DSO代码解读

龚益群

2019年09月20日

数据结构[6]：

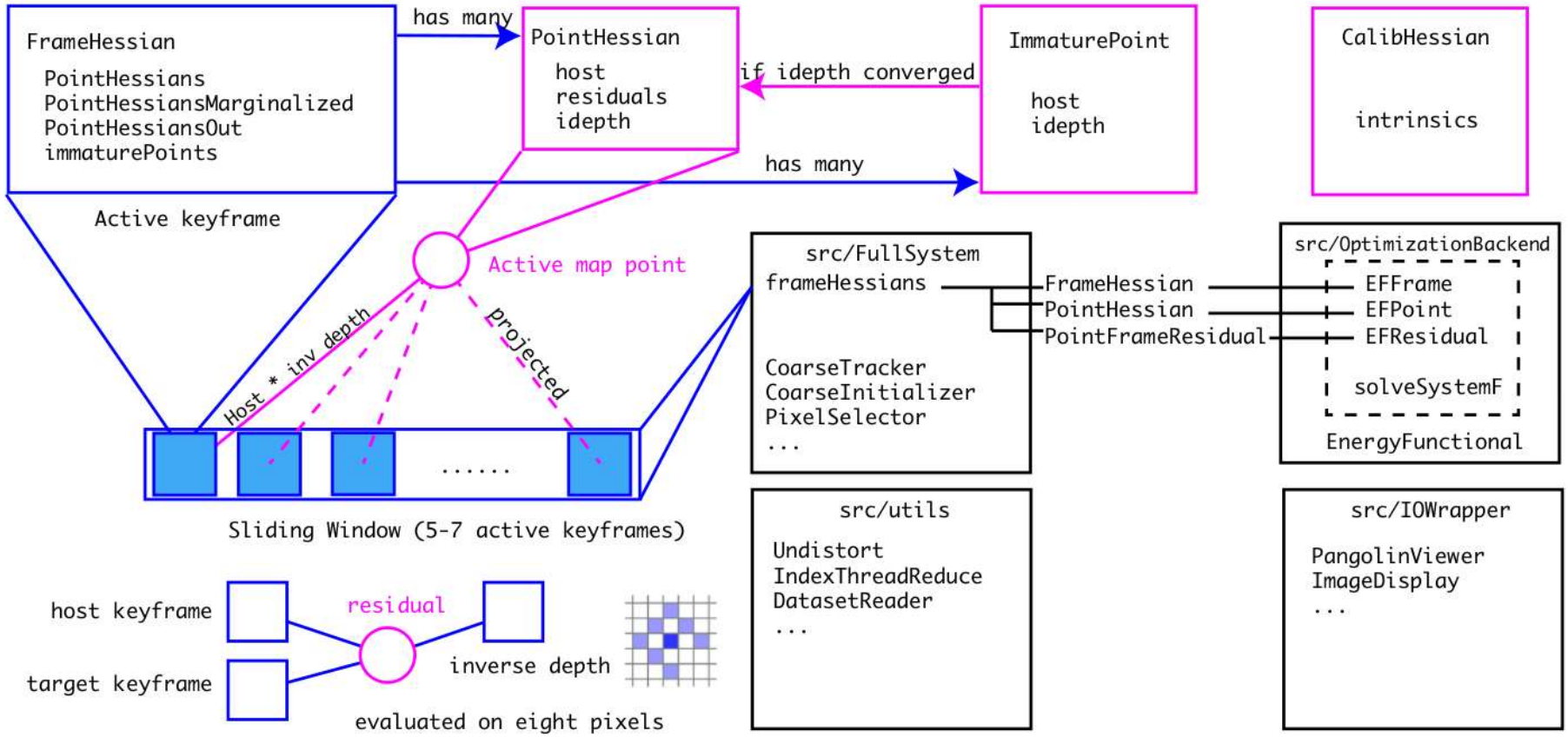


Figure 1: Framework of DSO.

算法流程[6]：

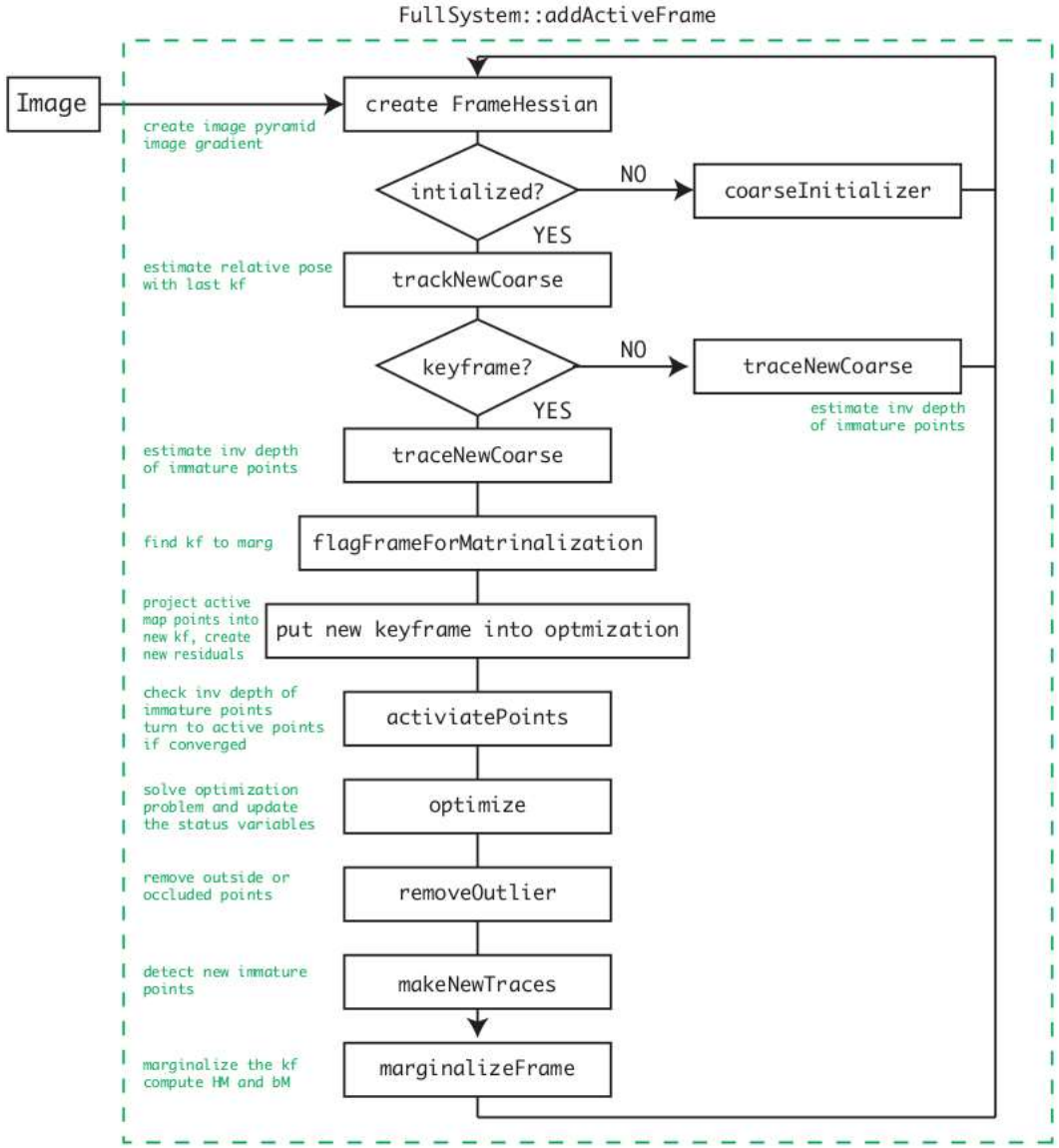
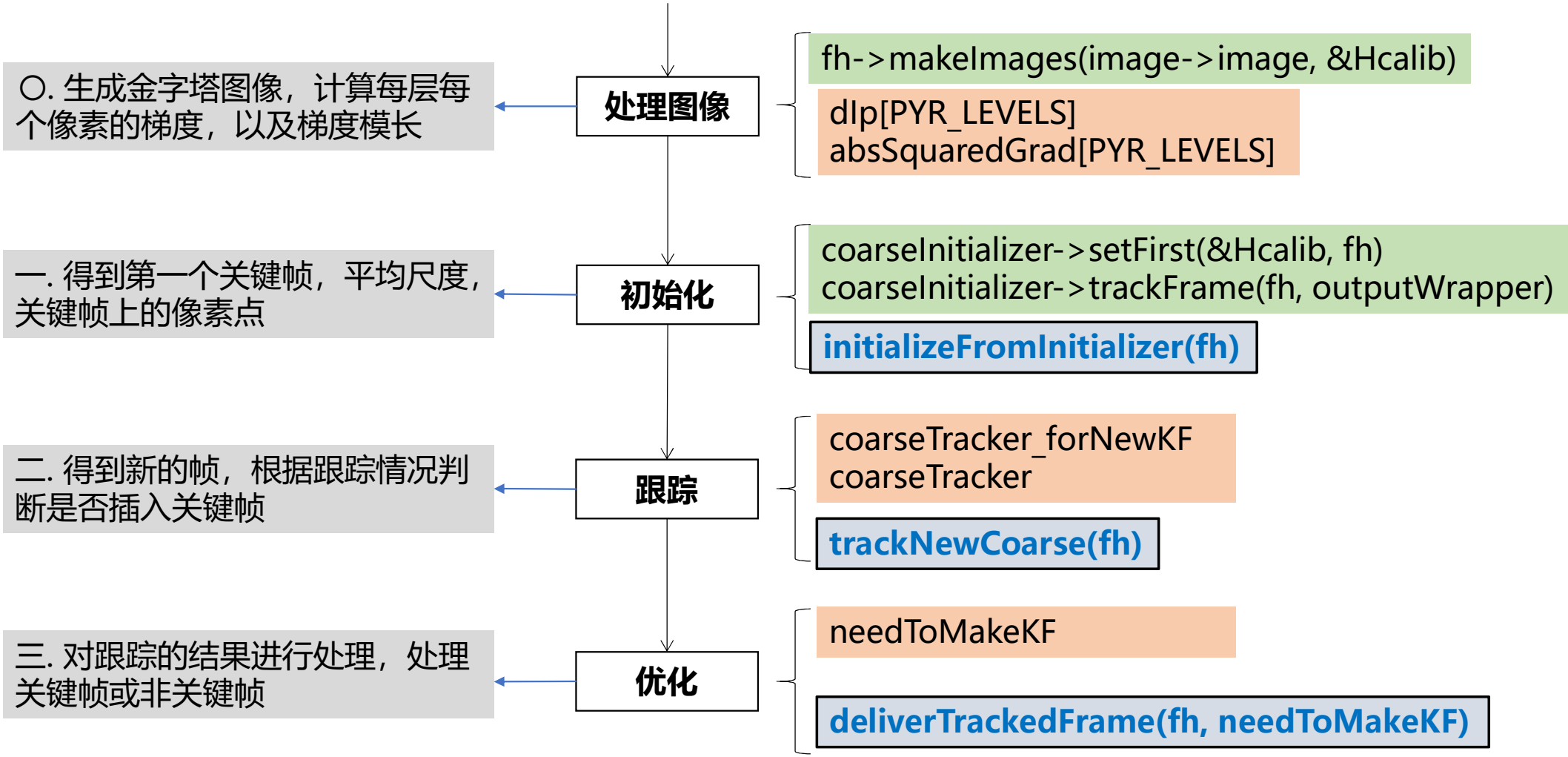


