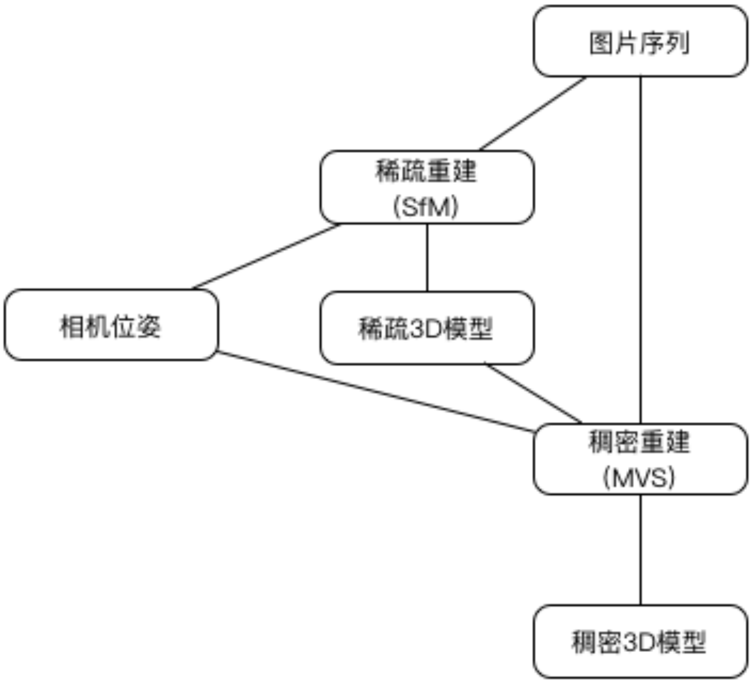
胡说八道——运动到结构（Structure from Motion）

什么是运动到结构

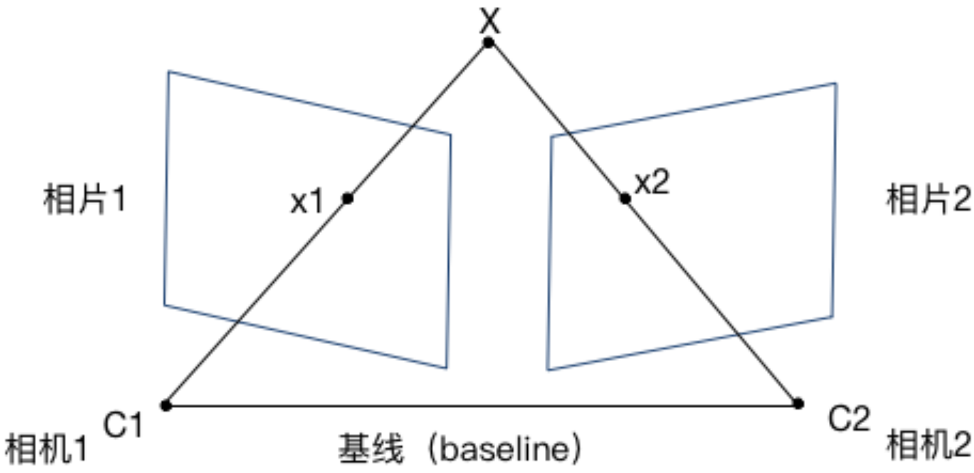
运动到结构（Structure from Motion，SfM）是利用2D图片序列估计3D结构的一种3D重建方法。由于这种方法只能产生较为稀疏（Sparse）的结构信息（因为只是图像上部分点），所以也称为稀疏重建（Sparse Reconstruction）。目前SfM方法最大的用途是**估计相机位姿（Camera Pose）**，结构重建反而是SfM的副产品。

简言之，SfM算法**读入图片序列**，输出**相机位姿**、**稀疏的3D模型**。而**稠密重建（Dense Reconstruction）**方法以**图片序列**、**相机位姿**、**稀疏的3D模型**为输入，输出**稠密的3D模型**。**稠密重建**的方法主要是**Multi-View Stereo（MVS）**方法。MVS方法超出本文的范畴，留待以后再谈。图片序列、SfM和MVS的关系如下图所示：



