

作者: 293311923@qq.com
URL: <https://medium.com/@kazuyukimiyazawa/revisit-structure-from-motion-revisite...>

Basic Pipeline

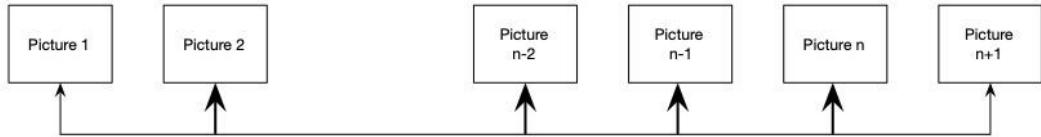
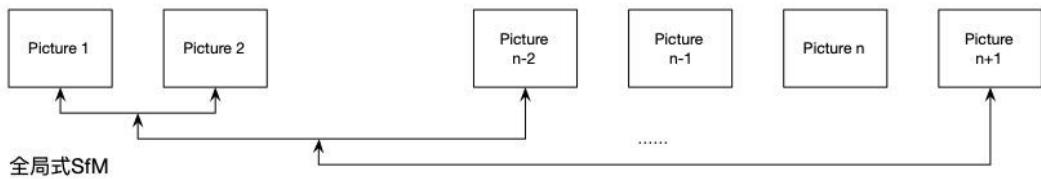
我们在之前文档“相机模型定义”中简述过这个概念：**一个SfM的basic pipeline可以描述为：**对每张图片检测特征点 (feature point)，对每对图片中的特征点进行匹配，只保留满足几何约束的匹配，最后执行一个迭代式的、鲁棒的SfM方法来恢复摄像机的内参 (intrinsic parameter) 和外参(extrinsic parameter)。

同时，我们描述了：Structure from Motion (SfM) 是一个估计相机参数及三维点位置的问题。**恢复相机的内参和外参是SfM算法的主要目标。**

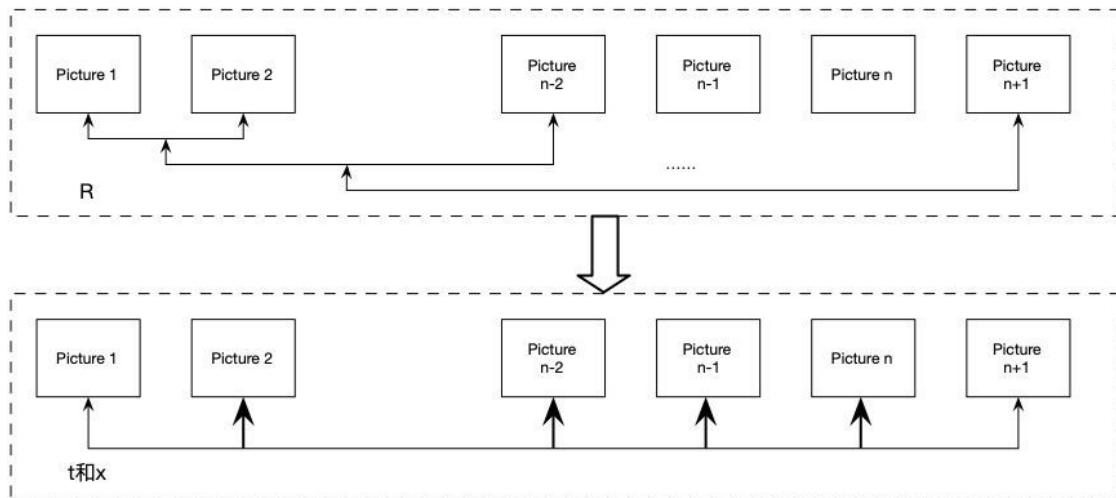
基于Basic Pipeline这个基本流程，有不同研究者描述了具体的SfM算法，**按照各类文献中算法的大致思路，SfM可以分为：**

- 增量式 (incremental/sequential)
- 全局式 (global)
- 混合式 (hybrid)
- 层次式 (hierarchical)
- 基于语义的(Semantic)