Figure 2: Pipeline of the frontend.

addActiveFrame:

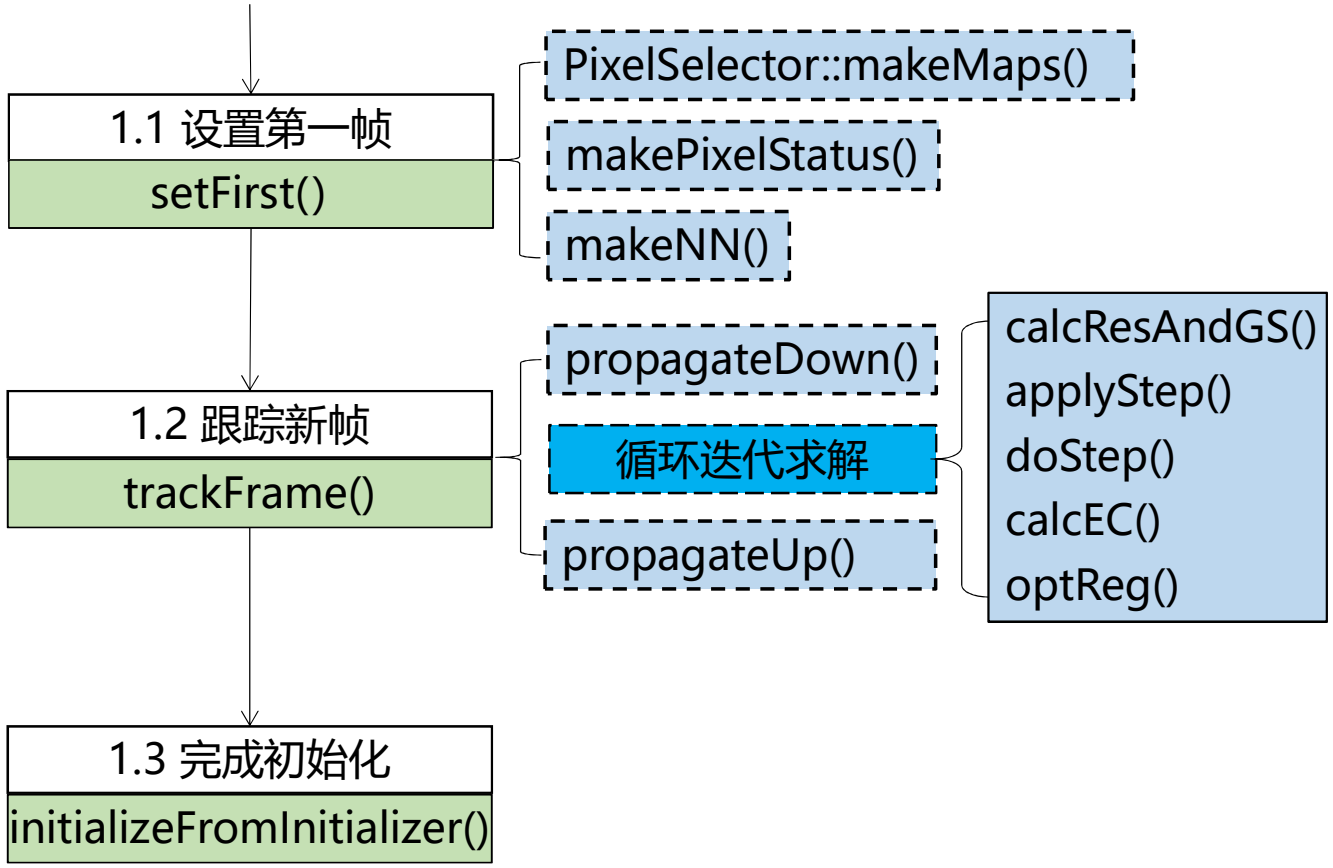


一、初始化：

- 提取第0层的特征，取网格内随机方向梯度最大点
- 提取1-5层的特征，取网格内具有dx/dy最大梯度点
- new用于初始化的点，得到每个点同层最近的10个点(neighbours)，和上一层最近的点(parent)

- 当检测到位移足够大时，开始从金字塔顶层向底层使用LM优化位姿，光度参数，逆深度。
- 然后将逆深度由底层向顶层传播逆深度，用于下次优化做初值。
- 优化到满足位移的后5帧，位移小或中间的帧删除fh。

- 将第一帧插入关键帧，插入能量方程中（后面也会把这个最新帧插入关键帧）
- 使用第0层点的均值作为归一尺度
- 把第0层点创建为PointHessian，并插入能量方程insertPoint()（有先验）
- 设置第一帧和最新帧的FrameShell信息，作为待估计量



【问题】：初始化一直跟踪第一帧好？还是根据优劣旋转跟踪参考帧好？

CoarseInitializer.cpp：

初始化中的能量方程

Jacobian

$$E_{\mathbf{p}j} = \sum_{\mathbf{p}_i \in \mathcal{N}_\mathbf{p}} w_p \left\| \left(I_j\left[\mathbf{p}_j\right] - b_j\right) - \frac{t_j e^{a_j}}{t_i e^{a_i}} \left(I_i\left[\mathbf{p}_i\right] - b_i\right) \right\|_r$$
$$= \sum_{\mathbf{p}_i \in \mathcal{N}_\mathbf{p}} w_p \left\| I_j\left[\mathbf{p}_j\right] - \frac{t_j e^{a_j}}{t_i e^{a_i}} I_i\left[\mathbf{p}_i\right] - \left(b_j - \frac{t_j e^{a_j}}{t_i e^{a_i}} b_i\right) \right\|_r \tag{1}$$

$$\mathbf{p}_j = \Pi_c\left(\mathbf{R}\Pi_c^{-1}\left(\mathbf{p}_i, d_{\mathbf{p}_i}\right) + \mathbf{t}\right) \quad \text{with} \quad \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} := \mathbf{T}_j \mathbf{T}_i^{-1} \tag{2}$$

按照代码中：

$$\mathbf{P}'_i = \pi_c^{-1}\left(\mathbf{p}_i\right) = \mathbf{K}^{-1}\begin{pmatrix} \mathbf{p}_i \\ 1 \end{pmatrix}$$
$$\mathbf{P}'_j = \mathbf{R}\mathbf{P}'_i + \mathbf{t}d_{\mathbf{p}_i} \tag{3}$$
$$\begin{pmatrix} \mathbf{p}_j \\ 1 \end{pmatrix} = \pi_c\left(\omega\left(\mathbf{P}'_j\right)\right)$$

初始化中的光度参数求导

光度求导

初始化对目标帧进行求导

$$\mathbf{J}_{\text{photo}} = \frac{\partial r_k}{\partial \delta_{\text{photo}}}$$
$$= \begin{pmatrix} \frac{\partial r_k}{\partial \delta a_j} & \frac{\partial r_k}{\partial \delta b_j} \end{pmatrix} \tag{4}$$
$$= \begin{pmatrix} -\frac{t_j e^{a_j}}{t_i e^{a_i}}\left(I_i\left[\mathbf{p}_i\right] - b_i\right) & -1 \end{pmatrix}$$

CoarseInitializer.cpp：

初始化中的位姿求导

初始化中的逆深度求导

位姿求导

$$\mathbf{J}_I = \frac{\partial I_j}{\partial \mathbf{p}_j} = \left(\frac{\partial I_j}{\partial \mathbf{p}'_x} \; \frac{\partial I_j}{\partial \mathbf{p}'_y} \right) = (d_x \; d_y) \tag{5}$$

对目标帧位姿求导有

$$\begin{aligned}
\frac{\partial r_k}{\partial \delta\xi} &= \mathbf{J}_I \cdot \frac{\partial \mathbf{p}_j}{\partial \mathbf{P}'_j} \cdot \frac{\partial \mathbf{P}'_j}{\partial \delta\xi} \\
&= (d_x f_x \quad d_y f_y) \begin{pmatrix} \frac{1}{P'_z} & 0 & -\frac{P'_x}{P'_z} \\ 0 & \frac{1}{P'_z} & -\frac{P'_y}{P'^2_z} \end{pmatrix} d_{\mathbf{p}_i} \begin{pmatrix} \mathbf{I} & -\frac{1}{d_{\mathbf{p}_i}} [\mathbf{P}'_j]_x \end{pmatrix} \\
&= (d_x f_x \quad d_y f_y) \begin{pmatrix} \frac{1}{P'_z} & 0 & -\frac{P'_x}{P'^2} \\ 0 & \frac{1}{P'_z} & -\frac{P'_y}{P'_z} \end{pmatrix} \begin{pmatrix} d_{\mathbf{p}_i} & 0 & 0 & 0 & P'_z & -P'_y \\ 0 & d_{\mathbf{p}_i} & 0 & -P'_z & 0 & P'_x \\ 0 & 0 & d_{\mathbf{p}_i} & P'_y & -P'_x & 0 \end{pmatrix} \\
&= (d_x f_x \quad d_y f_y) \begin{pmatrix} \frac{d_{\mathbf{p}_i}}{P'_z} & 0 & -\frac{d_{\mathbf{p}_i}}{P'_z}\frac{P'_x}{P'_z} & -\frac{P'_x P'_y}{P'^2_z} & 1+\frac{P'^2_x}{P'^2_z} & -\frac{P'_y}{P'_z} \\ 0 & \frac{d_{\mathbf{p}_i}}{P'_z} & -\frac{d_{\mathbf{p}_i}}{P'_z}\frac{P'_y}{P'_z} & -1-\frac{P'^2_y}{P'^2_z} & \frac{P'_x P'_y}{P'^2_z} & \frac{P'_y}{P'_z} \end{pmatrix}
\end{aligned} \tag{6}$$

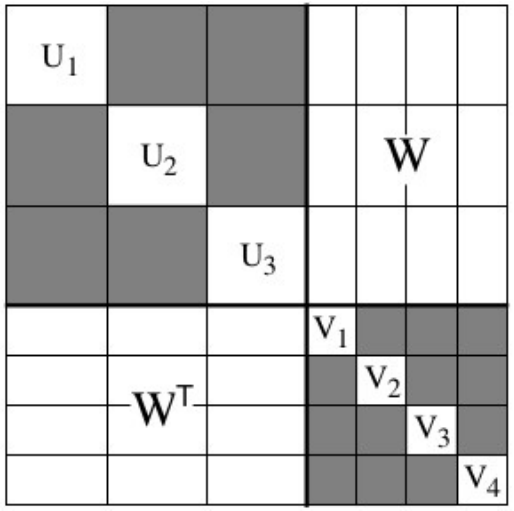
逆深度求导

$$\begin{aligned}
\frac{\partial r_k}{\partial \delta d_{\mathbf{p}_i}} &= \mathbf{J}_I \cdot \frac{\partial \mathbf{p}_j}{\partial \mathbf{P}'_j} \cdot \frac{\partial \mathbf{P}'_j}{\partial \delta d_{\mathbf{p}_i}} \\
&= (d_x \quad d_y) \begin{pmatrix} f_x & 0 \\ 0 & f_y \end{pmatrix} \begin{pmatrix} \frac{1}{P'_z} & 0 & -\frac{P'_x}{P'^2_z} \\ 0 & \frac{1}{P'_z} & -\frac{P'_y}{P'^2_z} \end{pmatrix} \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \\
&= (d_x f_x \quad d_y f_y) \frac{1}{P'_z} \begin{pmatrix} t_x - \frac{P'_x}{P'_z} t_z \\ t_y - \frac{P'_y}{P'_z} t_z \end{pmatrix} \\
&= \frac{1}{P'_z} d_x f_x \left(t_x - \frac{P'_x}{P'_z} t_z \right) + \frac{1}{P'_z} d_y f_y \left(t_y - \frac{P'_y}{P'_z} t_z \right)
\end{aligned} \tag{7}$$

