COMP0119 Assignment1

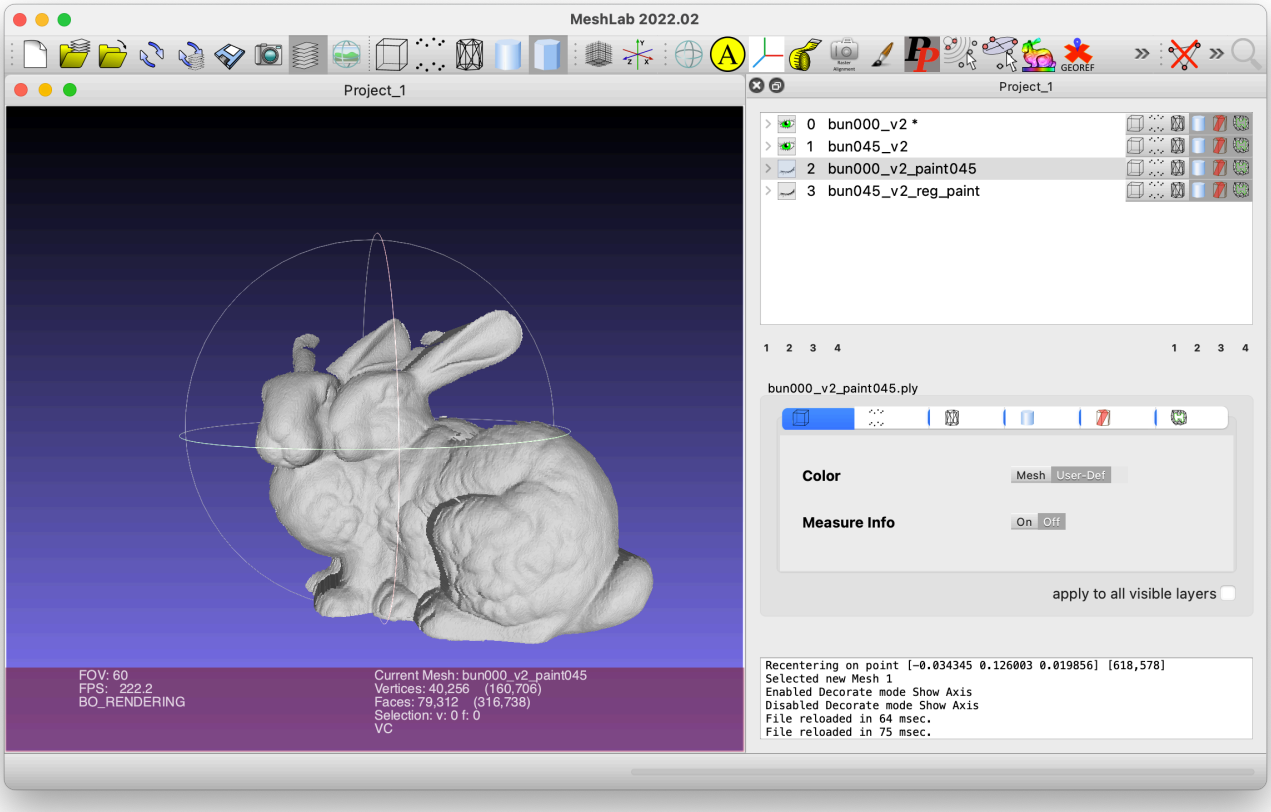
Platform Information

```
OS:                 macOS Ventura
Processor:          M2 Pro Simulating Intel by Rosetta
Environment:        Python 3.9.15
Python Packages:    open3d, numpy, sklearn, matplotlib
```

Part 1

Visualize the meshes

I use Meshlab to visualize the meshes. Below is the image of `bunny_v2/bun000_v2.ply` and `bunny_v2/bun045_v2.ply` being shown in a common coordinate system.



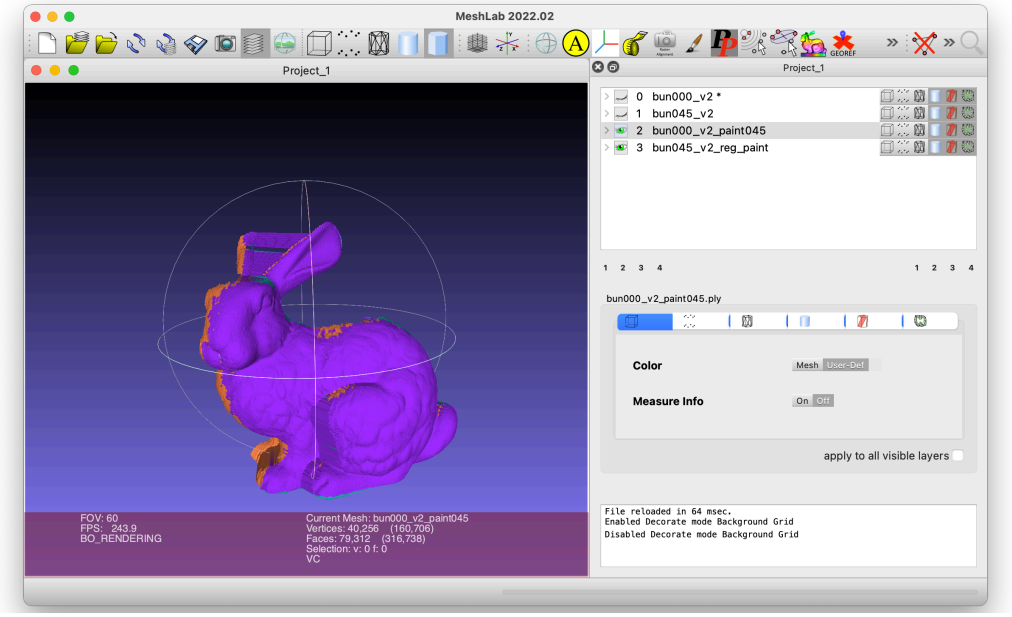
Implementation of the ICP algorithm

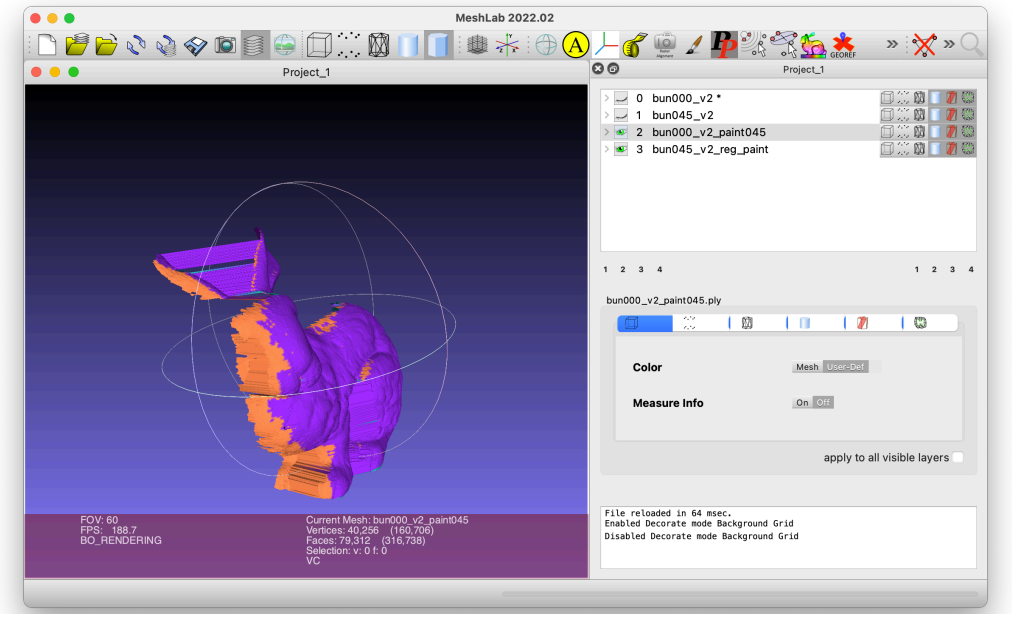
I implement ICP algorithm and use it to align `bunny_v2/bun045_v2.ply` to `bunny_v2/bun000_v2.ply` .

