# KinectFusion中Point-to-plane ICP的计算推导

## 目录



## 一、Point to plane ICP 误差函数

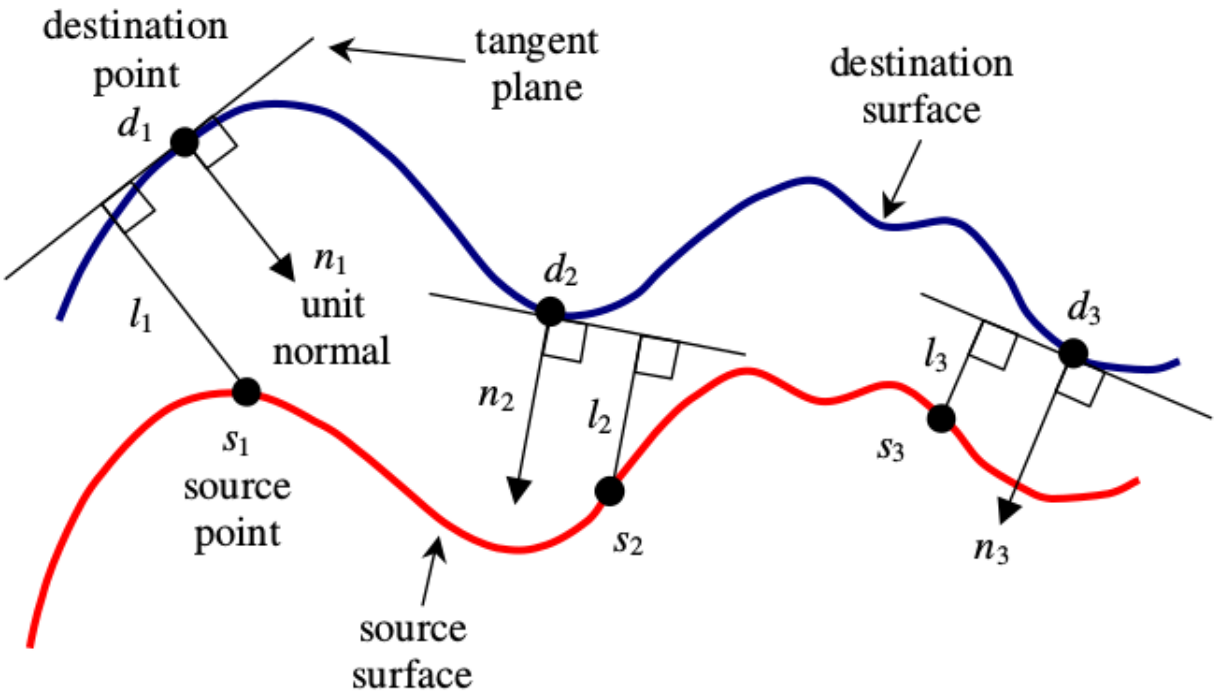
误差函数数学表达式为：

$$\mathbf{T}_{qp}^{*}=\arg\min_{\mathbf{T}_{qp}}\sum_{i}\left(\left(\mathbf{T}_{qp}\mathbf{p}_{i}-\mathbf{q}_{i}\right)\cdot\mathbf{n}_{i}\right)^{2} \tag{1}$$

其中：

- $\mathbf{p}_i$为当前帧表面点云的第$i$个点
- $\mathbf{q}_i$为上一帧推理的表面点云的第$i$个点
- $\mathbf{n}_i$为上一帧推理的表面点云第$i$个点的法向量
- $\mathbf{T}_{qp}$为将当前帧表面点云转换到上一帧推理的表面点云所需要进行的位姿变换

上式的直观意义是，在当前帧表面点云中的点$\mathbf{p}_i$和上一帧推理的表面点云中的一个点$\mathbf{q}_i$构成了关联关系之后，误差被定义为点$\mathbf{p}_i$到点$\mathbf{q}_i$所在表面的切平面的距离，或者也可以理解为点$\mathbf{p}_i$到点$\mathbf{q}_i$的欧式距离在点$\mathbf{q}_i$处法向量上的投影长度$\mathbf{n}_i$，如下图所示：（图像中使用$s_1$表示$\mathbf{p}_i$，使用$d_i$表示$q_i$）（参考文献[1]）




