中国科学院光电信息处理重点实验室
Key Laboratory of Opto-Electronic Information Processing,CAS

KinectFusion 和 ElasticFusion 三维重建方法

付兴银

2016/8/23

实时三维重建 和 SLAM 对比

基于RGB-D实时三维重建

直接法 ICP 特征匹配（GPU）

对点云融合去冗余操作

更关注重建的精度

位姿/闭环约束 优化重建 MAP（通过优化轨迹优化 Map 或者直接优化 Map）

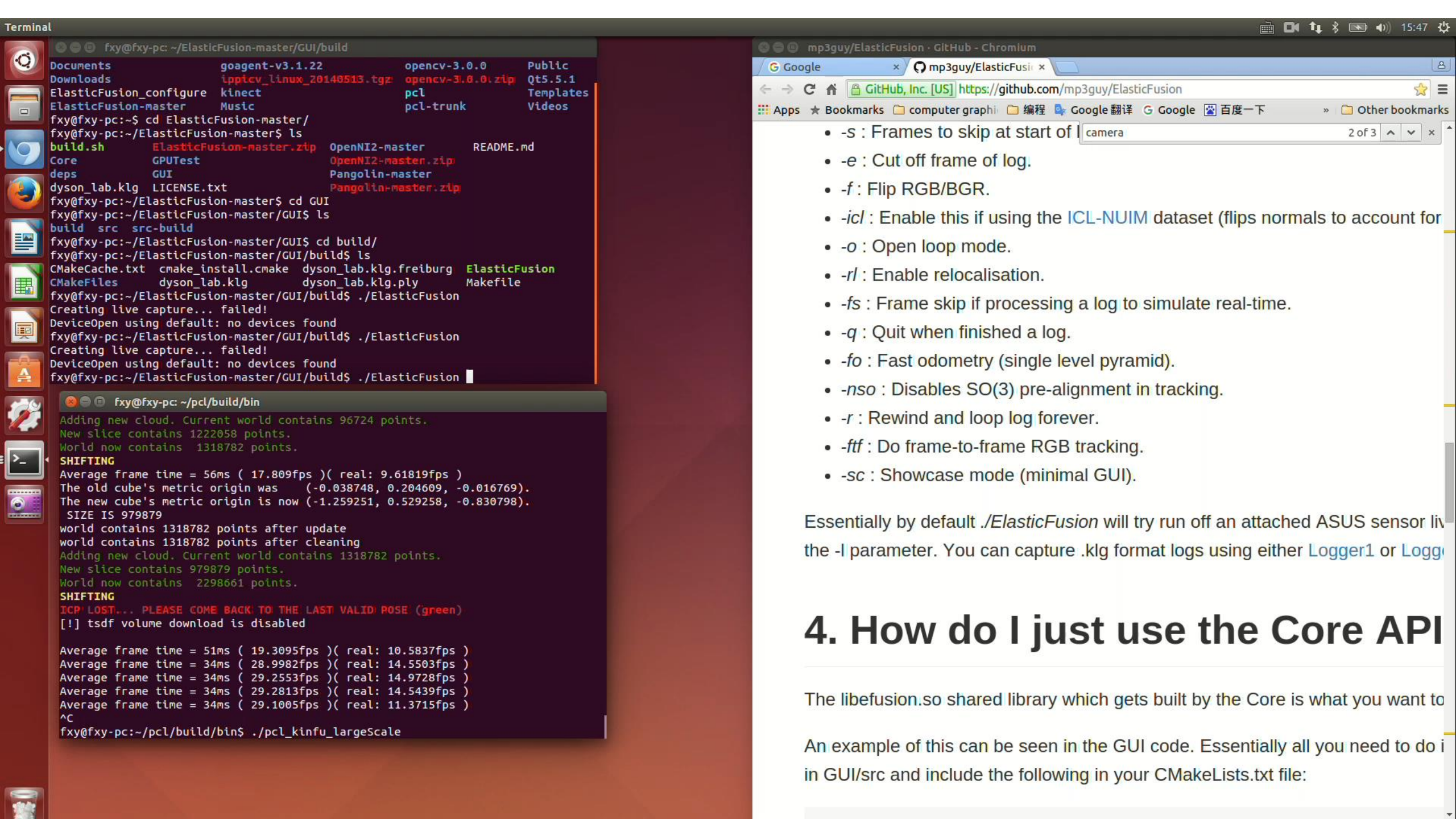
SLAM

特征配准 直接法 ICP（CPU）

简单将点堆在一起

更关注轨迹求解的精度

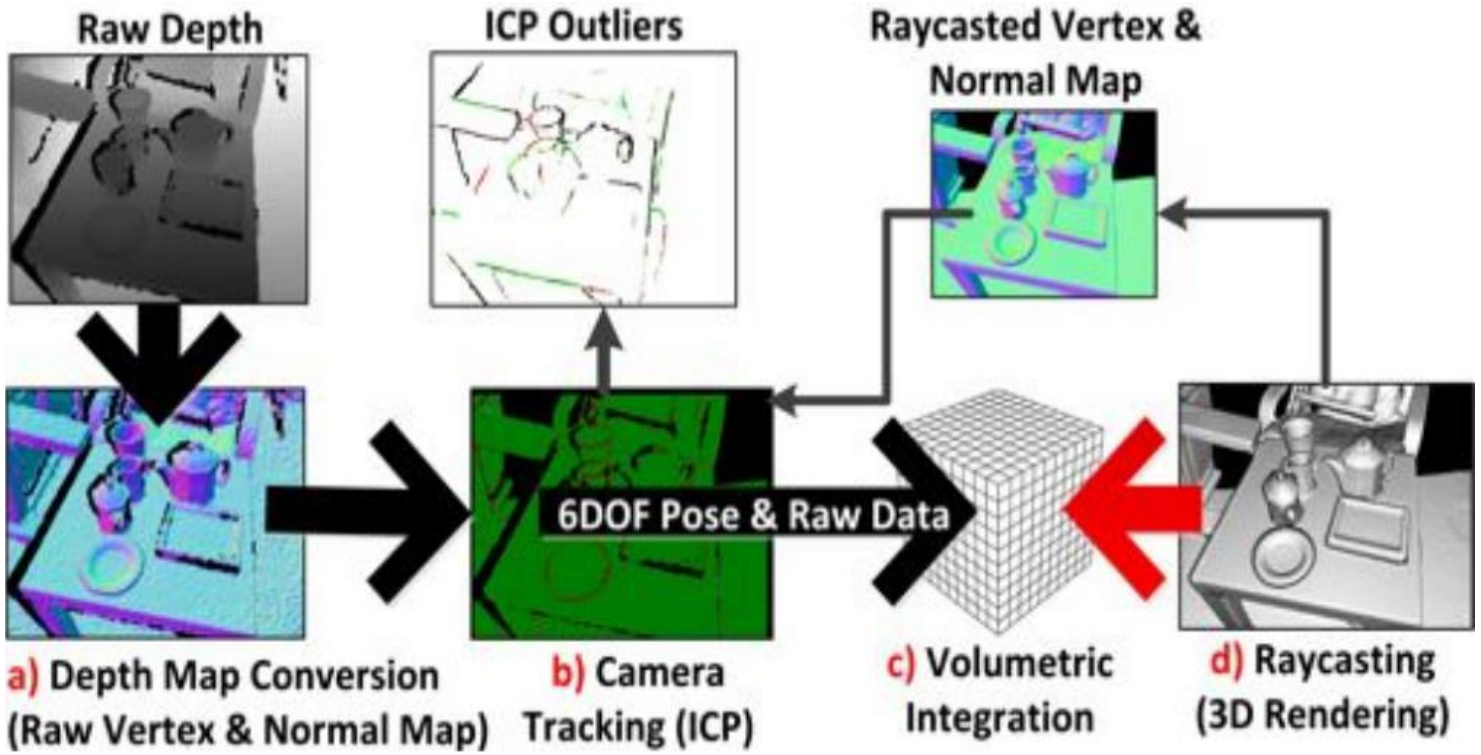
位姿/特征点约束用来优化相机轨迹


Terminal
15:47
fxy@fxy-pc: ~/ElasticFusion-master/GUI/build
Documents goagent-v3.1.22 opencv-3.0.0 Public
Downloads ippicv_linux_20140513.tgz opencv-3.0.0.zip Qt5.5.1
ElasticFusion_configure kinect pcl Templates
ElasticFusion-master Music pcl-trunk Videos
fxy@fxy-pc:~$ cd ElasticFusion-master/
fxy@fxy-pc:~/ElasticFusion-master$ ls
build.sh ElasticFusion-master.zip OpenNI2-master README.md
Core GPUTest OpenNI2-master.zip
deps GUI Pangolin-master
dyson_lab.klg LICENSE.txt Pangolin-master.zip
fxy@fxy-pc:~/ElasticFusion-master$ cd GUI
fxy@fxy-pc:~/ElasticFusion-master/GUI$ ls
build src src-build
fxy@fxy-pc:~/ElasticFusion-master/GUI$ cd build/
fxy@fxy-pc:~/ElasticFusion-master/GUI/build$ ls
CMakeCache.txt cmake_install.cmake dyson_lab.klg.freiburg ElasticFusion
CMakeFiles dyson_lab.klg dyson_lab.klg.ply Makefile
fxy@fxy-pc:~/ElasticFusion-master/GUI/build$ ./ElasticFusion
Creating live capture... failed!
DeviceOpen using default: no devices found
fxy@fxy-pc:~/ElasticFusion-master/GUI/build$ ./ElasticFusion
Creating live capture... failed!
