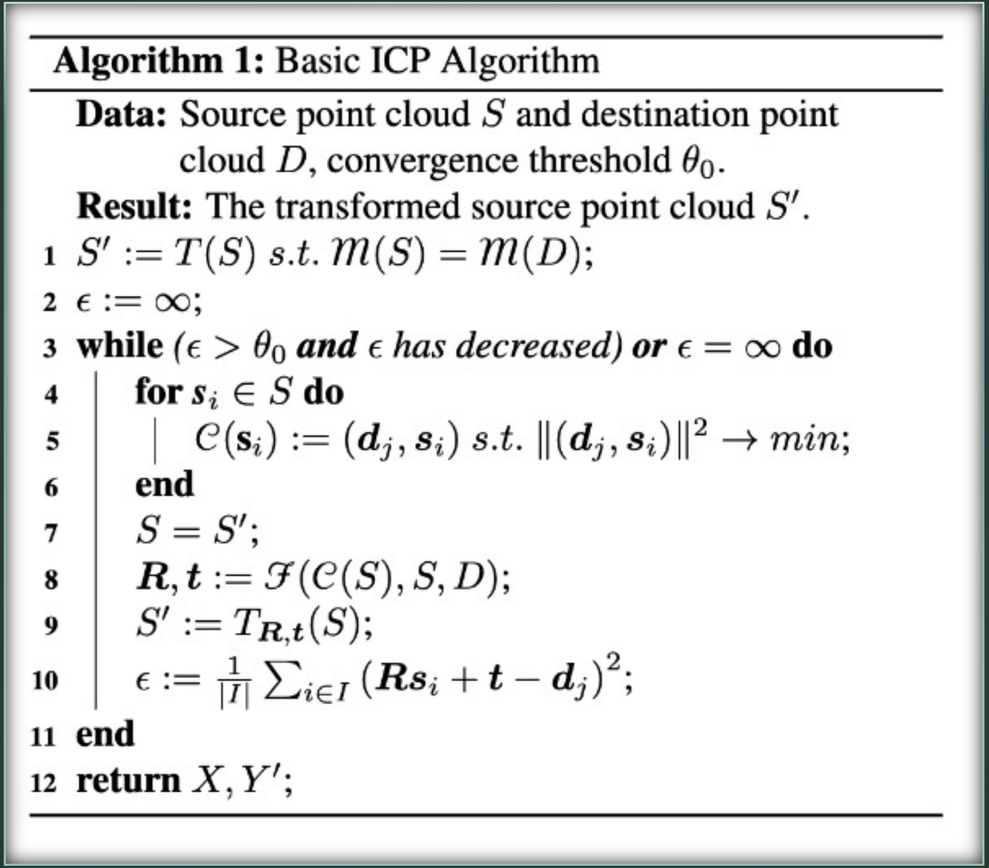
Jack BAI & Jiahe LIU

Project Presentation

Presentation Schedule

Algorithm & Exercise Overview

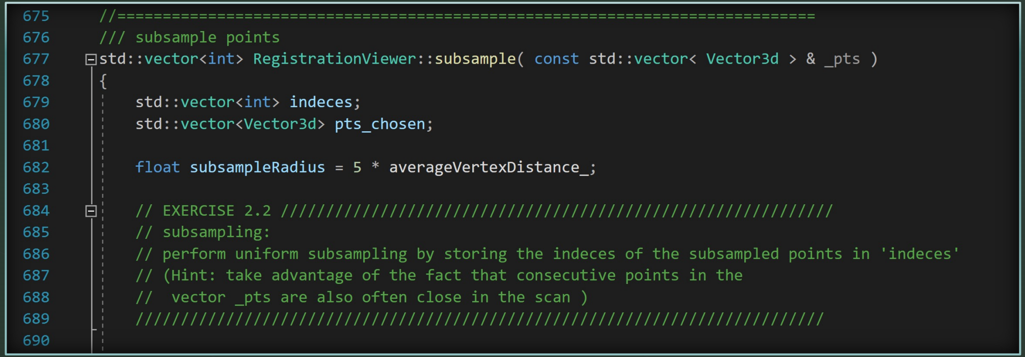
- **Input: point clouds X and Y**
- Initial guess (like making CM(x) == CM(y))
- **Subsampling (2)**
- Find correspondences (corresponding points) and **pruning (3)**
- Determine rigid transformation arguments (**registration (4, 5)**)
- Apply the rigid transformation and view results
- Go back to subsampling if still not reaching the convergence threshold.



```
Algorithm 1: Basic ICP Algorithm
  Data: Source point cloud $S$ and destination point
        cloud $D$, convergence threshold $\theta_0$.
  Result: The transformed source point cloud $S'$.
1  $S' := T(S)$ s.t. $\mathcal{M}(S) = \mathcal{M}(D)$;
2  $\epsilon := \infty$;
3  while ($\epsilon > \theta_0$ and $\epsilon$ has decreased) or $\epsilon = \infty$ do
4      for $\boldsymbol{s}_i \in S$ do
5          $\mathcal{C}(\mathbf{s}_i) := (\boldsymbol{d}_j, \boldsymbol{s}_i)$ s.t. $\|(\boldsymbol{d}_j, \boldsymbol{s}_i)\|^2 \to min$;
6      end
7      $S = S'$;
8      $\boldsymbol{R}, \boldsymbol{t} := \mathcal{F}(\mathcal{C}(S), S, D)$;
9      $S' := T_{\boldsymbol{R},\boldsymbol{t}}(S)$;
10     $\epsilon := \frac{1}{|I|} \sum_{i \in I} (\boldsymbol{R}\boldsymbol{s}_i + \boldsymbol{t} - \boldsymbol{d}_j)^2$;
11 end
12 return $X, Y'$;
```

Exercise 2.2 Subsampling

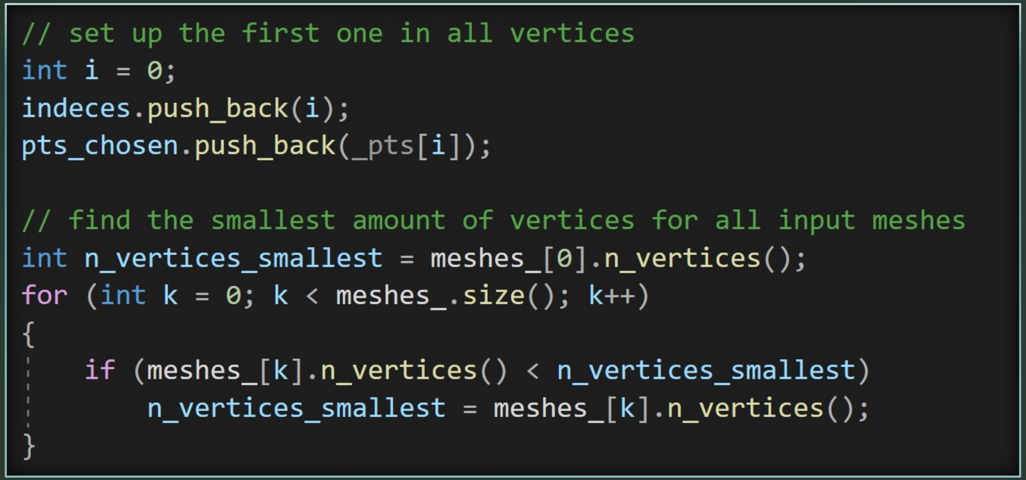
- Uniform subsampling
- *Ext: Normal subsampling*
- Given subsample radius
- *Opt: Consecutive pts*



```
//============================================================================
/// subsample points
std::vector<int> RegistrationViewer::subsample( const std::vector< Vector3d > & _pts )
{
    std::vector<int> indeces;
    std::vector<Vector3d> pts_chosen;

    float subsampleRadius = 5 * averageVertexDistance_;

    // EXERCISE 2.2 /////////////////////////////////////////////////////////////
    // subsampling:
    // perform uniform subsampling by storing the indeces of the subsampled points in 'indeces'
    // (Hint: take advantage of the fact that consecutive points in the
    //  vector _pts are also often close in the scan )
    ///////////////////////////////////////////////////////////////////////////