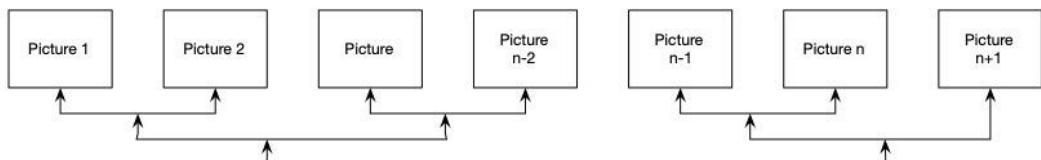
增量式SfM



混合式SfM



层次式SfM



前面四种SfM算法具体区别体现在图片的匹配方式和顺序上。基于语义的则是指引入物体的语义标签在SfM特征点匹配的基础之上减小匹配误差

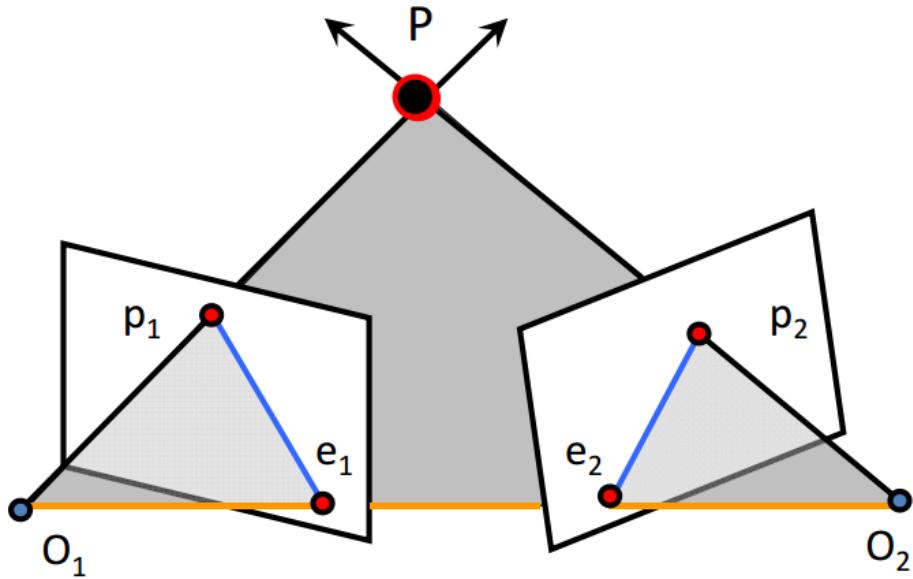
同时，在SfM算法中涉及到三个重要的矩阵（对极几何\计算机视觉）概念：

- 基本矩阵(Fundamental matrix)
- 本质矩阵(Essential Matrix)
- 单应矩阵(Homography matrix)

对极几何

下图就表示一个对极几何中的对极几何平面：

Epipolar geometry



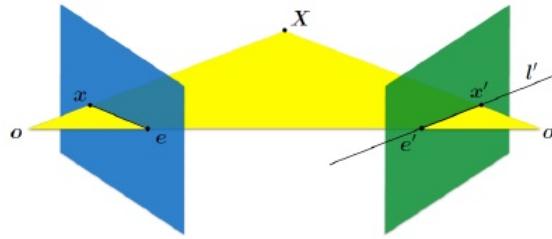
该图反映了摄像机从不同角度看物体 x 会形成对极平面，在这个平面上的多视图几何关系即是对极几何关系，因此SfM过程也是建立于对极几何关系之上，通过本质矩阵、基本矩阵、单应矩阵来表达坐标系因为相机运动所产生的在对极平面上的变换关系

SfM的核心就是从运动中恢复空间位置关系，因此**本质矩阵、基本矩阵，是对这种运动关系的关键映射对象。**

本质矩阵与基本矩阵

本质矩阵 E (Essential Matrix) 是一个包含了对极几何(epipolar geometry)关系的 3×3 大小的矩阵，给出一幅图像中的一个点，乘以本质矩阵就可以得到在第二图像中对应的极线 l' (epipolar line)。

$$Ex = l'$$



如果点在一条直线上满足 $ax + by + c = 0$

直线参数写成向量形式 $l = [a \ b \ c]^T$ 得到 $x^T l = 0$ ，由上图可以看到 x' 在直线 l' 上，则 $x'^T l' = 0$ ，带入 $Ex = l'$ 得到极限约束：

$$x'^T Ex = 0$$

这个式子里的 x 是图像点在归一化坐标 (normalized coordinates) 下表达的，归一化坐标就是这个图像像素点让 $z=1$ 转换到了相机坐标系下：

$$x = K^{-1} x \quad x' = K'^{-1} x'$$

其中 x 是图像坐标系下点的坐标， x 是相机坐标系下点的坐标， K 是相机内参矩阵。

把这两个点的坐标带入到本质矩阵表示的极线约束中得到：

$$x'^T K'^{-T} E K^{-1} x = 0$$

$$x'^T (K'^{-T} E K^{-1}) x = 0$$

$$x'^T F x = 0$$

这里的 F 为基本矩阵 (Fundamental matrix)，上式为基本矩阵表示的极线约束，所以本质矩阵和基本矩阵的区别是本质矩阵是使用归一化相机坐标系下的点表示极线约束，基本矩阵是使用图像坐标系下的点表示极线约束。

从上述过程中不难理解，基本矩阵(Fundamental Matrix)与本质矩阵(Essential Matrix)的关键区别：

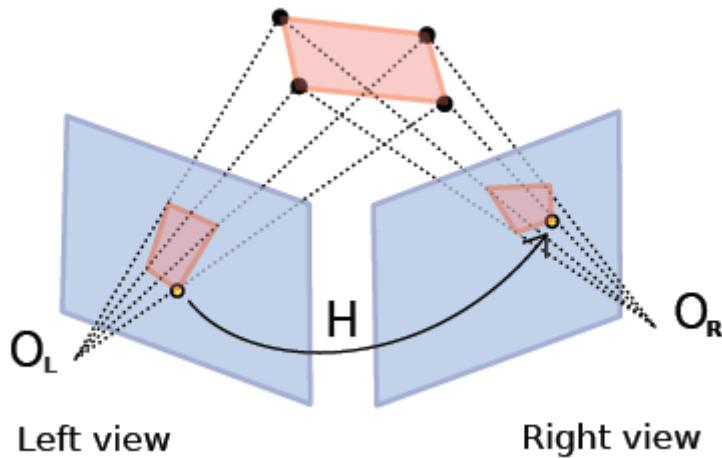
- 本质矩阵：空间中同一点在**2个相机光心坐标系中不同(三维)坐标**之间的关系
- 基本矩阵：空间中同一点在**2张照片(2个成像面)上的不同(二维)坐标**之间的关系

下面我们会针对不同类别的SfM算法，基于 Basic Pipeline，依据参考文献中的算法描述 (pseudo-code) 进行基本的思想描述

单应矩阵

单应矩阵是图像处理中描述同一个物体在世界坐标系和像素坐标系之间的位置映射关系。对应的变换矩阵称为单应性矩阵。

下面是一组二维单应性变换示例：



单应性矩阵主要用来解决两个问题：

- 一是表述真实世界中一个平面与对应它图像的透视变换
- 二是从通过透视变换实现图像从一种视图变换到另外一种视图

通俗的说，就是确定**两个不同平面之间**对应的空间变换关系，如上图中，左图Ql上的红色区域图像变换到右图Qr上的红色区域图像的变换矩阵，就可以说是这两个图像所对应的单应性矩阵。