DeviceOpen using default: no devices found
fxy@fxy-pc:~/ElasticFusion-master/GUI/build$ ./ElasticFusion
fxy@fxy-pc: ~/pcl/build/bin
Adding new cloud. Current world contains 96724 points.
New slice contains 1222058 points.
World now contains 1318782 points.
SHIFTING
Average frame time = 56ms (17.809fps)(real: 9.61819fps)
The old cube's metric origin was (-0.038748, 0.204609, -0.016769).
The new cube's metric origin is now (-1.259251, 0.529258, -0.830798).
SIZE IS 979879
world contains 1318782 points after update
world contains 1318782 points after cleaning
Adding new cloud. Current world contains 1318782 points.
New slice contains 979879 points.
World now contains 2298661 points.
SHIFTING
ICP LOST... PLEASE COME BACK TO THE LAST VALID POSE (green)
[!] tsdf volume download is disabled
Average frame time = 51ms (19.3095fps)(real: 10.5837fps)
Average frame time = 34ms (28.9982fps)(real: 14.5503fps)
Average frame time = 34ms (29.2553fps)(real: 14.9728fps)
Average frame time = 34ms (29.2813fps)(real: 14.5439fps)
Average frame time = 34ms (29.1005fps)(real: 11.3715fps)
^C
fxy@fxy-pc:~/pcl/build/bin$ ./pcl_kinfu_largeScale
mp3guy/ElasticFusion · GitHub - Chromium
Google
mp3guy/ElasticFusi
GitHub, Inc. [US] https://github.com/mp3guy/ElasticFusion
Apps Bookmarks computer graphi 编程 Google 翻译 Google 百度一下 Other bookmarks
camera 2 of 3
-s : Frames to skip at start of l
-e : Cut off frame of log.
-f : Flip RGB/BGR.
