2.Bundle Adjustment

2.1 文献阅读

1.为何说Bundle Adjustment is slow是不对的？

Such statements often appear in papers introducing yet another heuristic Structure from Motion (SFM) iteration. The claimed slowness is almost always due to the unthinking use of a general-purpose optimization routine that completely ignores the problem structure and sparseness. Real bundle routines are much more efficient than this, and usually considerably more efficient and flexible than the newly suggested mothod. That is why bundle adjustment remains the dominant structure refinement technique for real applications, after 40 years of research.
总结：忽略了问题本身的结构与稀疏性就可能导致"Bundle Adjustment is slow"。直接对H矩阵求逆来计算增量方程，会消耗很多计算资源，实际上，由于H具有稀疏性，是可以利用加速技巧来进行求解的。

2.BA中有哪些需要注意参数化的地方？Pose和Point各有哪些参数化方式？有何优缺点。

The bundle adjustment parameter space is generally a high-dimensional nonlinear manifold, which is a large Cartesian product of projective 3D feature, 3D rotation, and camera calibration manifolds, perhaps with nonlinear constraints, etc.
需要参数化的有3D points(也就是路标点y)，3D Rotation(也就是相机位姿R,t)，相机校准(camera calibration,内参数)，投影后的像素坐标(u,v)等

The many variants on Euler angles for ratations, the singularity of affine point coordinates at infinity, quaternions with ||q||^2 = 1, homogeneous projective quantities have a scale factor freedom, etc.
Pose: 变换矩阵、欧拉角、四元数
欧拉角的优点在于非常直观，缺点是会产生万向锁问题；变换矩阵的优点是描述方便，缺点是产生参数过多，需要16个参数来描述变换过程；四元数的优点是计算方便，没有万向锁问题，缺点是理解困难、不直观。

Point: 三维坐标点(x,y,z)、逆深度
三维坐标点优点是比较简单直观，缺点是无法描述无限远的点；
逆深度优点在于能够建模无穷远点，在实际应用中，逆深度也有较好的数值稳定性。