在一个三维空间任意两个面上，我们可以通过取对应的两个图像四个角点构成的对应两组向量计算如上图中的单应性矩阵H，在编程中也可以利用opencv中的findHomography函数确定两个图像之间的单应性矩阵。

例如：

```
// 原图像平面四点坐标src_corners[0] = Point(0, 0);  
src_corners[1] = Point(replaceImg.cols, 0);  
src_corners[2] = Point(0, replaceImg.rows);  
src_corners[3] = Point(replaceImg.cols, replaceImg.rows);  
  
// 目标平面四个角坐标dst_corners[0] = Point(218, 50);  
dst_corners[1] = Point(557, 30);  
dst_corners[2] = Point(273, 236);  
dst_corners[3] = Point(559, 215);  
  
// 计算单应性矩阵与透视变换  
Mat h = findHomography(src_corners, dst_corners);
```

单应性矩阵可以应用的领域：

- 用来解决拍照时候图像扭曲问题
- 计算机图形学的应用场景：纹理渲染与计算平面阴影
- 用来实现图像拼接时候解决对齐问题

incremental SfM

Incremental SfM (增量式SfM) 方法是目前最广为使用的方法。

增量式SfM主要涉及了**两种核心的算法**，在这里主要以增量式为例介绍算法细节，其他方法比较而言，涉及到的核心内容都是以之前文档所述的**basical pipeline**为基础并进行了改进。(可以参考[Johannes L. Schonberger and Jan-Michael Frahm, "Structure-from-Motion Revisited." CVPR2016这篇文章\(点击可以下载\)](#))：

接下来我们给出对应文献中的算法描述，然后进行翻译和说明：

Algorithm 1 Computation of geometry-consistent pairwise correspondences

Require: image set

Ensure: pairwise point correspondences that are geometrically consistent

Compute putative matches:

Detect features in each image and build their descriptor

Match descriptors (brute force or approximate nearest neighbor)

Filter geometric-consistent matches:

Estimate fundamental matrix F

Estimate homography matrix H

主旨思想：输入图像集(image set, which reflects the motion in CV)，通过成对匹配照片，寻找其满足几何一致的成对对应关系(pairwise correspondences)完成特征匹配过程。

对经推定的成对图像进行如下计算：

- 在每个图像中进行特征提取，并构建图像的描述器(descriptor)
- 利用构建的描述器将两两相邻的图像进行特征匹配

构建滤波器,依据下列两个关键矩阵，进行几何一致性匹配：

- 基本矩阵 (fundamental matrix) F
 - 单应矩阵 (homography matrix) H
-

Algorithm 2 Incremental Structure from Motion

Require: internal camera calibration (matrix K, possibly from EXIF data)

Require: pairwise geometry consistent point correspondences

Ensure: 3D point cloud

Ensure: camera poses
Compute correspondence tracks t
Compute connectivity graph G (1 node per view, 1 edge when enough matches)
Pick an edge e in G with sufficient baseline (compare F and H)
Robustly estimate essential matrix from images of e
Triangulate te , which provides an initial reconstruction
Contract edge e
While G contains an edge **do**
 Pick edge e in G that maximizes track $(e)\{3D\ points\}$
 Robustly estimate pose(external orientation/resection)
 Triangulate new tracks
 Contract edge e
 Perform bundle adjustment
End while

主旨思想：输入经过校准“标定过程”的摄像机（见文档“SfM核心思想与初始化一个相机模型”，输入参数包括了来自EXIF数据的矩阵K，注：EXIF是读入数据图像的头信息，通过EXIF可以读取数码相机产生的元数据。另：可用函数exif_read_data -- 从JPEG或TIFF文件中读取EXIF头信息），以及呈几何对偶（后续补充代数学中的对偶几何概念解释）一致性的匹配特征点，重建相机位姿(poses)与三维位置信息（3D稀疏点云，3D point cloud）

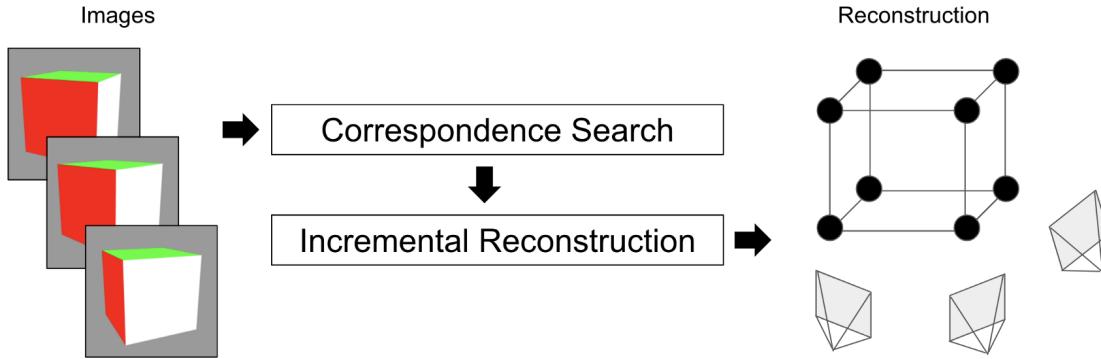
我们需要如下计算过程：

计算成对对应关系轨迹 t
建立连通图 G （对满足几何一致性的匹配点建立节点和边）
通过对本质矩阵和单应性矩阵进行比较，将一条基本边 e 选入图 G
用鲁棒性的方法通过 e 恢复图片的本质矩阵
三角剖分 te ，用于提供一个初始化的重建
逐步向连通图 G 中添加边 e