-icl : Enable this if using the ICL-NUIM dataset (flips normals to account for
-o : Open loop mode.
-rl : Enable relocalisation.
-fs : Frame skip if processing a log to simulate real-time.
-q : Quit when finished a log.
-fo : Fast odometry (single level pyramid).
-nso : Disables SO(3) pre-alignment in tracking.
-r : Rewind and loop log forever.
-ftf : Do frame-to-frame RGB tracking.
-sc : Showcase mode (minimal GUI).
Essentially by default ./ElasticFusion will try run off an attached ASUS sensor liv
the -l parameter. You can capture .klg format logs using either Logger1 or Logg
4. How do I just use the Core API
The libefusion.so shared library which gets built by the Core is what you want to
An example of this can be seen in the GUI code. Essentially all you need to do i
in GUI/src and include the following in your CMakeLists.txt file:

KinectFusion 算法流程

评价：

首次实现基于RGB-D 实时三维重建



a) 2D 深度图像转换成 3D 点云并求得每一点的法向量
b) ICP 迭代求当前帧相机位姿
c) 根据相机的位姿将点云数据融合到场景的三维模型中
d) 光线投影算法求当前视角下能够看到的场景表面

实时三维重建中的 ICP 算法

深度图像（双边滤波）+ 相机内参 ⟶ 点云 + 法向量

$$D_k(\mathbf{u}) = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in\mathscr{U}} \mathcal{N}_{\sigma_s}(\|\mathbf{u}-\mathbf{q}\|_2)\,\mathcal{N}_{\sigma_r}(\|R_k(\mathbf{u})-R_k(\mathbf{q})\|_2)\,R_k(\mathbf{q})$$

$$\mathbf{V}_k(\mathbf{u}) = D_k(\mathbf{u})\mathbf{K}^{-1}\dot{\mathbf{u}}\,.$$

$$\mathbf{N}_k(\mathbf{u}) = v\left[(\mathbf{V}_k(u+1,v)-\mathbf{V}_k(u,v))\times(\mathbf{V}_k(u,v+1)-\mathbf{V}_k(u,v))\right], \quad (4)$$