```

Exercise 2.2 Implementation

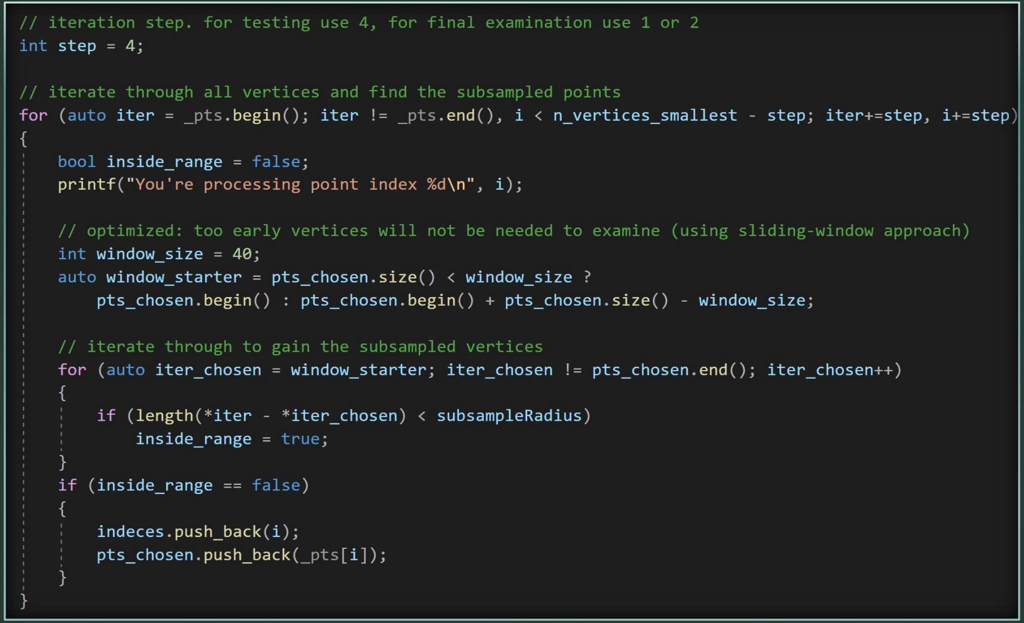
- Always add at least one point.
- Find least #vertice object.
- Ext: sort() algorithm in large number case.



```
// set up the first one in all vertices
int i = 0;
indeces.push_back(i);
pts_chosen.push_back(_pts[i]);

// find the smallest amount of vertices for all input meshes
int n_vertices_smallest = meshes_[0].n_vertices();
for (int k = 0; k < meshes_.size(); k++)
{
    if (meshes_[k].n_vertices() < n_vertices_smallest)
        n_vertices_smallest = meshes_[k].n_vertices();
}
```

Exercise 2.2 Implementation

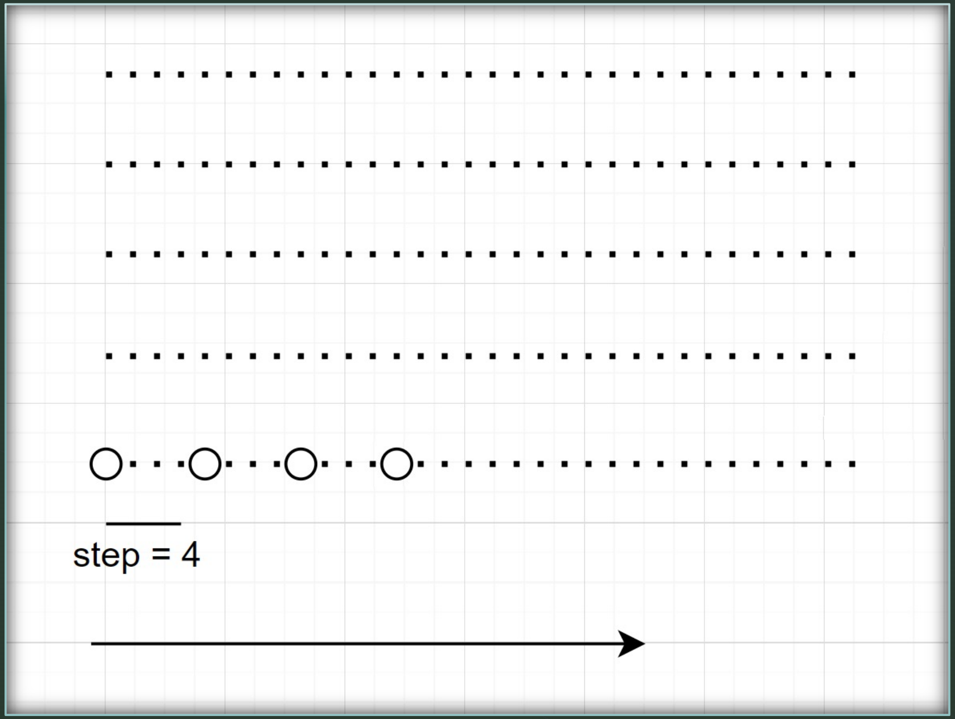
- Not every point need to be examined. A **step method** is utilized to accelerate the vertex iteration process.
- Utilize a **sliding window** method. Earlier subsampled vertices are omitted.
- Only when the vertex is outside all the subsampled vertices' subsample range, mark this vertex as blue.



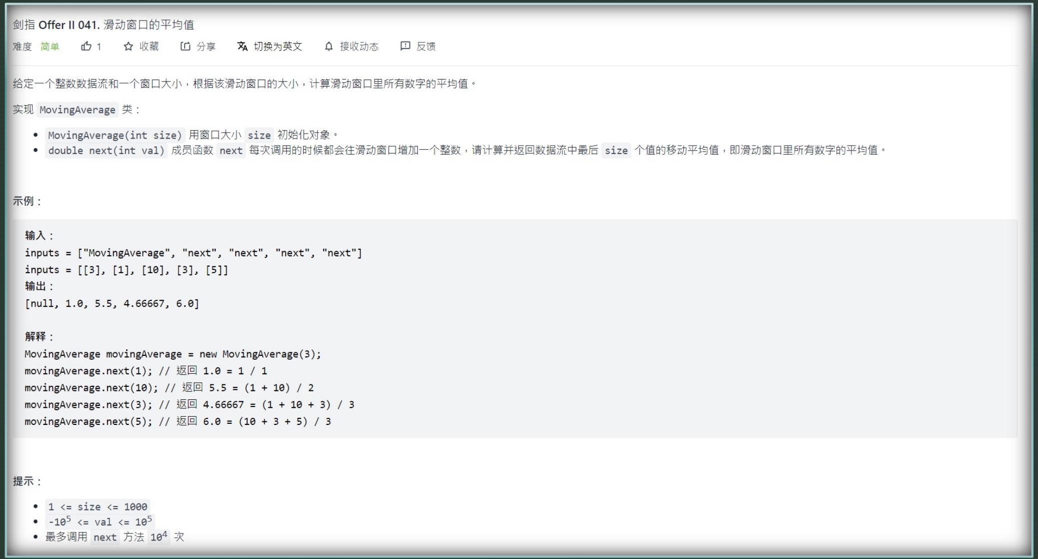
```cpp
// iteration step. for testing use 4, for final examination use 1 or 2
int step = 4;

// iterate through all vertices and find the subsampled points
for (auto iter = _pts.begin(); iter != _pts.end(), i < n_vertices_smallest - step; iter+=step, i+=step)
{
    bool inside_range = false;
    printf("You're processing point index %d\n", i);

    // optimized: too early vertices will not be needed to examine (using sliding-window approach)
    int window_size = 40;
    auto window_starter = pts_chosen.size() < window_size ?
        pts_chosen.begin() : pts_chosen.begin() + pts_chosen.size() - window_size;

    // iterate through to gain the subsampled vertices
    for (auto iter_chosen = window_starter; iter_chosen != pts_chosen.end(); iter_chosen++)
    {
        if (length(*iter - *iter_chosen) < subsampleRadius)
            inside_range = true;
    }
    if (inside_range == false)
    {
        indeces.push_back(i);
        pts_chosen.push_back(_pts[i]);
    }
}
```

Step Method

- Select 1 vertex for each 4 vertices.
- Result in a small loss but great efficiency amelioration.



Sliding Window Method (Learnt in *LeetCode*)



剑指 Offer II 041. 滑动窗口的平均值

难度 简单 1 收藏 分享 切换为英文 接收动态 反馈

给定一个整数数据流和一个窗口大小，根据该滑动窗口的大小，计算滑动窗口里所有数字的平均值。

实现 `MovingAverage` 类：

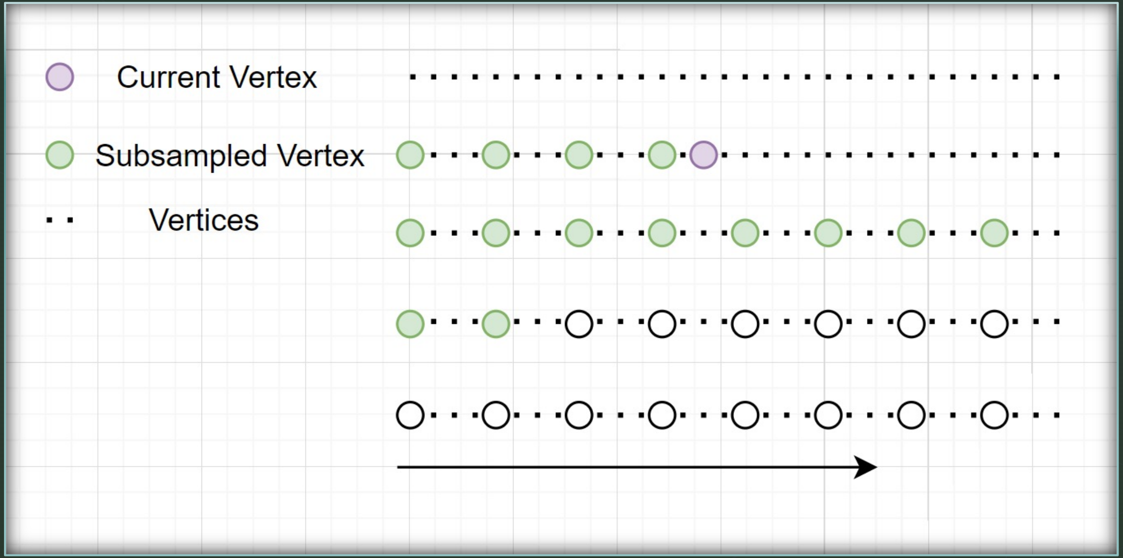
- `MovingAverage(int size)` 用窗口大小 `size` 初始化对象。
- `double next(int val)` 成员函数 `next` 每次调用的时候都会往滑动窗口增加一个整数，请计算并返回数据流中最后 `size` 个值的移动平均值，即滑动窗口里所有数字的平均值。