对于如何向图 G 中添加边 e ，我们进行 do...while 循环：

选取一个最大化轨迹 $e\{一个3D点\}$ 添加到图 G
用鲁棒性的方法估测相机位姿
剖分新轨迹
连接边 e
进行最小化重投影误差过程（slam 中的 bundle adjustment 过程）
结束该循环

上述两个算法就对应了下图中的 Correspondence Search (特征匹配) 和 Incremental Reconstruction (增量重建) 过程



global SfM

Require: internal camera calibration (possibly from EXIF data)

Require: pairwise geometry consistent point correspondences

Ensure: 3D point cloud

Ensure: camera poses

compute relative pairwise rotations

detect and remove false relative pairwise rotations

using composition error of triplet of relative rotations

compute the global rotation

using a dense least square and approximated rotations

compute relative translations

using triplet of views for stability and colinear motion support

compute the global translation

integration of the relative translation directions using a ℓ_∞ method.

final structure and motion

link tracks validated per triplets and compute global structure by triangulation,

refine estimated parameter in a 2 step Bundle Adjustment

refine structure and translations

refine structure and camera parameters (rotations, translations).

全局式和增量式的比较

增量式:

优--对错误的匹配点有较强鲁棒性，总体精度更高

劣--运行时间长， **drift**: error随着camera registration(相机校准)逐步积累

全局式:

优--避免了drift的问题 (更反映了图像的全局性), 速度快 (只需解决两个global synchronization(global SfM算法中的全局同步操作)+一次BA
(光束平差法))

劣--对错误的匹配点鲁棒性较差, 且错误难以修正 (error会沿着pipeline累积)

光束法平差 (BA)

光束法平差由Bundle Adjustment(BA)译得, 即**对所有光束进行平差**。有两层释义:

1. 对场景中任意三维点P, 由从每个视图所对应的摄像机的光心发射出来并经过图像中P对应的像素后的光线, 都将交于P这一点, 对于所有三维点, 则形成相当多的光束(bundle)
2. 实际过程中由于噪声等存在, 每条光线几乎不可能汇聚与一点, 因此在求解过程中, 需要不断对待求信息进行调整 (adjustment), 来使得最终光线能交于点P。

对n帧, 含k个特征点的**代价函数** (该优化下的损失函数) 如下:

$$\underset{P_i, M_i}{\operatorname{argmin}} \sum_{i=1}^k \sum_{j=1}^n v_{ij} d(Q(P_j, M_i), m_{ij})^2$$

光束法平差一般在各种重建算法的最后一步使用。这种优化方法的最大特点是**可以处理数据丢失情况并提供真正的最大似然估计**。

各参数含义如下:

- k为三维空间点个数;
- n为成像平面个数;
- mij表示第i个三维点在第j个成像平面对应的特征点坐标;
- vij表示点 i 在成像平面 j 上是否有投影, 如果有 vij=1, 否则vij=0;
- Pi表示每个成像平面对应的外参数向量;
- Mi表示每个三维点的坐标向量;
- Q(aj,bi)是重投影函数, 将三维点 Mi映射到成像平面;
- d(x,y)为距离度量函数, 一般为欧式距离函数。

需要优化求解的参数: 相机姿态/位置/世界点 X_{world} 坐标。

这三者决定的就是相机投影过程中的摄影光束, 所以我们在**调整的就是光束, 即从像平面创建光束, 并不断优化寻找光束对应的出发点 (相机三维坐标)**, 来使每个像素点触发的光线汇聚有一点P。这也是名字光束法平差的由来, 非常生动直观。

由于场景中特征点往往较多, 该问题是一个巨大的高维**非线性优化**问题。对上述函数进行求解, 这是光束平差法的核心内容。

解法: 一阶方法, 即对问题的目标函数进行泰勒一阶展开后进行迭代求解的方法。对雅克比矩阵使用梯度下降法是一阶方法之一。二阶优化方法, 即会将目标函数展开至泰勒二阶项然后进行优化求解, 有牛顿法、LM方法、LBFGS方法等。LM方法易于实现, 且效果较好, 应用广泛。这里不展开。

下面我们用BA求解pnp（相机位姿估计，通过给定若干图像，估计其中相机运动）问题，采用pnp问题中的经典开源图片：

待估测对应点深度的图片：



通过光束法最小化n个三维坐标点在两个图上投影的误差：

```

#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <Eigen/Core>
#include <Eigen/Dense>
#include "sophus/se3.h"

using namespace std;
using namespace Eigen;
using namespace cv;

typedef vector<Vector3d, Eigen::aligned_allocator<Vector3d>> VecVector3d;
typedef vector<Vector2d, Eigen::aligned_allocator<Vector3d>> VecVector2d;
typedef Matrix<double, 6, 1> Vector6d;

double fx = 520.9, fy = 521.0, cx = 325.1, cy = 249.7;

void getdata (
    VecVector2d& p2d,
    VecVector3d& p3d
);

int main ( int argc, char** argv )
{
    //生成3D点和像素坐标点
    VecVector2d p2d;
    VecVector3d p3d;
    getdata(p2d, p3d);

    //高斯牛顿迭代
    int iterations = 100;
    double cost = 0, lastCost = 0;
    int nPoints = p3d.size();
    cout << "points: " << nPoints << endl;

    Sophus::SE3 T_esti; // estimated pose 相机位姿 $R_t$ 李代数形式

    for (int iter = 0; iter < iterations; iter++) {
        Matrix<double, 6, 6> H = Matrix<double, 6, 6>::Zero();
        Vector6d b = Vector6d::Zero();

        cost = 0;
        // compute cost