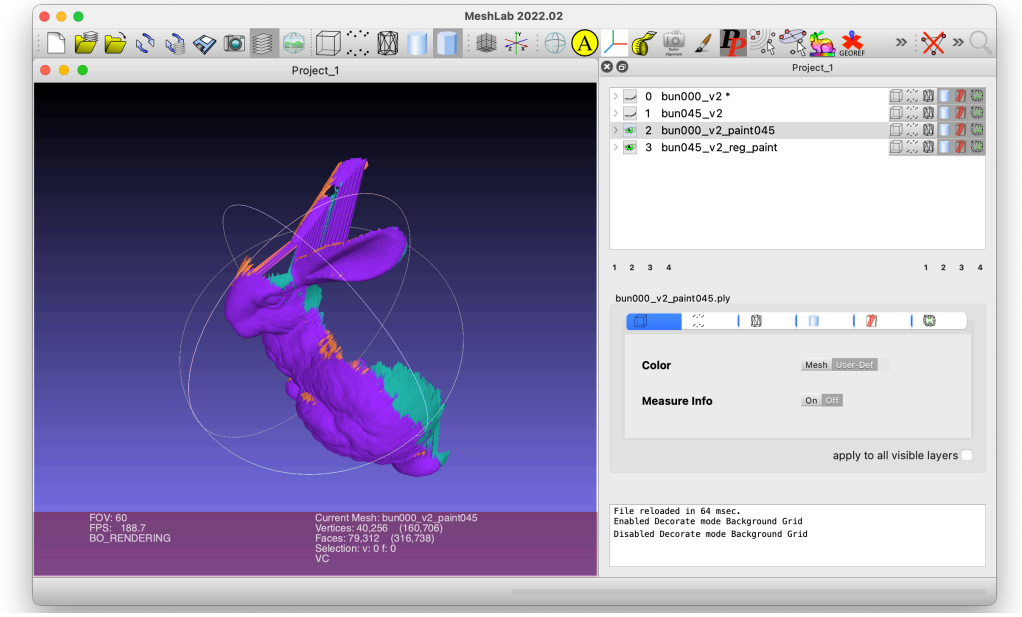
I also paint the meshes to mark out the overlapping and non-overlapping regions. The painted (and aligned) meshes are saved in `bunny_v2/bun000_v2_paint045.ply` (painted `bunny_v2/bun000_v2.ply`) and `bunny_v2/bun045_v2_reg_paint.ply` (painted and aligned `bunny_v2/bun045_v2.ply`). The colours used and their corresponding region is listed below.

Colour	Region
	Non-overlapping region in `bunny_v2/bun000_v2_paint045.ply`
	Overlapping region
	Non-overlapping region in `bunny_v2/bun045_v2_reg_paint.ply`

The result is shown below:







We can see that most of the regions are aligned. Most of the non-overlapping regions are because there is no corresponding vertices in another mesh. I also calculated the performance of my ICP by function in `open3d`, which is `open3d.pipelines.registration.evaluate_registration`. It shows that 91.5% of vertices in `bunny_v2/bun045_v2.ply` are got matched.

ICP algorithm with weights

After add the weights associated with points, the objective function becomes to

$$E(\mathbf{R},\mathbf{t}) = \sum_{i}^{n} w_i \|\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i\|^2 \tag{1}$$

.

We can use differential to minimize formula (1):

$$\begin{cases} \dfrac{\partial E}{\partial \mathbf{R}} = 0 \\ \dfrac{\partial E}{\partial \mathbf{t}} = 0 \end{cases} \tag{2}$$

.

First we solve $\frac{\partial E}{\partial \mathbf{t}} = 0$:

$$\frac{\partial E}{\partial \mathbf{t}} = \sum_{i}^{n} 2w_i(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) = 0 \tag{3}$$

, therefore,

$$\frac{\sum_i^n w_i \mathbf{R}\mathbf{p}_i}{n} + \frac{\sum_i^n w_i \mathbf{t}}{n} - \frac{\sum_i^n w_i \mathbf{q}_i}{n} = 0 \tag{4}$$

.

If we assume

$$\begin{cases} \overline{\mathbf{p}} = \dfrac{\sum_i^n w_i \mathbf{p}_i}{n} \\ \overline{\mathbf{q}} = \dfrac{\sum_i^n w_i \mathbf{q}_i}{n} \end{cases} \tag{5}$$

, which are in the form of weighted averages.

So that formula (4) can be written as

$$\mathbf{t} = \overline{\mathbf{q}} - \mathbf{R}\overline{\mathbf{p}} \tag{6}$$

.

Since $\overline{\mathbf{q}}$ and $\overline{\mathbf{p}}$ are constants, the remaining problems are the same as the derivation done in class. So that the solution is

$$\begin{cases} \mathbf{R} = \mathbf{V} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(\mathbf{V}\mathbf{U}^{\mathrm{T}}) \end{bmatrix} \mathbf{U}^{\mathrm{T}} \\ \mathbf{t} = \overline{\mathbf{q}} - \mathbf{R}\overline{\mathbf{p}} \end{cases} \tag{7}$$

, where

$$\begin{cases} \overline{\mathbf{p}} = \dfrac{\sum_i^n w_i \mathbf{p}_i}{n} \\ \overline{\mathbf{q}} = \dfrac{\sum_i^n w_i \mathbf{q}_i}{n} \end{cases} \tag{8}$$

, and

$$\mathrm{SVD}(\sum_i^n \widetilde{\mathbf{p}_i}\, \widetilde{\mathbf{q}_i}^{\mathrm{T}}) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} \tag{9}$$

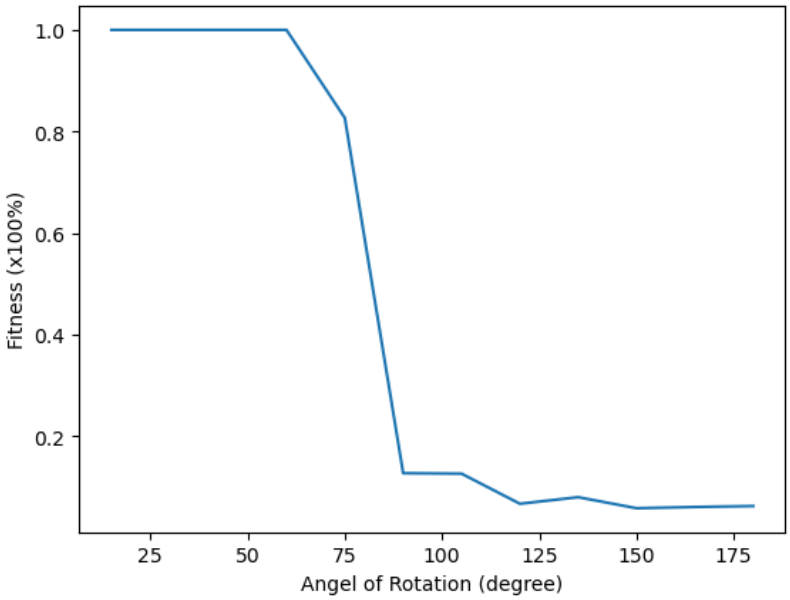
, where

$$\begin{cases} \widetilde{\mathbf{p}_i} = \mathbf{p}_i - \overline{\mathbf{p}} \\ \widetilde{\mathbf{q}_i} = \mathbf{q}_i - \overline{\mathbf{q}} \end{cases} \tag{10}$$

.

Part 2

I rotate the mesh `bunny_v2/bun000_v2.ply` with several angels and test my implementation of the ICP algorithm on them. The relationship between fitness and the angel is shown in the plot below.



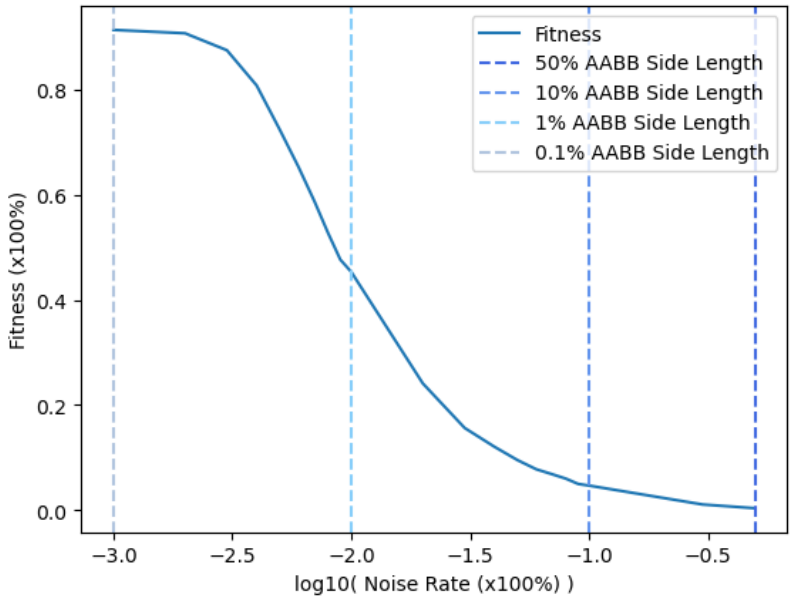
We can see that there is a huge drop in fitness when the angel of rotation is larger than 75°. It means if there is a big difference between the initial poses of two meshes, the ICP algorithm becomes very ineffective. It is because the method of using the nearest neighbour to find corresponding points is no longer reliable.

