ of p and $[\omega_q]$ of q, and derive their rotation matrices.

   (b) Using what you have above to verify that the following $\exp([\omega_1] + [\omega_2]) = \exp([\omega_1]) \exp([\omega_2])$ relationship does **not** hold for exponential map in general(Note: The condition for this equation to hold is $[\omega_1][\omega_2] = [\omega_2][\omega_1]$). Therefore, composing rotations in skew-symmetric representation should not be done in this way.

4. **Double-covering of quaternion:** What are the exponential coordinates of $p' = -p$ and $q' = -q$? What do you observe by comparing the exponential coordinates of $(p, -p)$ and $(q, -q)$? Does this relation hold for any quaternion pair $(r, -r)$? If it does, write down the statement and prove it.

4 Shape Approximation [7pt]

Problem 4. Given a mesh such as a Stanford Bunny, use spheres to approximate the given mesh. The objective is maximizing the volume covered while ensuring the total number of spheres does not exceed a predefined number N. Please locate the meshes to be used for this assignment in the folder `objs_approx`. You are free to choose the sphere radius and the maximum number of spheres N; however, you should provide clear and well-reasoned justifications for the values you choose. For each mesh, we expect you to create multiple problem settings with different N and sphere radius. Subsequently, solve the problem under these different configurations and perform a thorough analysis of both efficiency and quality.
(Sub Problem 1) Mesh Volume Estimation. Estimate the volume of a 3D mesh (e.g., Stanford Bunny).

Please implement the following two estimation methods.

- Voxelization. Estimate the volume by converting the mesh into a voxel. Please vary the resolution and perform comparisons and analysis regarding the efficiency and the accuracy.

- Signed Distance Field (SDF) + Monte Carlo.
  - Compute the SDF of the mesh.
  - Randomly sample points in the mesh's bounding box.
  - Use the SDF to classify points as inside/outside.
  - Volume $\approx$ (Fraction of inside points) $\times$ (Bounding box volume).

You can use Python packages such as `trimesh` and `PyVista` for SDF computing and sampling. In addition to simply implementing these methods, conduct some comparisons between these two methods and compare computational trade-offs.

Deliverables. 1) Code for voxelization and Monte Carlo volume estimation. 2) Report comparing accuracy, speed, and memory usage of both methods.

(Sub Problem 2) Mesh Intersection Volume Estimation. Compute the intersection volume between two meshes (e.g., an apple and the Stanford Bunny). Develop test cases for this subproblem using the meshes provided in the `objs_approx` directory. For example, place an apple and a bunny in the same global coordinate system and vary their poses to create intersecting configurations. Your test cases should cover a range of intersection ratios, from no intersection (0) to near the maximum possible intersection ratio.

For each mesh pair, create at least five test cases with intersection ratios spanning this range. Note that determining the exact maximum possible intersection ratio is not required for this assignment; simply attempting to maximize overlap is sufficient. Ensure your submitted package includes the generated test cases and that your report provides corresponding visualizations for each case.

Please implement the following two estimation methods.

- Voxelization. 1) Voxelize both meshes. 2) Compute the overlapping voxels. 3) Intersection volume = (Number of overlapping voxels) $\times$ (Voxel size)^3.
- Monte Carlo Sampling. 1) Sample points in the bounding box of the overlapping region. 2) Check if points lie inside both meshes (e.g., using SDFs). 3) Intersection volume $\approx$ (Fraction of overlapping points) $\times$ (Bounding box volume).

Deliverables. 1) Code to compute the intersection volume between two meshes. 2) Analysis of method accuracy vs. computational cost. 3) Created test cases and their visualizations.

(Sub Problem 3) Shape Approximation with Spheres. Approximate the mesh's volume using a limited number of spheres to maximize coverage. Spheres can overlap with each other.

Algorithm Design:

- **Constraints.** 1) All spheres must lie entirely within the mesh. 2) The number of utilized spheres should not exceed a pre-defined number N. 3) For each problem setting, all spheres must have the same radius.
- **Goal.** Maximize the union of sphere volumes.
- **Approaches.** Implement the following two methods:
  - Greedy Algorithm. Iteratively place the sphere in the largest uncovered region.

- Optimization. Formulate as a constrained optimization problem (e.g., maximize total sphere volume with the number of spheres constraint and the requirement of sphere inside the mesh). Solve the optimization problem (feel free to use any packages or libraries).

Coverage Calculation:

- Covered volume = (Sum of sphere volumes) − (Sum of pairwise overlaps) + ... (inclusion-exclusion principle).
- Coverage percentage = (Covered volume / Total mesh volume) $\times$ 100%.

Deliverables. 1) Code for sphere placement and coverage calculation. 2) Report analyzing coverage and efficiency under various problem settings with different maximum number N and sphere radius. 3) Results with visualizations.