```

```

for (int i = 0; i < nPoints; i++) {
    Vector2d p2 = p2d[i];
    Vector3d p3 = p3d[i]; //第i个数据

    Vector3d P = T_esti * p3;
    double x = P[0];
    double y = P[1];
    double z = P[2];

    Vector2d p2_ = { fx * (x/z) + cx, fy * (y/z) + cy };
    Vector2d e = p2 - p2_; //误差error = 观测值 - 估计值
    cost += (e[0]*e[0]+e[1]*e[1]);

    // compute jacobian
    Matrix<double, 2, 6> J;
    J(0, 0) = -(fx/z);
    J(0, 1) = 0;
    J(0, 2) = (fx*x/(z*z));
    J(0, 3) = (fx*x*y/(z*z));![d9e1a8843760d584a85dcc176be22c1e.png]
    (en-resource://database/4587:1)

    J(0, 4) = -(fx*x*x/(z*z)+fx);
    J(0, 5) = (fx*y/z);
    J(1, 0) = 0;
    J(1, 1) = -(fy/z);
    J(1, 2) = (fy*y/(z*z));
    J(1, 3) = (fy*y*y/(z*z)+fy);
    J(1, 4) = -(fy*x*y/(z*z));
    J(1, 5) = -(fy*x/z);

    H += J.transpose() * J;
    b += -J.transpose() * e;
}

// solve dx
Vector6d dx;
dx = H.ldlt().solve(b);

if (isnan(dx[0])) {
    cout << "result is nan!" << endl;
    break;
}

if (iter > 0 && cost >= lastCost) {

```

```

    // 误差增长了，说明近似的不够好
    cout << "cost: " << cost << ", last cost: " << lastCost << endl;
    break;
}

// 更新估计位姿
T_esti = Sophus::SE3::exp(dx)*T_esti;
lastCost = cost;
cout << "iteration " << iter << " cost=" << cout.precision(12) <<
cost << endl;
}

cout << "estimated pose: \n" << T_esti.matrix() << endl;
return 0;
}

void getdata (
    VecVector2d& p2d,
    VecVector3d& p3d
) {
    //-- 读取图像
    Mat img_1 = imread ( "./data/1.png", CV_LOAD_IMAGE_COLOR );
    Mat img_2 = imread ( "./data/2.png", CV_LOAD_IMAGE_COLOR );

    //初始化
    vector<KeyPoint> keypoints_1;
    vector<KeyPoint> keypoints_2;
    vector< DMatch > matches;
    Mat descriptors_1, descriptors_2;
    // used in OpenCV3
    Ptr<FeatureDetector> detector = ORB::create();
    Ptr<DescriptorExtractor> descriptor = ORB::create();
    // use this if you are in OpenCV2
    // Ptr<FeatureDetector> detector = FeatureDetector::create ( "ORB" );
    // Ptr<DescriptorExtractor> descriptor = DescriptorExtractor::create (
    "ORB" );
    Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create (
    "BruteForce-Hamming" );

    //检测 Oriented FAST 角点位置
    detector->detect ( img_1, keypoints_1 );
    detector->detect ( img_2, keypoints_2 );
}