示例：

```
输入：
inputs = ["MovingAverage", "next", "next", "next", "next"]
inputs = [[3], [1], [10], [3], [5]]
输出：
[null, 1.0, 5.5, 4.66667, 6.0]

解释：
MovingAverage movingAverage = new MovingAverage(3);
movingAverage.next(1); // 返回 1.0 = 1 / 1
movingAverage.next(10); // 返回 5.5 = (1 + 10) / 2
movingAverage.next(3); // 返回 4.66667 = (1 + 10 + 3) / 3
movingAverage.next(5); // 返回 6.0 = (10 + 3 + 5) / 3
```

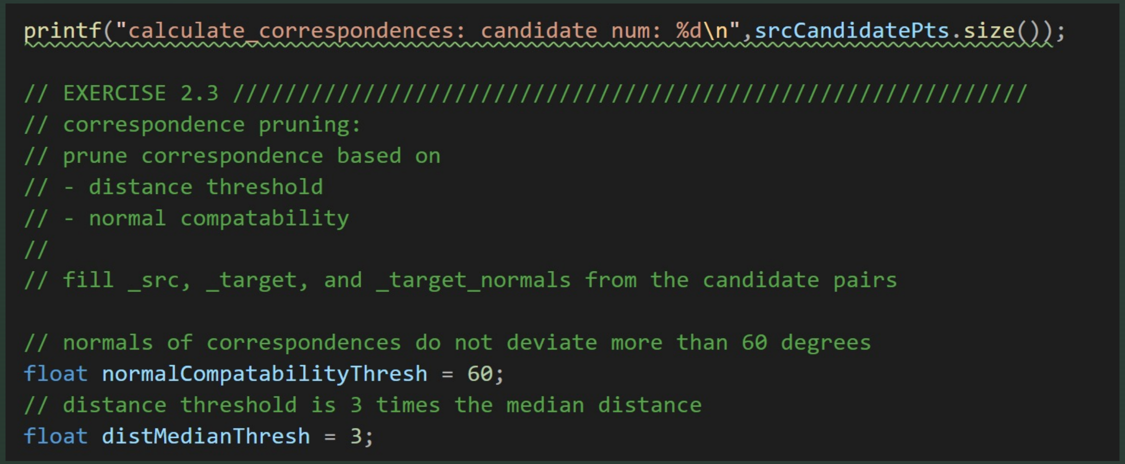
提示：

- `1 <= size <= 1000`
- -10^5 <= val <= 10^5
- 最多调用 `next` 方法 10^4 次

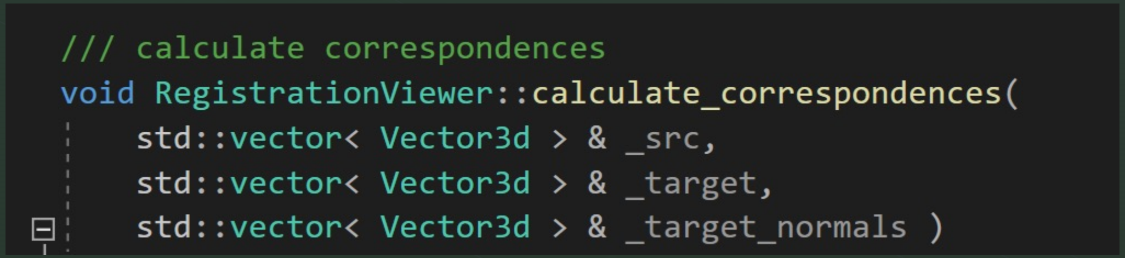


Exercise 2.3 Vertex Pruning

- Two pruning requirements.
- Input: source point set, target point set, normal vectors of target point set.
- Already calculated the correspondences.



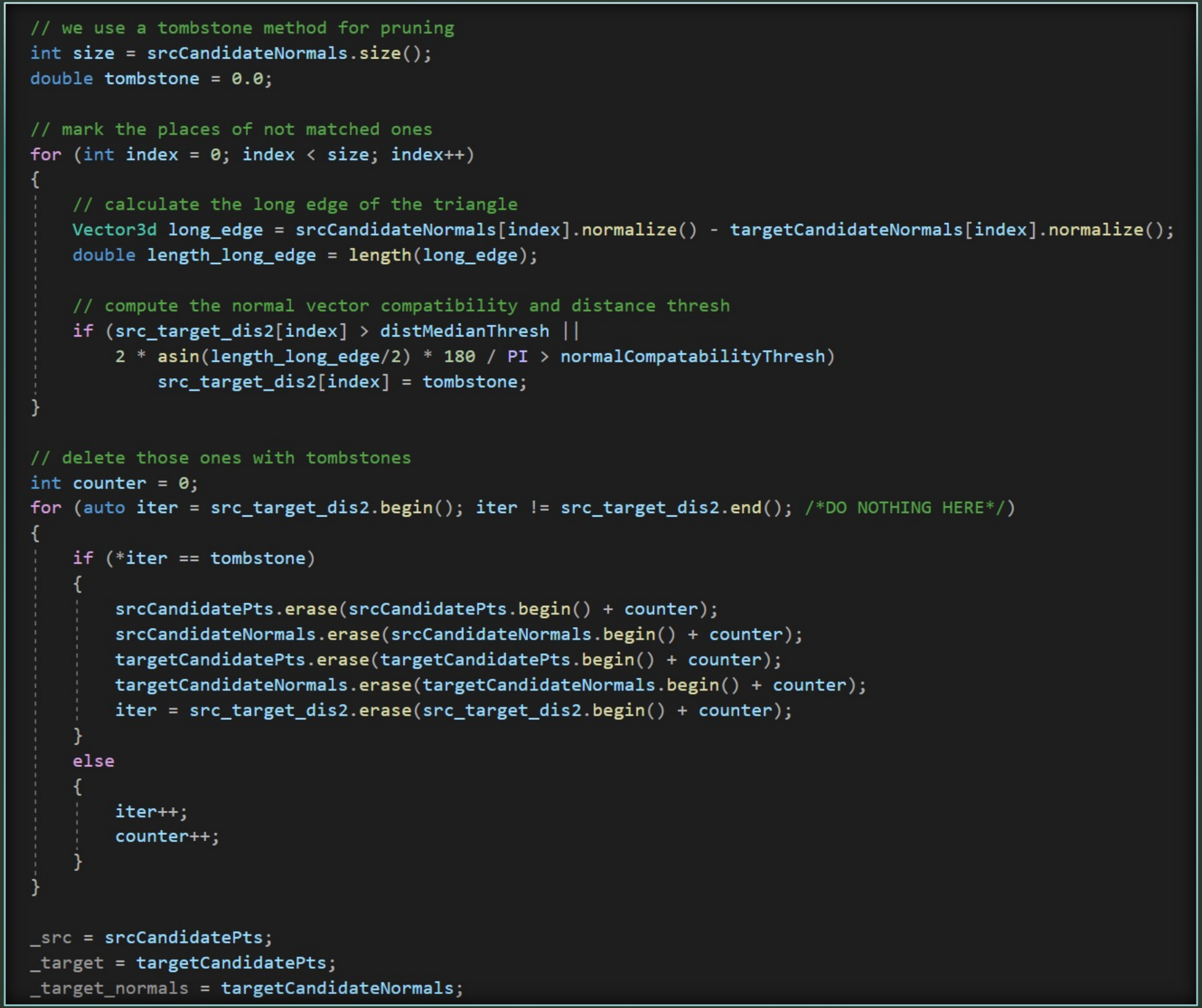
```
printf("calculate_correspondences: candidate num: %d\n",srcCandidatePts.size());

// EXERCISE 2.3 /////////////////////////////////////////////////////////////
// correspondence pruning:
// prune correspondence based on
// - distance threshold
// - normal compatability
//
// fill _src, _target, and _target_normals from the candidate pairs

// normals of correspondences do not deviate more than 60 degrees
float normalCompatabilityThresh = 60;
// distance threshold is 3 times the median distance
float distMedianThresh = 3;
```



```
/// calculate correspondences
void RegistrationViewer::calculate_correspondences(
    std::vector< Vector3d > & _src,
    std::vector< Vector3d > & _target,
    std::vector< Vector3d > & _target_normals )
```