对点云分层抽样并且匹配按照 coarse-to-fine 的方式

输入：p_i 当前点云、q_i 参考点云、n_i 参考点云法向量 T

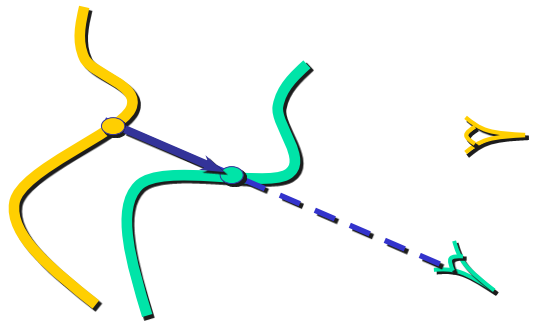
for 循环迭代 n 次

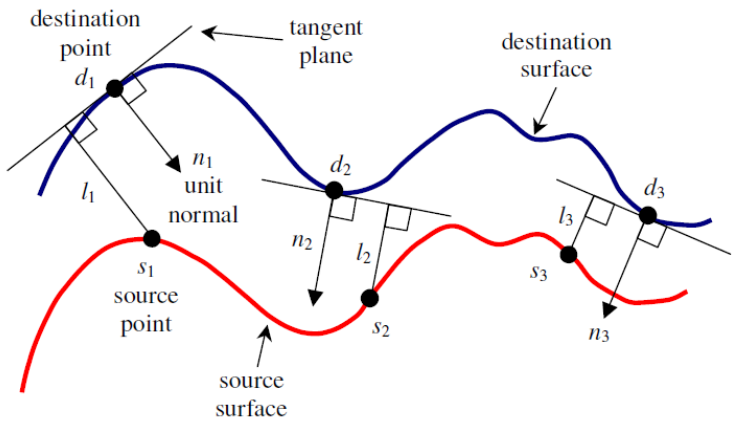
for 每个像素点 $(u\ v)$ 单个线程处理

1. 对于每个像素点用投影算法计算匹配点（两帧位姿小）
2. 对于匹配点极小化点到平面的距离计算位姿

$$E = \sum_{i=1}^{k} ((Rp_i + t - q_i)\cdot n_i)^2$$

3. 将位姿结果作用于当前点云，然后进入 1





实时三维重建中的 ICP 算法 GPU 实现 （PCL）

$$E = \sum_{i=1}^{k} ((Rp_i + t - q_i) \cdot n_i)^2$$

$$x_{opt} = \arg\min_x |Cx - b|^2$$

$$b = \sum_{i=1}^{k} ((p_i \times n_i)^\tau, n_i)^\tau \cdot ((p_i - q_i)^\tau \cdot n_i)$$

$$C = \sum_{i=1}^{k} \begin{pmatrix} p_i \times n_i \\ n_1 \end{pmatrix} \begin{pmatrix} (p_i \times n_i)^\top & n_i^\top \end{pmatrix}.$$

每个像素点用单个线程处理各自 Contribution

每个像素单个线程处理

计算匹配点

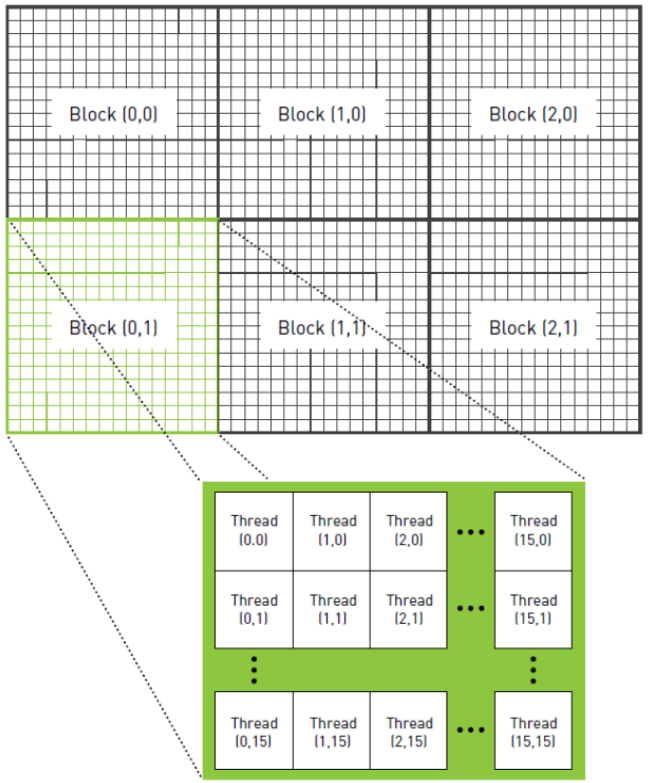
根据匹配点计算像素点的 contribution

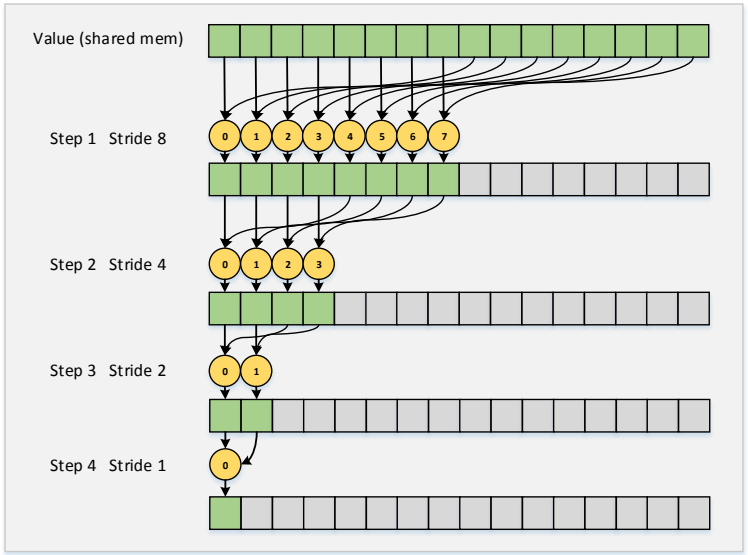
单个线程处理的结果在 GPU 中做累加（Reduce）

累加结果传到 CPU 计算最终位姿

实时三维重建中的 ICP 算法 GPU 实现 （PCL）

```
4  dim3 block (Combined::CTA_SIZE_X, Combined::CTA_SIZE_Y);
5  dim3 grid (1, 1, 1);
6
7  //  rows 和 cols 分别是图像行和列的维度
8  grid.x = divUp (cols, block.x);
9  grid.y = divUp (rows, block.y);
10
11 // 上述操作刚好分配了 rows 乘以 cols 个线程，每个线程处理图像中的单个像素点
12 combinedKernel<<<grid, block>>>(cs);
```





