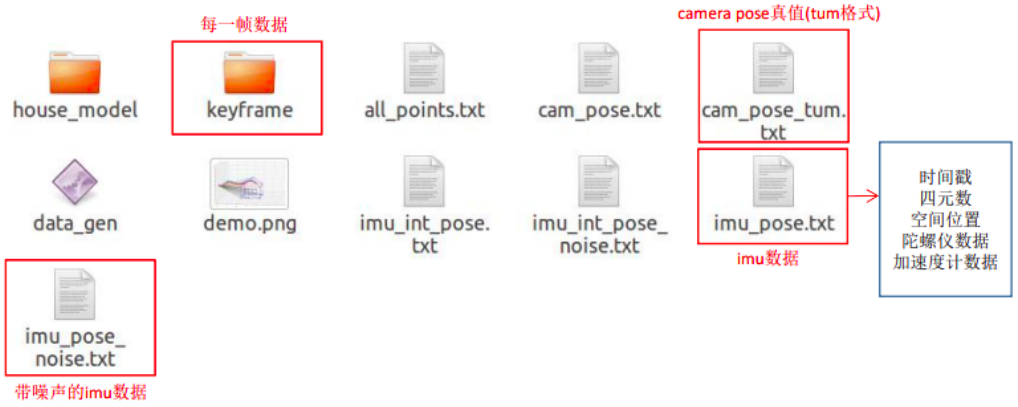
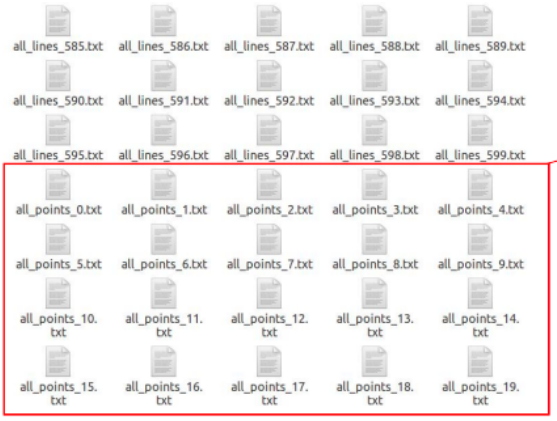
本次作业需要大家自己对于提供的代码框架进行“魔改”，来适应仿真数据。其实，采用仿真数据将会简化代码的处理逻辑，但是如果简化也需要基于大家对于代码和数据的理解。

本次作业的仿真数据次使用第二章的代码，建议同学们先看源码理解数据生成的流程，以及代码里定义的一些参数。

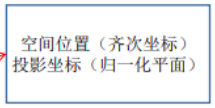
编译并运行第2章作业的vio_data_simulation，得到如下文件：



keyframe文件夹：



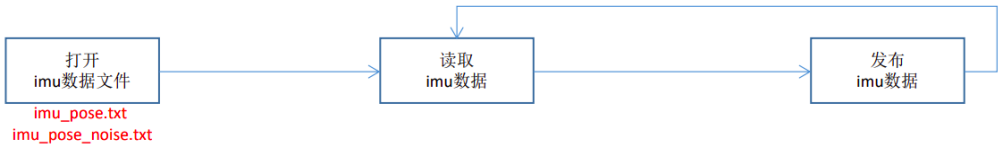
all_points_xx.txt是第xx帧观测到的空间点数据



1、直接获得归一化平面上的特征点坐标；

2、可以根据空间位置给points设置id，有了id就可以匹配上一帧的观测，也可以计算光流；

在对于数据的意义和生成过程理解到位后，我们可以对于vio系统接入imu和图像数据了

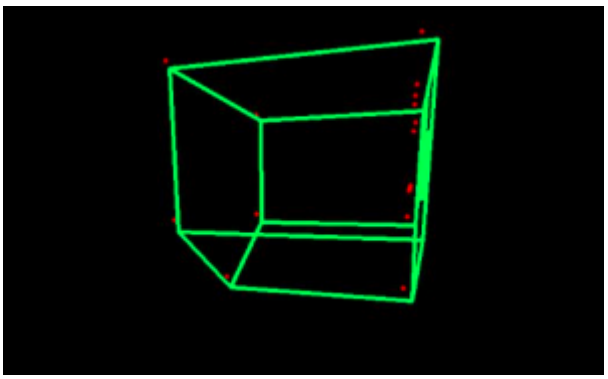



```
fsImu.open(sImu_data_file.c_str());
```