Exercise 2.3 Implementation

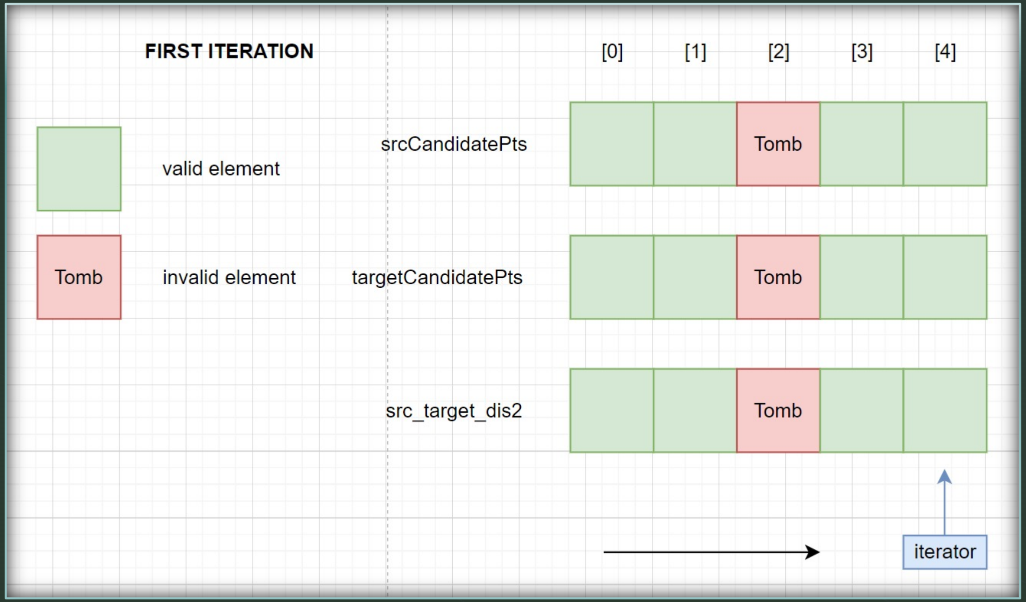
- Iterate through all point pairs.
- Use **tombstone method** for pruning.
- Distance Check.
- Normal Compatibility Thrash Check.



```cpp
// we use a tombstone method for pruning
int size = srcCandidateNormals.size();
double tombstone = 0.0;

// mark the places of not matched ones
for (int index = 0; index < size; index++)
{
    // calculate the long edge of the triangle
    Vector3d long_edge = srcCandidateNormals[index].normalize() - targetCandidateNormals[index].normalize();
    double length_long_edge = length(long_edge);

    // compute the normal vector compatibility and distance thresh
    if (src_target_dis2[index] > distMedianThresh ||
        2 * asin(length_long_edge/2) * 180 / PI > normalCompatabilityThresh)
            src_target_dis2[index] = tombstone;
}

// delete those ones with tombstones
int counter = 0;
for (auto iter = src_target_dis2.begin(); iter != src_target_dis2.end(); /*DO NOTHING HERE*/)
{
    if (*iter == tombstone)
    {
        srcCandidatePts.erase(srcCandidatePts.begin() + counter);
        srcCandidateNormals.erase(srcCandidateNormals.begin() + counter);
        targetCandidatePts.erase(targetCandidatePts.begin() + counter);
        targetCandidateNormals.erase(targetCandidateNormals.begin() + counter);
        iter = src_target_dis2.erase(src_target_dis2.begin() + counter);
    }
    else
    {
        iter++;
        counter++;
    }
}

_src = srcCandidatePts;
_target = targetCandidatePts;
_target_normals = targetCandidateNormals;
```

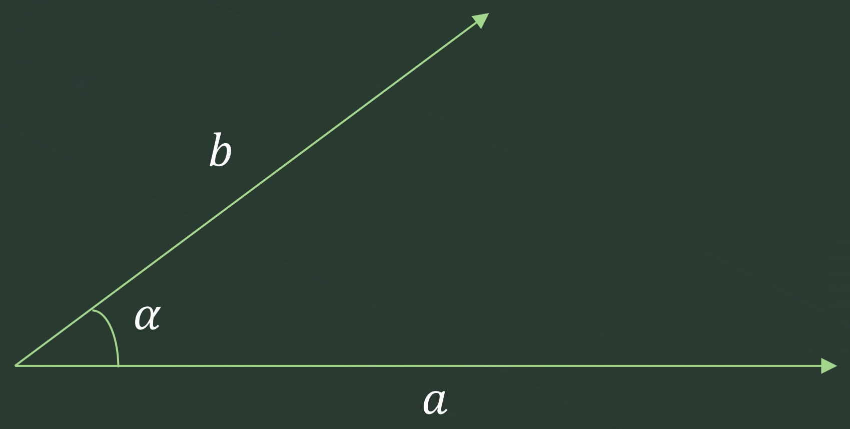
Tombstone Pruning Method

- Facing a question: how to delete elements with the same index in different arrays?
- TOMBSTONE method mentioned in CS225 (data structures).
- Two iterations. First iteration marks it, second iteration delete the marked elements.
- Also useful in relational database structure implementation.



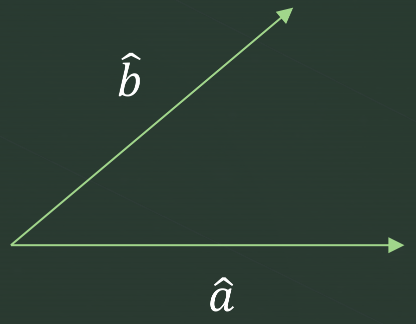
Ext: use another array for strong the elements that fulfill the requirements

Normal Compatibility Thrash Check



Normal Compatibility Thrash Check

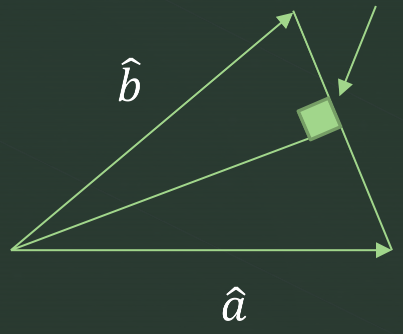
Normalization



$$|\hat{a}| = |\hat{b}| = 1$$

Normal Compatibility Thrash Check

Make into Isosceles triangle

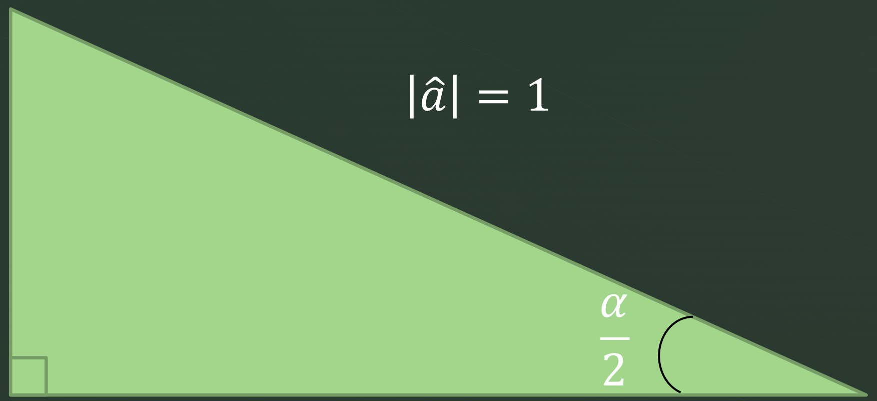


$$|\hat{a}| = |\hat{b}| = 1$$

Normal Compatibility Thrash Check

Make into right triangle

$$\cos\left(\frac{\alpha}{2}\right) = \frac{m}{|\hat{a}|}$$



$$\frac{|\boldsymbol{v}|}{2} = 1 \cdot \sin\left(\frac{\alpha}{2}\right)$$

$$\boldsymbol{v} = \hat{a} - \hat{b}$$

$$m = 1 \cdot \cos\left(\frac{\alpha}{2}\right)$$

Normal Compatibility Thrash Check

$$\frac{|\boldsymbol{v}|}{2} = 1 \cdot \sin\left(\frac{\alpha}{2}\right) \qquad \boldsymbol{v} = \hat{a} - \hat{b}$$

$$\sin\left(\frac{\alpha}{2}\right) = \frac{|\hat{a} - \hat{b}|}{2}$$

$$\alpha = 2 \cdot \arcsin\left(\frac{|\hat{a} - \hat{b}|}{2}\right) > 60° = 60 \cdot \frac{\pi}{180}$$

Exercise 2.4 P2P Registration

s denotes source point, while d denotes destination point.

$$d(3\times 1) = t(3\times 1) + R(3\times 3) \cdot s(3\times 1)$$

$$\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} + \begin{bmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x \\ s_y \\ s_z \end{bmatrix}$$

$$\Big\downarrow$$

$$d_x = t_x + s_x - \gamma s_y + \beta s_z$$
$$d_y = t_y + \gamma s_x + s_y - \alpha s_z$$
$$d_z = t_z - \beta s_x + \alpha s_y + s_z$$

Exercise 2.4 P2P Registration

s denotes source point, while d denotes destination point.

$$\begin{aligned} d_x &= t_x + s_x - \gamma s_y + \beta s_z \\ d_y &= t_y + \gamma s_x + s_y - \alpha s_z \\ d_z &= t_z - \beta s_x + \alpha s_y + s_z \end{aligned} \longrightarrow \begin{aligned} t_x + \beta s_z - \gamma s_y &= d_x - s_x \\ t_y - \alpha s_z + \gamma s_x &= d_y - s_y \\ t_z + \alpha s_y - \beta s_x &= d_z - s_z \end{aligned}$$

$$A(3\times6) \quad \cdot \quad x(6\times1) \quad = \quad b(3\times1)$$

$$\begin{bmatrix} 0 & s_z & -s_y & 1 & 0 & 0 \\ -s_z & 0 & s_x & 0 & 1 & 0 \\ s_y & -s_x & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha & \beta & \gamma & t_x & t_y & t_z \end{bmatrix}^T = \begin{bmatrix} d_x - s_x \\ d_y - s_y \\ d_z - s_z \end{bmatrix}$$