```

```

//根据角点位置计算 BRIEF 描述子
descriptor->compute ( img_1, keypoints_1, descriptors_1 );
descriptor->compute ( img_2, keypoints_2, descriptors_2 );

//对两幅图像中的BRIEF描述子进行匹配，使用 Hamming 距离
vector<DMatch> match;
// BFMatcher matcher ( NORM_HAMMING );
matcher->match ( descriptors_1, descriptors_2, match );

//匹配点对筛选
double min_dist=10000, max_dist=0;

//找出所有匹配之间的最小距离和最大距离，即是最相似的和最不相似的两组点之间的距离
for ( int i = 0; i < descriptors_1.rows; i++ )
{
    double dist = match[i].distance;
    if ( dist < min_dist ) min_dist = dist;
    if ( dist > max_dist ) max_dist = dist;
}

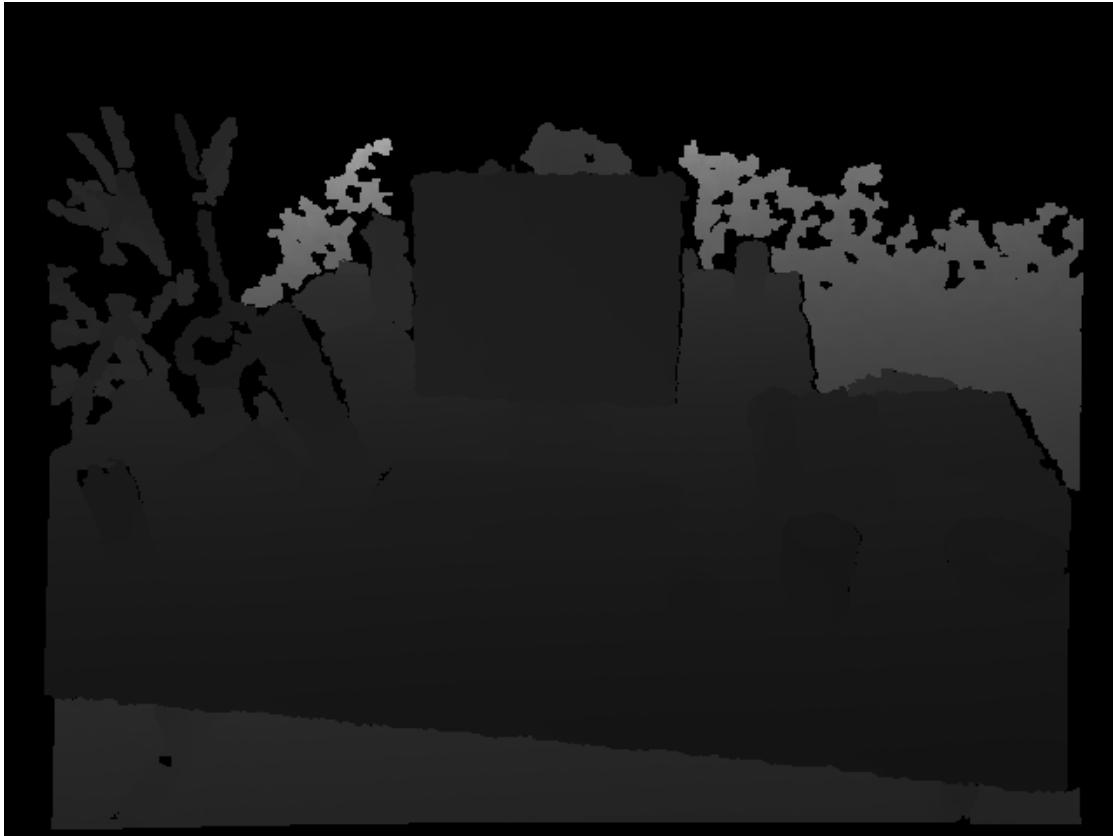
printf ( "-- Max dist : %f \n", max_dist );
printf ( "-- Min dist : %f \n", min_dist );

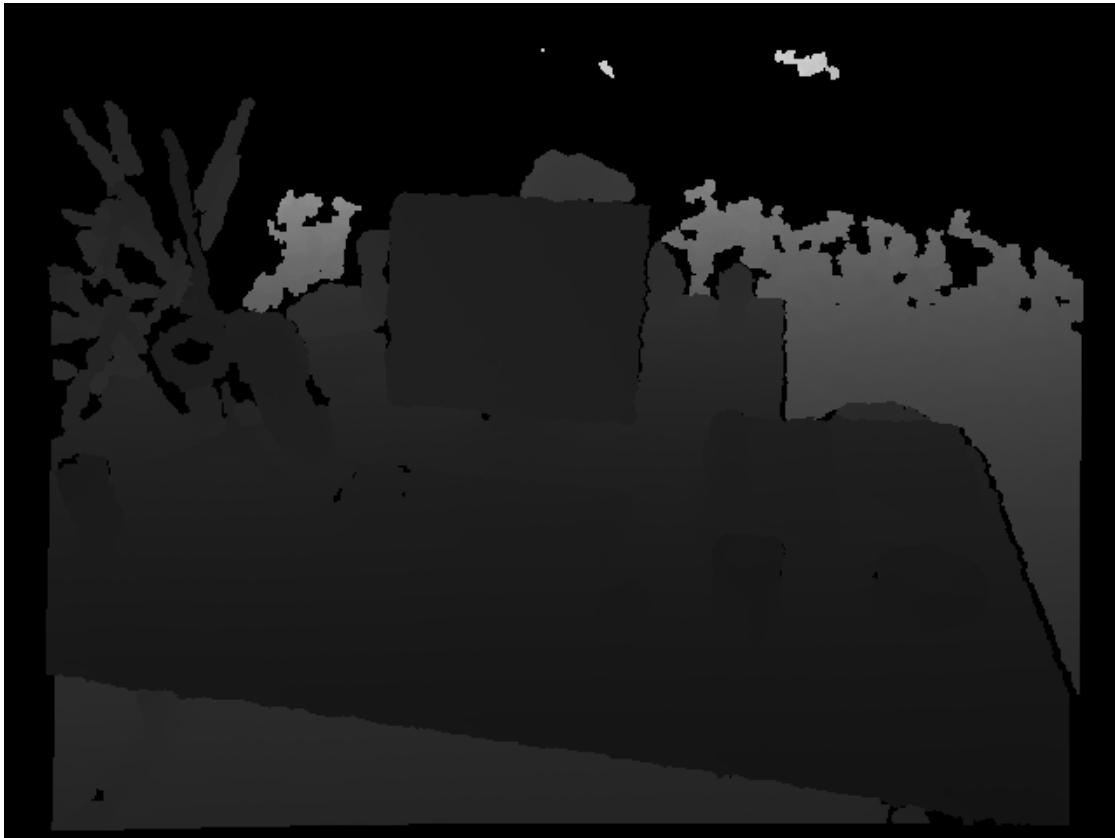
//当描述子之间的距离大于两倍的最小距离时，即认为匹配有误。但有时候最小距离会非常小，设置一个经验值30作为下限。
for ( int i = 0; i < descriptors_1.rows; i++ )
{
    if ( match[i].distance <= max ( 2*min_dist, 30.0 ) )
    {
        matches.push_back ( match[i] );
    }
}
cout<<"一共找到了"<<matches.size() <<"组匹配点"<<endl;
Mat d1 = imread ( "./data/1_depth.png", CV_LOAD_IMAGE_UNCHANGED );
for ( DMatch m:matches )
{
    ushort d = d1.ptr<unsigned short> ( int ( keypoints_1[m.queryIdx].pt.y
) ) [ int ( keypoints_1[m.queryIdx].pt.x ) ];
    if ( d == 0 ) // bad depth
        continue;
    float z = d/5000.0;
    float x = ( keypoints_1[m.queryIdx].pt.x - cx )* z / fx;
    float y = ( keypoints_1[m.queryIdx].pt.y - cx ) * z / fy;
}