Part 3

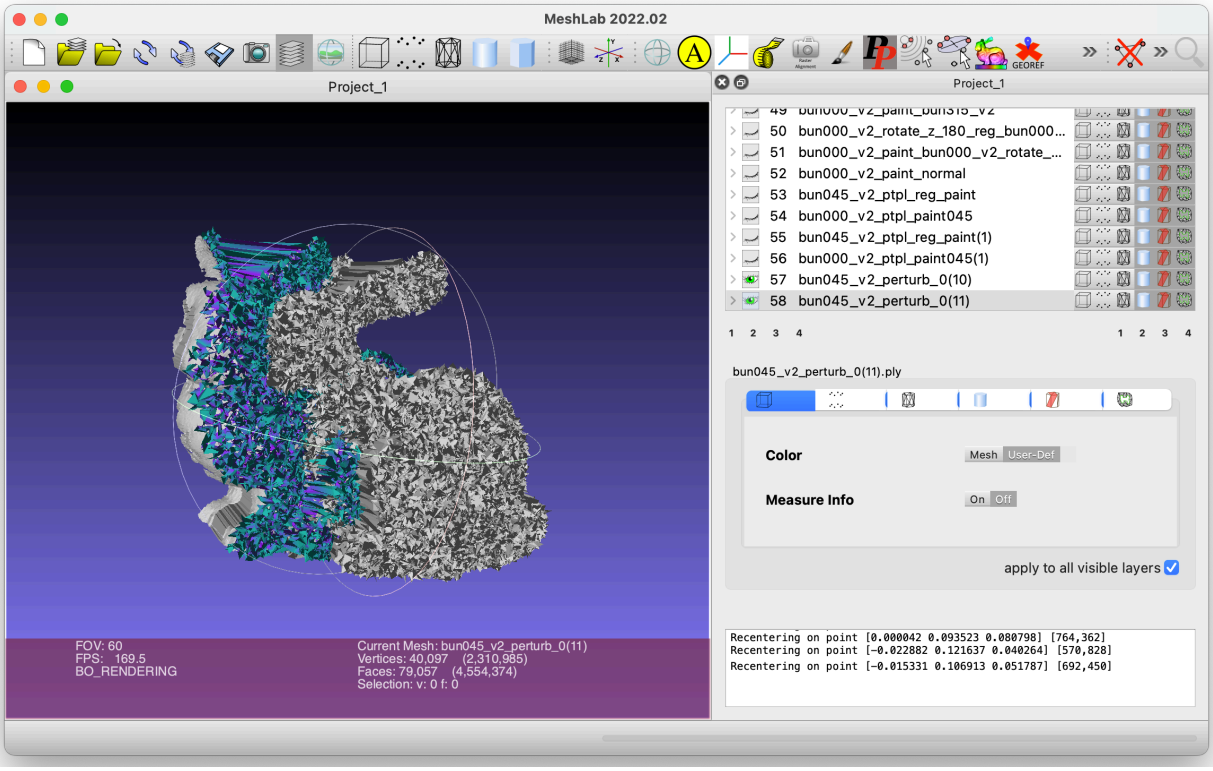
I use the side lengths of the axis-aligned bounding box as the standard to determine the variance of the noises. Assume the x-axis corresponding side length of the axis-aligned bounding box is lx, then set a Noise Rate (nr). Given a point ($p = (x, y, z)$) in a mesh, after adding the noise the x coordinate should be

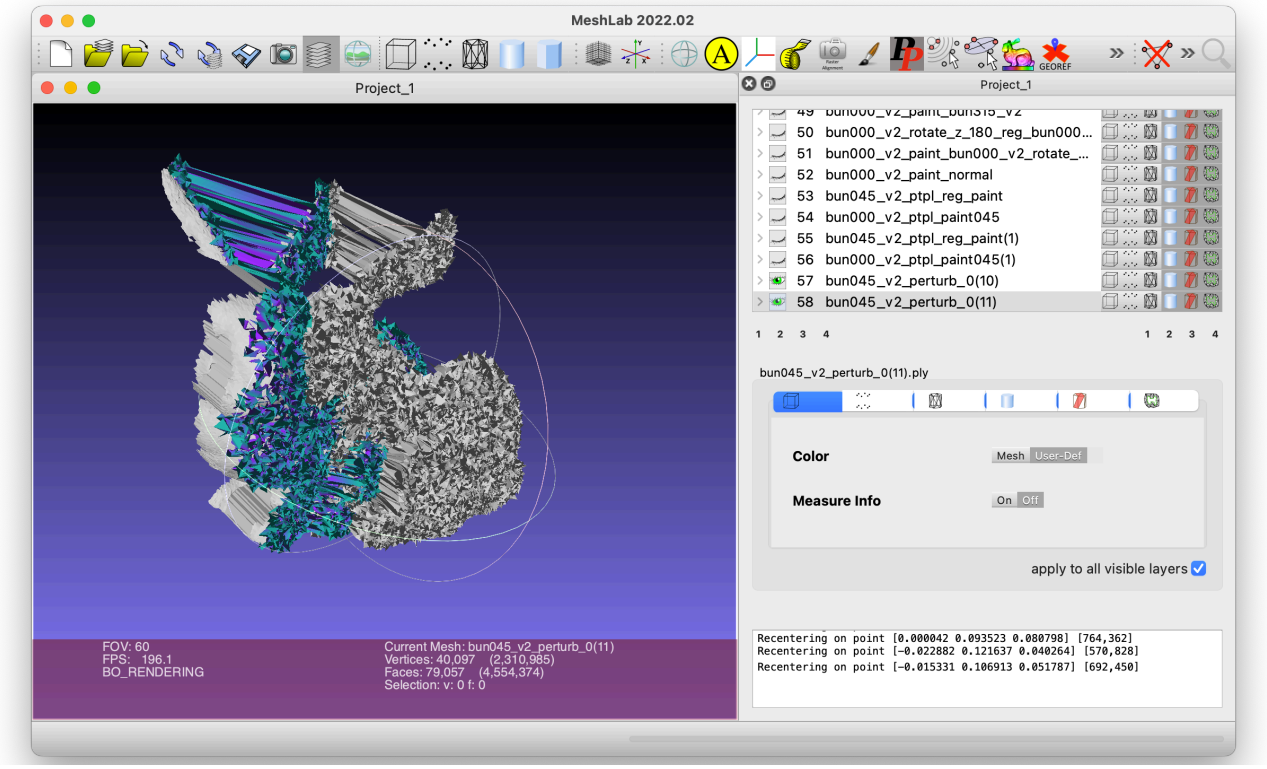
```
x + numpy.random.normal(0, lx*nr)
```

. The coordinates of the y-axis and z-axis are the same case as the x-axis. The relationship between fitness and the noise rate is shown in the plot below.

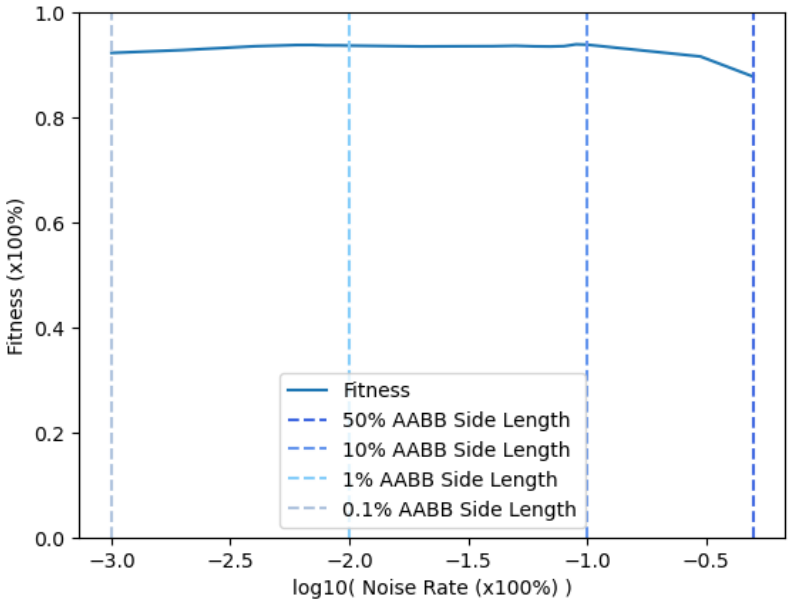


We can see when the noise rate increase to 1%, the fitness becomes less than 50%. However, it is because the noises are not taken into account when calculating fitness. Visualize the case of 1% in MeshLab:





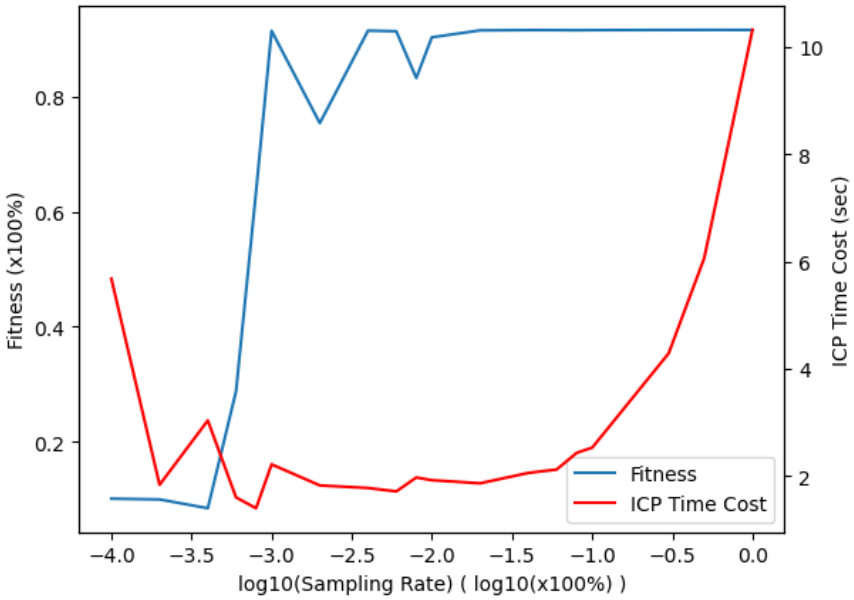
In the screenshot, the rightmost rabbit is the 'noisy rabbit ' before aligning. The coloured rabbit is the 'noisy rabbit ' after aligning. The leftmost is the target rabbit of which the most part is covered by the coloured 'noisy rabbit'. We can see after aligning by ICP, actually, the pose of the 'noisy rabbit' is aligned with the target rabbit. The reason for low fitness is that there is a threshold when judging whether a point has a corresponding point in the target point cloud, and the addition of noise could make the distance to the potential corresponding point larger than the threshold, thus reducing the fitness. So I add one noise length to the base threshold to observe how the noise will disturb the alignment of the pose of the ICP algorithm. The relationship between fitness considering noise and the noise rate is shown in the plot below.



We can see that the fitness has been maintained at a high level. However, the examples that have higher noise levels (such as a noise rate of 10%) are not so informative, since it is not so possible to meet these cases in the real life. The noise rate of 1% shown above is already quite noisy.

Part 4

I test the influence of the sampling rate by aligning `bunny_v2/bun045_v2.ply` to `bunny_v2/bun000_v2.ply` with different sampling rates. The relationship between fitness, the sampling rate and the time cost of ICP is shown in the plot below.



We can see that with the increase of sampling rate, the fitness increases but the ICP time cost increases too. We can also notice that when the sampling rate is low, the time cost is also high.

When the sampling rate is high, though the performance of aligning is pretty good, the number of points to be sampled may be very large, thus making progress in finding possible corresponding points by finding the nearest neighbour cost a large amount of time. When the sampling rate is very low, not only the fitness is low, but also the time cost is quite high. The reason is that too few sampling points are likely not to represent the whole model well, unless the sample points are specifically selected, such as certain feature points.

Therefore there is a trade-off between performance and time cost. We can see from the plot that, in this case, a sampling rate of 0.1%~1% is fairly enough, which keeps a balance of efficiency and performance.

Part 5

Given multiple scans and align all of them to a common global coordinate frame. I choose:

- `bunny_v2/bun000_v2.ply`,
- `bunny_v2/bun045_v2.ply`,
- `bunny_v2/bun270_v2.ply`,
- `bunny_v2/bun315_v2.ply`,
    - `bunny_v2/bun090_v2.ply` and `bunny_v2/bun180_v2.ply` cannot align to the others by rigid transformation
- and `bunny_v2/bun000_v2_rotate_z_180.ply`
    - is a mesh generated in **Part 2**, which is `bunny_v2/bun045_v2.ply` rotates 180°.

as the multiple scans, and `bunny_v2/bun000_v2.ply` is chosen as the **base scan**.

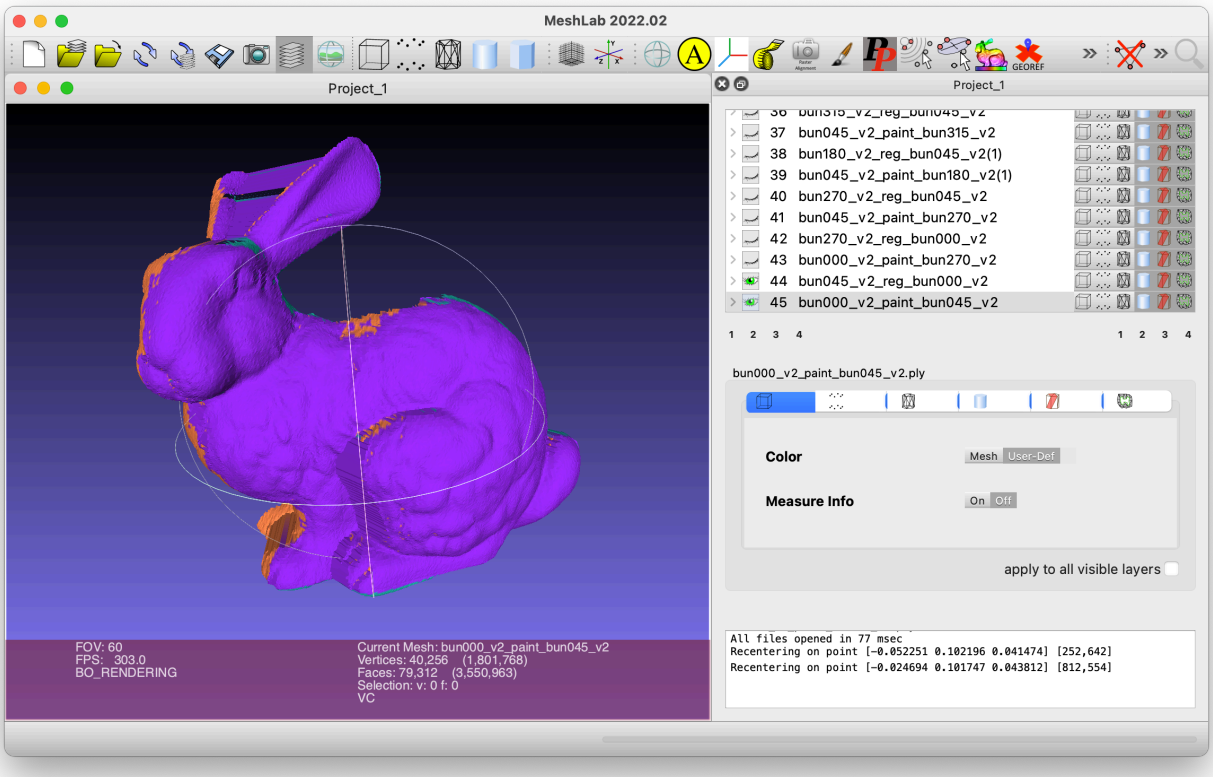
A simple method is choosing a scan as the base and aligning all the others to the **base scan**. But there is a problem, as discussed in **Part 2**, ICP algorithm will be ineffective if there is a big difference between the initial poses of two meshes. In the examples I choose, `bunny_v2/bun000_v2_rotate_z_180.ply` cannot be aligned to `bunny_v2/bun000_v2.ply` due to the difference in the initial pose by my ICP.

To solve the problem, we can make a rough alignment before using the ICP algorithm, which aims to get the initial pose of the source mesh as close as possible to the pose of the target. We can realize it with the help of **features**. Like the Harris Corner in Image Processing, we can find some specific features in the point clouds. One of the point cloud features is Point Feature Histograms (**PFH**), and it also has a more efficient version **FPFH**. We first extract feature points from the source and target point clouds, then we use these feature points to calculate a rough rigid transformation matrix between the source and target. It seems simple but finding the corresponding feature points is still a big problem. Fortunately, there already have some algorithms to do the rough alignment (or called Coarse/Global Registration) such as Sample Consensus Initial Alignment. In Python Library `open3d`, there is also a method to do the Global Registration by FPFH (feature points) and RANSAC (finding the corresponding feature points and calculating rigid transformation matrix).