Exercise 2.5 P2S Registration

s denotes source point, d denotes destination point, and n denotes the normal vector of the target plane.

$$d(3\times 1) \quad = \quad t(3\times 1) + \qquad R(3\times 3) \qquad \cdot\, s(3\times 1)$$

$$\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} + \begin{bmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x \\ s_y \\ s_z \end{bmatrix}$$

$$\big\downarrow$$

$$d_x = t_x + s_x - \gamma s_y + \beta s_z$$
$$d_y = t_y + \gamma s_x + s_y - \alpha s_z$$
$$d_z = t_z - \beta s_x + \alpha s_y + s_z$$

Exercise 2.5 P2S Registration

s denotes source point, d denotes destination point, and n denotes the normal vector of the target plane.

$$d_x = t_x + s_x - \gamma s_y + \beta s_z$$
$$d_y = t_y + \gamma s_x + s_y - \alpha s_z$$
$$d_z = t_z - \beta s_x + \alpha s_y + s_z$$

$$?\ \Big\downarrow$$ This approach is blocked.

$$(\alpha n_z s_y + \beta n_x s_z + \gamma n_y s_x) + (n_x s_x + n_y s_y + n_z s_z) + (n_x t_x + n_y t_y + n_z t_z)$$
$$= \left(\alpha n_y s_z + \beta n_z s_x + \gamma n_x s_y\right) + (n_x d_x + n_y d_y + n_z d_z)$$

Exercise 2.5 P2S Registration

s denotes source point, d denotes destination point, and n denotes the normal vector of the target plane.

$$(\alpha n_z s_y + \beta n_x s_z + \gamma n_y s_x) + (n_x s_x + n_y s_y + n_z s_z) + (n_x t_x + n_y t_y + n_z t_z) = (\alpha n_y s_z + \beta n_z s_x + \gamma n_x s_y) + (n_x d_x + n_y d_y + n_z d_z)$$

$$\downarrow$$

$$(\alpha n_z s_y - \alpha n_y s_z) + (\beta n_x s_z - \beta n_z s_x) + (\gamma n_y s_x - \gamma n_x s_y) + n_x t_x + n_y t_y + n_z t_z = n_x d_x + n_y d_y + n_z d_z - (n_x s_x + n_y s_y + n_z s_z)$$

Exercise 2.5 P2S Registration

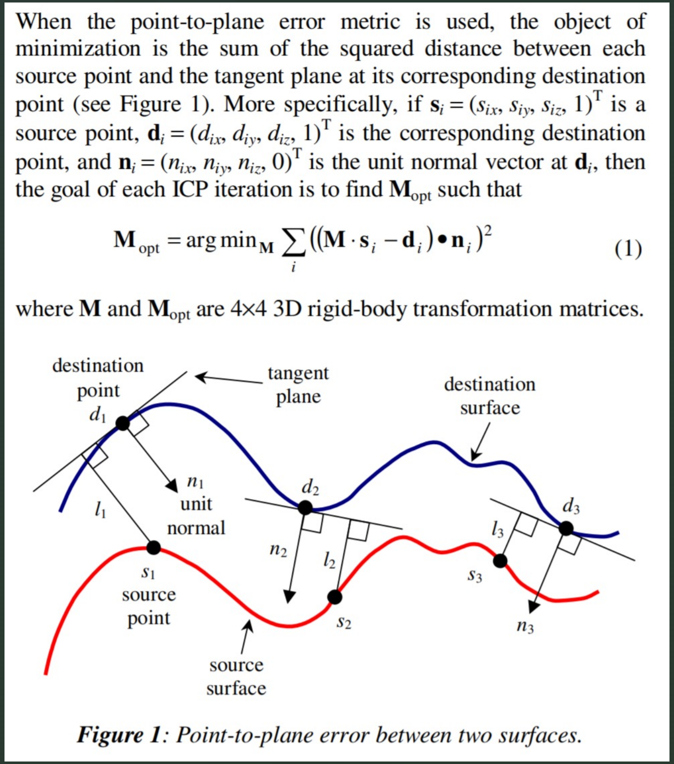
s denotes source point, d denotes destination point, and n denotes the normal vector of the target plane.

$$\left(\alpha n_z s_y - \alpha n_y s_z\right) + (\beta n_x s_z - \beta n_z s_x) + \left(\gamma n_y s_x - \gamma n_x s_y\right) + n_x t_x + n_y t_y + n_z t_z$$
$$= n_x d_x + n_y d_y + n_z d_z - (n_x s_x + n_y s_y + n_z s_z)$$

$$\downarrow$$

$$A(1\times 6) \quad \cdot \quad x(6\times 1) \quad = \quad b(1\times 1)$$
$$\begin{bmatrix} a_1 & a_2 & a_3 & n_x & n_y & n_z \end{bmatrix} \cdot \begin{bmatrix} \alpha & \beta & \gamma & t_x & t_y & t_z \end{bmatrix}^T = \begin{bmatrix} n_x d_x + n_y d_y + n_z d_z - (n_x s_x + n_y s_y + n_z s_z) \end{bmatrix}$$
$$a_1 = n_z s_y - n_y s_z$$
$$a_2 = n_x s_z - n_z s_x$$
$$a_3 = n_y s_x - n_x s_y$$

Exercise 2.5 P2S Registration



When the point-to-plane error metric is used, the object of minimization is the sum of the squared distance between each source point and the tangent plane at its corresponding destination point (see Figure 1). More specifically, if $\mathbf{s}_i = (s_{ix}, s_{iy}, s_{iz}, 1)^T$ is a source point, $\mathbf{d}_i = (d_{ix}, d_{iy}, d_{iz}, 1)^T$ is the corresponding destination point, and $\mathbf{n}_i = (n_{ix}, n_{iy}, n_{iz}, 0)^T$ is the unit normal vector at $\mathbf{d}_i$, then the goal of each ICP iteration is to find $\mathbf{M}_{\text{opt}}$ such that

$$\mathbf{M}_{\text{opt}} = \arg\min_{\mathbf{M}} \sum_i \left((\mathbf{M} \cdot \mathbf{s}_i - \mathbf{d}_i) \bullet \mathbf{n}_i \right)^2 \qquad (1)$$

where $\mathbf{M}$ and $\mathbf{M}_{\text{opt}}$ are 4×4 3D rigid-body transformation matrices.

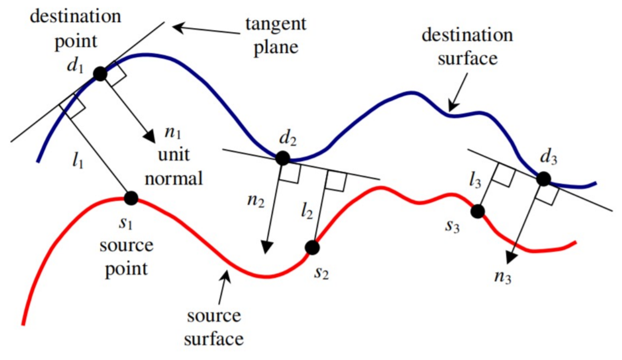
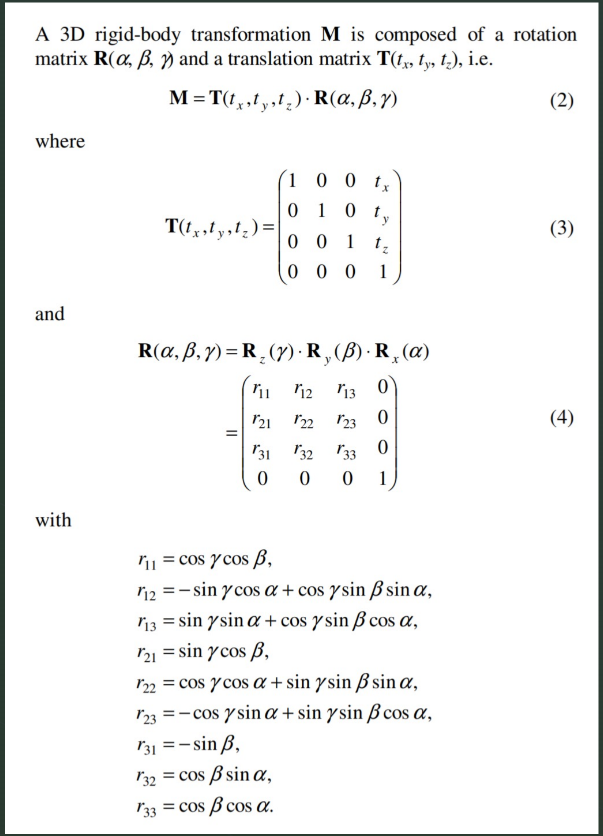


***Figure 1**: Point-to-plane error between two surfaces.*



A 3D rigid-body transformation $\mathbf{M}$ is composed of a rotation matrix $\mathbf{R}(\alpha, \beta, \gamma)$ and a translation matrix $\mathbf{T}(t_x, t_y, t_z)$, i.e.