3.本文写于2000年，但是文中提到的很多内容在后面十几年的研究中得到了验证，你能看到哪

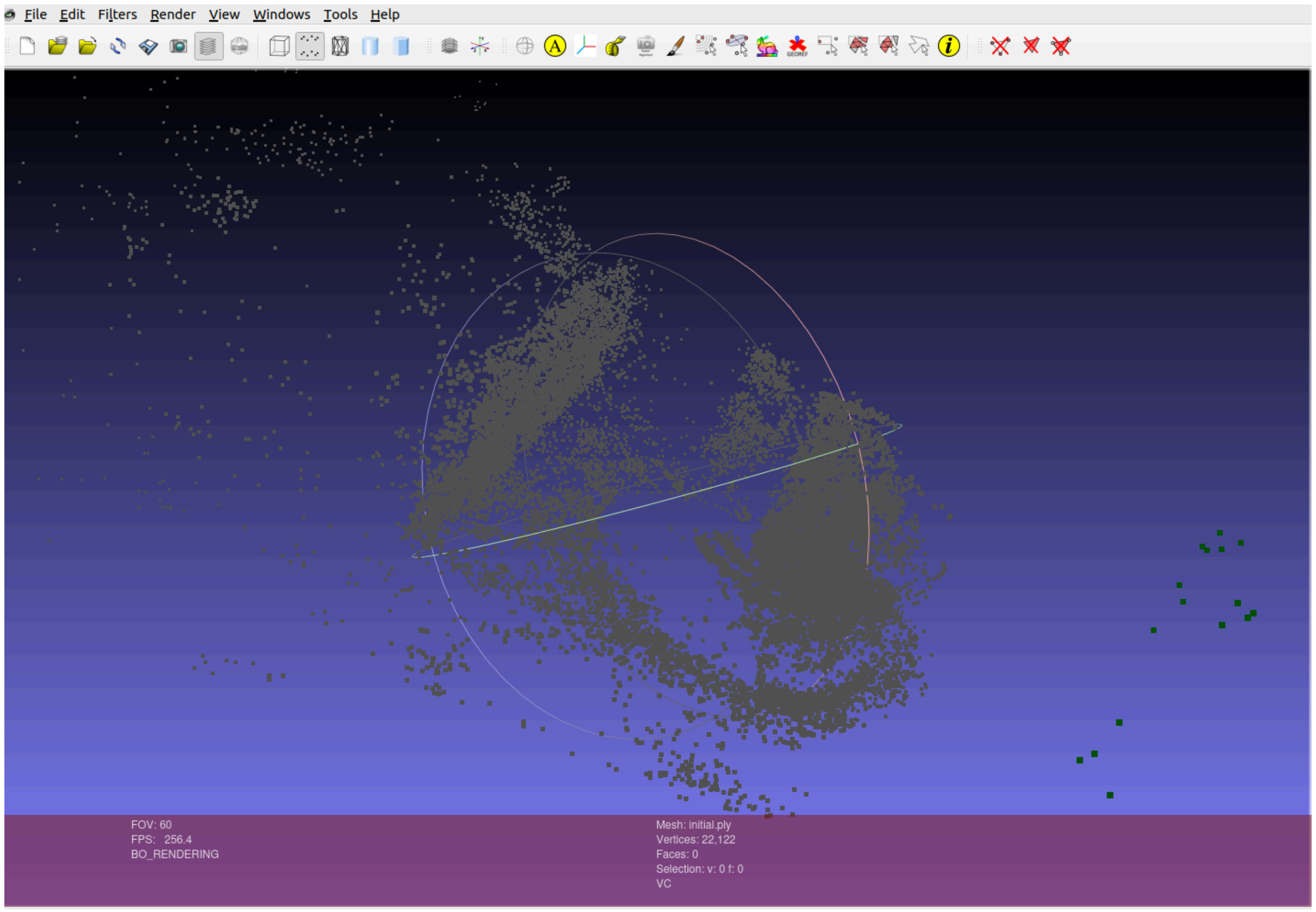
些方向在后续工作中有所体现呢？请举例说明

- Intensity-based方法就是直接法的Bundle Adjustment
- 文中提到的Network Structure对应现在应用比较广泛的图优化方法
- 利用H的稀疏性特性可以实现BA实时，在07年的PTAM上进行了实现