## 二、 线性化

(1)式中的$\mathbf{T}_{qp}$由一个旋转和一个平移组成：

$$\begin{aligned}\mathbf{T}_{qp} &= \begin{bmatrix}\mathbf{R} & \mathbf{t}\\ \mathbf{0}^{\mathrm{T}} & 1\end{bmatrix}\\ &= \begin{bmatrix}r_{11} & r_{12} & r_{13} & t_x\\ r_{21} & r_{22} & r_{23} & t_y\\ r_{31} & r_{32} & r_{33} & t_z\\ 0 & 0 & 0 & 1\end{bmatrix}\end{aligned} \tag{2}$$

对于旋转，还有对应的欧拉角$\alpha$、$\beta$、$\gamma$表示：

$$\mathbf{R}(\alpha,\beta,\gamma) = \mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha) \tag{3}$$

结合旋转矩阵和欧拉角的转换关系，(2)式旋转矩阵中的每一项都可以写成关于欧拉角的关系：

$$\begin{cases} r_{11} = \cos\gamma\cos\beta \\ r_{12} = -\sin\gamma\cos\alpha + \cos\gamma\sin\beta\sin\alpha \\ r_{13} = \sin\gamma\sin\alpha + \cos\gamma\sin\beta\cos\alpha \\ r_{21} = \sin\gamma\cos\beta \\ r_{22} = \cos\gamma\cos\alpha + \sin\gamma\sin\beta\sin\alpha \\ r_{23} = -\cos\gamma\sin\alpha + \sin\gamma\sin\beta\cos\alpha \\ r_{31} = -\sin\beta \\ r_{32} = \cos\beta\sin\alpha \\ r_{33} = \cos\beta\cos\alpha \end{cases} \tag{4}$$

(4)式是一个非线性的表达式。在KinectFusion中假设了两帧之间的相机运动非常小，即三个轴上的旋转角都很小，这意味着我们可以得到近似形式：

$$\begin{cases} sin\theta \approx \theta \\ cos\theta \approx 1 \end{cases} \tag{5}$$

进而我们可以对$\mathbf{T}_{qp}$中的旋转矩阵进行线型近似：

$$\begin{aligned}\tilde{\mathbf{T}}_{qp} &= \begin{bmatrix}\tilde{\mathbf{R}} & \mathbf{t}\\ \mathbf{0}^{\mathrm{T}} & 1\end{bmatrix}\\ &\approx \begin{bmatrix}1 & \alpha\beta-\gamma & \alpha\gamma-\beta & t_x\\ \gamma & \alpha\beta\gamma+1 & \beta\gamma-\alpha & t_y\\ -\beta & \alpha & 1 & t_z\\ 0 & 0 & 0 & 1\end{bmatrix}\\ &\approx \begin{bmatrix}1 & -\gamma & \beta & t_x\\ \gamma & 1 & -\alpha & t_y\\ -\beta & \alpha & 1 & t_z\\ 0 & 0 & 0 & 1\end{bmatrix}\end{aligned} \tag{6}$$

那么对于式(1)现在获得其近似形式：

$$\tilde{\mathbf{T}}_{qp}^{*} = \arg\min_{\tilde{\mathbf{T}}_{qp}} \sum_{i} \left( \left( \tilde{\mathbf{T}}_{qp}\mathbf{p}_i - \mathbf{q}_i \right) \cdot \mathbf{n}_i \right)^2 \tag{7}$$