$$\mathbf{M} = \mathbf{T}(t_x, t_y, t_z) \cdot \mathbf{R}(\alpha, \beta, \gamma) \qquad (2)$$

where

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (3)$$

and

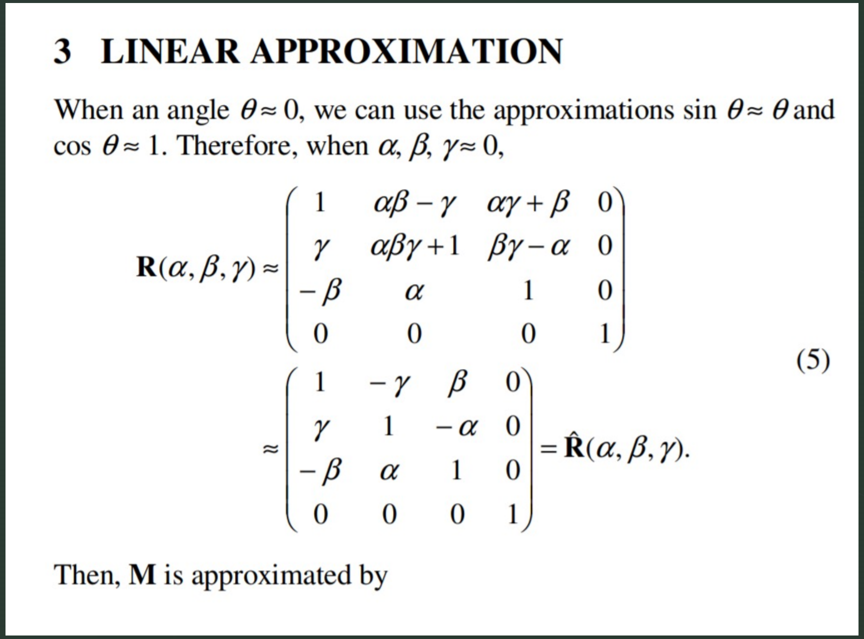
$$\mathbf{R}(\alpha, \beta, \gamma) = \mathbf{R}_z(\gamma) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_x(\alpha) = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (4)$$

with

$$\begin{aligned}
r_{11} &= \cos\gamma\cos\beta, \\
r_{12} &= -\sin\gamma\cos\alpha + \cos\gamma\sin\beta\sin\alpha, \\
r_{13} &= \sin\gamma\sin\alpha + \cos\gamma\sin\beta\cos\alpha, \\
r_{21} &= \sin\gamma\cos\beta, \\
r_{22} &= \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha, \\
r_{23} &= -\cos\gamma\sin\alpha + \sin\gamma\sin\beta\cos\alpha, \\
r_{31} &= -\sin\beta, \\
r_{32} &= \cos\beta\sin\alpha, \\
r_{33} &= \cos\beta\cos\alpha.
\end{aligned}$$

Low, K. L. (2004). Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, *4*(10), 1-3.

Exercise 2.5 P2S Registration

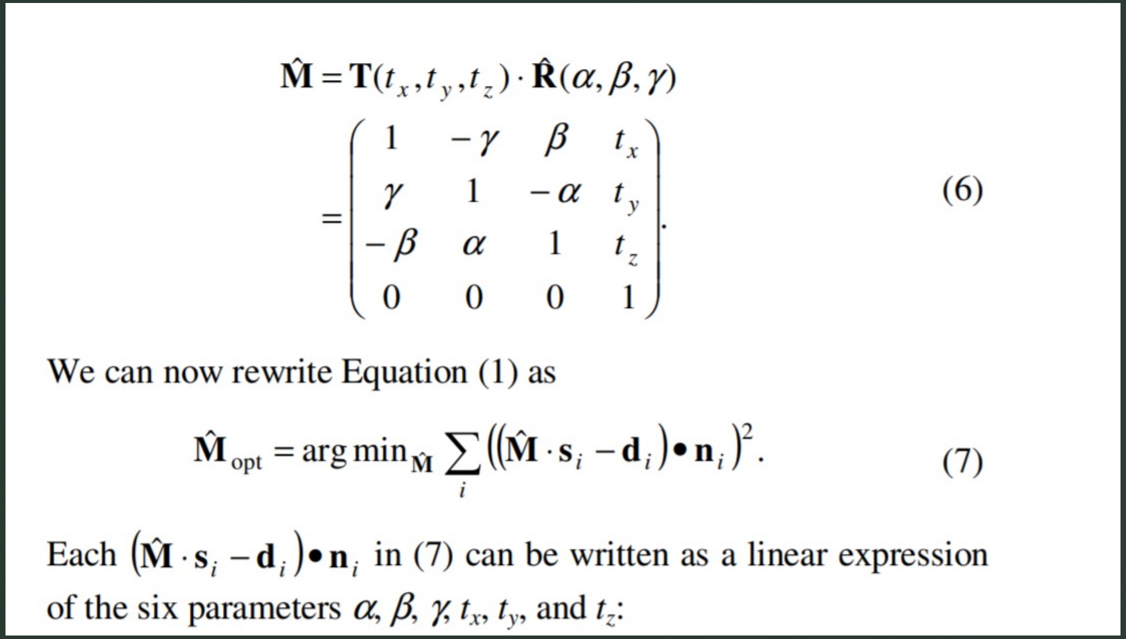


3 LINEAR APPROXIMATION

When an angle $\theta \approx 0$, we can use the approximations $\sin\theta \approx \theta$ and $\cos\theta \approx 1$. Therefore, when $\alpha, \beta, \gamma \approx 0$,

$$\mathbf{R}(\alpha,\beta,\gamma) \approx \begin{pmatrix} 1 & \alpha\beta-\gamma & \alpha\gamma+\beta & 0 \\ \gamma & \alpha\beta\gamma+1 & \beta\gamma-\alpha & 0 \\ -\beta & \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$
$$\approx \begin{pmatrix} 1 & -\gamma & \beta & 0 \\ \gamma & 1 & -\alpha & 0 \\ -\beta & \alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \hat{\mathbf{R}}(\alpha,\beta,\gamma). \tag{5}$$

Then, **M** is approximated by



$$\hat{\mathbf{M}} = \mathbf{T}(t_x,t_y,t_z)\cdot\hat{\mathbf{R}}(\alpha,\beta,\gamma)$$
$$= \begin{pmatrix} 1 & -\gamma & \beta & t_x \\ \gamma & 1 & -\alpha & t_y \\ -\beta & \alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{6}$$

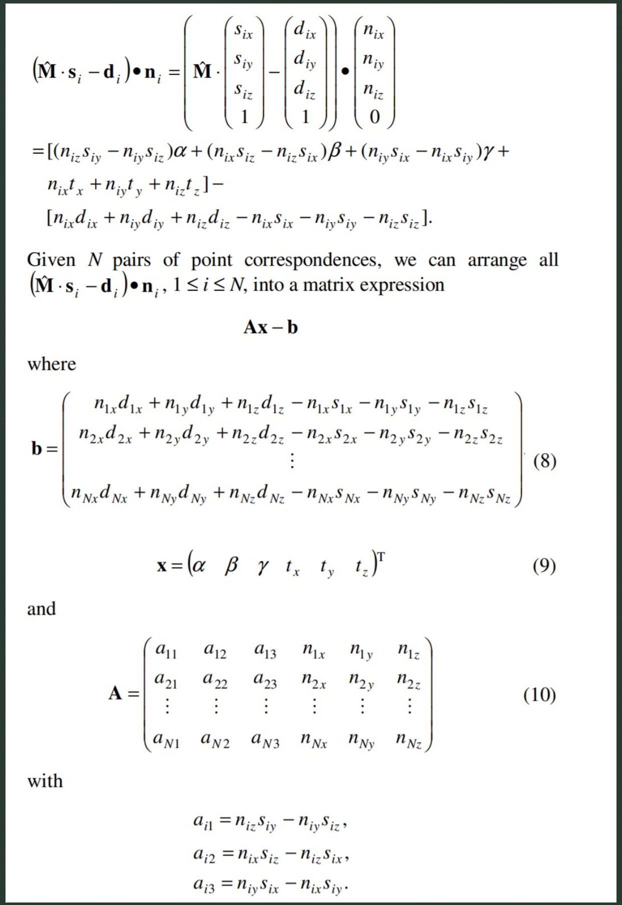
We can now rewrite Equation (1) as

$$\hat{\mathbf{M}}_{\text{opt}} = \arg\min_{\hat{\mathbf{M}}} \sum_i \left(\left(\hat{\mathbf{M}}\cdot\mathbf{s}_i - \mathbf{d}_i\right)\bullet\mathbf{n}_i \right)^2. \tag{7}$$

Each $\left(\hat{\mathbf{M}}\cdot\mathbf{s}_i - \mathbf{d}_i\right)\bullet\mathbf{n}_i$ in (7) can be written as a linear expression of the six parameters α, β, γ, t_x, t_y, and t_z:

Low, K. L. (2004). Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, *4*(10), 1-3.

