2 Bundle Adjustment (5 分，约 3 小时)

2.1 文献阅读 (2 分)

1. 为何说 Bundle Adjustment is slow 是不对的?

BA is slow 是因为考虑这个问题的时候只考虑了的 BA 常规形式，没有考虑具体 BA 问题的结构和稀疏性。

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

BA 问题中点的位移，旋转，甚至相机内参都能参数化。根据具体的形式，我们的状态可能受到奇异性，内部约束和不必要的内部自由度的影响。最合适的参数化应该要做到统一，有限，行为良好且接近当前状态。
对于点来说，可以参数化成 XYZ（仿射形式）或者 XYZW（射影形式），在距离相机近的点我们使用 xyz 形式就可以了。但是对于很远的点，因为很远的点和无限远的点本身就不容易区分，还加上噪声问题就可能让我们混肴。在很远的位置，xyz 的 cost function 十分的平坦，steps 会变得很大。用 XYZW 形式的话它会变得自然，有限，条件良好。
对于 Pose（位移部分跟点同理）的旋转部分。用 Euler angle 就有很多数值问题，除非能保证避免奇异性和不平坦空间。可以用单位四元数或者在旋转矩阵 R 上面加个小扰动来表示。

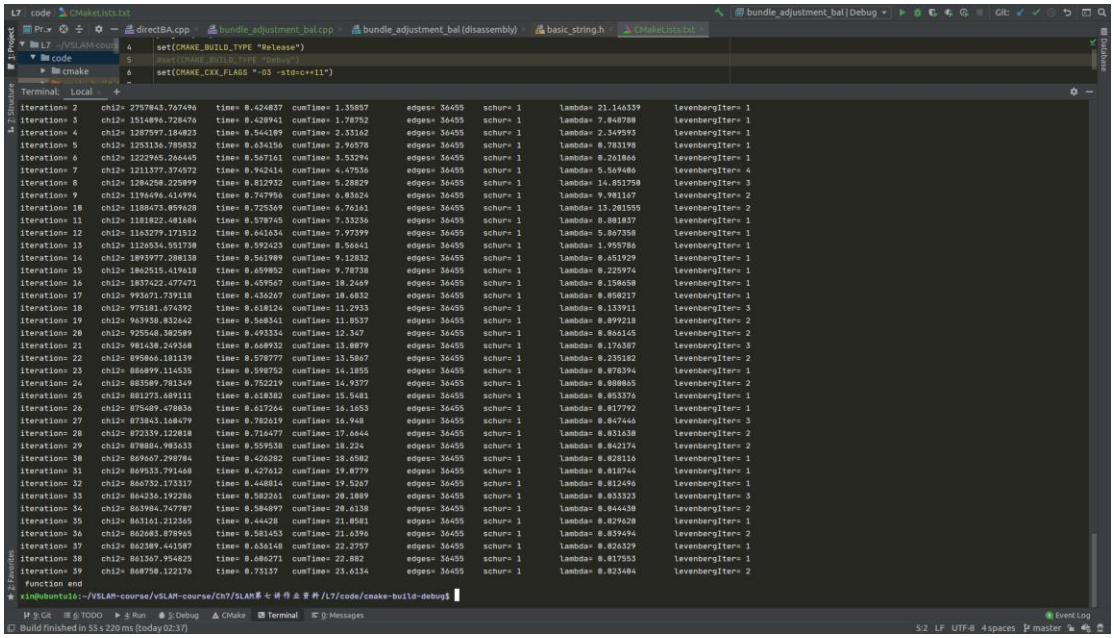
3.* 本文写于 2000 年，但是文中提到的很多内容在后面十几年的研究中得到了印证。你能看到哪些 方向在后续工作中有所体现？请举例说明