CoarseInitializer.cpp：

DSO中正规方程维护

- DSO中维护的是把逆深度部分舒尔消元之后的矩阵，因此分别计算H/b，HSC/bSC.



$$H = J^T\Sigma^{-1}J = \left[\begin{array}{c|c} A^T\Sigma^{-1}A & A^T\Sigma^{-1}B \\ \hline B^T\Sigma^{-1}A & B^T\Sigma^{-1}B \end{array}\right] = \begin{bmatrix} U & W \\ W^T & V \end{bmatrix}$$

这样公式可以表示为

$$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix}\begin{bmatrix} \Delta x_c \\ \Delta x_p \end{bmatrix} = \begin{bmatrix} b_A \\ b_B \end{bmatrix} \tag{8}$$

计算V的舒尔补进行分块消元：

$$(U - WV^{-1}W^T)\Delta x_c = b_A - WV^{-1}b_B \tag{9}$$

通过（9）式解出Δx_c代入公式（10）可以解出Δx_p

$$V\Delta x_p = b_B - W^T\Delta x_c \tag{10}$$

公式（8）变为：

$$\begin{bmatrix} U - WV^{-1}W^T & 0 \\ W^T & V \end{bmatrix}\begin{bmatrix} \Delta x_c \\ \Delta x_p \end{bmatrix} = \begin{bmatrix} b_A - WV^{-1}b_B \\ b_B \end{bmatrix} \tag{11}$$

PixelSelector2.cpp：

PixelSelector::makeMaps( )

- **makeHists(fh)**计算直方图, 以及选点的阈值
- **select()**当前帧上选择符合条件的像素，densities[] = {0.03,0.05,0.15,0.5,1}
- 计算当前得到的点和需要的点数目比，据此来动态调节网格大小

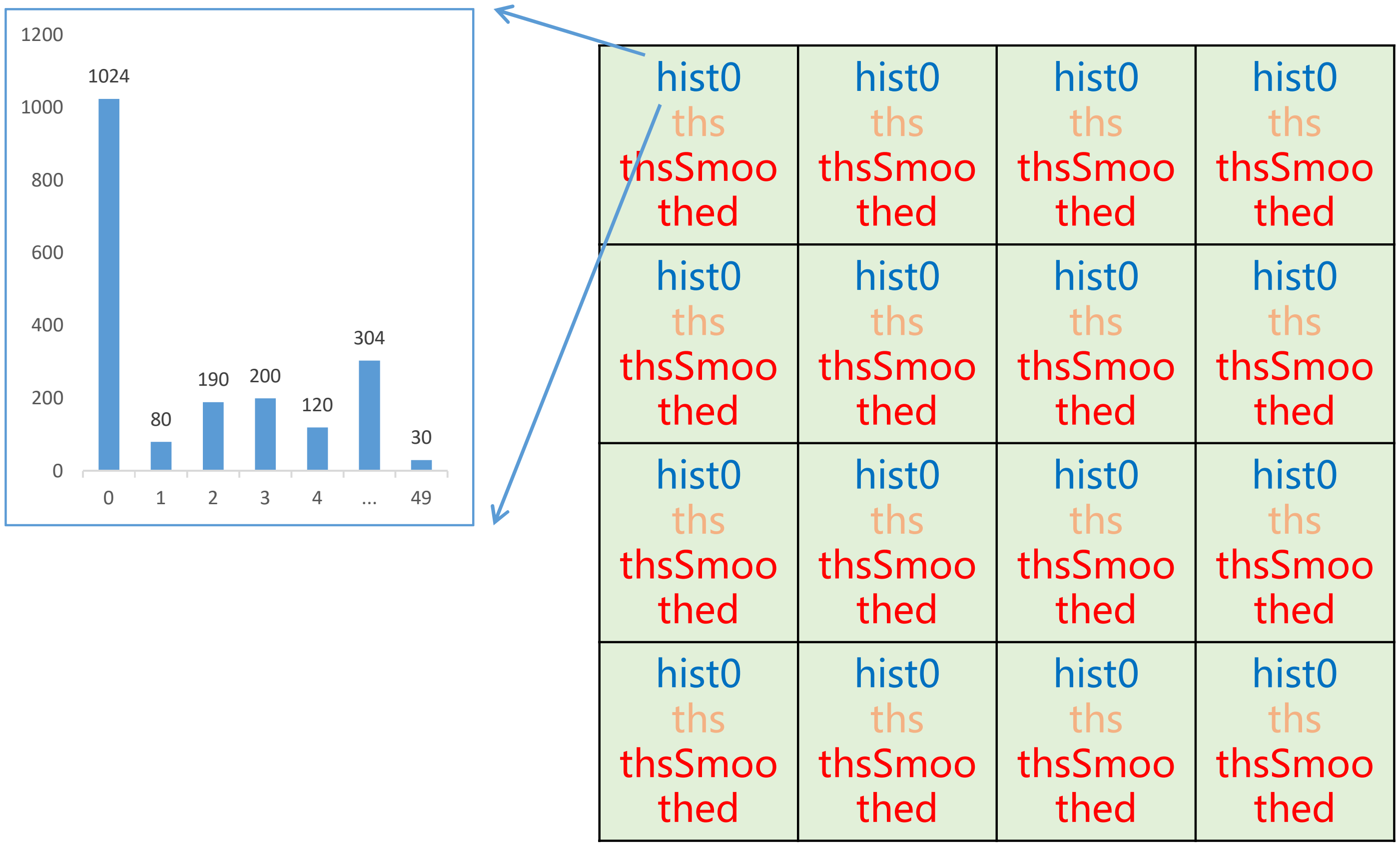
```
// 相当于覆盖的面积, 每一个像素对应一个pot*pot
K = numHave * (currentPotential+1) * (currentPotential+1);
// 除以目标点数，得到应该设置的pot大小
idealPotential = sqrtf(K/numWant)-1;
```

- 比例超过或小于0.25就会递归重新计算
- 如果点还是多，就随机删除一些点

PixelSelector2.cpp：

PixelSelector::makeHists( )

- 每个格32*32大小
- 在格内创建直方图**hist0**
- 统计直方图中的像素点占50%位置的梯度作为阈值**ths**
- 对阈值进行3*3的均值滤波**thsSmoothed**



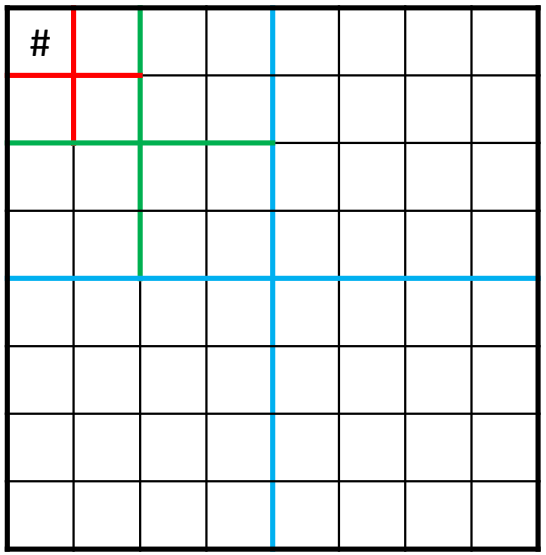
PixelSelector2.cpp：

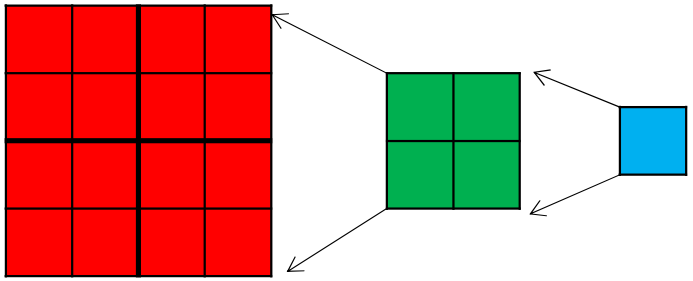
PixelSelector::select( )

对于第0层的金字塔图像

- 遍历每一个像素，取大于所求阈值，且在pot内最大的
- 红pot对应4个像素，绿pot对应4个红pot，蓝pot对应4个绿pot
- 红色的使用金字塔0层上的梯度，阈值为thsSmoothed
- 绿色的使用金字塔1层上的梯度，阈值为红色的0.75倍
- 蓝色的使用金字塔2层上的梯度，阈值为绿色的0.75倍
- 比较大小都使用零层*随机方向*上的梯度
- 优先级红 > 绿 > 蓝，优先级高的找到了就不在低的里面找

4*pot





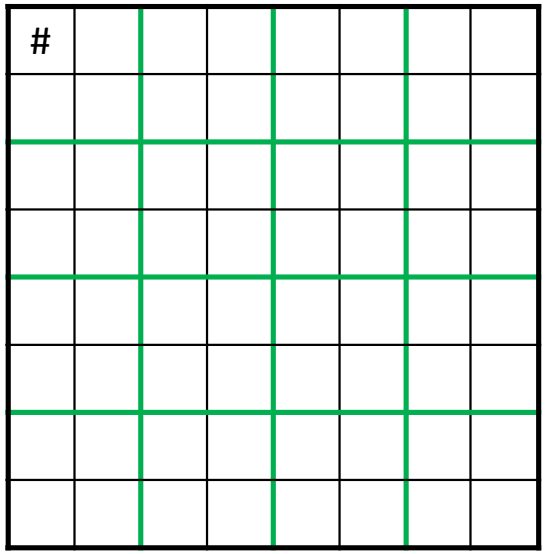
PixelSelector.h：

makePixelStatus()	template<int pot> inline int gridMaxSelection()

对于1-5层的金字塔图像

- 同样动态调节pot的大小，来保证提取合适的数目
- 每个pot内梯度大于阈值，且gradx / grady / gradx-grady / gradx+grady 中有最大值的则被选中

4*pot



CoarseInitializer.cpp：

calcResAndGS()

DSO在初始化时为了使得**尺度收敛**增加两个**能量项**

当位移不足够大(snapped=false)：

$$E_{\mathbf{p}j} := E_{\mathbf{p}j} + \alpha_w \left[(d_{\mathbf{p}_i} - 1)^2 + \|t\|_2 \cdot N \right] \tag{12}$$