SfM的原理

SfM基于极线几何，原理如下图所示：



假设相机在$C1$和$C2$分别拍摄了两张图片，空间点X在相片上的投影点分别为x_1和x_2，则x_1和x_2满足如下的公式：

$$x_1^T F x_2 = 0 \tag{1}$$

公式(???)中F是3×3的矩阵，并且秩为2。F被称为**基础矩阵（Fundamental Matrix）**。F描述了x_1和x_2之间的关系。那么F是什么？这需要从F的推导说起。假设相机1的外参由矩

阵P_1描述，相机2的外参由矩阵P_2描述，并且：

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} I_{3x3} & 0_{3x1} \end{bmatrix} \quad (20)$$

$$P_2 = \begin{bmatrix} R_{3x3} & t_{3x1} \end{bmatrix} \quad (3)$$

K_1和K_2分别是两个相机的内参，则x_1和x_2之间存在如下关系：

$$\lambda_1 x_1 = K_1 P_1 X \quad (4)$$

$$\lambda_2 x_2 = K_2 P_2 X \quad (5)$$

公式(???)和公式(???)中有相同的X，因此若消去X，就可以将公式(???)和公式(???)合并为一个公式，这个公式就是(???)。**Multiple View Geometry in Computer Vision 2nd**第9章有详细的推导，本文就不再推导了。根据该书的推导，F的值为：

$$F = K_2^{-T}[t]_{\times} R K_1^{-1} \quad (???)$$

其中$[]_{\times}$是叉积算子。若$t = (t_x, t_y, t_z)$，则

$$[n]_{\times} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \quad (7)$$

初学SfM总是纠结F是什么？F是干什么的？F怎么求（稍后告诉你）？其实呢，这些纠结都是没有必要的。**F就是一个3×3的矩阵（且秩为2），并且F与x_1、x_2间的关系满足公式(???)**。记住这一点就够了啦！至于它怎么来的，留给给数学家去折腾，和我们没有一毛钱关系（我们是做软件的，是搞算法的！）。什么与我们有关系？**算F和用F**。

基础矩阵的计算

基础矩阵F的计算是一个非常简单、非常简单的问题。直接按照公式(???)算就ok啦，难不成你不会矩阵乘法吗？打住！现实世界如果如此美好，那就是天堂啦。因为现实中极少有机会直接用公式(???)计算F，甚至大多数情况，我们还指望从公式(???)中恢复K_1、$K2$、R和t。既然公式(???)不让用，那怎么办？只能用公式(???)。忘记公式(???)啦？再贴一下。

$$x_1^T F x_2 = 0$$

这是一个方程不是吗？如果x_1和x_2是已知的，那么这个方程只有一个未知数F。F是3×3的

矩阵，换言之，有9个未知数。9个未知数，就需要9个方程。一组x_1和x_2可以提供1个方程，故此需要9组对应的x_1和x_2。换言之，只需要找到3D空间中9个点，在图像上的9组对应2D点就可以解出这个方程。这个方程还是线性方程哦。还不懂？线性代数真的是还给老师啦。慢慢来展开看看，F有9个未知数，即：

$$F=\begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \tag{8}$$

然后有9个点$X_1, X_2, X_3, \ldots, X_9$，它们在两张相片上的像分别是$x_1^1$和$x_2^1$、$x_1^2$和$x_2^2$、$x_1^3$和$x_2^3$、……、$x_1^9$和$x_2^9$。$x_1^1$和$x_2^1$的坐标分别是$(p_x^1, p_y^1, 1)$和$(q_x^1, q_y^1, 1)$，$x_1^2$和$x_2^2$的坐标分别是$(p_x^2, p_y^2, 1)$和$(q_x^2, q_y^2, 1)$……以此类推，搞个表格：

序号	3D点	2D点	坐标值
1	X_1	x_1^1,x_2^1	$(p_x^1, p_y^1, 1)$,$(q_x^1, q_y^1, 1)$
2	X_2	x_1^2,x_2^2	$(p_x^2, p_y^2, 1)$,$(q_x^2, q_y^2, 1)$
…	…	…	…
9	X_9	x_1^9,x_2^9	$(p_x^9, p_y^9, 1)$,$(q_x^9, q_y^9, 1)$