2.2 BAL dataset

选取的数据集为problem-16-22106

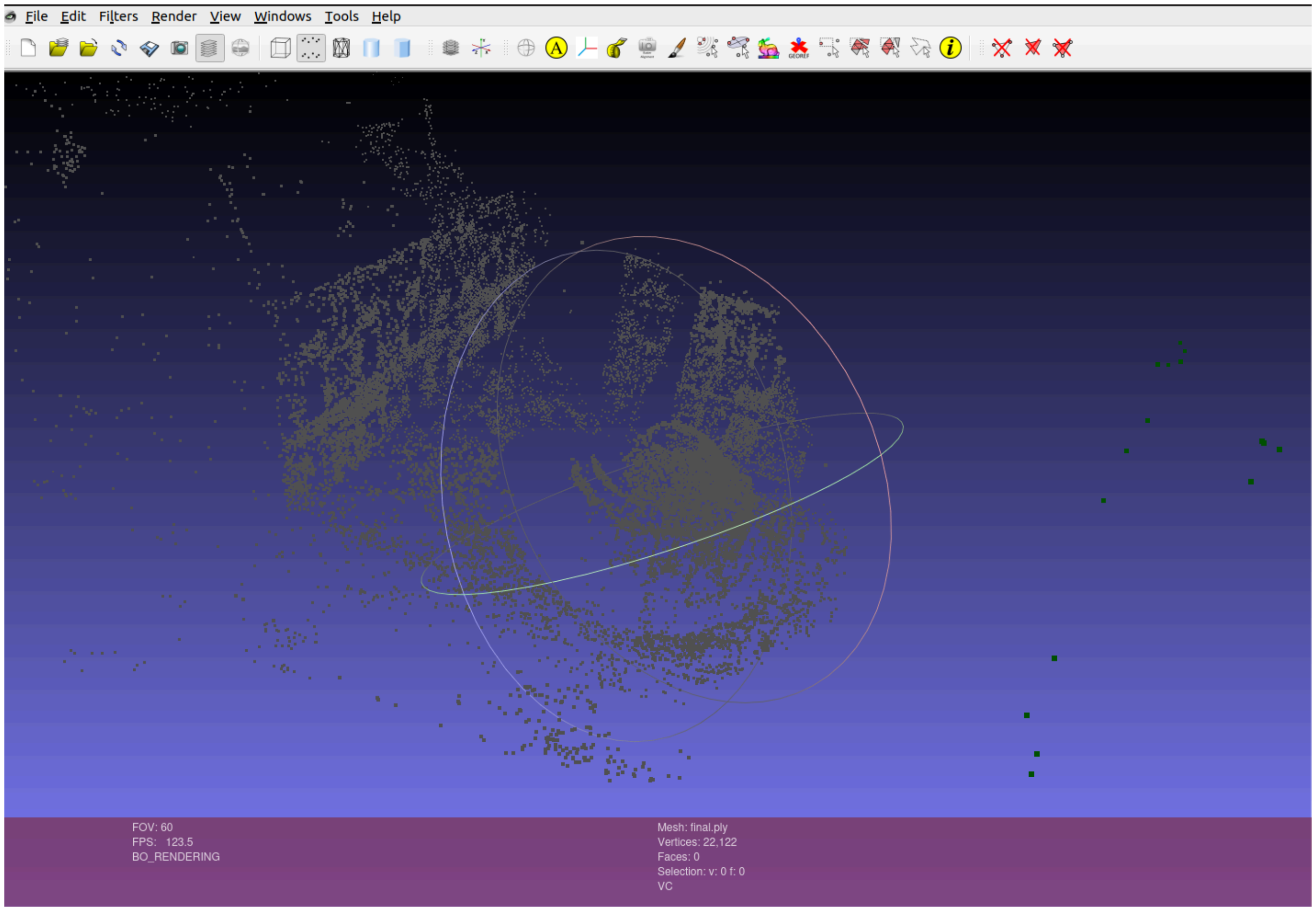
BA之前的点云图为



FOV: 60
FPS: 256.4
BO_RENDERING

Mesh: initial.ply
Vertices: 22,122
Faces: 0
Selection: v: 0 f: 0
VC

BA之后的点云图为



```
FOV: 60
FPS:   123.5
BO_RENDERING
```

```
Mesh: final.ply
Vertices: 22,122
Faces: 0
Selection: v: 0 f: 0
VC
```

可以看出点云的结构特征有了非常明显的改善

3.直接法的Bundle Adjustment

3.1 数学模型

1.如何描述任意一点投影在任一图像中形成的error?

第i个特征点投影在第j个相机中，投影后点周围小块与原始的4x4小块之间灰度值的差异之和

$$e_{ij} = \sum_{W} ||I(p_i) - I_j(\pi(KT_jp_i))||_2^2$$

2.每个error关联几个优化变量?

第i个特征点投影在第j个相机中，关联第i个3D坐标点(3维)和第j个相机位姿(6维)这两个优化变量

3.error关于各变量的雅可比是什么?

$$\frac{\partial u}{\partial P} = \begin{bmatrix} \frac{\partial u}{\partial X} & \frac{\partial u}{\partial Y} & \frac{\partial u}{\partial Z} \\ \frac{\partial v}{\partial X} & \frac{\partial v}{\partial Y} & \frac{\partial v}{\partial Z} \end{bmatrix} = \begin{bmatrix} \frac{f_x}{Z} & 0 & \frac{-f_x X}{Z^2} \\ 0 & \frac{f_y}{Z} & \frac{-f_y Y}{Z^2} \end{bmatrix}$$

$$\frac{\partial P}{\partial \delta\xi} = \begin{bmatrix} I & -P^\wedge \end{bmatrix}$$

$$\frac{\partial u}{\partial \delta\xi} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} & -\frac{f_x XY}{Z^2} & f_x + \frac{f_x X^2}{Z^2} & -\frac{f_x Y}{Z} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} & -f_y - \frac{f_y Y^2}{Z^2} & \frac{f_y XY}{Z^2} & \frac{f_y X}{Z} \end{bmatrix}$$

对位姿ξ_j的Jacobian为：

$$J_{\xi_j} = -\frac{\partial I_j}{\partial u}\frac{\partial u}{\partial \delta\xi}$$