因此有代码：

```
JbBuffer_new[i][8] += alphaOpt*(point->idepth_new - 1); // r*dd
JbBuffer_new[i][9] += alphaOpt; // 对逆深度导数为1 // dd*dd

// t*t*ntps
// 给 t 对应的Hessian，对角线加上一个数，b也加上
H_out(0,0) += alphaOpt*npts;
H_out(1,1) += alphaOpt*npts;
H_out(2,2) += alphaOpt*npts;

// 李代数，平移部分（上一次的位姿值）
Vec3f tlog = refToNew.log().head<3>().cast<float>();
b_out[0] += tlog[0]*alphaOpt*npts;
b_out[1] += tlog[1]*alphaOpt*npts;
b_out[2] += tlog[2]*alphaOpt*npts;
```

当位移足够大(snapped=true)时：

$$E_{\mathbf{p}j} := E_{\mathbf{p}j} + (d_{\mathbf{p}_i} - d_{iR})^2 \tag{13}$$

因此有代码：

```
JbBuffer_new[i][8] += couplingWeight*(point->idepth_new - point->iR);
JbBuffer_new[i][9] += couplingWeight;
```

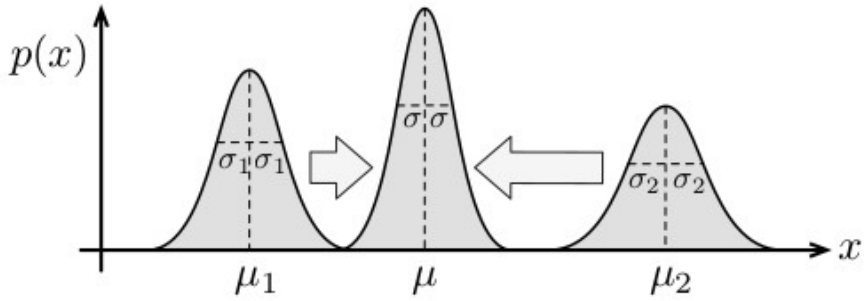
CoarseInitializer.cpp：

propagateDown()	propagateUp()

逆深度在不同层之间传递使用parent点来作为关联，融合策略采用高斯归一化积

$$\boldsymbol{\Sigma}^{-1} = \sum_{k=1}^{K} \boldsymbol{\Sigma}_k^{-1},$$

$$\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} = \sum_{k=1}^{N} \boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k,$$



$$\frac{1}{\sigma^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}$$

$$\frac{\mu}{\sigma^2} = \frac{\mu_1}{\sigma_1^2} + \frac{\mu_2}{\sigma_2^2}$$

optReg()

函数使用同金字塔层之间的neighbours点的中位值对iR进行平滑处理

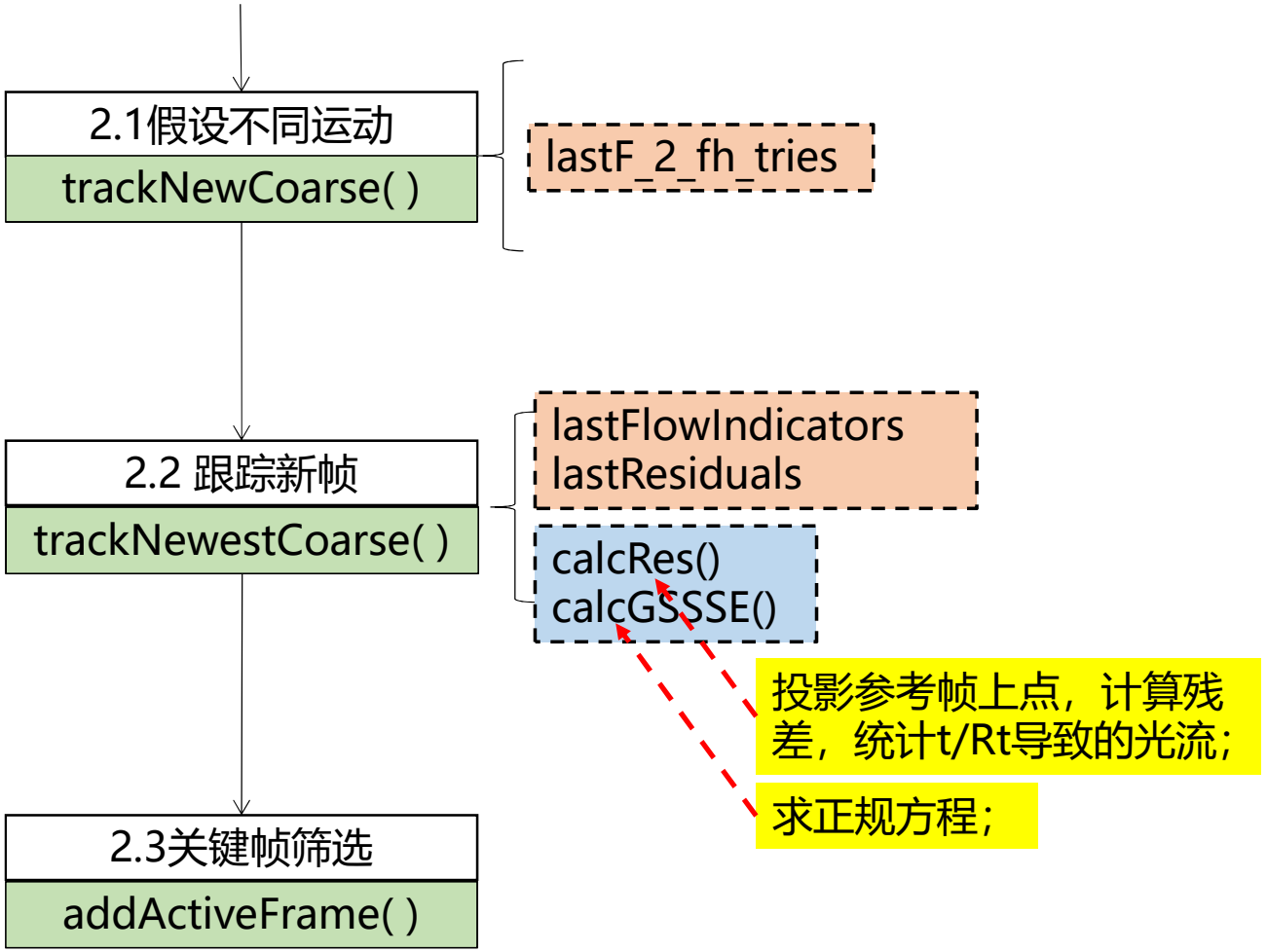
二、跟踪：

- 分别假设5种：匀速、半速、倍速、零速、以及从参考帧就没动。
- 另假设匀速基础上有26种旋转运动。例：Quaterniond(1,rotDelta,0,0)

- 由粗到精迭代优化，如果大于能量阈值的点超过60%，则放大阈值，且该层多优化一遍。
- 如果某一层的能量值大于1.5倍最小值，直接判断为失败，结束节省时间。
- 如果跟踪成功且第0层好于当前，则保留结果，并更新每一层的最小能量值。
- 第0层最小值小于阈值则停止，把目前最好的值作为下次跟踪的阈值。

- 可以设置定时插入

- 通过像素移动的大小判断，位移较大、平移+旋转较大、曝光变化大、跟踪得到的能量变化大，则插入（*论文中说只考虑位移，有效处理遮挡/去遮挡*）



【问题】：如何理解这个判断关键帧策略有效处理遮挡?

CoarseTracker.cpp：

- 没用pattern，只使用投影过来的中心点像素误差
- 优化的变量只有相机位姿和光度参数，没有逆深度
- 其中用到Huber函数

$$\rho_{\mathrm{H}}(e)=\begin{cases}\frac{1}{2}e^2 & \text{for } |e|\le k\\ k|e|-\frac{1}{2}k^2 & \text{for } |e|>k\end{cases}$$

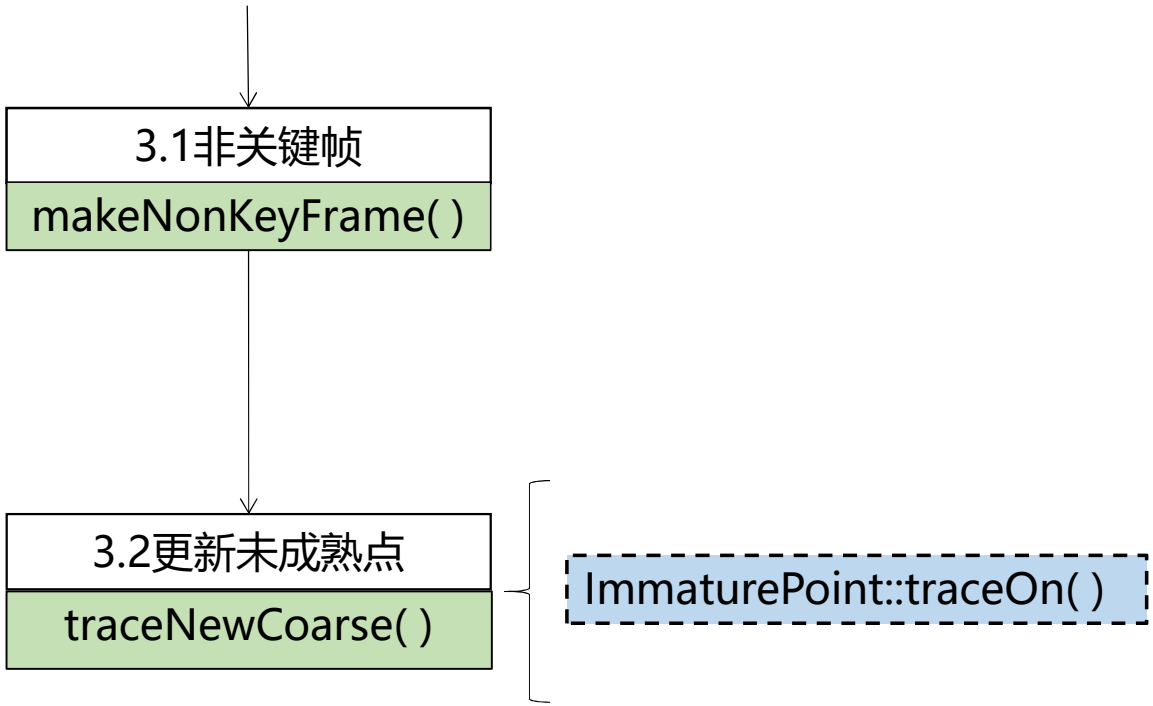
$$w_{\mathrm{H}}(e)=\begin{cases}1 & \text{for } |e|\le k\\ k/|e| & \text{for } |e|>k\end{cases}$$

$$w_H(e)\times(2-w_H(e))\times e^2=2\times\rho_H(e)$$

三、优化：

- 更新未成熟点ImmaturePoint的逆深度范围后，删除fh，只保留FrameShell。

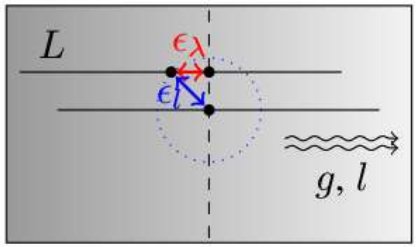
- 根据逆深度范围得到极线搜索的范围
- 计算图像梯度和极线夹角的大小，如果太大则误差会很大
- 在极线上按照一定步长进行搜索能量最小的位置，和大于设置半径(2)的第二小的位置，后/前作为质量，越大越好
- 沿着极线进行GN优化，直到增量足够小
- 根据搜索得到的投影位置计算新的逆深度范围