然后每一组点均符合公式(???)，分别代入可得9个方程。我们将第1组点代入展开看看，其他点类似。如下：

$$\begin{bmatrix} p_x^1 & p_y^1 & 1 \end{bmatrix}\begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix}\begin{bmatrix} q_x^1 \\ q_y^1 \\ 1 \end{bmatrix}=0 \tag{9}$$

将公式(???)完全展开可得如下：

$$\begin{aligned} q_x^1 p_x^1 f_1 + q_x^1 p_y^1 f_2 + q_x^1 f_3 + q_y^1 p_x^1 f_4 \\ + q_y^1 p_y^1 f_5 + q_y^1 f_6 + p_x^1 f_7 + p_y^1 f_8 + f_9 = 0 \end{aligned} \tag{10}$$

上面这个展开式不需要什么技巧，很显然这是9个未知数的线性方程。于是这个方程还可以重写为如下形式：

$$\begin{bmatrix} q_x^1 p_x^1 & q_x^1 p_y^1 & q_x^1 & q_y^1 p_x^1 & q_y^1 p_y^1 & q_y^1 & p_x^1 & p_y^1 & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \end{bmatrix} = 0 \qquad (14)$$

写到这里我们发现公式(???)可以写作$Af = 0$的形式，这是一个齐次线性方程。那么**齐次线性方程有非0解的条件是什么呢**？A的秩等于8啊。所以陡然间发现，其实并不需要9组点，只需要8组即可。换言之，只需要这样的方程就可以求得F:

$$Af = \begin{bmatrix} q_x^1 p_x^1 & q_x^1 p_y^1 & q_x^1 & q_y^1 p_x^1 & q_y^1 p_y^1 & q_y^1 & p_x^1 & p_y^1 & 1 \\ q_x^2 p_x^2 & q_x^2 p_y^2 & q_x^2 & q_y^2 p_x^2 & q_y^2 p_y^2 & q_y^2 & p_x^2 & p_y^2 & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ q_x^8 p_x^8 & q_x^8 p_y^8 & q_x^8 & q_y^8 p_x^8 & q_y^8 p_y^8 & q_y^8 & p_x^8 & p_y^8 & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \end{bmatrix} \quad (???)$$
$$= 0$$

公式(???)怎么解呢？一般齐次线性方程组的解，可以用SVD来求（奇异值分解）。可以证明（见Multiple View Geometry in Computer Vision 2nd，A5.3）对A做奇异值分解，得到$A = U\Sigma V^T$，V^T的最后一行就是方程的解。写成python代码就是如下:

```
U,S,Vt=svd(A)
f=Vt[-1]
f=f/f[8]
F=f.reshape(3,3)
```

推到了那么多步骤，实际上只需要写这么几行代码，是不是很让人泄气？原来还是数学多于代码啊！其实不是这样的。SVD是一个很复杂的计算过程，上面这个代码，之所以简单，完全是因为现代程序语言已经将SVD做成函数啦。若要自己实现SVD的算法，那么代码就会很长。同时我们还会遇到一个更为头大的问题：找到8组点就可以求F，但不幸我们找到了9组，10组，11组，12组……甚至更多组点，怎么办？还这么求吗？显然不行！这个时候就

只能求最优解啦！计算机算法也就华丽登场啦！

F的最优解

现实中往往可以在张照片中找到8个以上的点对（如果少于8个，那就不用做啦），这种情况下，任意选择8个点对，都可以求得一个F值。怎么办呢？方法其实很简单。假设有9个点对，选前8个点，采用公式(???)求得一个F。那么对应第9个点对，必然有下面的公式：

$$e_9 = (x_2^9)^T F x_1^9 \tag{13}$$

同理对于前8个点，有相同方法计算的e_1，e_2，e_3，…，e_8(F是前8个点对精确求出的，所以e_1，e_2，e_3，…，e_8等于0)。则对应F总体的误差为：

$$e = \sqrt{e_1^2 + e_2^2 + e_3^2 + \ldots + e_9^2} \tag{43}$$

因此对应9个点对所求得的F是使得公式(???)最小的那个F。换言之，先用8个点求得初始值F，然后调整F的值使得，e最小即可。我们还可以发现e实际上就是$\|Af\|$（表示Af的模）。换言之，优化的目标就是：

$$\underset{f}{argmin}\|Af\| \tag{44}$$

这个优化目标恰好可以采用最小二乘法进行求解，这里还是最简单线性最小二乘。另一种求最优解的方法是用牛顿法。

如果误差函数不采用公式(???)，问题可能要复杂一些（Multiple View Geometry in Computer Vision 2nd中有各种误差函数的讨论），但是牛顿法一般都能解决。