Exercise 2.5 P2S Registration



$$(\hat{\mathbf{M}} \cdot \mathbf{s}_i - \mathbf{d}_i) \bullet \mathbf{n}_i = \left(\hat{\mathbf{M}} \cdot \begin{pmatrix} s_{ix} \\ s_{iy} \\ s_{iz} \\ 1 \end{pmatrix} - \begin{pmatrix} d_{ix} \\ d_{iy} \\ d_{iz} \\ 1 \end{pmatrix} \right) \bullet \begin{pmatrix} n_{ix} \\ n_{iy} \\ n_{iz} \\ 0 \end{pmatrix}$$

$$= [(n_{iz}s_{iy} - n_{iy}s_{iz})\alpha + (n_{ix}s_{iz} - n_{iz}s_{ix})\beta + (n_{iy}s_{ix} - n_{ix}s_{iy})\gamma + n_{ix}t_x + n_{iy}t_y + n_{iz}t_z] - [n_{ix}d_{ix} + n_{iy}d_{iy} + n_{iz}d_{iz} - n_{ix}s_{ix} - n_{iy}s_{iy} - n_{iz}s_{iz}].$$

Given N pairs of point correspondences, we can arrange all $(\hat{\mathbf{M}} \cdot \mathbf{s}_i - \mathbf{d}_i) \bullet \mathbf{n}_i$, $1 \le i \le N$, into a matrix expression

$$\mathbf{Ax} - \mathbf{b}$$

where

$$\mathbf{b} = \begin{pmatrix} n_{1x}d_{1x} + n_{1y}d_{1y} + n_{1z}d_{1z} - n_{1x}s_{1x} - n_{1y}s_{1y} - n_{1z}s_{1z} \\ n_{2x}d_{2x} + n_{2y}d_{2y} + n_{2z}d_{2z} - n_{2x}s_{2x} - n_{2y}s_{2y} - n_{2z}s_{2z} \\ \vdots \\ n_{Nx}d_{Nx} + n_{Ny}d_{Ny} + n_{Nz}d_{Nz} - n_{Nx}s_{Nx} - n_{Ny}s_{Ny} - n_{Nz}s_{Nz} \end{pmatrix} \quad (8)$$

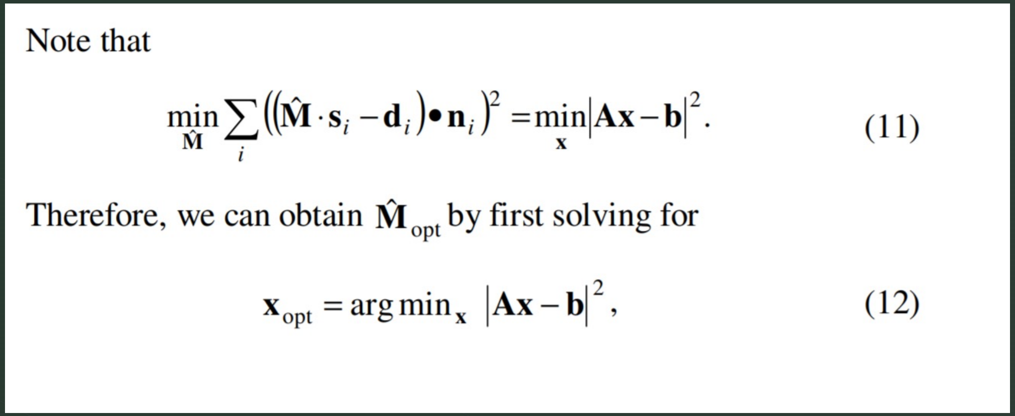
$$\mathbf{x} = (\alpha \quad \beta \quad \gamma \quad t_x \quad t_y \quad t_z)^{\mathrm{T}} \quad (9)$$

and

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & n_{1x} & n_{1y} & n_{1z} \\ a_{21} & a_{22} & a_{23} & n_{2x} & n_{2y} & n_{2z} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & n_{Nx} & n_{Ny} & n_{Nz} \end{pmatrix} \quad (10)$$

with

$$a_{i1} = n_{iz}s_{iy} - n_{iy}s_{iz},$$
$$a_{i2} = n_{ix}s_{iz} - n_{iz}s_{ix},$$
$$a_{i3} = n_{iy}s_{ix} - n_{ix}s_{iy}.$$



Note that

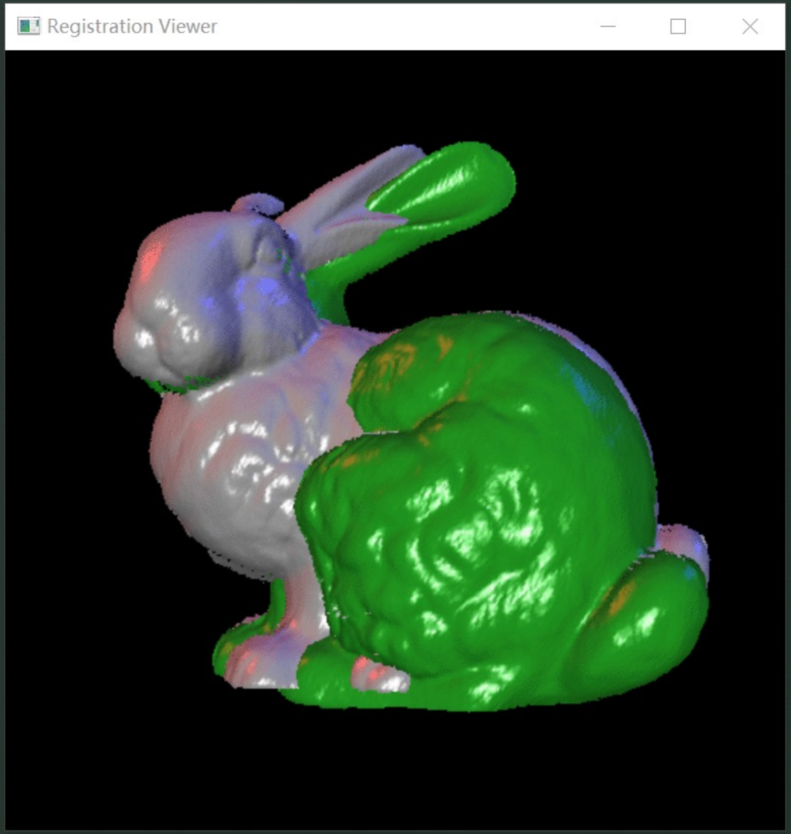
$$\min_{\hat{\mathbf{M}}} \sum_i \left((\hat{\mathbf{M}} \cdot \mathbf{s}_i - \mathbf{d}_i) \bullet \mathbf{n}_i \right)^2 = \min_{\mathbf{x}} |\mathbf{Ax} - \mathbf{b}|^2. \quad (11)$$

Therefore, we can obtain $\hat{\mathbf{M}}_{\text{opt}}$ by first solving for

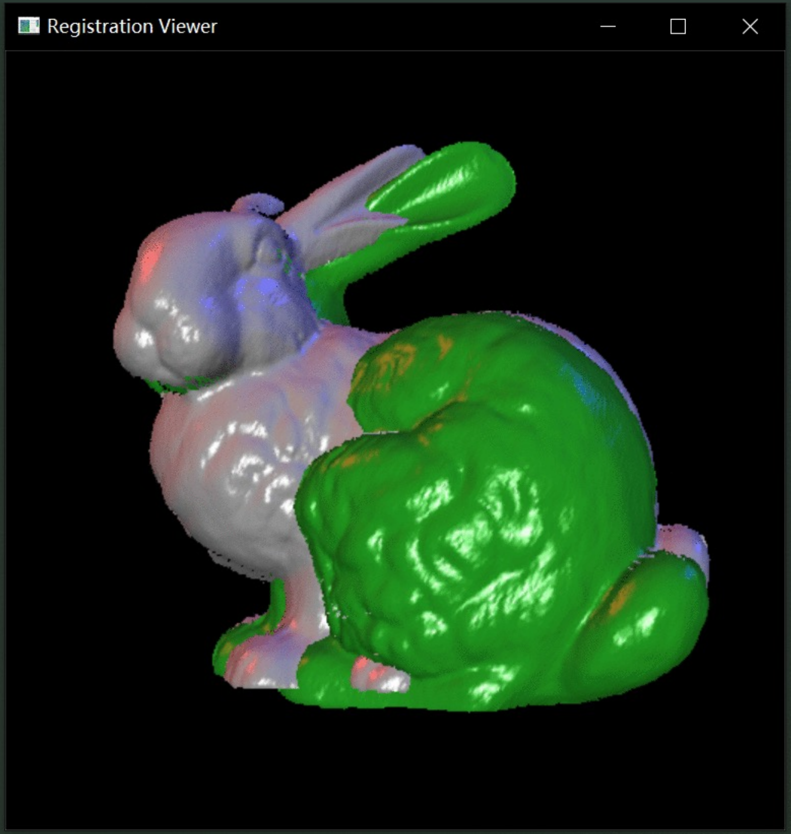
$$\mathbf{x}_{\text{opt}} = \arg\min_{\mathbf{x}} |\mathbf{Ax} - \mathbf{b}|^2, \quad (12)$$

Low, K. L. (2004). Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, *4*(10), 1-3.