单个像素点的contribution

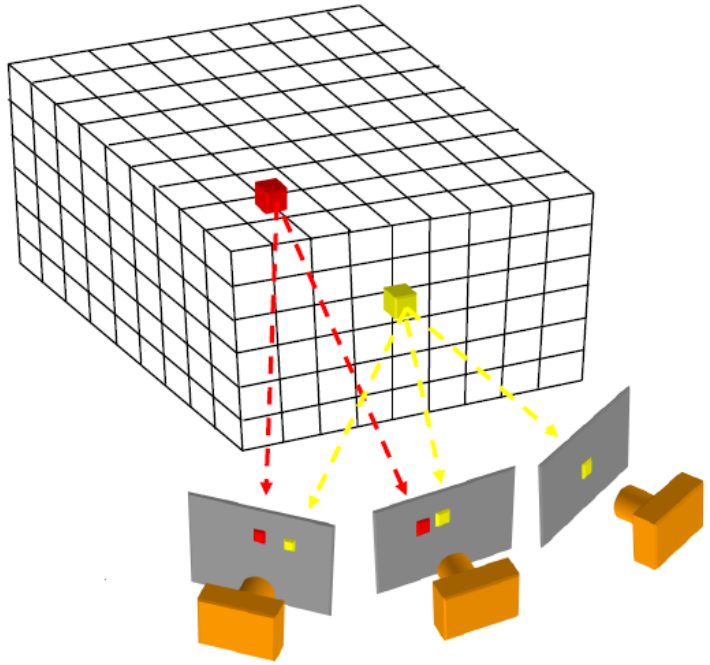
每个 Block 中单个线程的 contribution 做累加

每个 Block 累加和存储到 GPU 显存

单个 Block 对每个 C 和 b 其中一个元素做累加

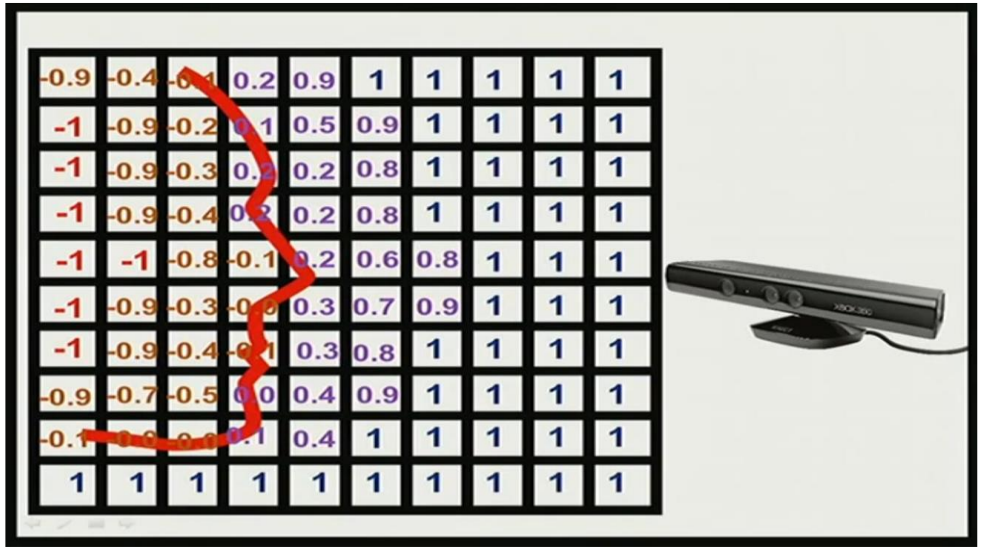
@张也东

根据相机的位姿将点云数据融合到全局模型中



- 三维的 TSDF 模型，将待重建的三维场景划分成网格
- 对于每一帧的深度图像做融合

- TSDF 模型：网格中的数值代表离重建场景表面的距离，网格中从正到负的穿越点代表重建的表面



根据相机的位姿将点云数据融合到全局模型中

for 对于每个 (x, y) 长方形单元并行处理 ⟹ GPU 每个线程处理一个长方形单元

while 对于长方形上的单元从前向后逐个处理

将单元从全局坐标系转到相机坐标系 $v \leftarrow T_i^{-1} v^g$