无论用哪种误差函数，哪种求解方法，现在编程都很简单，只需要调用现成的函数库即可。 所以F最优解的求法是一个**需要了解，但无需操心**的问题。或者说，通过这一小节的描述，可以学习到**如何缩短理想和现实的距离。**

F计算的健壮性问题

上一节解决了点对多余8个的情况下如何计算F的问题，但是还有问题。假设我们找到了N个点对（$N > 8$），这里面很可能有几个点对找错了。现实中，找错啦，是比较常见的情况，全部找对那是碰到大奖啦。如果点对中有几个是错的，势必影响F的计算，这就是健壮性问题（Robust，也叫鲁棒性）。

怎么办呢？好办，把有问题的点对滤掉。关键问题在于，哪些是有问题的？好办，先求一个F出来，然后把点对带到公式(???)里面去，看误差有多大。误差太大的就舍弃，然后用剩下的点再重新算法。如此循环多次，最终就得到一个比较靠谱的值。这个算法思路称为RANSAC，许多需要健壮性的算法都在用。

F最终的计算

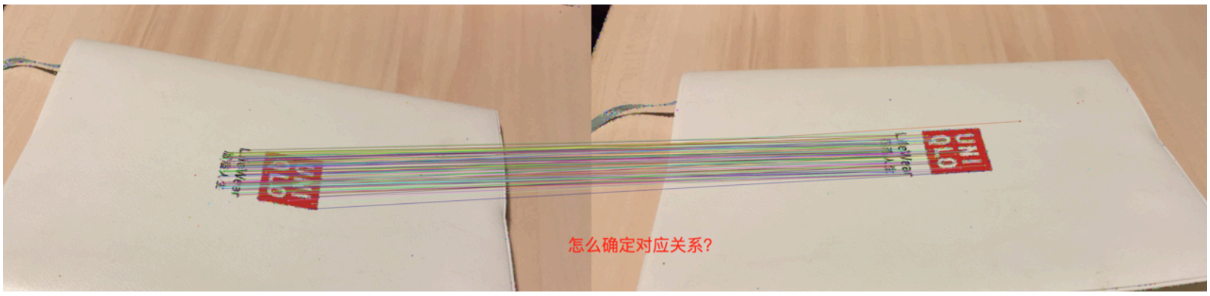
上面说的内容，在今天都是一堆废话。是属于你应该了解，但是了解了也没什么用的废话。因为现在F的计算只需要调用一个函数即可。以OpenCV为例：

```
Mat findFundamentalMat(InputArray points1, InputArray points2, int method=FM_RANSAC, double param1=3., double param2=0.99, OutputArray mask=noArray() )
```

可以解决你的所有问题，你要做的只是传递一个组对应点即可。原谅我说了那么多废话。

点对从哪里来？

如果F可以用OpenCV的函数搞定，但是函数还是需要传递点对啊。点对怎么算呢？换句话说，我怎么知道点a和点b是对应关系呢？



这是一个比较复杂的问题，复杂到可以研究几十年啦。最近有一个比较常规的方法，那就是求图像的SIFT特征点，然后求特征点之间的匹配。使用Opencv很容易计算SIFT特征以及对应的匹配。参考代码如下：

```
Mat im1=imread(argv[1]);
Mat im2=imread(argv[2]);
SIFT sift;
vector<KeyPoint> kp1,kp2;
Mat des1,des2;
```

```
    sift(im1,Mat(),kp1,des1);
    sift(im2,Mat(),kp2,des2);
    BFMatcher matcher(NORM_L2,true);
    vector<DMatch> matches;
    matcher.match(des1,des2,matches); //matches里面就是匹配结果
```