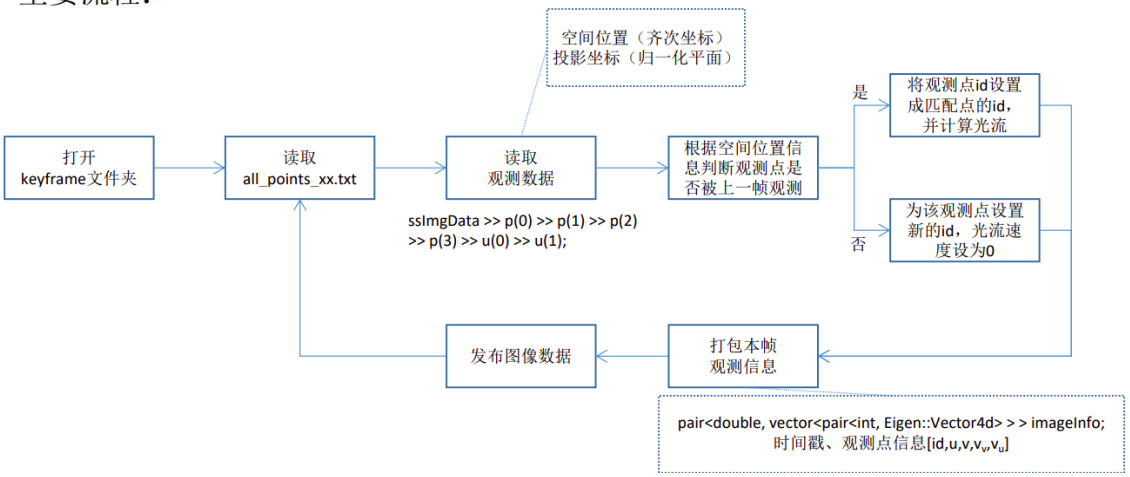
```
ssImuData >> dStampNSec >> q.w() >> q.x() >> q.y()
>> q.z() >> t(0) >> t(1) >> t(2) >> vGyr.x() >> vGyr.y()
>> vGyr.z() >> vAcc.x() >> vAcc.y() >> vAcc.z();
```

```
pSystem->PubImuData(dStampNSec, vGyr, vAcc);
```

图像数据的可视化效果大家可以看下



主要流程：



发布图像：

由于我们已将观测信息打包了，System::PubImageData不再适用，可以重载这个函数，并作简单修改，然后将观测信息填入feature_points即可。

```
for (size_t i = 0; i < Imageinfo.second.size(); i++)
{
    int p_id = Imageinfo.second[i].first;
    double x = Imageinfo.second[i].second( index: 0);
    double y = Imageinfo.second[i].second( index: 1);
    double z = 1;
    double velocity_x = Imageinfo.second[i].second( index: 2);
    double velocity_y = Imageinfo.second[i].second( index: 3);

    feature_points->points.push_back(Vector3d(x, y, z));
    feature_points->id_of_point.push_back(p_id);
    feature_points->u_of_point.push_back(x);
    feature_points->v_of_point.push_back(y);
    feature_points->velocity_x_of_point.push_back(velocity_x);
    feature_points->velocity_y_of_point.push_back(velocity_y);
}
```

在修改了前端的接口后，后端要修改的其实就没了，轨迹效果如下


```
338             if (estimator.solver_flag == Estimator::SolverFlag::NON_LINEAR)
339             {
340                 Vector3d p_wi;
341                 Quaterniond q_wi;
342                 q_wi = Quaterniond(estimator.Rs[WINDOW_SIZE]);
343                 p_wi = estimator.Ps[WINDOW_SIZE];
344                 vPath_to_draw.push_back(p_wi);
345                 double dStamp = estimator.Headers[WINDOW_SIZE];
346                 cout << "1 BackEnd processImage dt: " << fixed << t_processImage.toc() << " stamp: " <<  dStamp << " p_wi: " << p_wi.transpose() << endl;
347                 ofs_pose << fixed << dStamp << " " << p_wi(0) << " " << p_wi(1) << " " << p_wi(2) << " "
348                         << q_wi.w() << " " << q_wi.x() << " " << q_wi.y() << " " << q_wi.z() << endl;
349             }
350         }
351         m_estimator.unlock();
```

本行代码将非初始化阶段的camera位姿输出到了pose_output.txt中，可以使用evo工具评估精度，命令如下：

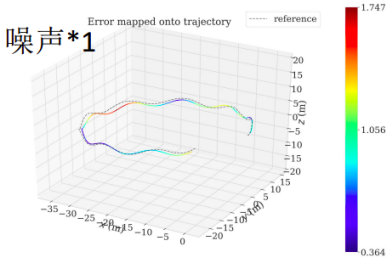
evo_ape tum pose_output.txt cam_pose_tum.txt -va --plot --plot_mode xyz

参考链接：

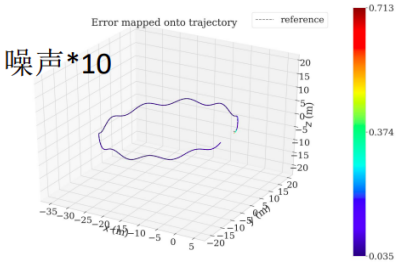
C++文件读写：
https://www.cplusplus.com/reference/fstream/fstream/
Evo评估
https://github.com/MichaelGrupp/evo

最后的实验·结果·1当然是有噪声时轨迹估计会有较大误差

噪声*1



噪声*10



噪声*100

	max	mean	min	rmse
噪声*1	1.746751	0.944979	0.364448	1.004950
噪声*10	0.712855	0.072721	0.035496	0.095686
噪声*100	1.055822	0.124855	0.029540	0.172392