对于第$i$对匹配点，将式(6)代入，对最终结果提取出$\alpha$、$\beta$、$\gamma$和$t_x$、$t_y$、$t_z$得：

$$\begin{aligned}\left( \tilde{\mathbf{T}}_{qp} \cdot \mathbf{p}_i - \mathbf{q}_i \right) \cdot \mathbf{n}_i =& \left( \tilde{\mathbf{T}}_{qp} \cdot \begin{bmatrix}p_{ix}\\ p_{iy}\\ p_{iz}\\ 1\end{bmatrix} - \begin{bmatrix}q_{ix}\\ q_{iy}\\ q_{iz}\\ 1\end{bmatrix} \right) \cdot \begin{bmatrix}n_{ix}\\ n_{iy}\\ n_{iz}\\ 0\end{bmatrix}\\ =& \alpha\left(n_{iz}p_{iy} - n_{iy}p_{iz}\right) + \beta\left(n_{ix}p_{iz} - n_{iz}p_{ix}\right) + \gamma\left(n_{iy}p_{ix} - n_{ix}p_{iy}\right) +\\ & t_x n_{ix} + t_y n_{iy} + t_z n_{iz} -\\ & \left(n_{ix}q_{ix} + n_{iy}q_{iy} + n_{iz}q_{iz} - n_{ix}p_{ix} - n_{iy}p_{iy} - n_{iz}p_{iz}\right)\end{aligned} \tag{8}$$

# 三、最小二乘SVD方式求解

得到式(8)之后，我们可以发现对于其中任意一对点的误差，可以分成三个部分组成：带有因子$\alpha$、$\beta$、$\gamma$的部分，带有因子$t_x$、$t_y$、$t_z$的部分，以及两种因子都不包含的部分。记$\mathbf{x} = [\alpha, \beta, \gamma, t_x, t_y, t_z]^{\mathrm{T}}$， 若在理想情况下$\mathbf{x}$为准确的相机位姿，那么对于第$i$对点的误差为0，可以写为：

$$\begin{bmatrix} a_{i1} & a_{i2} & a_{i3} & n_{ix} & n_{iy} & n_{iz} \end{bmatrix} \cdot \mathbf{x} - b_i = 0 \tag{9}$$

其中：

$$\begin{cases} a_{i1} = n_{iz}p_{iy} - n_{iy}p_{iz} \\ a_{i2} = n_{ix}p_{iz} - n_{iz}p_{ix} \\ a_{i3} = n_{iy}p_{ix} - n_{ix}p_{iy} \\ b_i = n_{ix}q_{ix} + n_{iy}q_{iy} + n_{iz}q_{iz} - n_{ix}p_{ix} - n_{iy}p_{iy} - n_{iz}p_{iz} \end{cases} \tag{10}$$

对于全部的$N$对匹配点，则式(9)有其对应的矩阵形式：

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & n_{1x} & n_{1y} & n_{1z} \\ a_{21} & a_{22} & a_{23} & n_{2x} & n_{2y} & n_{2z} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & n_{Nx} & n_{Ny} & n_{Nz} \end{bmatrix} \cdot \mathbf{x} - \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} = \mathbf{0} \tag{11}$$

可以写成紧凑形式：

$$\mathbf{A}\mathbf{x}-\mathbf{b}=\mathbf{0} \tag{12}$$

注意到，式(12)所蕴含的最优化问题和式(7)中的是一样的，终极目标都是使得每一对点的误差之和最小，理想情况下使得误差为0：

$$\min_{\tilde{\mathbf{T}}_{qp}}\sum_{i}\left(\left(\tilde{\mathbf{T}}_{qp}\mathbf{p}_i-\mathbf{q}_i\right)\cdot\mathbf{n}_i\right)^2=\min_{\mathbf{x}}\left|\mathbf{A}\mathbf{x}-\mathbf{b}\right|^2 \tag{13}$$

因此式(12)的解$\mathbf{x}^*$对应于$\tilde{\mathbf{T}}_{qp}^*$，因此问题转换成为对式(12)的求解。由于在实际应用中，ICP的匹配点往往远远多于未知量的个数（6个），因此式(12)是一个超定方程，一般无解；实际应用中通常求其最小二乘解。

由于矩阵$\mathbf{A}$不一定可逆，于是求解上述方程的一个思想就是同求解矩阵$\mathbf{A}$的伪逆来求解上述方程组。首先对矩阵$\mathbf{A}$进行SVD分解：

$$\mathbf{A}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} \tag{14}$$

得到其伪逆$\mathbf{A}^+$：

$$\mathbf{A}^+=\mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^{\mathrm{T}} \tag{15}$$