当然这个匹配结果还是比较粗糙的，有许多匹配不是正确的。当然findFundamentalMat函数具有一定的健壮性，也可以过滤掉大多数错误的匹配。

用F恢复相机位姿

F可以由相机的位姿信息导出（见公式(???)）。反过来，如果F已知（通过点对计算），那么也可以反向求出相机位姿。假设相机的K_1和K_2已知，那么实际上需要求的变量就是R和t（一般假定相机1的位姿是$P_1 = \begin{bmatrix} I & 0 \end{bmatrix}$）。依据Multiple View Geometry in Computer Vision 2nd的推导，可以按照如下的方式计算：

$$E = K_2^T F K_1 \tag{16}$$

$$E = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T \tag{17}$$

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{18}$$

$$\begin{aligned} P_2 = &\begin{bmatrix} UWV^T & +u_3 \end{bmatrix} or \begin{bmatrix} UWV^T & -u_3 \end{bmatrix} \\ or &\begin{bmatrix} UW^TV^T & +u_3 \end{bmatrix} or \begin{bmatrix} UW^TV^T & -u_3 \end{bmatrix} \end{aligned} \tag{19}$$

其中E被称为本质矩阵（Essential Matrix），公式(???)是E的奇异值分解，u_3是U的第三列，最终所得的P_2有4个可能解，但是只有1个解是正确的。回忆一下中学解方程，有时候方程有多个解，但是某些解要舍去。那么怎么找到P_2的正确解呢？需要把X求出来。

三角化

我们重写一下公式(???)和(???)，如下：

$$\lambda_1 x_1 = K_1 P_1 X = K_1 \begin{bmatrix} I & 0 \end{bmatrix} X \tag{20}$$

$$\lambda_2 x_2 = K_2 P_2 X = K_2 \begin{bmatrix} R & t \end{bmatrix} X \tag{21}$$

根据前面的推导，x_1、x_2、K_1、K_2、P_1、P_2已知，X就是未知量。这就是一个简单的解方程问题，其中λ_1、λ_2、X是未知量。那么一共有几个未知数呢？如果认为$X = (x, y, z, w)^T$，未知数个数就是6。把公式(???)和(???)做如下的变换：

$$\begin{bmatrix} K_1 P_1 X & -\lambda_1 x_1 & \lambda_2 * 0 \\ K_2 P_2 X & -\lambda_1 * 0 & \lambda_2 * x_2 \end{bmatrix} = \begin{bmatrix} K_1 P_1 & -x_1 & 0 \\ K_2 P_2 & 0 & -x_2 \end{bmatrix} \begin{bmatrix} X \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = A\hat{X} \\ = 0 \tag{22}$$

方程$A\hat{X} = 0$有非零解的条件是A的行秩小于A的列秩，即如果A是$m \times n$的矩阵，$m < n$说明方程个数小于未知数个数（行秩为m，列秩为n）。但非常不幸，公式(???)中的A是6×6的矩阵，而$\hat{X}$有6个未知数，若如此$\hat{X}$就只能是全0解啦。真的吗？显然不是！如果我们假设$X = (x, y, z, 1)^T$，那么方程就只有5个未知数。这种假设合理吗？合理。因为：

$$\frac{A\hat{X}}{w} = A \begin{bmatrix} x \\ y \\ z \\ 1 \\ \lambda_1/w \\ \lambda_2/w \end{bmatrix} = 0 \tag{23}$$

用奇异值分解就可以解这个方程：

```
U,S,Vt=svd(A)
X=Vt[-1,:4]
X=X/X[3]
```

当然如果对解方程没有自信，直接用OpenCV的函数就可以了：

```
void triangulatePoints(InputArrayprojMatr1,InputArrayprojMatr2,InputArrayprojPoints1,InputArray projPoints2, OutputArray points4D)
```

3视图怎么办？

至此我们已经搞定了两视图的稀疏重建，但如果视图有3个怎么办呢？也就是，如果K_3已知，我们需要求出P_3。可能很多人想当然的认为可以将视图1和视图3看做两视图问题，然后用前面的算法就可以求出P_3。对不起，这是不行的。为什么呢？原理如下：

$$\lambda_3 x_3 = K_3 P_3 X = K_3 P_3 H H^{-1} X = K_3 \bar{P}_3 \bar{X} \qquad (24)$$