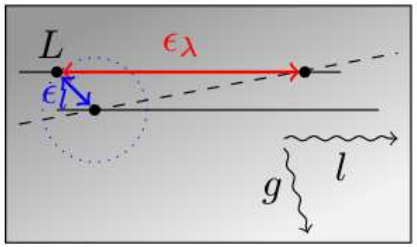


ImmaturePoint.cpp：

traceOn( )

公式（14）可知极线方向和梯度方向**夹角过大**误差会变大





$$l_0 + \lambda^* \begin{pmatrix} l_x \\ l_y \end{pmatrix} \stackrel{!}{=} g_0 + \gamma \begin{pmatrix} -g_y \\ g_x \end{pmatrix}, \quad \gamma \in \mathbb{R} \tag{14}$$

$$\begin{aligned} \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} &= \Pi_c \left(KR \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} + Kt \cdot d_p \right) \\ &= \Pi_c \left(\begin{bmatrix} pr_x \\ pr_y \\ pr_z \end{bmatrix} + d_p \cdot \begin{bmatrix} kt_x \\ kt_y \\ kt_z \end{bmatrix} \right) \end{aligned} \tag{15}$$

$$\begin{aligned} u' &= \frac{pr_x + d_p \cdot kt_x}{pr_z + d_p \cdot kt_z} \\ v' &= \frac{pr_y + d_p \cdot kt_y}{pr_z + d_p \cdot kt_z} \\ \Rightarrow d_p &= \frac{u' \cdot pr_z - pr_x}{kt_x - u' \cdot kt_z} \\ d_p &= \frac{v' \cdot pr_z - pr_y}{kt_y - v' \cdot kt_z} \end{aligned} \tag{16}$$

已知匹配点计算**逆深度范围**（会在公式（16）点上再加误差）

2019.10.10

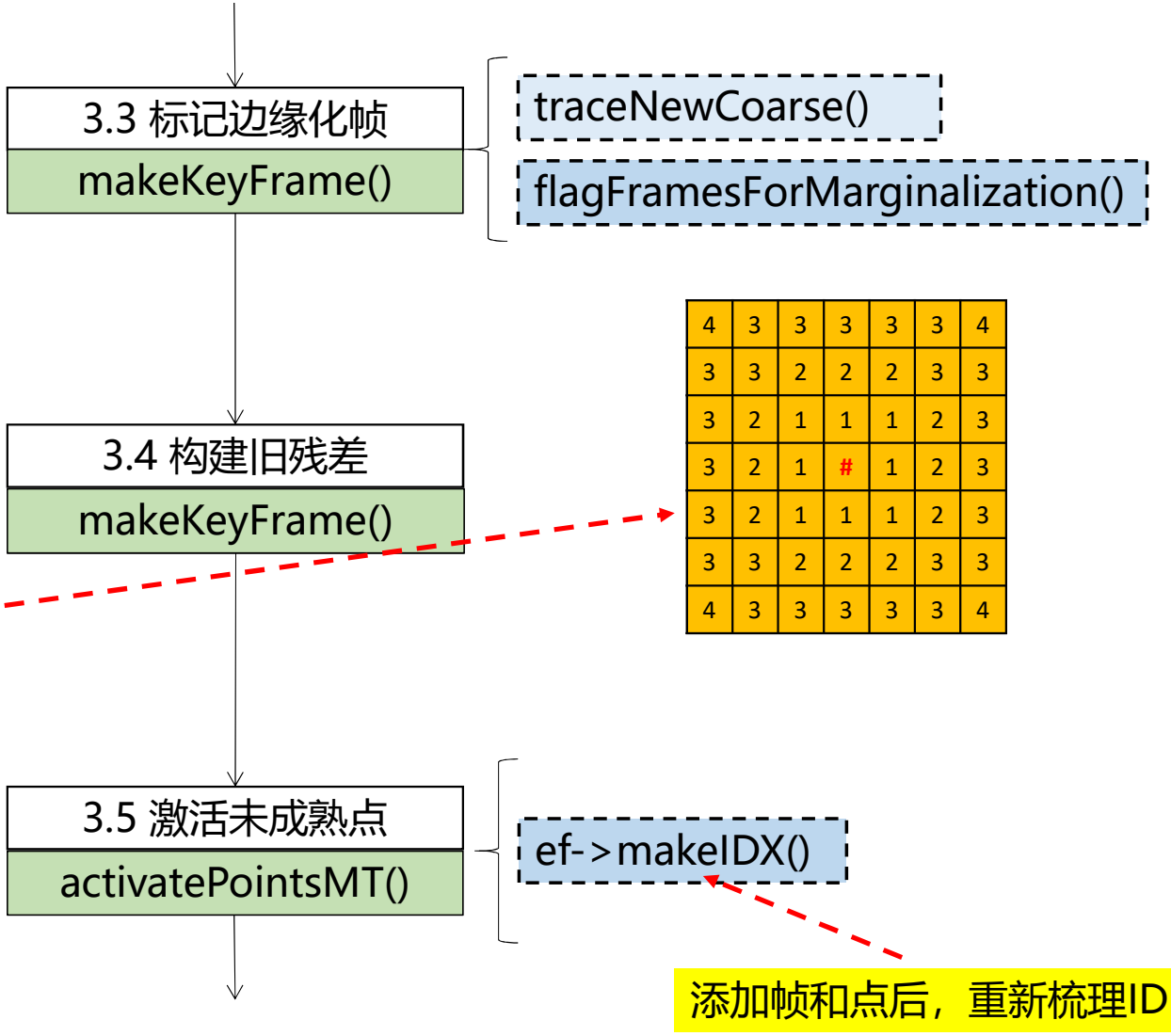
【问题】：坐标加0.5的作用?

三、优化：

- 留下的点(去除边缘化+删除)占所有点小于5%
- 和参考帧比曝光变化较大
- 保证滑窗内有5个关键帧
- 如果还大于7个关键帧，则边缘化掉到最新关键帧的距离占所有距离比最大的。保证良好的空间结构

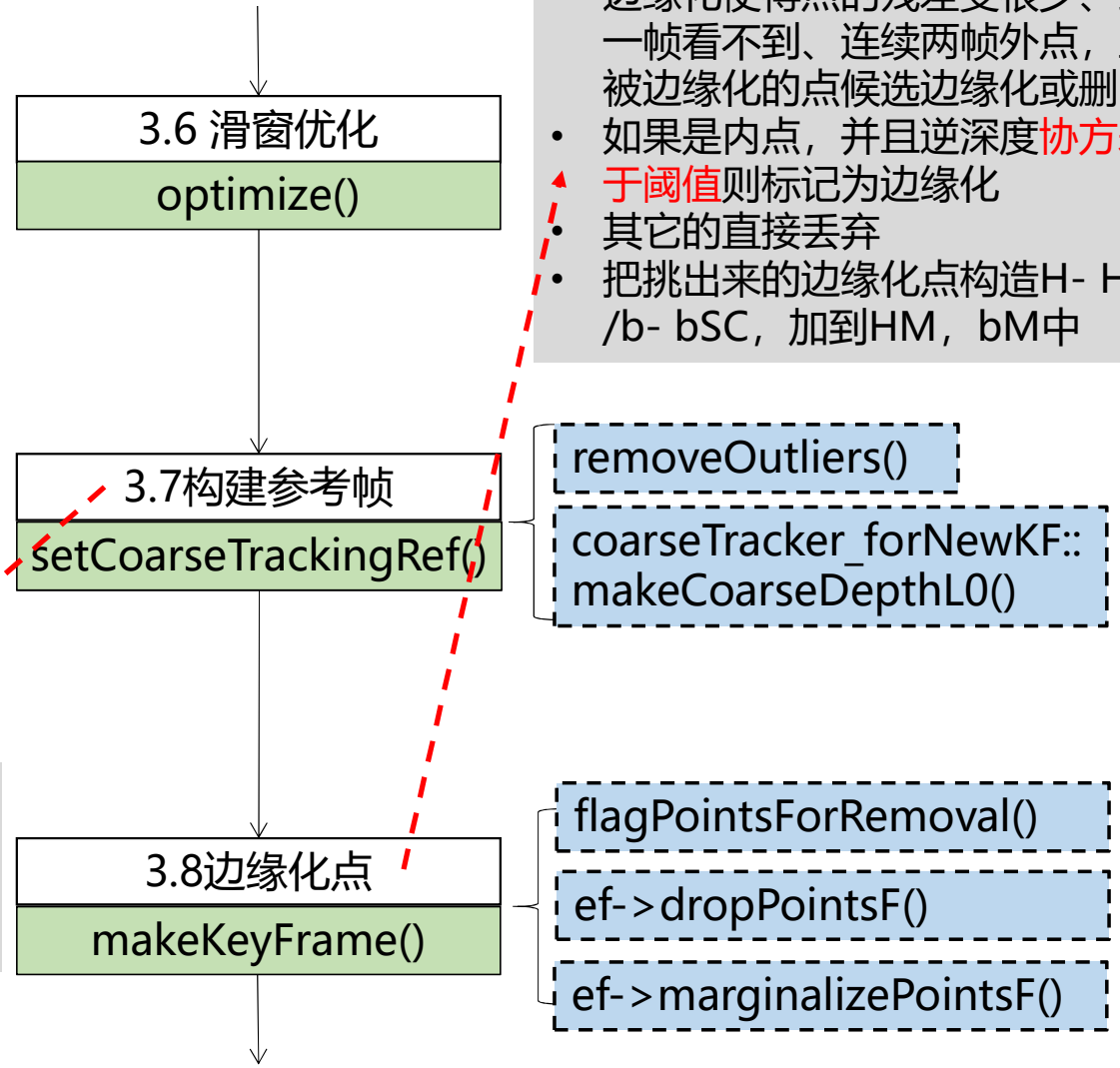
- 加入关键帧，能量函数中insertFrame()，设置相对位姿和线性化点
- 构建新关键帧和之前关键帧之间的残差PointFrameResidual，并插入能量函数insertResidual()，DSO选择先全部构建，之后再删除

- 在最新关键帧上生成距离地图(点附近第一层是1，第二层2,.....)
- 满足搜索范围小于8，质量好，逆深度为正，删除外点和边缘化帧上的点
- 将点投影到距离地图上，大于阈值(阈值根据当前点数量调节)激活
- 使用LM优化选出来点的逆深度，将内点构建PointHessian和残差一起加入能量函数
- 删除未收敛的点



三、优化：

- 使用GN法对位姿、光度参数、逆深度、相机内参进行优化，由于边缘化需要维护两个H矩阵和b向量
- 其中位姿和相机内参使用FEJ，除了最新一帧，相关H矩阵固定在上一次优化，残差仍然使用更新后的状态求
- 被边缘化部分残差更新：b=b-H*delta
- 其中第一帧位姿和其上点逆深度，由于初始化具有先验，光度参数有先验
- 使用伴随性质将相对位姿变为世界系下的(local -> global)
- 减去求解出的增量零空间部分，防止在零空间乱飘



- 边缘化使得点的残差变很少、最新一帧看不到、连续两帧外点，主帧被边缘化的点候选边缘化或删除
- 如果是内点，并且逆深度协方差小于阈值则标记为边缘化
- 其它的直接丢弃
- 把挑出来的边缘化点构造H- HSC /b- bSC，加到HM，bM中

- 将最新帧设置为参考帧，并将所有的点向最新帧投影，并且在金字塔从下向上使用协方差加权生成逆深度，然后对于每一层使用周围点来尽可能生成像素逆深度，这样保证有足够多得点来跟踪，鲁棒。

【问题】：这种先计算posegraph的和直接H矩阵上边缘化的区别?