其中$\mathbf{\Sigma}^+$为$\mathbf{\Sigma}$中所有不为0的元素取逆之后得到。接着即可以求解式(12)所示的方程：

$$\mathbf{x}^*=\mathbf{A}^+\mathbf{b} \tag{16}$$

得到$\mathbf{x}^*$之后，使用其中的$\alpha^*$、$\beta^*$、$\gamma^*$和$t_x^*$、$t_y^*$、$t_z^*$来构造$\tilde{\mathbf{T}}_{qp}$即可。

# 四、另一种可用于并行处理的最小二乘求解

由于前面对于非线性的旋转已经进行了线性化，因此式(12)本质上是一个线型最小二乘的问题。如果不考虑式(13)中进行的优化问题等价，即如果考虑式(7)，参考文献[2]206页，那么此线型最小二乘问题可以描述为：

$$\mathbf{x}^*=\min_{\mathbf{x}}\parallel\mathbf{A}\mathbf{x}-\mathbf{b}\parallel^2 \tag{17}$$

数学上可以证明，$\mathbf{x}^*$为极小点的充要条件是$\mathbf{x}^*$是下面方程组

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x}=\mathbf{A}^{\mathrm{T}}\mathbf{b} \tag{18}$$

的解。为方便起见，这里对式(18)使用新的表示：

$$\mathbf{C}\mathbf{x}=\mathbf{d} \tag{19}$$

其中

$$\begin{aligned}\mathbf{C}&=\mathbf{A}^{\mathrm{T}}\mathbf{A}\\\mathbf{d}&=\mathbf{A}^{\mathrm{T}}\mathbf{b}\end{aligned} \tag{20}$$

其实对于式(19)，本质上和式(12)是相同的。KinectFusion中的ICP计算是利用GPU进行并行加速实现的，因此对于式(11)中的矩阵$\mathbf{A}$和向量$\mathbf{b}$中的表达，不方便进行并行化，需要进行一些处理。

# 五、并行化处理

此部分处理参考文献[3]。回顾向量叉乘的定义，对于向量$\mathbf{p}_i$和向量$\mathbf{n}_i$：

$$\mathbf{p}_i = \begin{bmatrix} p_{ix} \\ p_{iy} \\ p_{iz} \end{bmatrix} \quad \mathbf{n}_i = \begin{bmatrix} n_{ix} \\ n_{iy} \\ n_{iz} \end{bmatrix} \tag{21}$$

那么根据向量叉乘的定义，结合式(10)有：

$$\mathbf{p}_i \times \mathbf{n}_i = \begin{bmatrix} n_{iz}p_{iy} - n_{iy}p_{iz} \\ n_{ix}p_{iz} - n_{iz}p_{ix} \\ n_{iy}p_{ix} - n_{ix}p_{iy} \end{bmatrix} = \begin{bmatrix} a_{i1} \\ a_{i2} \\ a_{i3} \end{bmatrix} \tag{22}$$

发现如果设置：

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} \\ \mathbf{t} \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ t_x \\ t_y \\ t_z \end{bmatrix} \tag{23}$$

结合式(6)，将原代价函数中第$i$对匹配点的贡献写成如下形式：

$$\begin{aligned} \left(\tilde{\mathbf{T}}_{qp} \cdot \mathbf{p}_i - \mathbf{q}_i\right) \cdot \mathbf{n}_i &= \tilde{\mathbf{R}}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i \\ &= \left(\mathbf{p}_i \times \mathbf{n}_i\right)^{\mathrm{T}} \cdot \mathbf{r} + \mathbf{n}_i^{\mathrm{T}} \cdot \mathbf{t} - \left(\mathbf{q}_i - \mathbf{p}_i\right)^{\mathrm{T}} \cdot \mathbf{n}_i \end{aligned} \tag{24}$$

参考式(12)的表达形式，有：

$$\begin{bmatrix} \left(\mathbf{p}_i \times \mathbf{n}_i\right)^{\mathrm{T}} & \mathbf{n}_i^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{t} \end{bmatrix} - \begin{bmatrix} \left(\mathbf{q}_i - \mathbf{p}_i\right)^{\mathrm{T}} \cdot \mathbf{n}_i \end{bmatrix} = 0 \tag{25}$$