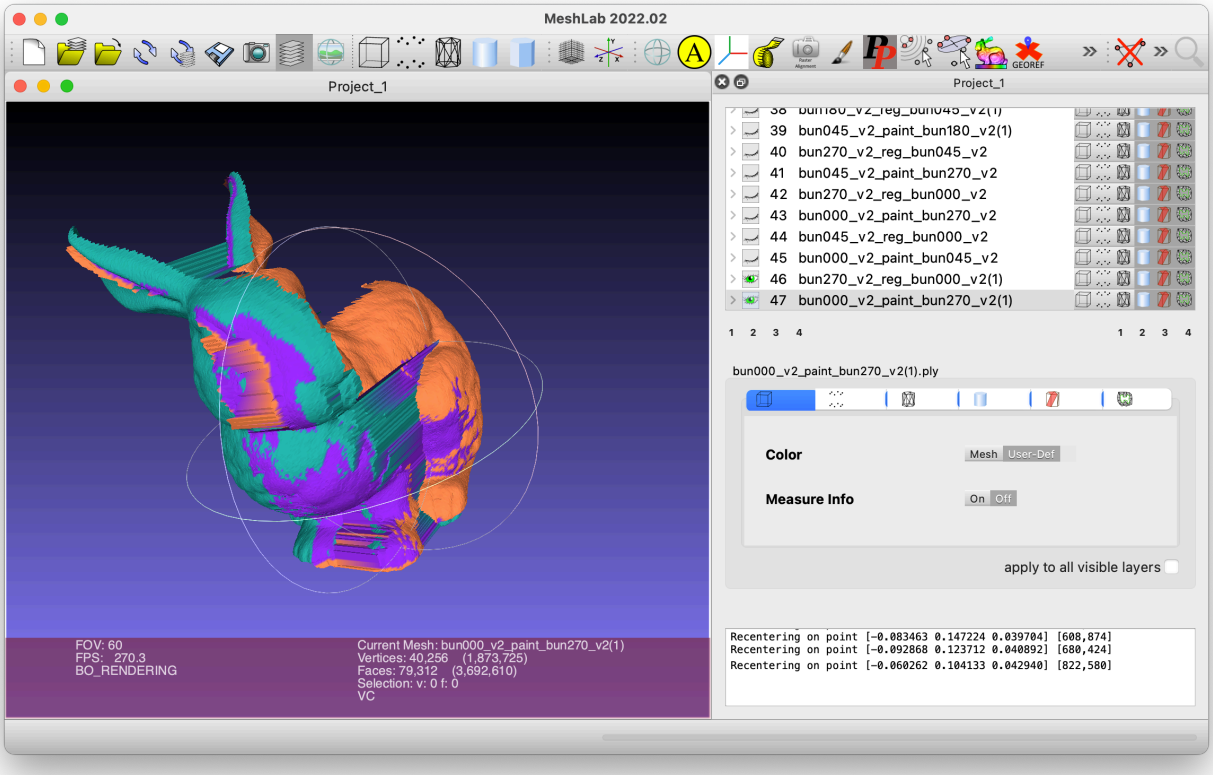
Therefore, I use `open3d` to roughly align the point clouds before using ICP. Then all the scans in this example are aligned to the **base scan** successfully. Below are the results visualized in the MeshLab.

Colour	Region
	Non-overlapping region in **base scan**
	Overlapping region
	Non-overlapping region in other scans

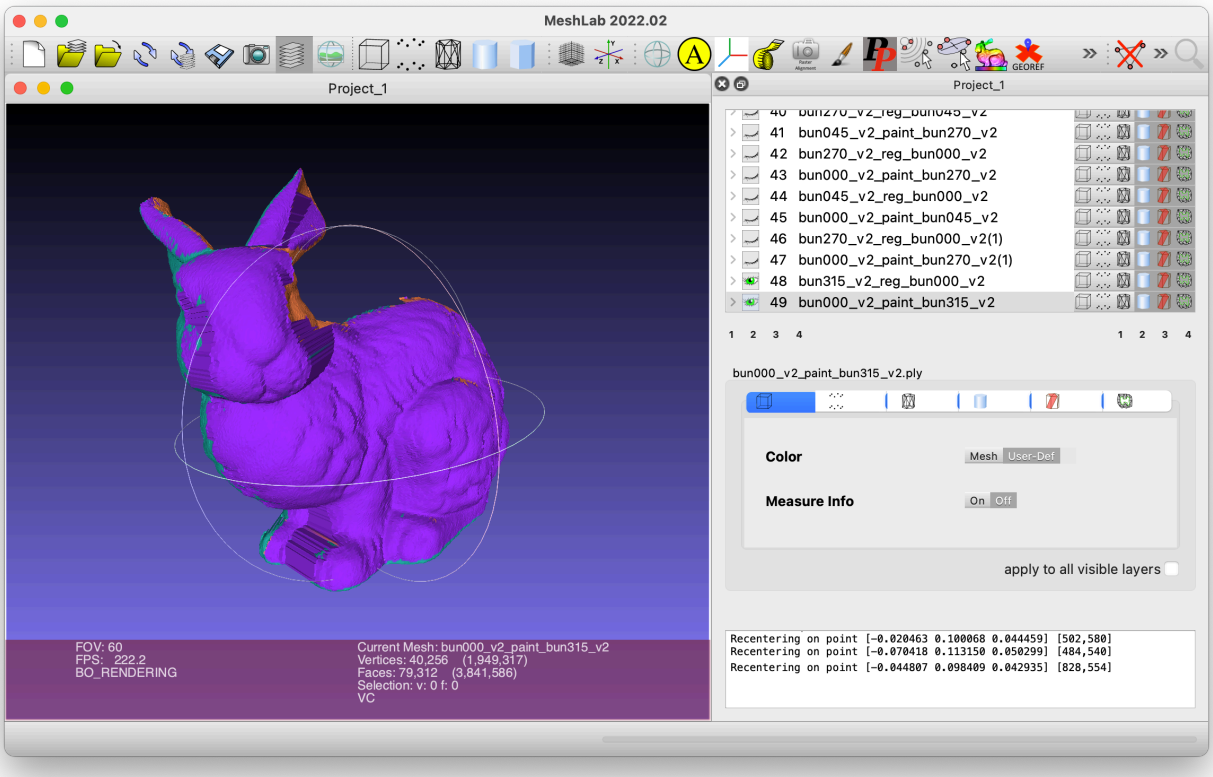
- `bunny_v2/bun045_v2.ply` to `bunny_v2/bun000_v2.ply`



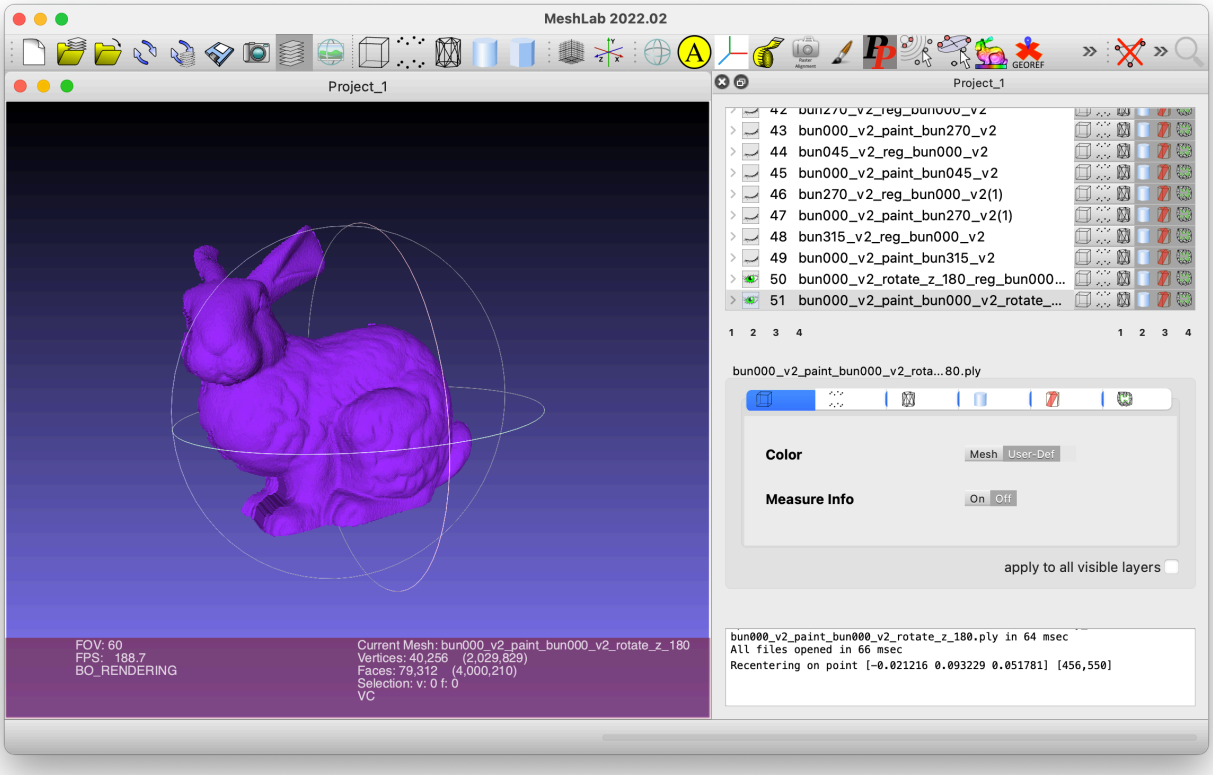
- `bunny_v2/bun270_v2.ply` to `bunny_v2/bun000_v2.ply`