对三维点P_i的Jacobian为：

$$J_{P_i} = -\frac{\partial I_j}{\partial u}\frac{\partial u}{\partial P}$$

3.2 实现

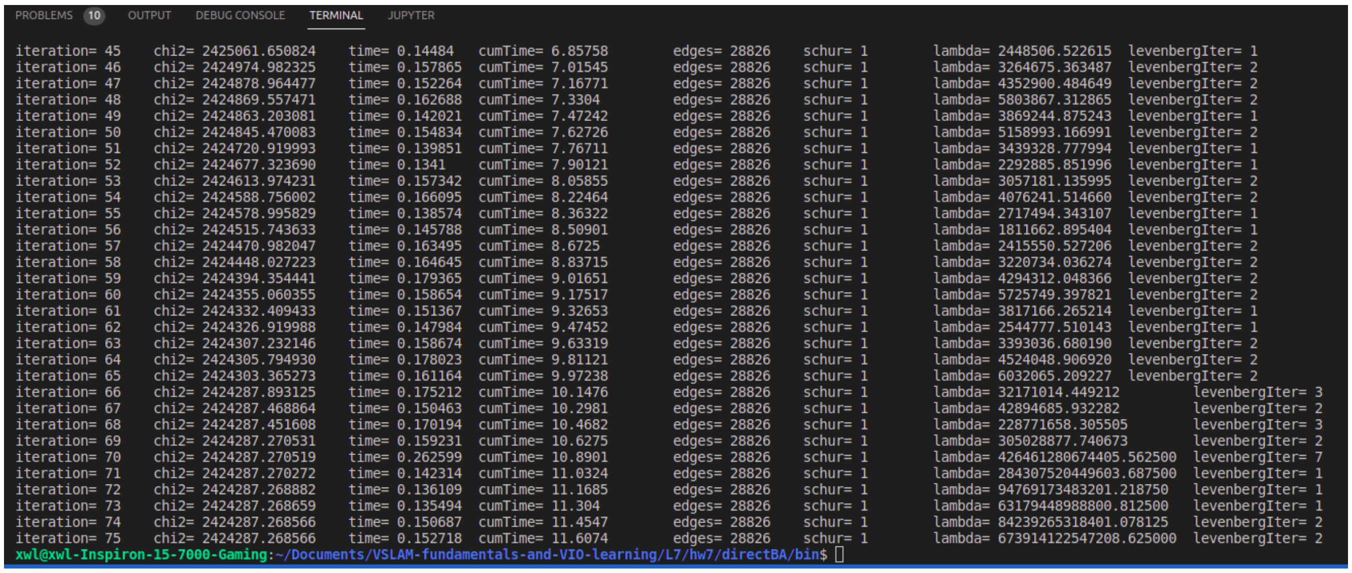
1.能否不要以$[x, y, z]^T$的形式参数化每个点?
可以用逆深度的方式参数化