对于全部的$N$对匹配点，有：

$$\begin{bmatrix} \left(\mathbf{p}_1 \times \mathbf{n}_1\right)^{\mathrm{T}} & \mathbf{n}_1^{\mathrm{T}} \\ \left(\mathbf{p}_2 \times \mathbf{n}_2\right)^{\mathrm{T}} & \mathbf{n}_2^{\mathrm{T}} \\ \vdots & \vdots \\ \left(\mathbf{p}_N \times \mathbf{n}_N\right)^{\mathrm{T}} & \mathbf{n}_N^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \mathbf{r} \\ \mathbf{t} \end{bmatrix} - \begin{bmatrix} \left(\mathbf{q}_1 - \mathbf{p}_1\right)^{\mathrm{T}} \cdot \mathbf{n}_1 \\ \left(\mathbf{q}_2 - \mathbf{p}_2\right)^{\mathrm{T}} \cdot \mathbf{n}_2 \\ \vdots \\ \left(\mathbf{q}_N - \mathbf{p}_N\right)^{\mathrm{T}} \cdot \mathbf{n}_N \end{bmatrix} = 0 \tag{26}$$

其实就是式(12)中的矩阵$\mathbf{A}$和向量$\mathbf{b}$的另外一种表达。

回到式(19)所表示的最小二乘形式，有：

$$
\begin{aligned}
\mathbf{C} &= \mathbf{A}^{\mathrm{T}}\mathbf{A} \\
&= \begin{bmatrix} \mathbf{p}_1 \times \mathbf{n}_1 & \cdots & \mathbf{p}_N \times \mathbf{n}_N \\ \mathbf{n}_1 & \cdots & \mathbf{n}_N \end{bmatrix} \begin{bmatrix} (\mathbf{p}_1 \times \mathbf{n}_1)^{\mathrm{T}} & \mathbf{n}_1^{\mathrm{T}} \\ \vdots & \vdots \\ (\mathbf{p}_N \times \mathbf{n}_N)^{\mathrm{T}} & \mathbf{n}_N^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{p}_1 \times \mathbf{n}_1)(\mathbf{p}_1 \times \mathbf{n}_1)^{\mathrm{T}} + \cdots + (\mathbf{p}_N \times \mathbf{n}_N)(\mathbf{p}_N \times \mathbf{n}_N)^{\mathrm{T}} & (\mathbf{p}_1 \times \mathbf{n}_1)\mathbf{n}_1^{\mathrm{T}} + \cdots + (\mathbf{p}_N \times \mathbf{n}_N)\mathbf{n}_N^{\mathrm{T}} \\ \mathbf{n}_1(\mathbf{p}_1 \times \mathbf{n}_1)^{\mathrm{T}} + \cdots + \mathbf{n}_N(\mathbf{p}_N \times \mathbf{n}_N)^{\mathrm{T}} & \mathbf{n}_1\mathbf{n}_1^{\mathrm{T}} + \cdots + \mathbf{n}_N\mathbf{n}_N^{\mathrm{T}} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=0}^{N}(\mathbf{p}_i \times \mathbf{n}_i)(\mathbf{p}_i \times \mathbf{n}_i)^{\mathrm{T}} & \sum_{i=0}^{N}(\mathbf{p}_i \times \mathbf{n}_i)\mathbf{n}_i^{\mathrm{T}} \\ \sum_{i=0}^{N}\mathbf{n}_i(\mathbf{p}_i \times \mathbf{n}_i)^{\mathrm{T}} & \sum_{i=0}^{N}\mathbf{n}_i\mathbf{n}_i^{\mathrm{T}} \end{bmatrix} \\
&= \sum_{i=0}^{N} \begin{bmatrix} (\mathbf{p}_i \times \mathbf{n}_i)(\mathbf{p}_i \times \mathbf{n}_i)^{\mathrm{T}} & (\mathbf{p}_i \times \mathbf{n}_i)\mathbf{n}_i^{\mathrm{T}} \\ \mathbf{n}_i(\mathbf{p}_i \times \mathbf{n}_i)^{\mathrm{T}} & \mathbf{n}_i\mathbf{n}_i^{\mathrm{T}} \end{bmatrix} \\
&= \sum_{i=0}^{N} \begin{bmatrix} \mathbf{p}_i \times \mathbf{n}_i \\ \mathbf{n}_i \end{bmatrix} \begin{bmatrix} (\mathbf{p}_i \times \mathbf{n}_i)^{\mathrm{T}} & \mathbf{n}_i^{\mathrm{T}} \end{bmatrix}
\end{aligned} \tag{27}
$$