Residuals.cpp：

- 该文件主要求对各个状态相对量的Jacobian，因此对光度参数的Jacobian结果有变化，其它的和对Target帧的结果相同。
- 在得到针对相对量的Jacobian之后，使用伴随性质将相对的变为绝对的，其中使用FEJ的部分，伴随也需要使用固定的线性化点位置来求。

滑窗中的光度参数求导

将$\delta a = \frac{t_j e^{a_j}}{t_i e^{a_i}}$，　$\delta b = b_j - \frac{t_j e^{a_j}}{t_i e^{a_i}} b_i$看成整体，分别对增量a, b求导：

$$\begin{aligned} \mathbf{J}_{\text{photo}} &= \frac{\partial r_k}{\partial \delta_{\text{photo}}} \\ &= \begin{pmatrix} \frac{\partial r_k}{\partial \delta a} & \frac{\partial r_k}{\partial \delta b} \end{pmatrix} \\ &= \begin{pmatrix} -\left(I_i[\mathbf{p}] - b_i\right) & -1 \end{pmatrix} \end{aligned} \tag{4'}$$

Residuals.cpp：

滑窗中相机内参求导

相机内参部分

设：

$$\omega\left(\mathbf{P}_j'\right)=\begin{bmatrix}\frac{P_{jx}'}{P_{jz}'}\\ \frac{P_{jy}'}{P_{jz}'}\\ 1\end{bmatrix}=\begin{bmatrix}u_j\\ v_j\\ 1\end{bmatrix} \tag{17}$$

根据公式（3）对相机内参求导，包括两部分π_c，π_c^{-1}，分别求偏导有

$$\frac{\partial \mathbf{p}_j}{\partial \delta \mathbf{c}}=\begin{pmatrix}u_j & 0 & 1 & 0\\ 0 & v_j & 0 & 1\end{pmatrix}+\begin{pmatrix}f_x & 0\\ 0 & f_y\end{pmatrix}\begin{pmatrix}\frac{\partial u_j}{\partial \delta \mathbf{c}}\\ \frac{\partial v_j}{\partial \delta \mathbf{c}}\end{pmatrix} \tag{18}$$

对于第二部分

$$\begin{aligned}\frac{\partial \omega\left(\mathbf{P}_j'\right)}{\partial \mathbf{P}_\mathbf{i}'}&=\frac{\partial \omega\left(\mathbf{P}_j'\right)}{\partial \mathbf{P}_j'}\frac{\partial \mathbf{P}_j'}{\partial \mathbf{P}_i'}\\ &=\frac{1}{P_{jz}'}\begin{pmatrix}1 & 0 & -u_j\\ 0 & 1 & -v_j\\ 0 & 0 & 0\end{pmatrix}\begin{pmatrix}r_{00} & r_{01} & r_{02}\\ r_{10} & r_{11} & r_{12}\\ r_{20} & r_{21} & r_{22}\end{pmatrix}\end{aligned} \tag{19}$$

$$\begin{aligned}\frac{\partial \mathbf{P}_i'}{\partial \delta \mathbf{c}}&=\frac{\partial \mathbf{K}^{-1}\mathbf{p}_i}{\partial \delta \mathbf{c}}\\ &=\begin{pmatrix}-f_x^{-2}\left(p_{ix}-c_x\right) & 0 & -f_x^{-1} & 0\\ 0 & -f_y^{-2}\left(p_{iy}-c_y\right) & 0 & -f_y^{-1}\\ 0 & 0 & 0 & 0\end{pmatrix}\\ &=\begin{pmatrix}-f_x^{-1}P_{ix}' & 0 & -f_x^{-1} & 0\\ 0 & -f_y^{-1}P_{iy}' & 0 & -f_y^{-1}\\ 0 & 0 & 0 & 0\end{pmatrix}\end{aligned} \tag{20}$$

Residuals.cpp：

滑窗中相机内参求导

结合公式（17） （19） （20）得到

$$\frac{\partial u_j}{\partial \delta \mathbf{c}} = \frac{\partial u_j}{\partial \mathbf{P}_i'} \frac{\partial \mathbf{P}_i'}{\partial \delta \mathbf{c}}$$

$$= \frac{1}{P_z'} \begin{pmatrix} 1 & 0 & -u_j \end{pmatrix} \mathbf{R} \begin{pmatrix} -f_x^{-1} P_x & 0 & -f_x^{-1} & 0 \\ 0 & -f_y^{-1} P_y & 0 & -f_y^{-1} \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{21}$$

$$= \frac{1}{P_z'} \begin{pmatrix} P_x f_x^{-1} (r_{20} u_j - r_{00}) \\ P_y f_y^{-1} (r_{21} u_j - r_{01}) \\ f_x^{-1} (r_{20} u_j - r_{00}) \\ f_y^{-1} (r_{21} u_j - r_{01}) \end{pmatrix}^T$$

$$\frac{\partial v_j}{\partial \delta \mathbf{c}} = \frac{\partial v_j}{\partial \mathbf{P}_i'} \frac{\partial \mathbf{P}_i'}{\partial \delta \mathbf{c}}$$

$$= \frac{1}{P_z'} \begin{pmatrix} P_x f_x^{-1} (r_{20} v_j - r_{10}) \\ P_y f_y^{-1} (r_{21} v_j - r_{11}) \\ f_x^{-1} (r_{20} v_j - r_{10}) \\ f_y^{-1} (r_{21} v_j - r_{11}) \end{pmatrix}^T \tag{22}$$

结合公式（18） （21） （22）得到结果：

$$\frac{\partial \mathbf{p}_j}{\partial \delta \mathbf{c}} = \begin{pmatrix} u_j & 0 & 1 & 0 \\ 0 & v_j & 0 & 1 \end{pmatrix} + \begin{pmatrix} f_x & 0 \\ 0 & f_y \end{pmatrix} \begin{pmatrix} \frac{\partial u_j}{\partial \delta \mathbf{c}} \\ \frac{\partial v_j}{\partial \delta \mathbf{c}} \end{pmatrix}$$

$$= \begin{pmatrix} u_j & 0 & 1 & 0 \\ 0 & v_j & 0 & 1 \end{pmatrix} \tag{23}$$

$$+ \frac{1}{P_z'} \begin{pmatrix} P_x (r_{20} u - r_{00}) & P_y \frac{f_x}{f_y} (r_{21} u - r_{01}) & (r_{20} u - r_{00}) & \frac{f_x}{f_y} (r_{21} u - r_{01}) \\ P_x \frac{f_y}{f_x} (r_{20} v - r_{10}) & P_y (r_{21} v - r_{11}) & \frac{f_y}{f_x} (r_{20} v - r_{10}) & (r_{21} v - r_{11}) \end{pmatrix}$$

Residuals.cpp：

FrameFramePrecalc::set()

该函数设置关键帧间相对（ host 到 target）的状态变量。

- 因为DSO部分状态使用FEJ，所以需要保存不同状态的值。
- 其中位姿 / 光度参数使用固定线性化点，逆深度 / 内参 / 图像导数都没有固定线性化点。
- 其中逆深度、内参、位姿这些几何参数使用中心点的值代替pattern内的。

FrameFramePrecalc::PRE_RTll_0 / PRE_tTll_0

该变量是固定的线性化点位置的状态增量。

FrameFramePrecalc::PRE_RTll / PRE_tTll

该变量是优化更新状态后的状态增量。

注：逆深度每次都会重新设置线性化点，相当于没有固定，虽然代码写的很像。

【问题】：如何选择FEJ的变量？NOFEJ只影响不可观的？会不会把可观的变不可观的？

EnergyFunctional.cpp：

伴随得到**相对量**对**绝对量**导数

伴随

定义:

$$\mathrm{Exp}(\mathrm{Ad}_{\mathbf{T}} \cdot \xi) \doteq \mathbf{T}\,\mathrm{Exp}(\xi)\mathbf{T}^{-1} \tag{24}$$

线性化点相对位姿和绝对位姿的关系:

$$\mathbf{T}_{th} = \mathbf{T}_{tw}\mathbf{T}_{hw}^{-1} \tag{25}$$

第一部分相对位姿se3对host se3导数:

对姿态进行左乘更新:

$$\mathrm{Exp}(\delta\boldsymbol{\xi}_{th})\mathbf{T}_{th} = \mathbf{T}_{tw}(\mathrm{Exp}(\delta\boldsymbol{\xi}_{h})\mathbf{T}_{hw})^{-1} \tag{26}$$

$$\begin{aligned}\mathrm{Exp}(\delta\boldsymbol{\xi}_{th}) &= \mathbf{T}_{tw}\mathbf{T}_{hw}^{-1}\mathrm{Exp}(-\delta\boldsymbol{\xi}_{h})\mathbf{T}_{th}^{-1} \\ &= \mathbf{T}_{th}\mathrm{Exp}(-\delta\boldsymbol{\xi}_{h})\mathbf{T}_{th}^{-1} \\ &= \mathrm{Exp}\left(-\mathrm{Ad}_{\mathbf{T}_{th}}(\delta\boldsymbol{\xi}_{h})\right)\end{aligned} \tag{27}$$

$$\delta\boldsymbol{\xi}_{th} = -\mathrm{Ad}_{\mathbf{T}_{th}}\delta\boldsymbol{\xi}_{h} \tag{28}$$

所以:

$$\frac{\partial\boldsymbol{\xi}_{th}}{\partial\boldsymbol{\xi}_{h}} = -\mathrm{Ad}_{\mathbf{T}_{th}} \tag{29}$$

EnergyFunctional.cpp：

伴随得到**相对量**对**绝对量**导数

第二部分相对位姿se3对target se3求导:

同样类似公式(21) - (24)的过程:

$$\mathrm{Exp}(\delta\boldsymbol{\xi}_{th})\mathbf{T}_{th} = \mathrm{Exp}(\delta\boldsymbol{\xi}_{t})\mathbf{T}_{tw}\mathbf{T}_{hw}^{-1} \tag{30}$$

$$\begin{aligned}\mathrm{Exp}(\delta\boldsymbol{\xi}_{th}) &= \mathrm{Exp}(\delta\boldsymbol{\xi}_{t})\mathbf{T}_{tw}\mathbf{T}_{hw}^{-1}\mathbf{T}_{th}^{-1} \\ &= \mathrm{Exp}(\delta\boldsymbol{\xi}_{t})\end{aligned} \tag{31}$$

$$\delta\boldsymbol{\xi}_{th} = \delta\boldsymbol{\xi}_{t} \tag{32}$$

最终有:

$$\frac{\partial\boldsymbol{\xi}_{th}}{\partial\boldsymbol{\xi}_{t}} = \mathbf{I} \tag{33}$$

第三部分相对光度参数对host和target求导

host和target之间的相对光度参数为

$$\delta a = -\frac{t_j e^{a_j}}{t_i e^{a_i}}, \quad \delta b = -b_j + \frac{t_j e^{a_j}}{t_i e^{a_i}} b_i \tag{34}$$