2.取4x4的patch好吗?取更大的patch好还是取小一点的patch好?
挺好的。
取更大的patch增加计算量；更小的patch鲁棒性变差

3.从本题中，你看到直接法与特征点法在BA阶段有何不同?
直接法计算光度误差，特征点法计算重投影误差，误差形式和雅可比形式不一样。

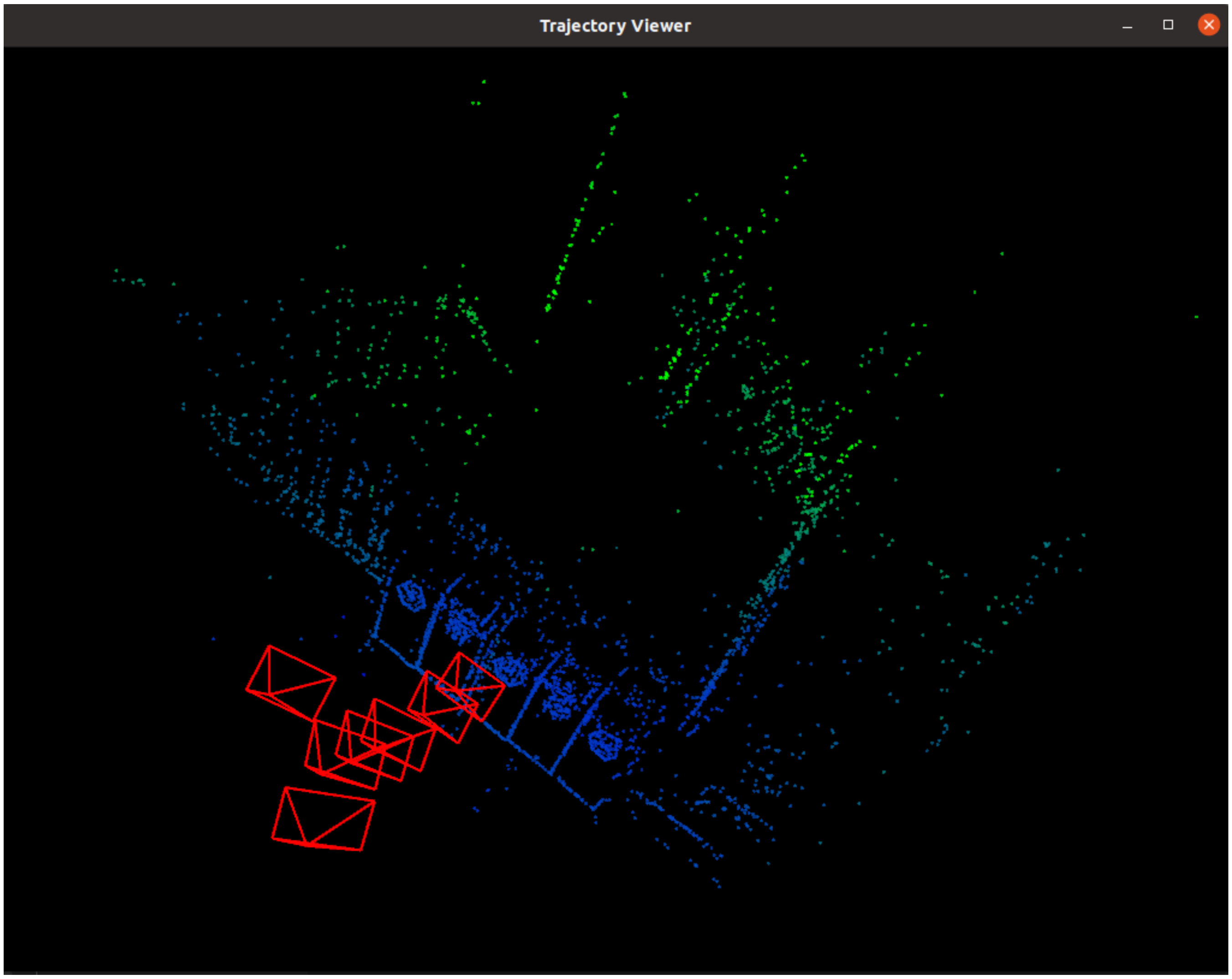
4.由于图像的差异，你可能需要鲁棒核函数，例如Huber。此时Huber的阈值如何选取?
可以先计算一步得到总误差除以点数获得一个平均误差，再根据该值设置Huber的阈值