将单元从相机坐标系投影到相平面 $(u \ \ v) \leftarrow A v$

如果 $(u \ \ v)$ 在相机视场内：

$$sdf_i \leftarrow \| t_i - v_g \| - D_i(u\ v)$$

$$if\ (sdf_i > 0)\ then$$

$$tsdf_i \leftarrow \min(1,\ sdf_i / \max\_truncation)$$

$$else$$

$$tsdf_i \leftarrow \max(-1,\ sdf_i / \max\_truncation)$$

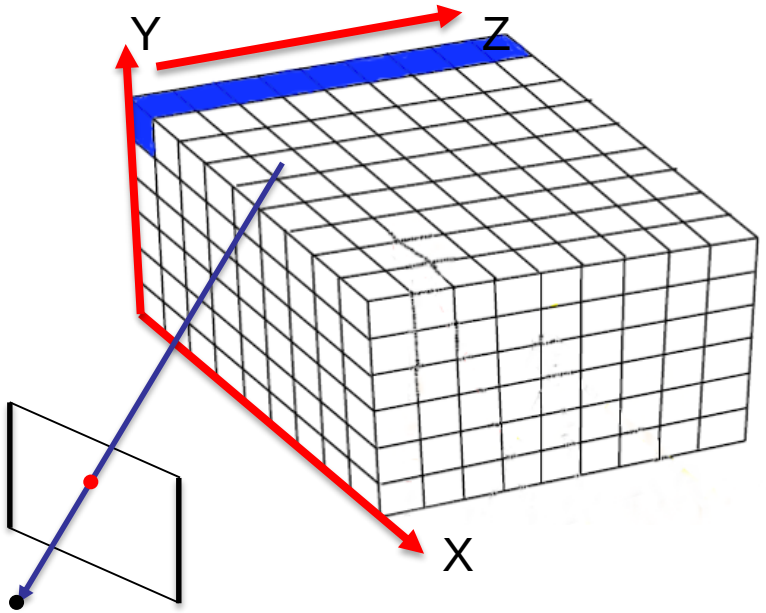
$$w_i \leftarrow \min(\max)$$

$$tsdf^{avg} \leftarrow \frac{tsdf_{i-1} w_{i-1} + tsdf_i w_i}{w_{i-1} + w_i}$$

t_i：相机光心全局坐标

v_g： 单元全局坐标