相对量对绝对量求导:

$$\begin{aligned}\frac{\partial\delta a}{\partial a_i} &= \frac{t_j e^{a_j}}{t_i e^{a_i}} = e^{\delta a} \\ \frac{\partial\delta b}{\partial b_i} &= \frac{t_j e^{a_j}}{t_i e^{a_i}} = e^{\delta a} \\ \\ \frac{\partial\delta a}{\partial a_j} &= -\frac{t_j e^{a_j}}{t_i e^{a_i}} = -e^{\delta a} \\ \frac{\partial\delta b}{\partial b_j} &= \frac{t_j e^{a_j}}{t_i e^{a_i}} = -1\end{aligned} \tag{35}$$

EnergyFunctional.cpp：

accumulateAF_MT( )

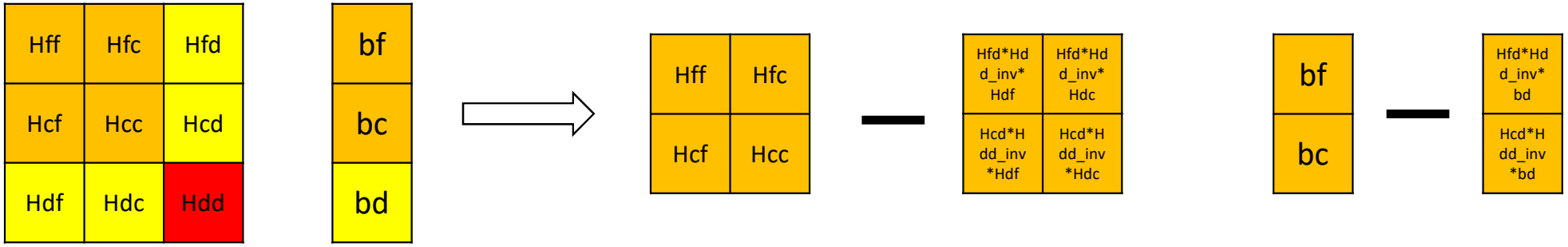
函数利用AccumulatedTopHessian.cpp中AccumulatedTopHessianSSE(mode=0)类对滑窗内的点计算正规方程，计算H和b，然后使用Adj转为绝对量。这里不包括逆深度相关量，因为被舒尔消元掉了。

accumulateLF_MT()

同样调用AccumulatedTopHessianSSE类只不过是mode=1，实际该函数中加入的点(isLinearized=true)是不存在的，因为被这样的点在边缘化时都别删除了。剩下的作用包括：添加先验信息和把p->*LF相关的量置零。

accumulateSCF_MT()

调用AccumulatedSCHessian.cpp中AccumulatedSCHessianSSE类对滑窗内点逆深度相关的量进行计算，来完成Schur消元过程，同样使用Adj转为绝对量。



EnergyFunctional.cpp：

bM_top = (bM+ HM * getStitchedDeltaF()

这是对于被边缘化部分的残差像更新，HM和bM是边缘化得到的，后面详细说明。因为被边缘化后没办法project计算新的残差，因此使用一阶泰勒方式更新。注意，其中delta应该是绝对量。

EF->cPrior　EF->cDeltaF

相机参数的先验信息：Hessian是很大的常数；delta估计值与设置的初值的差；

EF->frames->prior　EF->frames->delta_prior

位姿光度的先验信息：第一帧位姿Hessian常数，光度Hessian常数(1e14); 其它帧位姿Hessian为零，光度Hessian常数(1e12/1e8)；delta估计值与固定线性化点的差

p->priorF; p->deltaF

点逆深度的先验信息：第一帧上的有(2500)，其它帧上的为0；delta值为0(因为线性化点被更新了)

【问题】：如何设置先验值?

FullSystemOptimize.cpp：

optimize()

- 在完成一次滑窗优化后，对于最新关键帧会设置当前的状态为FEJ状态，更新当前的待估计量，并且重新线性化一次，因为创建residual时是全部创建，这里删除不好的值。
- 对于其它的关键帧，将优化的状态更新到FrameShell，线性化点不变。
- 每次变动关键帧都会调用setPrecalcValues()，更新关键帧之间的相对状态。

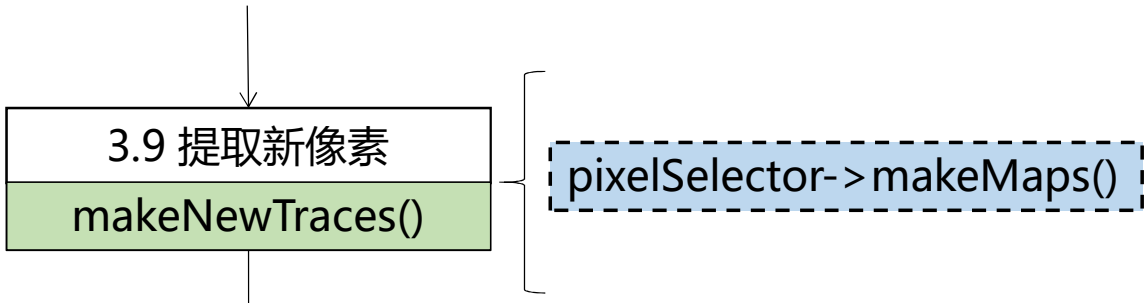
EnergyFunctional::orthogonalize()

- 零空间：求的方法是对状态进行数值求导，即 (正扰动-逆扰动) / (2*扰动) 。我理解这部分就相当于观测方程y=cx中的c，然后 [c] 就是能观性方程，因此c应该是秩为0的，多出来的都是，因为零空间，从正规方程中减去零空间，即可抑制零空间漂移。

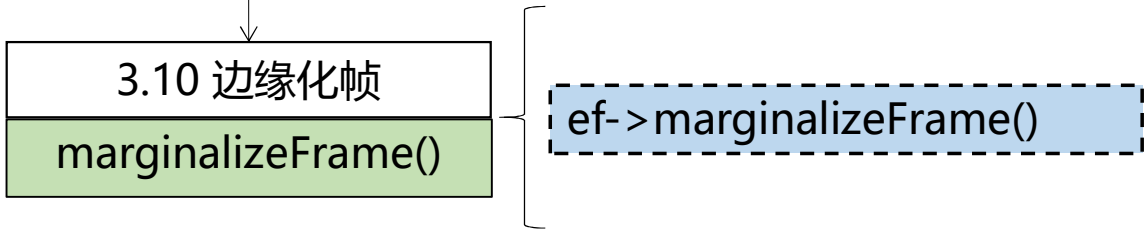
```
// 比最大奇异值小setting_solverModeDelta(e-5)倍，则认为是0
for(int i=0;i<SNN.size();i++)
    { if(SNN[i] > setting_solverModeDelta*maxSv) SNN[i] = 1.0 / SNN[i]; else SNN[i] = 0; } // 求逆

MatXX Npi = svdNN.matrixU() * SNN.asDiagonal() * svdNN.matrixV().transpose();   // [dim] x 7.
//! Npi.transpose()是N的伪逆
MatXX NNpiT = N*Npi.transpose();    // [dim] x [dim].
MatXX NNpiTS = 0.5*(NNpiT + NNpiT.transpose()); // = N * (N' * N)^-1 * N'.

//TODO 为什么这么做?
//* 把零空间从H和b中减去??? 以免乱飘?
if(b!=0) *b -= NNpiTS * *b;
if(H!=0) *H -= NNpiTS * *H * NNpiTS;
```

三、优化：

- 在最新帧第0层提取随机方向梯度最大的像素，并且构造成ImmaturePoint



- 将被边缘化的帧的8个状态挪到右下角，然后计算Schur Complement，将其消掉
- 删除在被边缘化帧上的残差
- DSO里面操作的都是PoseGraph



EnergyFunctional.cpp：

marginalizePointsF()

函数调用AccumulatedTopHessianSSE(mode=2)和AccumulatedSCHessianSSE类分别针对被边缘化的点残差计算H/b和HSC/bSC，然后舒尔消元后，加权加在HM和bM上。

marginalizeFrame()

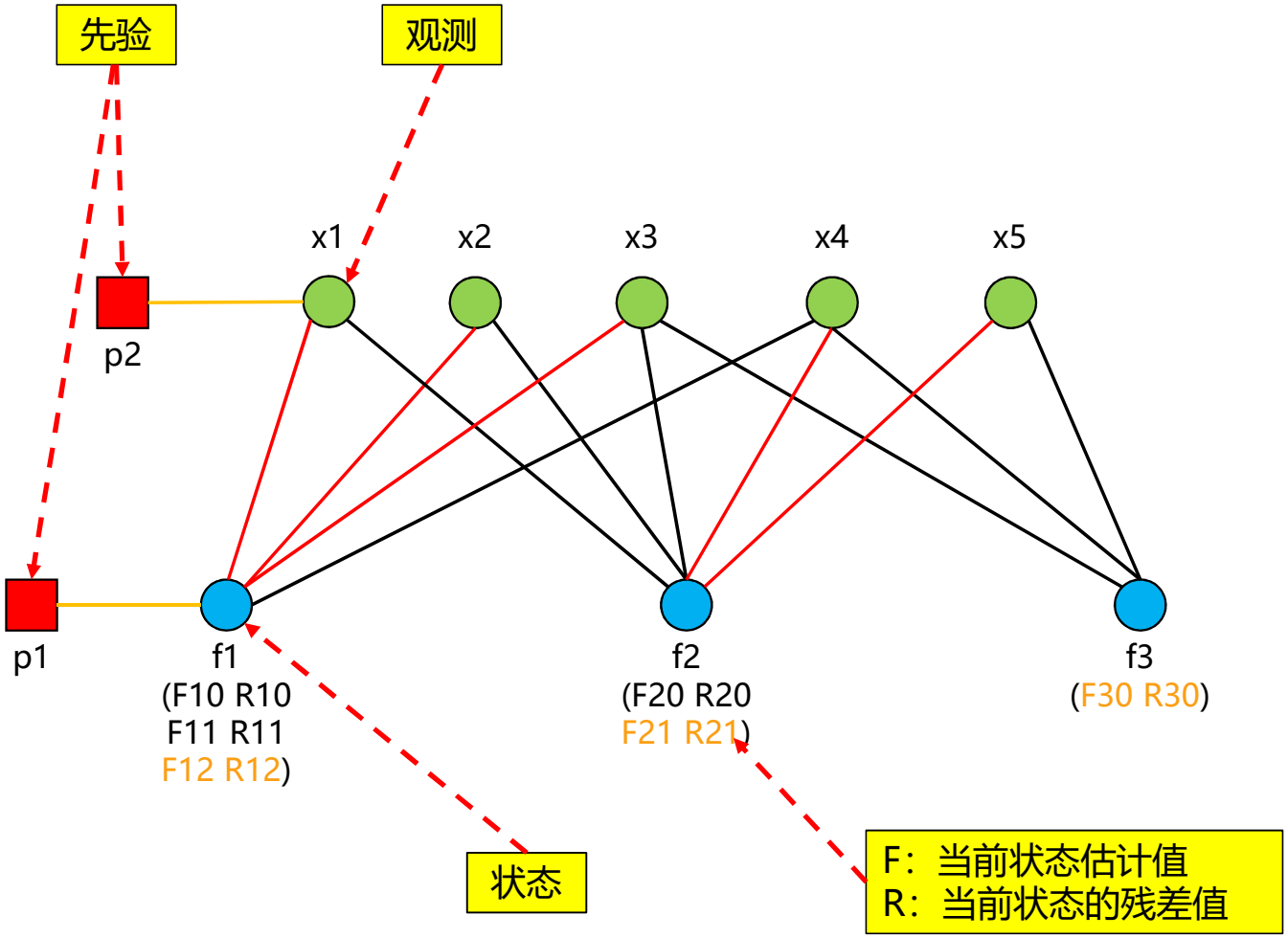
对于HM和bM进行边缘化（消掉边缘化的帧）操作。

fixLinearizationF()