程序运行结果为：



```
iteration= 45    chi2= 2425061.650824    time= 0.14484   cumTime= 6.85758        edges= 28826    schur= 1        lambda= 2448506.522615  levenbergIter= 1
iteration= 46    chi2= 2424974.982325    time= 0.157865  cumTime= 7.01545        edges= 28826    schur= 1        lambda= 3264675.363487  levenbergIter= 2
iteration= 47    chi2= 2424878.964477    time= 0.152264  cumTime= 7.16771        edges= 28826    schur= 1        lambda= 4352900.484649  levenbergIter= 2
iteration= 48    chi2= 2424869.557471    time= 0.162688  cumTime= 7.3304         edges= 28826    schur= 1        lambda= 5803867.312865  levenbergIter= 2
iteration= 49    chi2= 2424863.203081    time= 0.142021  cumTime= 7.47242        edges= 28826    schur= 1        lambda= 3869244.875243  levenbergIter= 1
iteration= 50    chi2= 2424845.470083    time= 0.154834  cumTime= 7.62726        edges= 28826    schur= 1        lambda= 5158993.166991  levenbergIter= 2
iteration= 51    chi2= 2424720.919993    time= 0.139851  cumTime= 7.76711        edges= 28826    schur= 1        lambda= 3439328.777994  levenbergIter= 1
iteration= 52    chi2= 2424677.323690    time= 0.1341    cumTime= 7.90121        edges= 28826    schur= 1        lambda= 2292885.851996  levenbergIter= 1
iteration= 53    chi2= 2424613.974231    time= 0.157342  cumTime= 8.05855        edges= 28826    schur= 1        lambda= 3057181.135995  levenbergIter= 2
iteration= 54    chi2= 2424588.756002    time= 0.166095  cumTime= 8.22464        edges= 28826    schur= 1        lambda= 4076241.514660  levenbergIter= 2
iteration= 55    chi2= 2424578.995829    time= 0.138574  cumTime= 8.36322        edges= 28826    schur= 1        lambda= 2717494.343107  levenbergIter= 1
iteration= 56    chi2= 2424515.743633    time= 0.145788  cumTime= 8.50901        edges= 28826    schur= 1        lambda= 1811662.895404  levenbergIter= 1
iteration= 57    chi2= 2424470.982047    time= 0.163495  cumTime= 8.6725         edges= 28826    schur= 1        lambda= 2415550.527206  levenbergIter= 2
iteration= 58    chi2= 2424448.027223    time= 0.164645  cumTime= 8.83715        edges= 28826    schur= 1        lambda= 3220734.036274  levenbergIter= 2
iteration= 59    chi2= 2424394.354441    time= 0.179365  cumTime= 9.01651        edges= 28826    schur= 1        lambda= 4294312.048366  levenbergIter= 2
iteration= 60    chi2= 2424355.060355    time= 0.158654  cumTime= 9.17517        edges= 28826    schur= 1        lambda= 5725749.397821  levenbergIter= 2
iteration= 61    chi2= 2424332.409433    time= 0.151367  cumTime= 9.32653        edges= 28826    schur= 1        lambda= 3817166.265214  levenbergIter= 1
iteration= 62    chi2= 2424326.919988    time= 0.147984  cumTime= 9.47452        edges= 28826    schur= 1        lambda= 2544777.510143  levenbergIter= 1
iteration= 63    chi2= 2424307.232146    time= 0.158674  cumTime= 9.63319        edges= 28826    schur= 1        lambda= 3393036.680190  levenbergIter= 2
iteration= 64    chi2= 2424305.794930    time= 0.178023  cumTime= 9.81121        edges= 28826    schur= 1        lambda= 4524048.906920  levenbergIter= 2
iteration= 65    chi2= 2424303.365273    time= 0.161164  cumTime= 9.97238        edges= 28826    schur= 1        lambda= 6032065.209227  levenbergIter= 2
iteration= 66    chi2= 2424287.893125    time= 0.175212  cumTime= 10.1476        edges= 28826    schur= 1        lambda= 32171014.449212         levenbergIter= 3
iteration= 67    chi2= 2424287.468864    time= 0.150463  cumTime= 10.2981        edges= 28826    schur= 1        lambda= 42894685.932282         levenbergIter= 2
iteration= 68    chi2= 2424287.451608    time= 0.170194  cumTime= 10.4682        edges= 28826    schur= 1        lambda= 228771658.305505        levenbergIter= 3
iteration= 69    chi2= 2424287.270531    time= 0.159231  cumTime= 10.6275        edges= 28826    schur= 1        lambda= 305028877.740673        levenbergIter= 2
iteration= 70    chi2= 2424287.270519    time= 0.262599  cumTime= 10.8901        edges= 28826    schur= 1        lambda= 426461280674405.562500  levenbergIter= 7
iteration= 71    chi2= 2424287.270272    time= 0.142314  cumTime= 11.0324        edges= 28826    schur= 1        lambda= 284307520449603.687500  levenbergIter= 1
iteration= 72    chi2= 2424287.268882    time= 0.136109  cumTime= 11.1685        edges= 28826    schur= 1        lambda= 94769173483201.218750   levenbergIter= 1
iteration= 73    chi2= 2424287.268659    time= 0.135494  cumTime= 11.304         edges= 28826    schur= 1        lambda= 63179448988800.812500   levenbergIter= 1
iteration= 74    chi2= 2424287.268566    time= 0.150687  cumTime= 11.4547        edges= 28826    schur= 1        lambda= 84239265318401.078125   levenbergIter= 2
iteration= 75    chi2= 2424287.268566    time= 0.152718  cumTime= 11.6074        edges= 28826    schur= 1        lambda= 673914122547208.625000  levenbergIter= 2
xwl@xwl-Inspiron-15-7000-Gaming:~/Documents/VSLAM-fundamentals-and-VIO-learning/L7/hw7/directBA/bin$ 
```

BA之后的效果如下：


Trajectory Viewer