因此如果用视图1和视图3做一次两视图的SfM，求得的可能是正确的P_3，也可能是错误的$\bar{P}_3$。总之，所有视图必须在同一个基准上面搞定（也就是上面的H必须是同一个）。怎么保证这一点呢？

如果x_1、x_2、x_3对应于X，那么利用x_3和X能够重建P_3吗？完全没有问题，因为：

$$\lambda_3 x_3 = K_3 P_3 X \qquad (25)$$

又是一个解方程的问题，这是一个$AX = b$形式的非齐次线性方程，所解起来比较容易。P_3有12个未知数，所以理论上需要4组这样的方程。方程凑齐，求一个逆即可。如果方程数超过12个，那就用最小二乘法，或者牛顿法。但最后我要告诉你，其实P_3只有6个未知数（这又是一个很长的故事），所以用不了那么多方程。Anyway，OpenCV提供了一个函数来搞定这一切：

```
void solvePnPRansac(InputArray objectPoints, InputArray imagePoints, InputArray cameraMatrix, InputArray distCoeffs, OutputArray rvec, OutputArray tvec, bool useExtrinsicGuess=false, int iterationsCount=100, float reprojectionError=8.0, int minInliersCount=100, OutputArray inliers=noArray(), int flags=ITERATIVE
)
```

所以3视图或者N视图问题最关键的步骤是**求2D点的匹配**，只要在已知视图中找到足够多的对应点，那么就可以求出未知视图的相机位姿。相机位姿求得之后，可以对未知点重建X。

算法总结

假设输入有N个视图，对其进行SfM算法的基本步骤是：

1. 用某一个特征点提取算法对N个视图进行计算（通常选用SIFT）。
2. 对N个视图的特征点两两求匹配，计算点在视图中的轨迹。例如点X_i在视图1、3、5、7中可见，则记为$X_i \mapsto x_1^i \mapsto x_3^i \mapsto x_5^i \mapsto x_7^i$。
3. 选择匹配点最多的两个视图作为初始视图，记为I_1和I_2，求基础矩阵F，求P_1和P_2，对视图中的匹配点重建求得集X。
4. 选择下一个视图I_3（通常选择与初始视图匹配点较多的），对于$X_j \in X$，若X_j在I_3中可见，则利用X_j在I_3中的对应点x_3^j，求P_3，重建I_3中的其他点，并加入到点集X中。
5. 重复第4步，直到全部视图均计算完毕。

Bundle Adjustment

这是算法的最后一步，当所有视图计算完毕之后，有必要对计算结果进行优化。优化的方法是将计算所得点X_i重新投影到视图上，然后计算重投影后的点与视图上对应的坐标差，将坐标差的平方和作为优化的目标。假设n个视图，重建m个三维点，则K_i、P_i分别对应视图i的内参和外参，点X_j在视图i上的投影是x_i^j，则优化的目标函数如下：

$$\underset{P_i,X_j}{argmin}\sum_{i=1}^{n}\sum_{j=1}^{m}\|K_iP_iX_j - x_i^j\| \qquad (26)$$

Bundle Adjustment的优化参数可以包括K_i、P_i、X_j和x_i^j。当然一般只优化P_i和X_j。

Bundle Adjustment的程序超级难写，SBA(Sparse Bundle Adjustment)是解决该问题的C语言库。单纯写这一个库都够发表一篇不错的论文啦。

One More Thing

求P_2的时候留了一个尾巴，P_2有4个可能值怎么选呢？

1. 对于任意备选矩阵P_2，用x_1、x_2、P_1、P_2三角化（算法详见三角化），重建三维点X。
2. 用P_1和P_2将重构的三维点X，投影到二维，记为$\bar{x}_1 = K_1P_1X$、$\bar{x}_2 = K_2P_2X$。
3. $\bar{x}_1$和$\bar{x}_2$中点的齐次坐标形式为$(a_1, b_1, c_1)^T$和$(a_2, b_2, c_2)^T$，统计$\bar{x}_1$和$\bar{x}_2$中$c_1 > 0$且$c_2 > 0$的顶点数量记为S。
4. 若P_2使得S的值最大，则P_2就是最佳的矩阵。

总结

学完这些东西，大概掌握了十年前相关领域的技术。