- `bunny_v2/bun315_v2.ply` to `bunny_v2/bun000_v2.ply`



- `bunny_v2/bun000_v2_rotate_z_180.ply` to `bunny_v2/bun000_v2.ply`

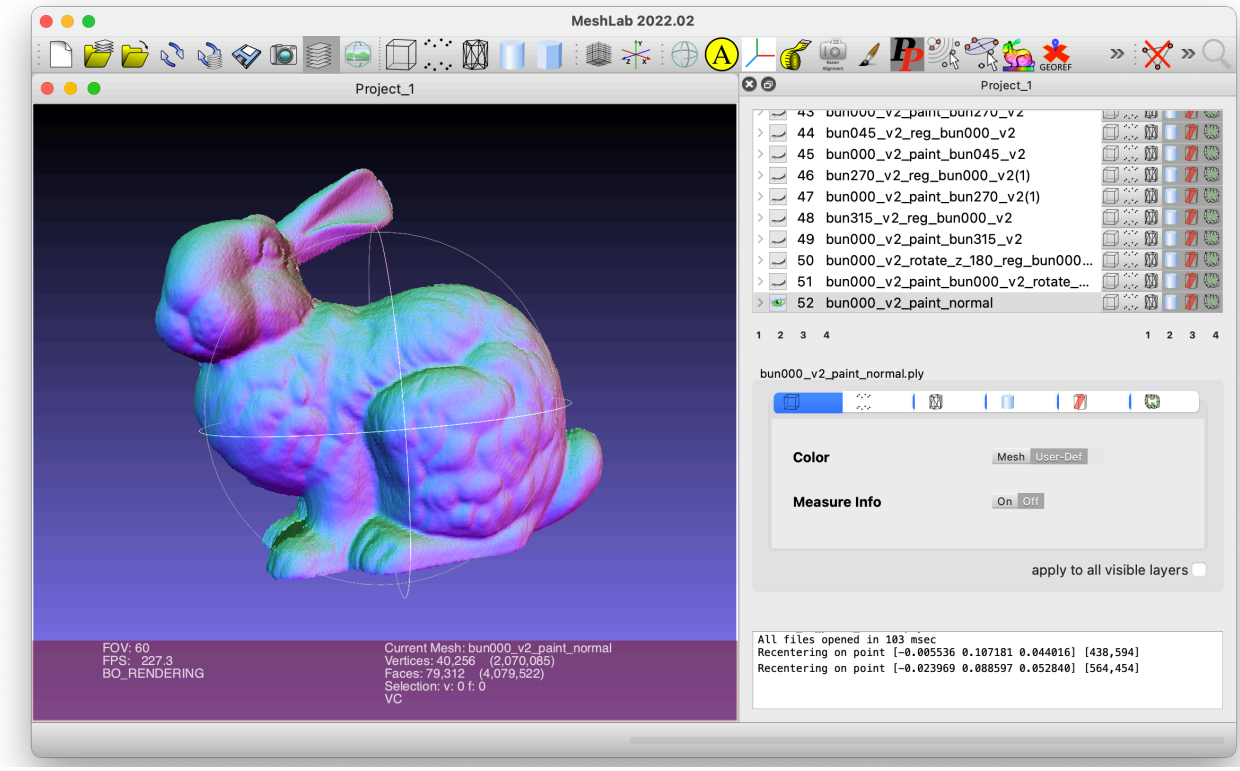


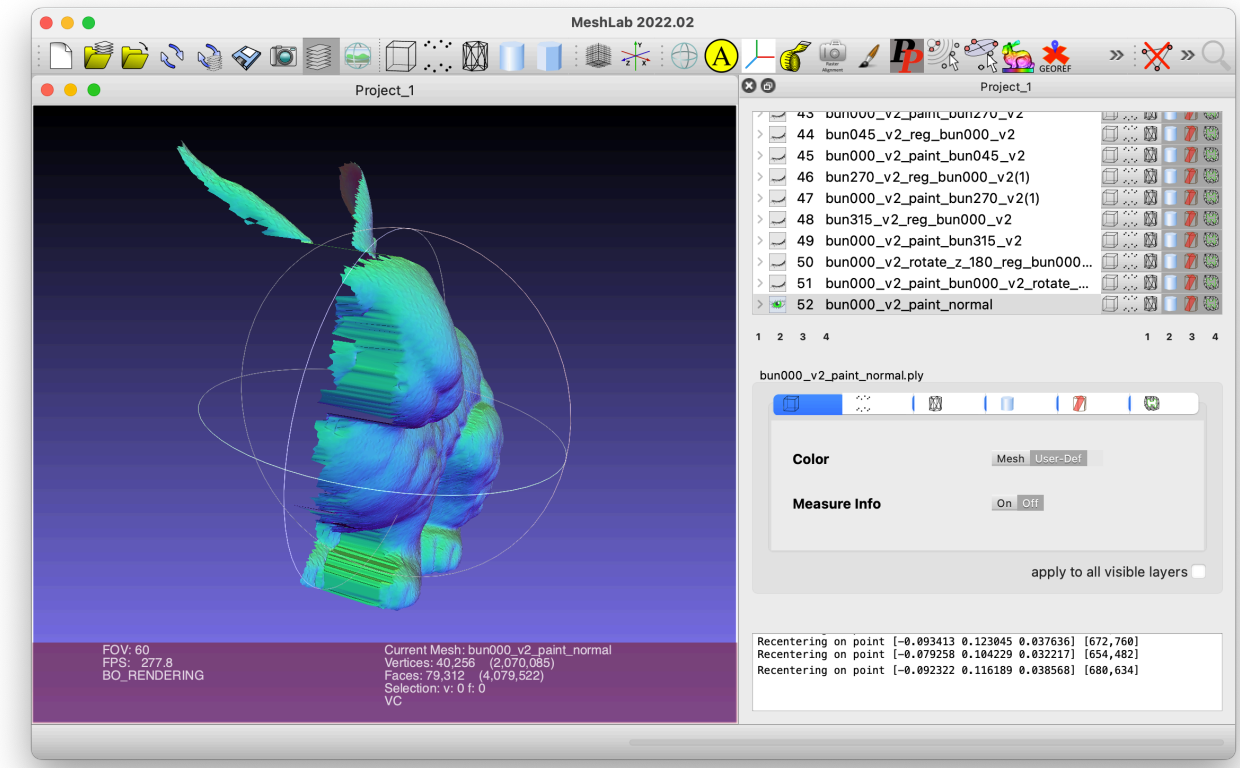
Until now, all the scans in this example can be aligned to the **base scan**.