$D_i(p)$：$(u \ \ v)$处深度值



光线投影算法求得当前视角下场景的表面

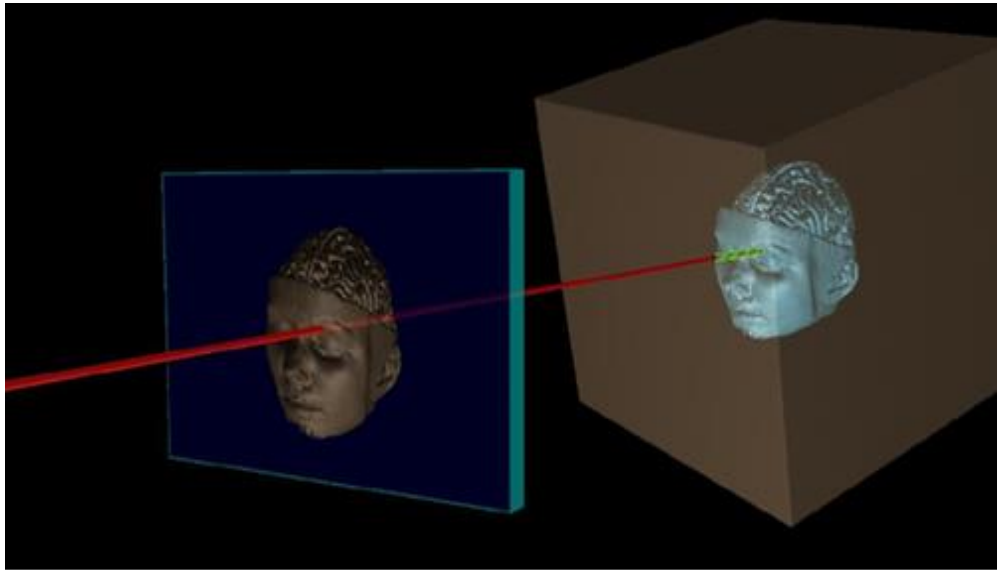
对于图像每个 $(u\ v)$ 像素点 并行处理

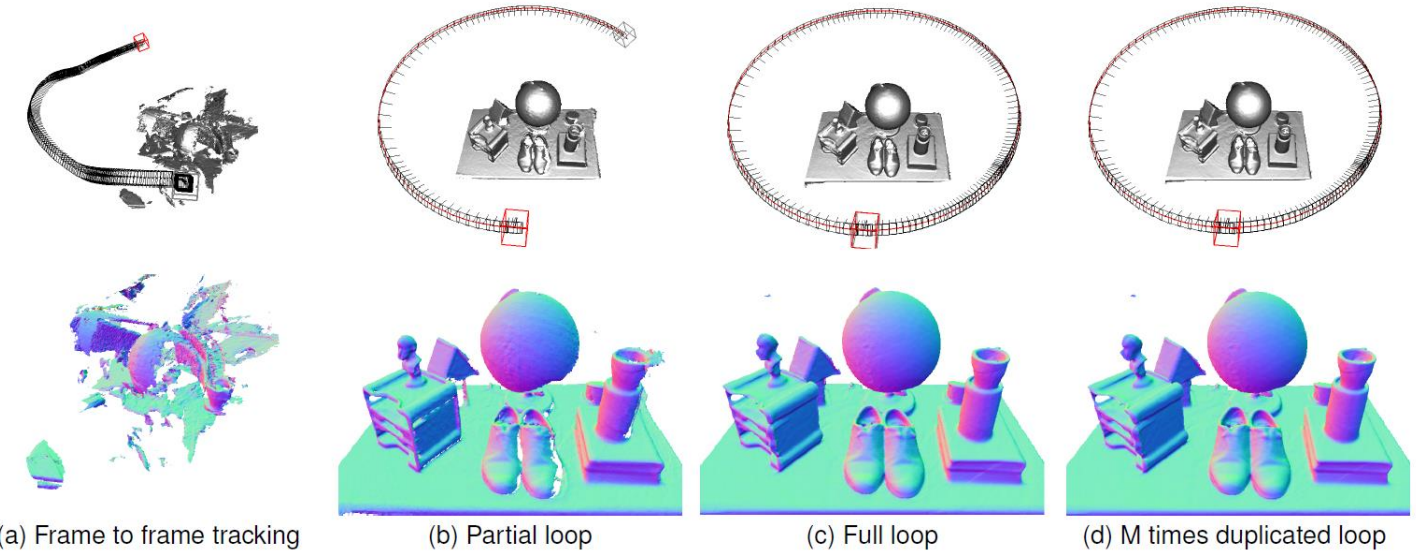
从光心出发穿过像素点的光线在 TSDF 模型中从正到负的穿越点既是目标表面

- 当前视角下光线投影得到的 RGB-D 和下一帧深度图像进行配准求下一帧相机位姿

GPU 每个线程处理一个像素点

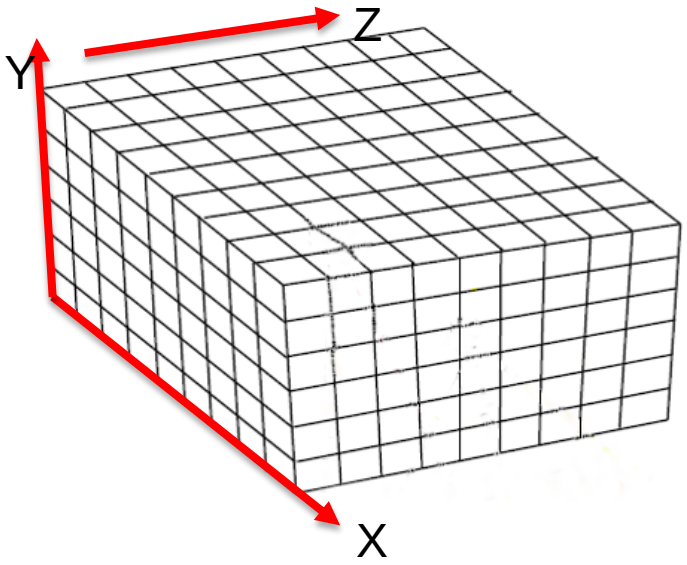
KinectFusion 算法用帧和模型投影计算位姿的方法要比用帧和帧之间的方法要精确



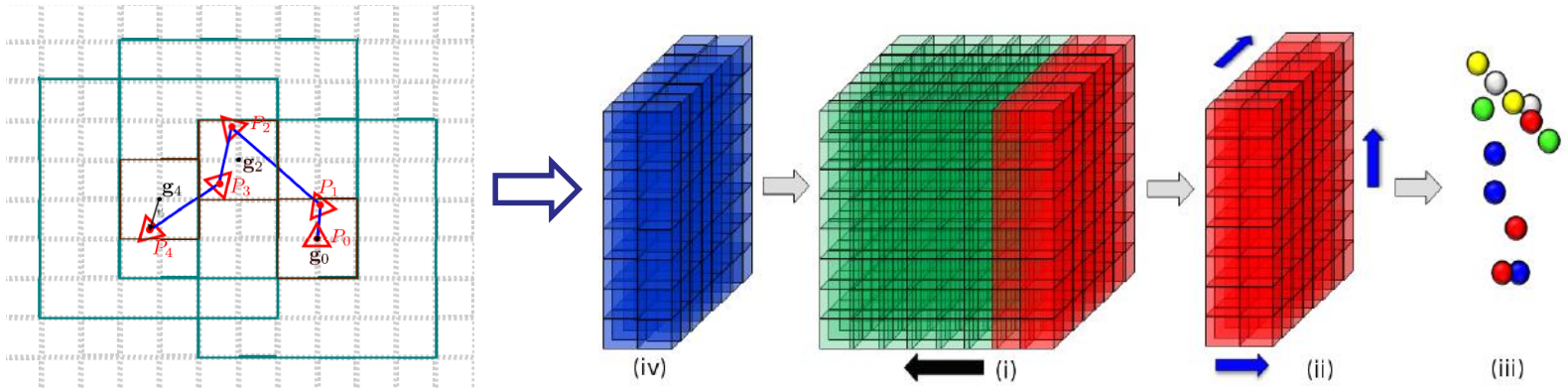


(a) Frame to frame tracking (b) Partial loop (c) Full loop (d) M times duplicated loop