模型要考虑 outlier
控制步长
可以探索问题的结构性，善用因式分解，使用稳定的局部参数，缩放和预处理。

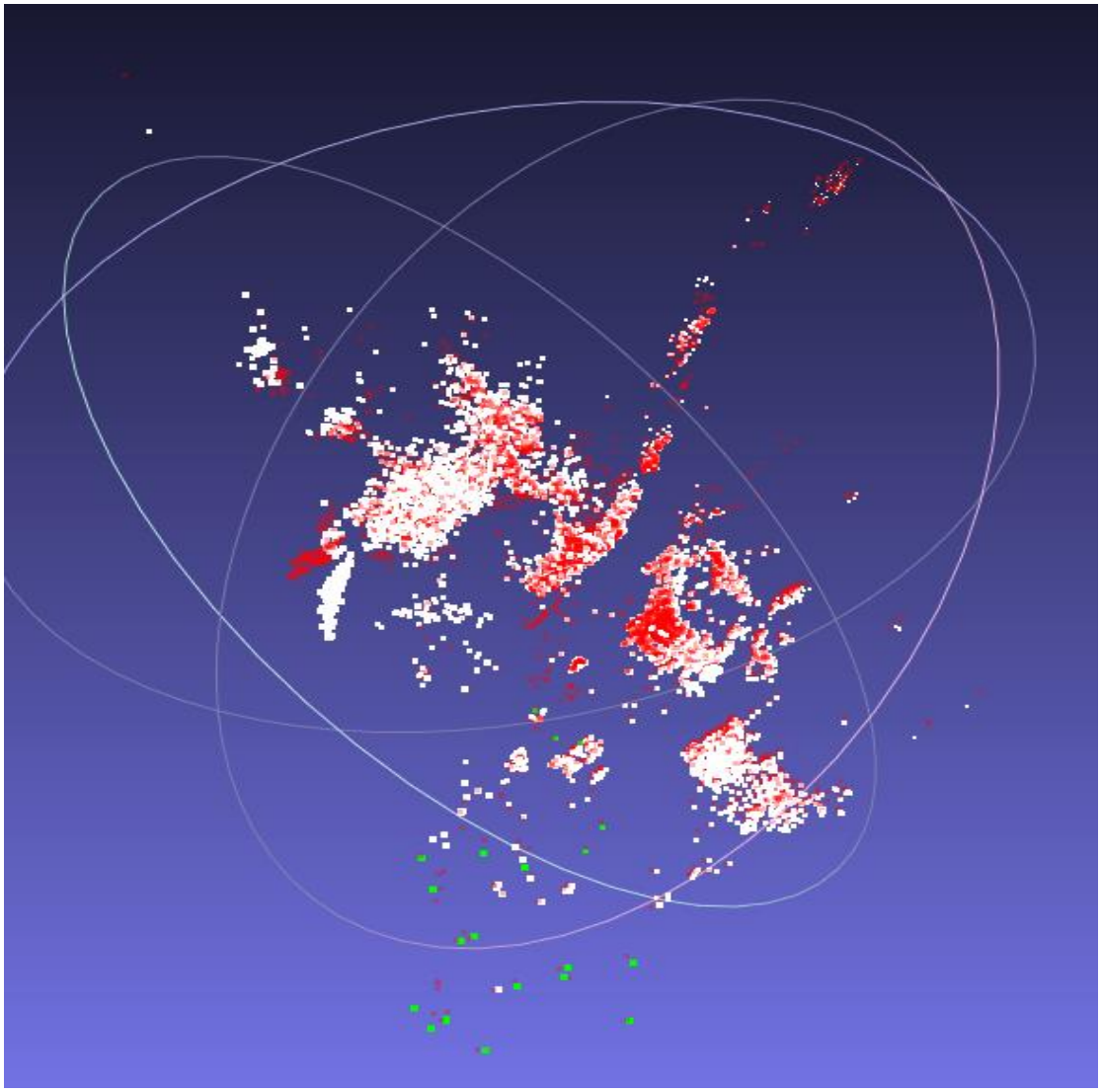
2.2 BAL-dataset（3 分）

请你使用 g2o，自己定义 Vertex 和 Edge 书写 BAL 上的 BA 程序，运行你的 BA，并给出优化后的点云图。

代码参考了书上示例，详情参见.cpp 文件
程序运行结果可见误差在缩小



点云地图，白色为原始位姿，红色为优化后的位姿，主要是同一面上的点云经过优化后都集中到一个平面了。



3 直接法的 Bundle Adjustment (5 分，约 3 小时)

3.1 数学模型

1. 如何描述任意一点投影在任意一图像中形成的 error?

即任意一点$\mathrm{p_j}$投影到某相机$\mathrm{T_i}$的成像平面后，计算该点（及其 patch）的像素和参考像素的差值。

$$\mathrm{error} = \mathrm{z_{ij}} - h(T_i, p_j)$$

H 函数为观测方程

2. 每个 error 关联几个优化变量?

两个，一个是相机位姿(translation 和 rotation 各 3 维)
另一个是点的位置（3 维）

3. error 关于各变量的雅可比是什么

$$\frac{\partial \mathrm{error}}{\partial \mathrm{q}} = \frac{\partial I_2}{\partial u} * \frac{\partial u}{\partial p} = \begin{matrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} \end{matrix} * \mathrm{R}$$

$$\frac{\partial \mathrm{error}}{\partial \mathrm{T}} = \begin{matrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} & -\frac{f_x XY}{Z^2} & f_x + \frac{f_x X^2}{Z^2} & -\frac{f_x Y}{Z} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} & -f_y - \frac{f_x Y^2}{Z^2} & -\frac{f_y XY}{Z^2} & -\frac{f_x X}{Z} \end{matrix}$$

此处 XYZ 为特征点在相机系坐标，R 为世界系到相机系的旋转矩阵。

3.2 实现-根据上述说明，使用 g2o 实现上述优化，并用 pangolin 绘制优化结果

这题遇到了 error，暂时没有解决。

```
// build optimization problem
typedef g2o::BlockSolver<g2o::BlockSolverTraits<6, 3>> DirectBlock;  // 求解的向量是6＊1的
DirectBlock::LinearSolverType *linearSolver = new g2o::LinearSolverDense<DirectBlock::PoseMatrixType>();
DirectBlock *solver_ptr = new DirectBlock(linearSolver);
g2o::OptimizationAlgorithmLevenberg *solver = new g2o::OptimizationAlgorithmLevenberg(solver_ptr); // L-M
```