```

```
    p3d.push_back ( Vector3d( x, y, z ) );
    p2d.push_back ( Vector2d( keypoints_2[m.trainIdx].pt.x,
keypoints_2[m.trainIdx].pt.y ) );
}
}
```

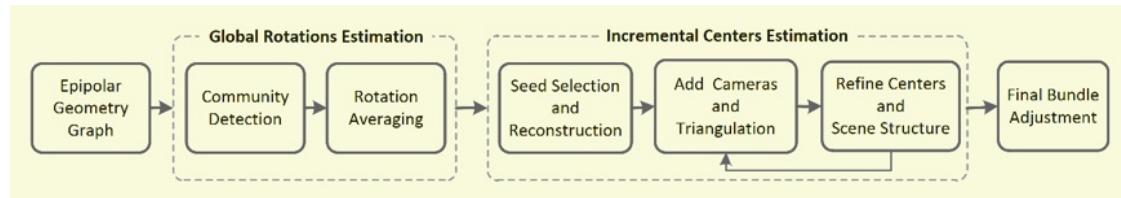
算法返回对应点深度（到P点距离）：





混合式与层次式

混合式SfM在一定程度上综合了incremental SfM和global SfM各自的优点。它的整个pipeline可以概括为**全局估计摄像机旋转矩阵，增量估计摄像机中心**。如下图：



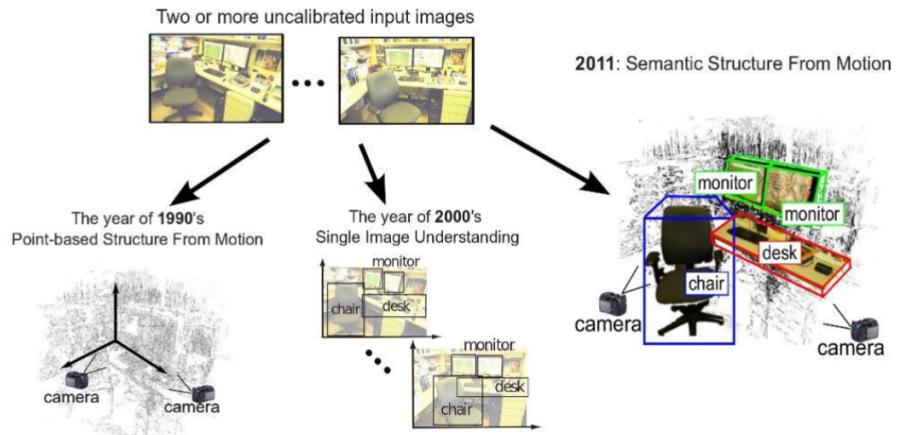
层次式SfM在特征点匹配和增量重建上和增量式大同小异，但其在执行顺序上采用了层次式的聚类策略 (clustering)。其先生成一棵聚类二叉树 (binary cluster tree)，然后算法自底向上进行处理:算法的每次迭代合并具有最小距离的两个clusters，每个cluster可以是一张图片，也可以是一个合并之后的cluster。组成cluster的两个视图需要满足两个要求：足够多的匹配点和足够宽的基线。由于合并的两个clusters的坐标系不同，因此合并的时候需要做相似变换。

语义

正所谓“横看成岭侧成峰，远近高低各不同”，研究者们在对SfM的逐渐研究过程中，发现了不仅仅可以对点进行匹配来恢复相机外参，引用语义标签(Semantic labels)更可以帮助

助改善算法:

Semantic SfM



其中，语义用SfM中的作用有三种：

- 用语义物体标签改善SfM算法
- 通过SfM重建三维模型，帮助标定语义标签，以让人从三维模型中获取更多信息
- 要求更高的方案，结合以上两者进行协同性的优化

具体的语义应用举例：

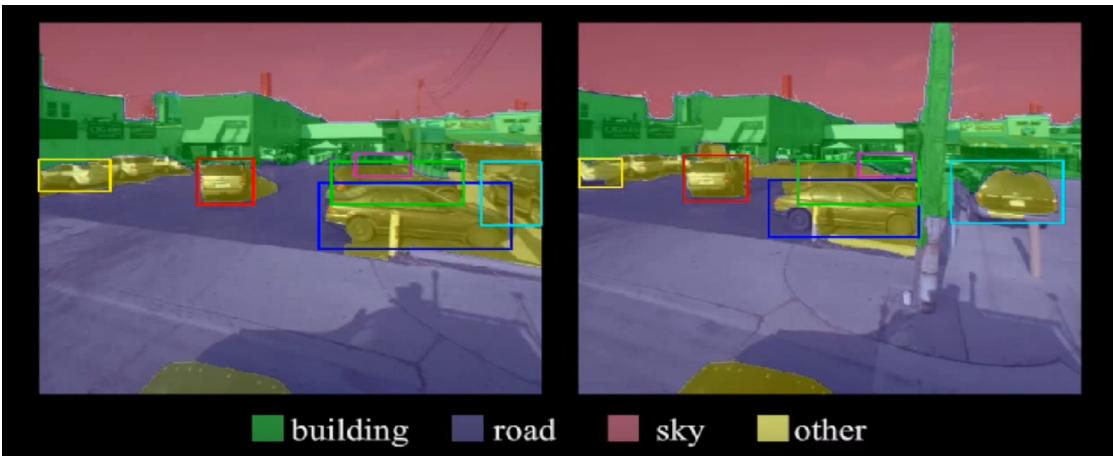
1，在SfM中通过语义实现对应场景的分割



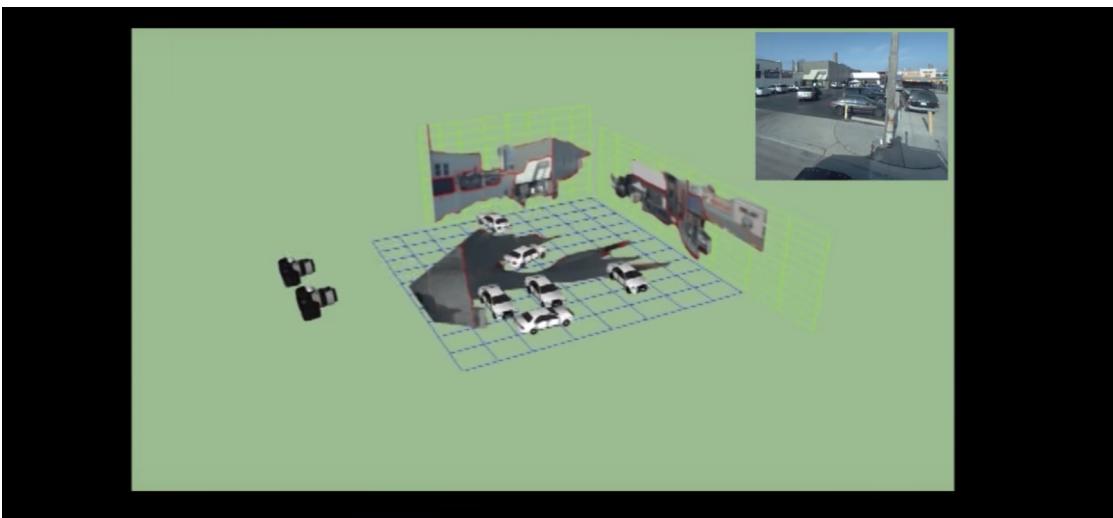
2, 通过语义因子检测出对应的模型, 以帮助特征匹配与相机标定。



3, 将语义标签指派到对应的分割场景中



4, 利用场景分割的位置关系矫正恢复后的三维模型位置



References

- [1] Seitz S M, Szeliski R, Snavely N. Photo Tourism: Exploring Photo Collections in 3D[J]. Acm Transactions on Graphics, 2006, 25(3):835-846.
- [2] Agarwal S, Snavely N, Simon I, et al. Building Rome in a day[J]. Communications of the Acm, 2011, 54(10):105-112.

- [3] Snavely K N. Scene reconstruction and visualization from internet photo collections[M]. University of Washington, 2008.
- [4] Frahm J M, Fite-Georgel P, Gallup D, et al. Building Rome on a cloudless day[C]// European Conference on Computer Vision. Springer-Verlag, 2010:368-381.
- [5] Wu C. Towards Linear-Time Incremental Structure from Motion[C]// International Conference on 3dtv-Conference. IEEE, 2013:127-134.
- [6] Moulon P, Monasse P, Marlet R. Adaptive structure from motion with a contrario, model estimation[C]// Asian Conference on Computer Vision. Springer Berlin Heidelberg, 2012:257-270.
- [7] Schönberger J L, Frahm J M. Structure-from-Motion Revisited[C]// Computer Vision and Pattern Recognition. IEEE, 2016.
- [8] Lowe D G, Lowe D G. Distinctive Image Features from Scale-Invariant Keypoints[J]. International Journal of Computer Vision, 2004, 60(2):91-110.
- [9] Fischler M A, Bolles R C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography[J]. Readings in Computer Vision, 1987:726-740.
- [10] Hartley R, Zisserman A. Multiple view geometry in computer vision[J]. Kybernetes, 2003, 30(9/10):1865 - 1872.
- [11] Beder C, Steffen R. Determining an Initial Image Pair for Fixing the Scale of a 3D Reconstruction from an Image Sequence[J]. 2006, 4174:657-666.
- [12] Triggs B, McLauchlan P F, Hartley R I, et al. Bundle Adjustment — A Modern Synthesis[C]// International Workshop on Vision Algorithms: Theory and Practice. Springer-Verlag, 1999:298-372.
- [13] Wu C, Agarwal S, Curless B, et al. Multicore bundle adjustment[C]// Computer Vision and Pattern Recognition. IEEE, 2011:3057-3064.
- [14] Moisan L, Moulon P, Monasse P. Automatic Homographic Registration of a Pair of Images, with A Contrario Elimination of Outliers[J]. Image Processing on Line, 2012, 2:329-352.
- [15] H Cui , X Gao, S Shen, Z Hu. HSfM: Hybrid Structure-from-Motion[C]// Computer Vision and Pattern Recognition. IEEE, 2017.
- [16] Zhu S, Shen T, Zhou L, et al. Parallel Structure from Motion from Local Increment to Global Averaging[J]. 2017.
- [17] Farenzena M, Fusiello A, Gherardi R. Structure-and-motion pipeline on a hierarchical cluster tree[C]// IEEE, International Conference on Computer Vision Workshops. IEEE, 2009:1489-1496.
- [18] Gherardi R, Farenzena M, Fusiello A. Improving the efficiency of hierarchical structure-and-motion[C]// Computer Vision and Pattern Recognition. IEEE,

2010:1594-1600.

- [19] Chen Y, Chan A B, Lin Z, et al. Efficient tree-structured SfM by RANSAC generalized Procrustes analysis[J]. Computer Vision & Image Understanding, 2017, 157(C):179-189.