因为状态需要固定在线性化点位置，边缘化一个点前会重新线性化一次，这时得到的resF是使用最新状态线性化的，使用该函数减去J×delta得到在线性化点状态的residual。

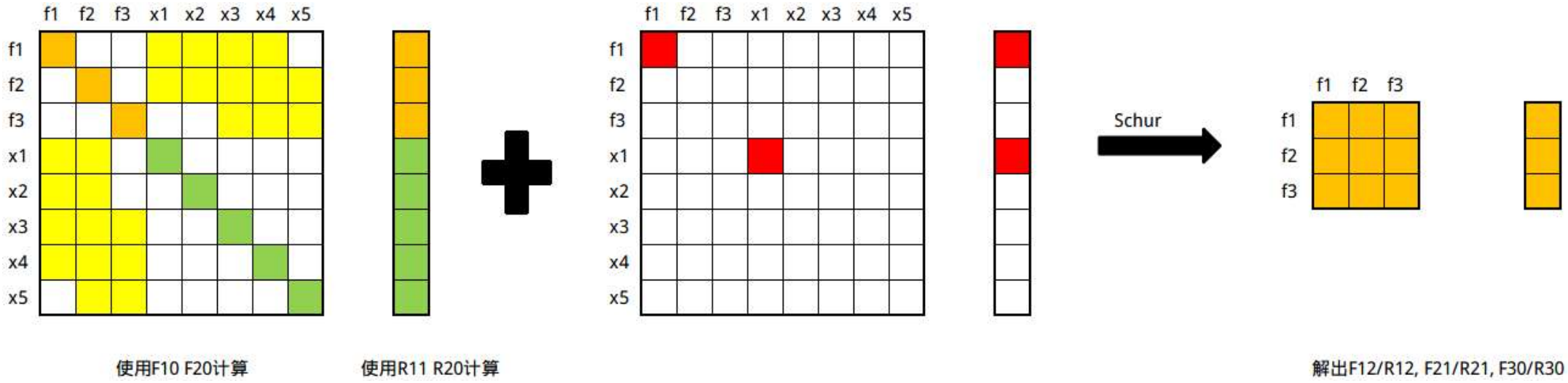
边缘化例子：

f3为新加入的帧，完成优化后得到状态F12/R12, F21/R21, F30/R30，将F30/R30设置为**线性化点**



边缘化例子：

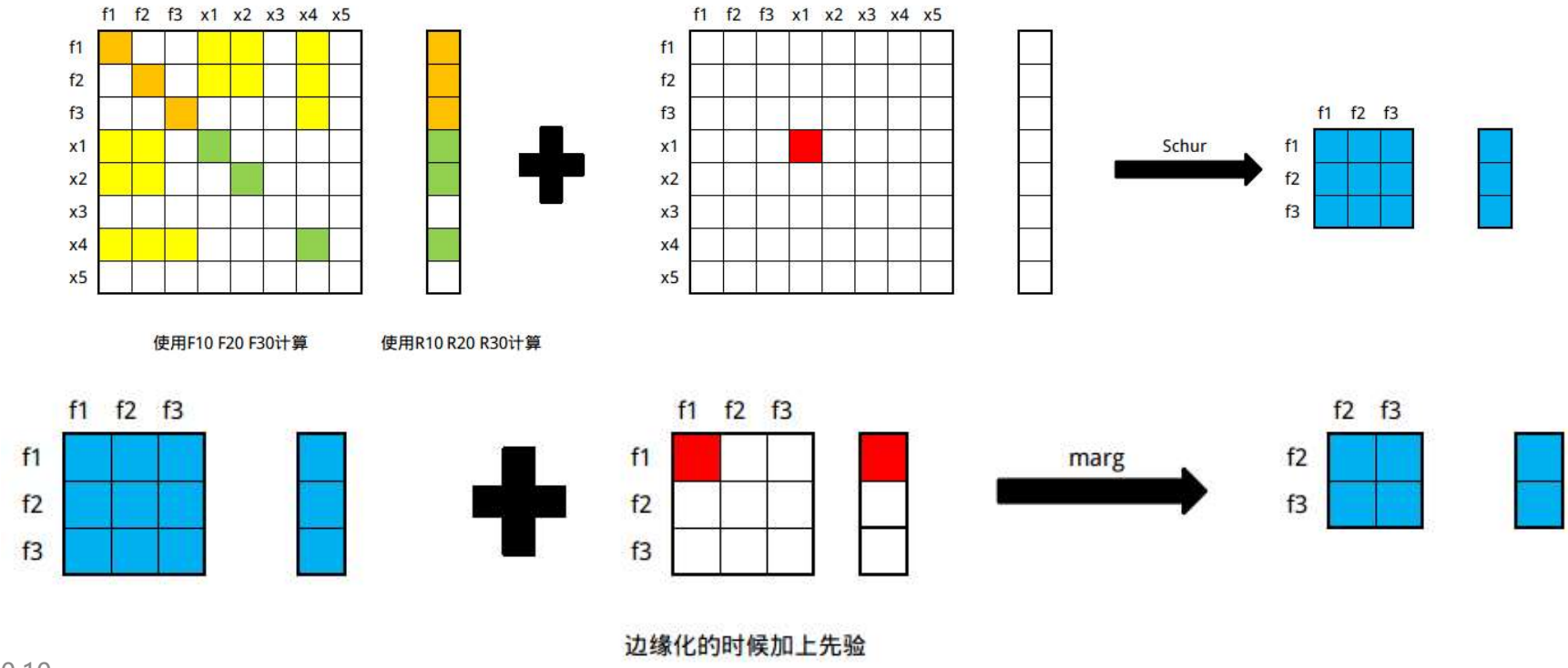
使用Hessian来表示上述过程



边缘化例子：

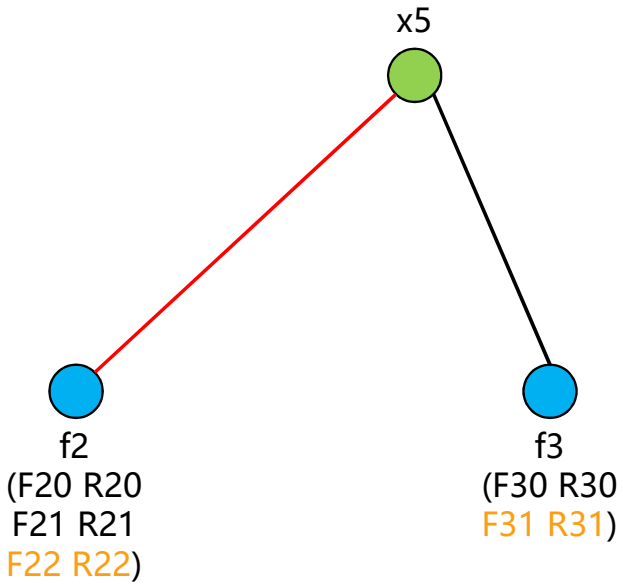
x1、x2、x4边缘化掉，x3丢掉，构造边缘化先验HM，bM

边缘化：



边缘化例子：

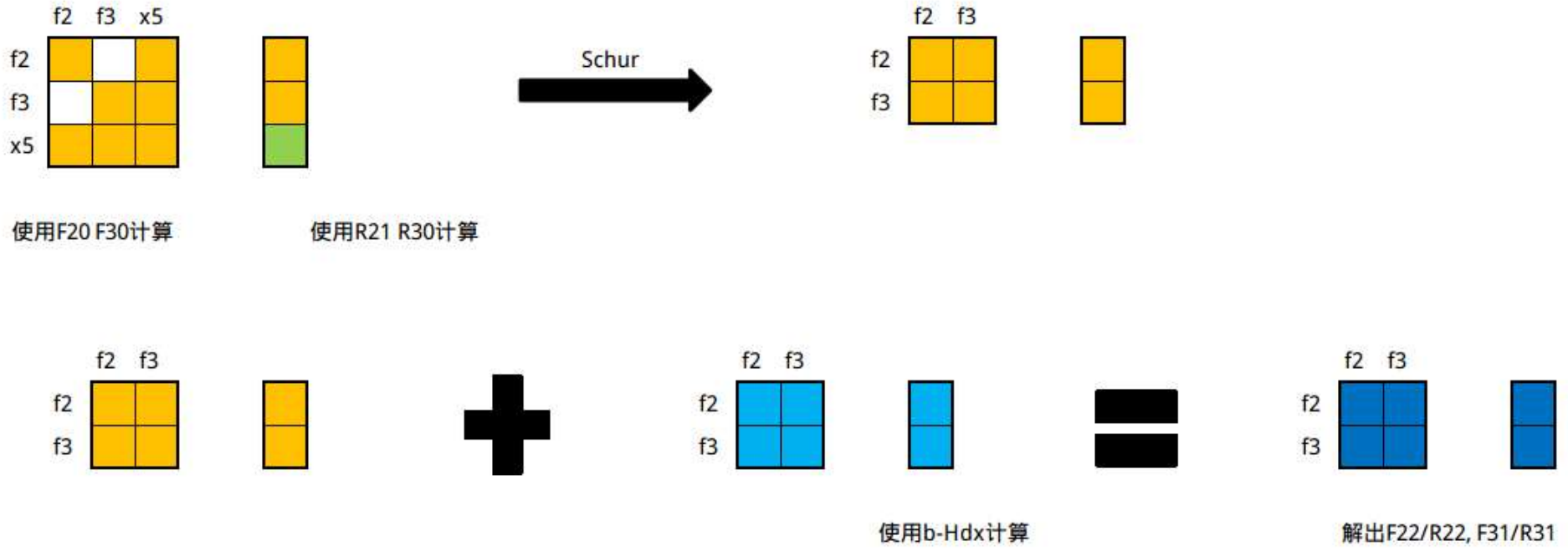
边缘化之后会得到边缘化的先验矩阵，然后求解得状态：F22/R22，F31/R31



边缘化例子：

使用Hessian来表示上述过程

边缘化后：



Reference：

[1] Engel J, Koltun V, Cremers D. Direct sparse odometry[J]. IEEE transactions on pattern analysis and machine intelligence, 2017, 40(3): 611-625.
[2] https://blog.csdn.net/xxxlinttp/article/details/90640350
[3] https://www.cnblogs.com/JingeTU/
[4] https://zhuanlan.zhihu.com/p/29177540
[5] https://zhuanlan.zhihu.com/p/74709586
[6] Xiang Gao. Notes on DSO. October 4, 2018

DSO注释：https://github.com/alalagong/DSO
在线光度标定注释：https://github.com/alalagong/online_photometric_calibration

谢 谢 聆 听

2019.10.10