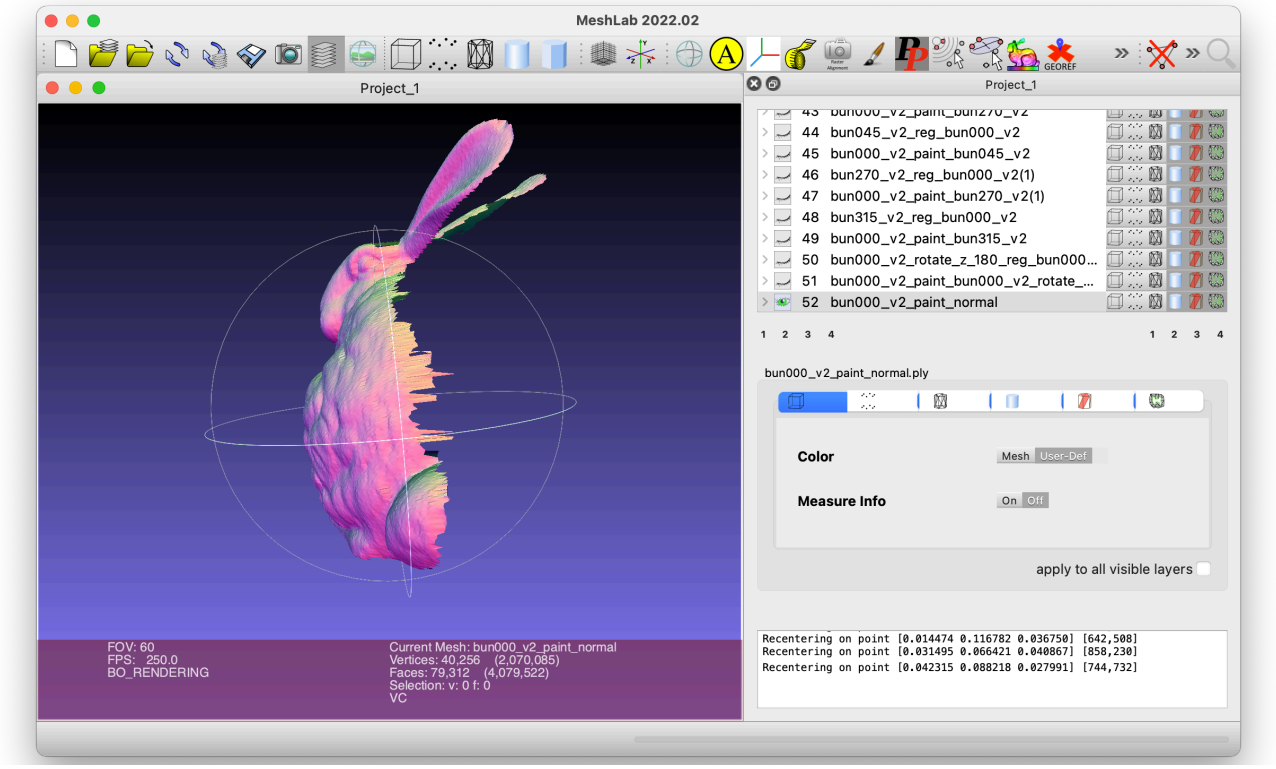
Part 6

Shade the models based on these normals

Paint `bunny_v2/bun000_v2.ply` by normals:







Derivation of Point-to-Plane ICP

Objective function:

$$E(\mathbf{R},\mathbf{t}) = \sum_{i}^{n} \|(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i)\cdot \mathbf{n}_i^q\|^2 \tag{11}$$

Step 1: Parameterising $\mathbf{R}$ and $\mathbf{t}$

Assume that the rotation matrix $\mathbf{R}$ rotates the target by α degree around the x-axis, β degree around the y-axis and γ degree around the z-axis, then $\mathbf{R}$ can be rewritten as:

$$\mathbf{R} = \mathbf{R}_\mathrm{x}(\alpha)\mathbf{R}_\mathrm{y}(\beta)\mathbf{R}_\mathrm{z}(\gamma) \tag{12}$$

, where

$$\begin{cases} \mathbf{R}_\mathrm{x}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \\ \mathbf{R}_\mathrm{y}(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \\ \mathbf{R}_\mathrm{z}(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{cases} \tag{13}$$

.

Assume α, β and γ are fairly small angels, i.e $\alpha, \beta, \gamma \to 0$. When $\theta \to 0$, there are $\cos(\theta) \to 1, \sin(\theta) \to \theta$ and $\theta^2 \to 0$. So that

$$\begin{cases} \mathbf{R}_{\mathrm{x}}(\alpha) \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\alpha \\ 0 & \alpha & 1 \end{bmatrix} \\ \mathbf{R}_{\mathrm{y}}(\beta) \approx \begin{bmatrix} 1 & 0 & \beta \\ 0 & 1 & 0 \\ -\beta & 0 & 1 \end{bmatrix} \\ \mathbf{R}_{\mathrm{z}}(\gamma) \approx \begin{bmatrix} 1 & -\gamma & 0 \\ \gamma & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{cases} \tag{14}$$

.

Substituting (14) into (12):

$$\mathbf{R} \approx \begin{bmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{bmatrix} \tag{15}$$

.

For translation $\mathbf{t}$, assume

$$\mathbf{t} = \begin{bmatrix} \mathrm{t}_{\mathrm{x}} \\ \mathrm{t}_{\mathrm{y}} \\ \mathrm{t}_{\mathrm{z}} \end{bmatrix} \tag{16}$$

Step 2: Converting to a least squares problem

First, we assume

$$\begin{cases} \mathbf{p}_i = \begin{bmatrix} \mathrm{p}_i^x & \mathrm{p}_i^y & \mathrm{p}_i^z \end{bmatrix}^{\mathrm{T}} \\ \mathbf{q}_i = \begin{bmatrix} \mathrm{q}_i^x & \mathrm{q}_i^y & \mathrm{q}_i^z \end{bmatrix}^{\mathrm{T}} \\ \mathbf{n}_i^q = \begin{bmatrix} \mathrm{n}_i^x & \mathrm{n}_i^y & \mathrm{n}_i^z \end{bmatrix}^{\mathrm{T}} \end{cases} \tag{17}$$

, then we substitute $(15), (16)$ and (17) into $(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) \cdot \mathbf{n}_i^q$:

$$\begin{aligned} & (\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) \cdot \mathbf{n}_i^q \\ & = (\begin{bmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{bmatrix} \begin{bmatrix} \mathrm{p}_i^x \\ \mathrm{p}_i^y \\ \mathrm{p}_i^z \end{bmatrix} + \begin{bmatrix} \mathrm{t}_{\mathrm{x}} \\ \mathrm{t}_{\mathrm{y}} \\ \mathrm{t}_{\mathrm{z}} \end{bmatrix} - \begin{bmatrix} \mathrm{q}_i^x \\ \mathrm{q}_i^y \\ \mathrm{q}_i^z \end{bmatrix}) \cdot \begin{bmatrix} \mathrm{n}_i^x \\ \mathrm{n}_i^y \\ \mathrm{n}_i^z \end{bmatrix} \\ & = (\mathrm{n}_i^z \mathrm{p}_i^y - \mathrm{n}_i^y \mathrm{p}_i^z)\alpha + (\mathrm{n}_i^x \mathrm{p}_i^z - \mathrm{n}_i^z \mathrm{p}_i^x)\beta + (\mathrm{n}_i^y \mathrm{p}_i^x - \mathrm{n}_i^x \mathrm{p}_i^y)\gamma + (\mathrm{n}_i^x)\mathrm{t}_{\mathrm{x}} + (\mathrm{n}_i^y)\mathrm{t}_{\mathrm{y}} + (\mathrm{n}_i^z)\mathrm{t}_{\mathrm{z}} \\ & \quad - (\mathrm{n}_i^x \mathrm{q}_i^x + \mathrm{n}_i^y \mathrm{q}_i^y + \mathrm{n}_i^z \mathrm{q}_i^z - \mathrm{n}_i^x \mathrm{p}_i^x - \mathrm{n}_i^y \mathrm{p}_i^y - \mathrm{n}_i^z \mathrm{p}_i^z) \end{aligned} \tag{18}$$

.

Assume

$$\begin{cases} \mathbf{a}_i = \begin{bmatrix} \mathrm{n}_i^z \mathrm{p}_i^y - \mathrm{n}_i^y \mathrm{p}_i^z & \mathrm{n}_i^x \mathrm{p}_i^z - \mathrm{n}_i^z \mathrm{p}_i^x & \mathrm{n}_i^y \mathrm{p}_i^x - \mathrm{n}_i^x \mathrm{p}_i^y & \mathrm{n}_i^x & \mathrm{n}_i^y & \mathrm{n}_i^z \end{bmatrix}^{\mathrm{T}} \\ \mathrm{b}_i = \mathrm{n}_i^x \mathrm{q}_i^x + \mathrm{n}_i^y \mathrm{q}_i^y + \mathrm{n}_i^z \mathrm{q}_i^z - \mathrm{n}_i^x \mathrm{p}_i^x - \mathrm{n}_i^y \mathrm{p}_i^y - \mathrm{n}_i^z \mathrm{p}_i^z \\ \mathrm{x} = \begin{bmatrix} \alpha & \beta & \gamma & \mathrm{t_x} & \mathrm{t_y} & \mathrm{t_z} \end{bmatrix}^{\mathrm{T}} \end{cases} \tag{19}$$

, so that

$$\begin{aligned} E(\mathbf{R}, \mathbf{t}) &= \sum_i^n \| (\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) \cdot \mathbf{n}_i^q \|^2 \\ &= \sum_i^n [(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) \cdot \mathbf{n}_i^q]^{\mathrm{T}} [(\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i) \cdot \mathbf{n}_i^q] \\ &= \| \mathbf{A}\mathrm{x} - \mathbf{b} \|^2 \end{aligned} \tag{20}$$

, where

$$\begin{cases} \mathbf{A} = \begin{bmatrix} \mathbf{a}_0 & \mathbf{a}_1 & \dots & \mathbf{a}_i & \dots & \mathbf{a}_{n-1} \end{bmatrix}^{\mathrm{T}} \\ \mathbf{b} = \begin{bmatrix} \mathrm{b}_0 & \mathrm{b}_1 & \dots & \mathrm{b}_i & \dots & \mathrm{b}_{n-1} \end{bmatrix}^{\mathrm{T}} \end{cases} \tag{21}$$

.