以下为代码和思路

把估计点投影到相机系统，加上内参放到成像平面，找到 patch 内对应的位置，判断是否出界，结算该像素值和参考像素值的差。

```cpp
virtual void computeError() override {
    // TODO START YOUR CODE HERE
    // compute projection error ...
    float shifterTable[2][16] = {{-2, -2, -2, -2, -1, -1, -1, -1, 0,  0,  0, 0, 1,  1,  1, 1},
                                 {-2, -1, 0,  1,  -2, -1, 0,  1,  -2, -1, 0, 1, -2, -1, 0, 1,}};
    for (int i = 0; i < 16; i++) {
        //拿到指向两个顶点的指针
        g2o::VertexSBAPointXYZ *vertex_point = (g2o::VertexSBAPointXYZ *) _vertices[1];
        VertexSophus *vertex_camera = (VertexSophus *) _vertices[0];
        //估计位姿transform到相机系
        Eigen::Vector3d p_undis = vertex_camera->estimate() * vertex_point->estimate();
        //投影到图像上
        p_undis = -p_undis / p_undis[2];
        float u = fx * p_undis[0] + cx;
        float v = fy * p_undis[1] + cy;
        //找到patch中对应的位置
        u = u + shifterTable[0][i];
        v = v + shifterTable[1][i];
        //越界点跳过
        if (u < 0 || u > this->targetImg.cols || v < 0 || v > this->targetImg.rows) {
            _error[i] = 0;
            continue;
        }
        //投影像素值和固定像素值相减
        float estimate_Pixel = GetPixelValue(this->targetImg, u, v);
        //用_measurement还是用this->origColor?
        _error[i] = this->origColor[i] - estimate_Pixel;
    }
    // END YOUR CODE HERE
}
```

往 optimizer 里面加入各个顶点（特征点和位姿），把边和顶点相关联，加入 optimizer。

```
// TODO add vertices, edges into the graph optimizer
// START YOUR CODE HERE
vector<VertexSophus *> vertex_cameras;
vector<g2o::VertexSBAPointXYZ *> vertex_points;
//往optimizer里面增加顶点
for(int i=0;i<poses.size();i++)
{
    VertexSophus *vc = new VertexSophus();
    vc->setId(i);
    //设置初始值
    vc->setEstimate(poses[i]);
    optimizer.addVertex(vc);
    vertex_cameras.push_back(vc);
}
for(int i=0;i<points.size();i++)
{
    g2o::VertexSBAPointXYZ *vp = new g2o::VertexSBAPointXYZ();
    vp->setId( id: i+poses.size());
    vp->setEstimate(points[i]);
    //vp->setMarginalized(true);
    optimizer.addVertex(vp);
    vertex_points.push_back(vp);
}
//往opitimizer里面增加边
for (int i = 0; i < points.size(); i++) {
    for (int c = 0; c < vertex_cameras.size(); c++) {
        EdgeDirectProjection *edge = new EdgeDirectProjection(color[i], &: images[c]);
        edge->setVertex( i: 0, vertex_cameras[c]);
        edge->setVertex( i: 1, vertex_points[i]);
        Vector16d measurement;
        for(int g = 0;g<16;g++)
        {
            measurement[g] = color[i][g];
        }
        edge->setMeasurement(measurement);
        edge->setInformation( information: Eigen::Matrix<double,16,16>::Identity());
        //防outlier
        edge->setRobustKernel( ptr: new g2o::RobustKernelHuber());
        optimizer.addEdge(edge);
    }
}
// END YOUR CODE HERE
```

把优化完的数据塞回 poses 和 poitns 方便显示函数调用。

```
// TODO fetch data from the optimizer
// START YOUR CODE HERE
for (int i = 0; i < vertex_cameras.size(); i++) {
    VertexSophus *vertexC = vertex_cameras[i];
    auto estimate = vertexC->estimate();
    poses[i] = estimate;
}
for (int i = 0; i < vertex_points.size(); ++i) {
    auto vertex = vertex_points[i];
    points[i] = vertex->estimate();
}
// END YOUR CODE HERE
```

1. 能否不要以 [x,y,z]T 的形式参数化每个点？

参考阅读题，可以参数化为 XYZW 形式

2. 取 4x4 的 patch 好吗？取更大的 patch 好还是取小一点的 patch 好？

应该跟直接法本身的 patch 一样，大了小了都不一定更好。
主要是尽量取到跟"局部光度特征"大小差不多的 patch 最好。

3. 从本题中，你看到直接法与特征点法在 BA 阶段有何不同？

Error 原理不同，直接法是光度误差，特征点法是特征点的重投影误差。
Error 不同之后求偏导的各项都会不一样。

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？

这应该是个经验值吧，考虑到像素变化不应该太大，要我的话可能会取 255/5=51。