Experimental Results / bunny01+02

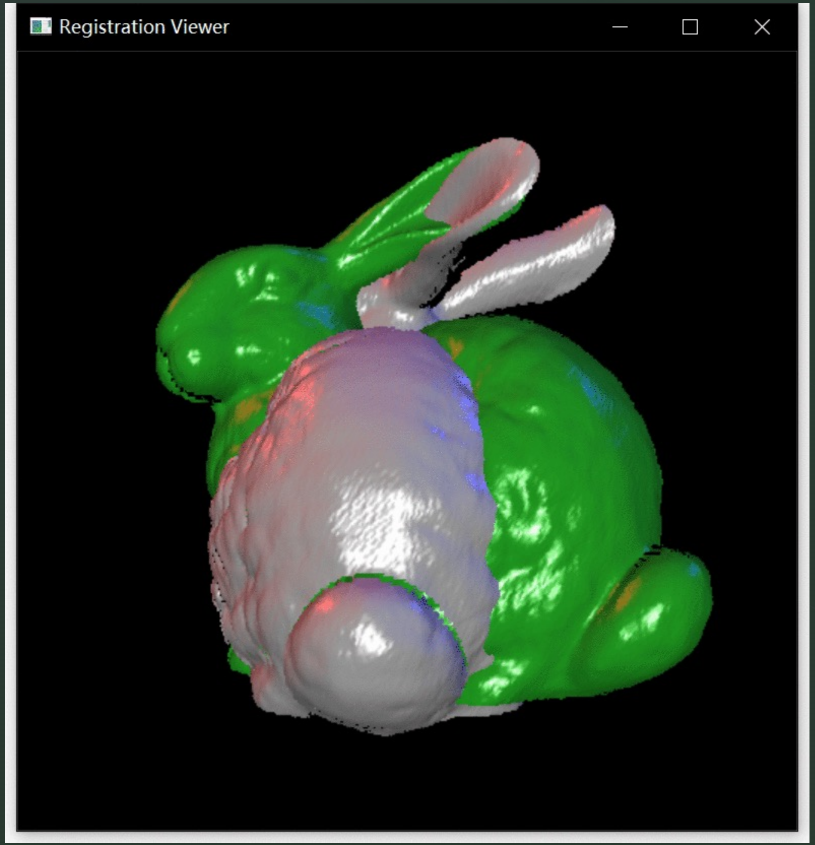


Point to point, 30 epochs

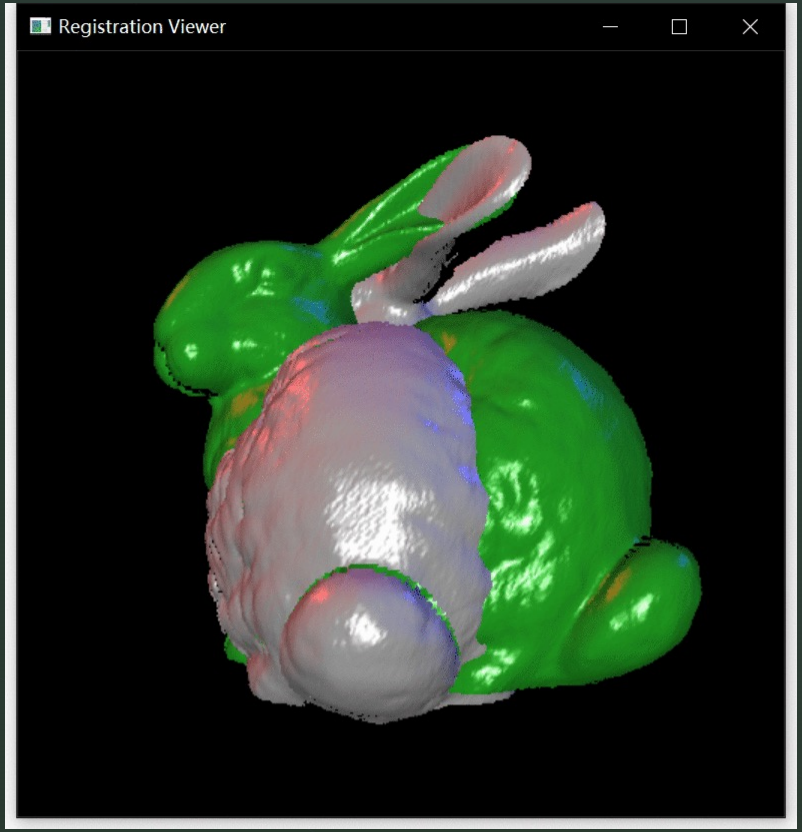


Point to surface, **4 EPOCHS**

Experimental Results / bunny02+03

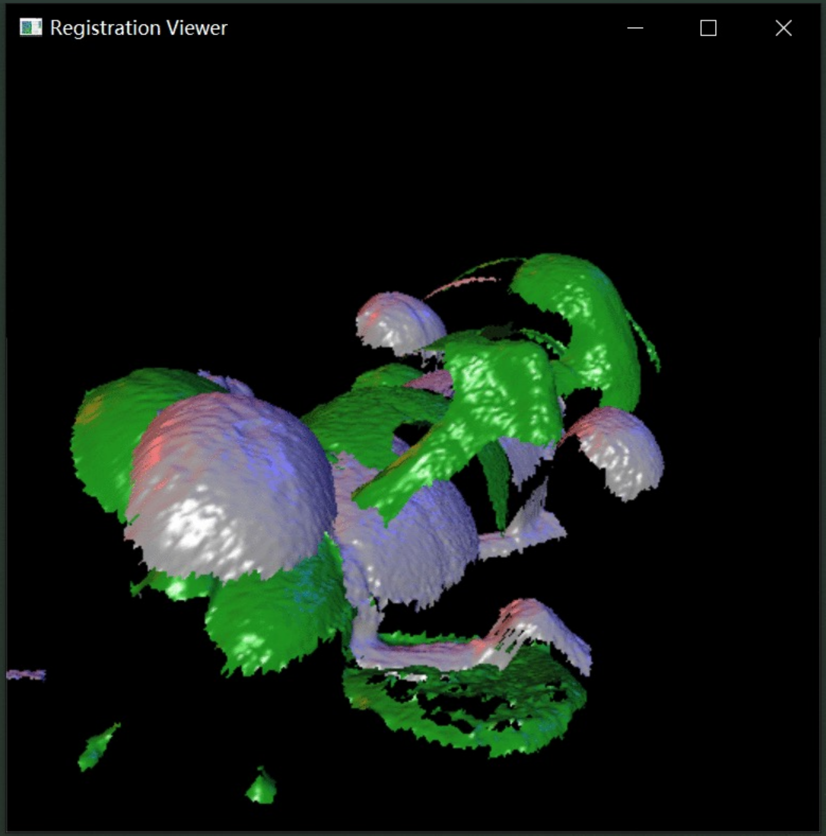


Point to point, 100 epochs

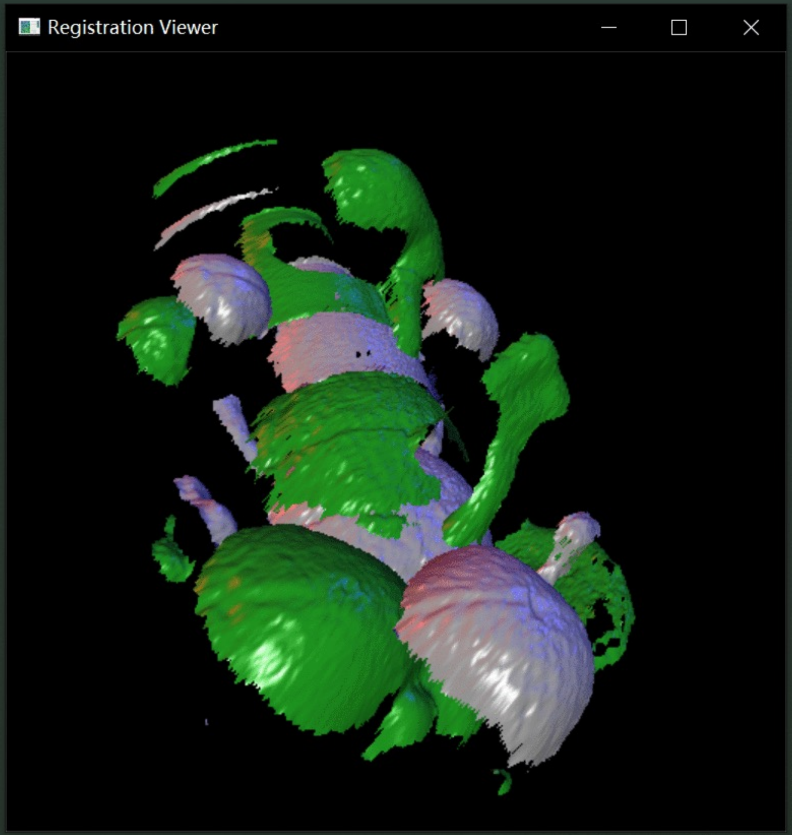


Point to surface, **20 EPOCHS**

Experimental Results / bee1+2



Point to point, 100 epochs



Point to surface, **20 EPOCHS**

Conclusions

- For normal objects with small rigid distance, both algorithm converges, and point-to-surface approach converges ~7 times faster than the point-to-point approach. (bunny01+02)
- For normal objects with large rigid distance, point-to-point approach does **not** converge, but point-to-surface approach still converges, although the convergence rate is lower. (bunny02+03)
- For complex objects, even with a small rigid distance, both algorithm does **not** converge. (bee1+2)
- With manual adjustments, the convergence rate is a bit better, but still greatly dependent on the complexity of the object. (not illustrated)

Future Work

- Ext: Normal subsampling
- Ext: sort() algorithm in large number case.
- Ext: use another array for strong the elements that fulfill the requirements
- More experiments on the animals!

Accessing Github

`https://github.com/BiEchi/IterativeClosestPoint`

Inherit GPL License!