Until now, the origin problem is converted to a least squares problem $E(\mathbf{R}, \mathbf{t}) = \| \mathbf{A}\mathrm{x} - \mathbf{b} \|^2$.

Step 3: Solve the least squares problem

Use SVD to solve the least squares problem:

$$\mathrm{SVD}(\mathbf{A}) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} \tag{22}$$

$$\mathbf{A}^{\dagger} = \mathbf{V}\mathbf{\Sigma}^{\dagger}\mathbf{U}^{\mathrm{T}} \tag{23}$$

, where $\mathbf{\Sigma}^{\dagger}$ is the inverse or pseudoinverse of $\mathbf{\Sigma}$.

So that,

$$\mathrm{x} = \mathbf{A}^{\dagger}\mathbf{b} \tag{24}$$

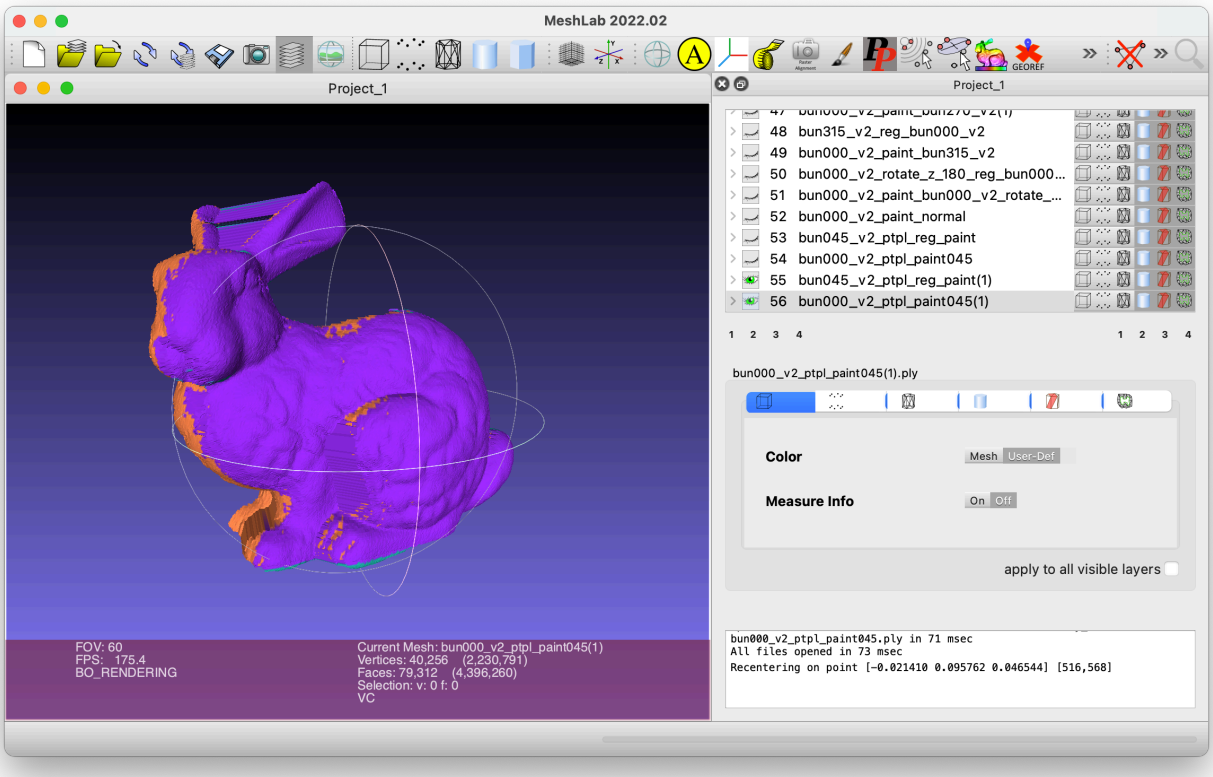
Step 4: Calculate $\mathbf{R}$ and $\mathbf{t}$ from x

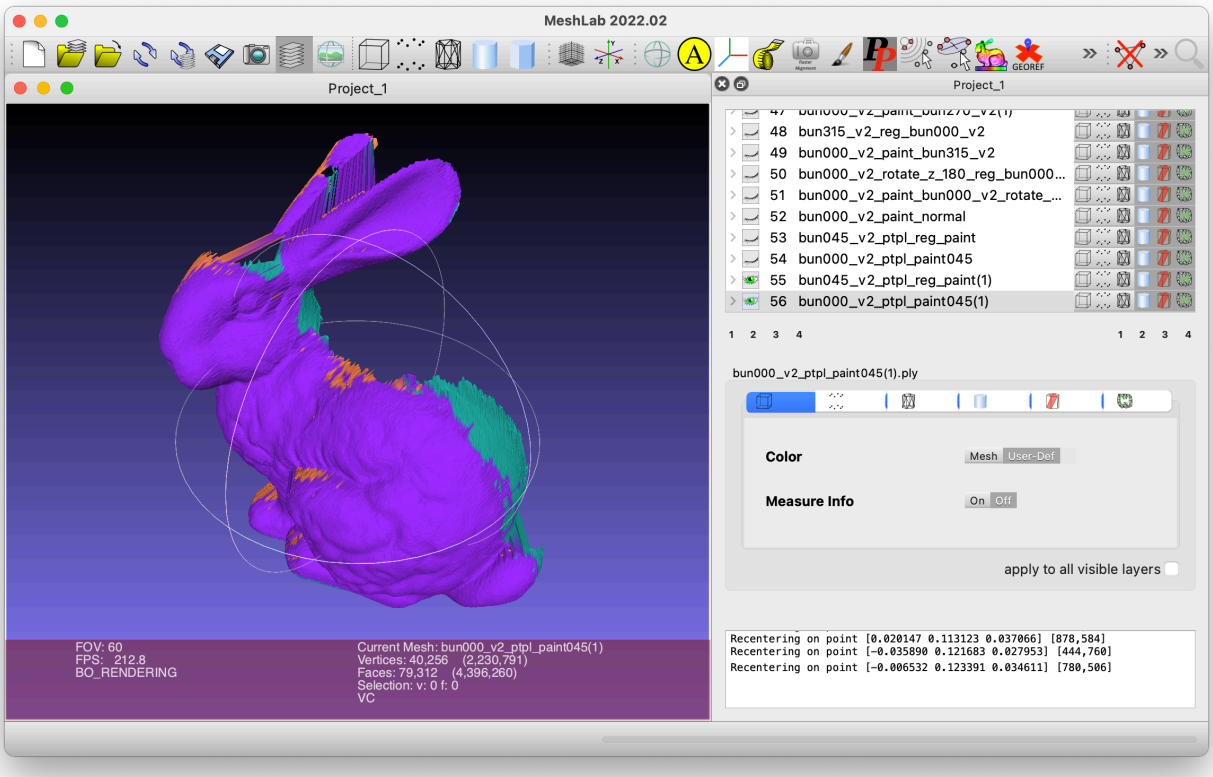
$\mathbf{t}$ can be extracted from x directly by (16).

After getting α, β and γ from x, we cannot calculate $\mathbf{R}$ by (15) which is only an approximation of $\mathbf{R}$. We should calculate $\mathbf{R}$ by (12) and (13).

Visualize the result of Point-to-Plane ICP

I align `bunny_v2/bun045_v2.ply` to `bunny_v2/bun000_v2.ply` by the new Point-to-Plane ICP, below is the screenshot of MeshLab.





The performance is as well as the point-to-point version, but the convergence speed is slower. The possible reason may be the method using an approximation, which assumes that in each iteration we only rotate the point cloud with a quite small angle. This method will be effective when the initial pose of the source is close to the target. Conversely, it may take longer to converge.