由上式可以发现，矩阵$\mathbf{C}$的维度是$6 \times 6$，并且是一个对称矩阵，KinectFusion论文[4]中提到了这种对称关系可以节约内存(3.5节，式26后面)。同时我们还得到了对于其中每一对匹配点对整个矩阵$\mathbf{C}$的贡献，因此可以在GPU中使用每一个核函数来单独处理其中一对匹配点的计算。

类似地，对于向量$\mathbf{d}$:

$$
\begin{aligned}
\mathbf{d} &= \mathbf{A}^{\mathrm{T}}\mathbf{b} \\
&= \begin{bmatrix} \mathbf{p}_1 \times \mathbf{n}_1 & \cdots & \mathbf{p}_N \times \mathbf{n}_N \\ \mathbf{n}_1 & \cdots & \mathbf{n}_N \end{bmatrix} \begin{bmatrix} (\mathbf{q}_1 - \mathbf{p}_1) \cdot \mathbf{n}_1 \\ \vdots \\ (\mathbf{q}_N - \mathbf{p}_N) \cdot \mathbf{n}_N \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{p}_1 \times \mathbf{n}_1)((\mathbf{q}_1 - \mathbf{p}_1) \cdot \mathbf{n}_1) + \cdots + (\mathbf{p}_N \times \mathbf{n}_N)((\mathbf{q}_N - \mathbf{p}_N) \cdot \mathbf{n}_N) \\ \mathbf{n}_1((\mathbf{q}_1 - \mathbf{p}_1) \cdot \mathbf{n}_1) + \cdots + \mathbf{n}_N((\mathbf{q}_N - \mathbf{p}_N) \cdot \mathbf{n}_N) \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i=0}^{N}(\mathbf{p}_i \times \mathbf{n}_i)((\mathbf{q}_i - \mathbf{p}_i) \cdot \mathbf{n}_i) \\ \sum_{i=0}^{N}\mathbf{n}_i((\mathbf{q}_i - \mathbf{p}_i) \cdot \mathbf{n}_i) \end{bmatrix} \\
&= \sum_{i=0}^{N} \begin{bmatrix} (\mathbf{p}_i \times \mathbf{n}_i)((\mathbf{q}_i - \mathbf{p}_i) \cdot \mathbf{n}_i) \\ \mathbf{n}_i((\mathbf{q}_i - \mathbf{p}_i) \cdot \mathbf{n}_i) \end{bmatrix}
\end{aligned} \tag{28}
$$

可以发现向量$\mathbf{d}$是一个$6 \times 1$的向量，并且我们也得到了其中的每个像素对向量$\mathbf{d}$的贡献，可以使用GPU对其进行并行处理。

综上GPU中并行计算得到$\mathbf{C}$和$\mathbf{d}$后，就可以在CPU中求解如式(19)所示的方程了，得到的解即为$\mathbf{x}^*$，进而可以得到$\tilde{\mathbf{T}}_{qp}$。

# 参考资料


[1] Low, Kok-Lim. "Linear least-squares optimization for point-to-plane icp surface registration." *Chapel Hill, University of North Carolina* 4.10 (2004): 1-3.

[2] 张薇，薛家庆. 最优化方法. 东北大学出版社, 2004.

[3] 付兴银. RGB-D实时三维重建/SLAM中的ICP算法解析. https://blog.csdn.net/fuxingyin/article/details/51425721. 2016.05.16, 2019.11.26.

[4] Newcombe, Richard A., et al. "Kinectfusion: Real-time dense surface mapping and tracking." *ISMAR*. Vol. 11. No. 2011. 2011.