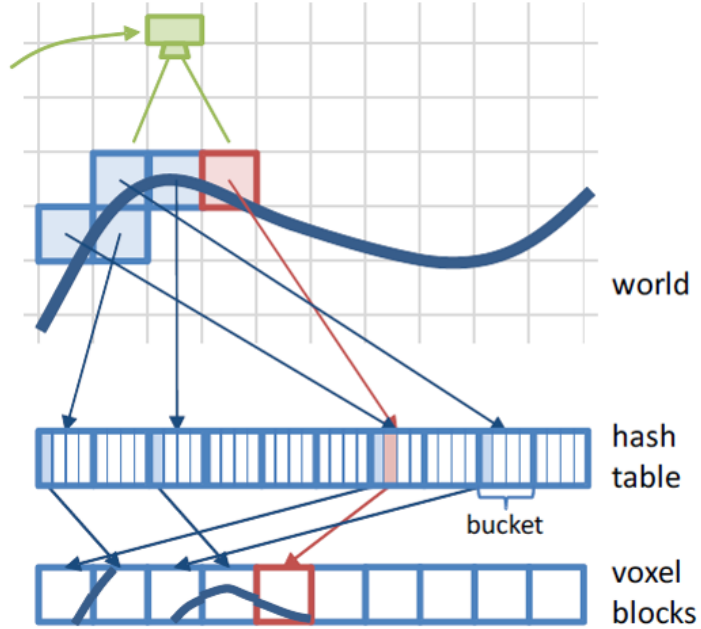
网格模型优缺点



Kintinuous



voxel hashing



优点：

- 网格模型隐含了重建好的表面，加权融合消除误差
- 便于做并行计算

缺点：

- 非常消耗显存，显存使用率和重建场景的体积成正比
- 回环、位姿优化做的差

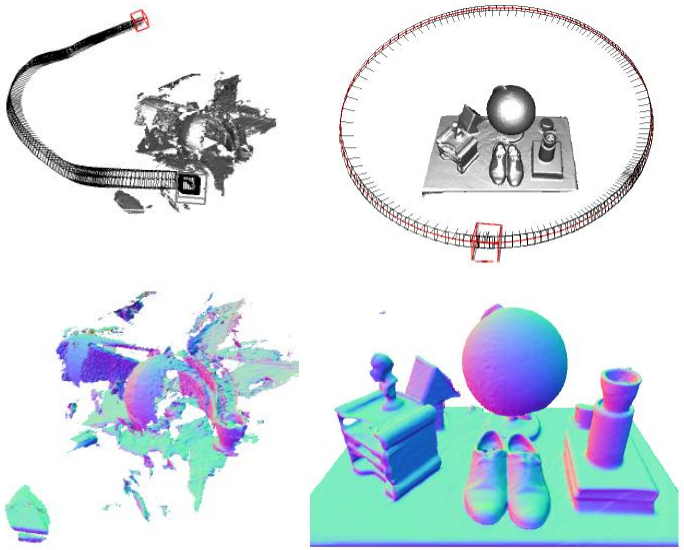
位姿优化和回环检测/重定位

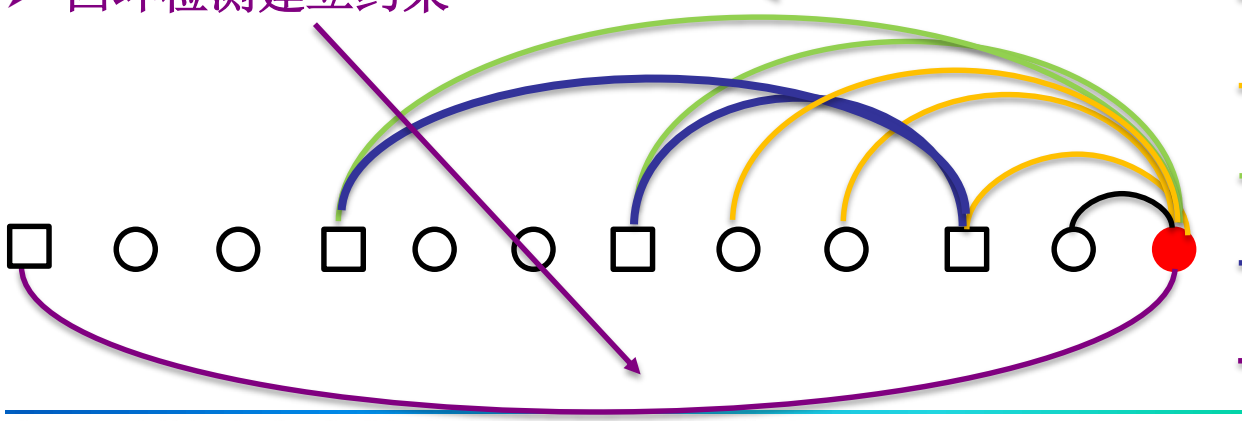
问题：

逐帧和模型或者帧和帧配准，上一帧配准的误差会叠加到这一帧上，误差会逐步累加

解决方法：

- 通过当前帧和模型投影配准 比 通过当前帧和上一帧之间配准累积误差会小很多
- 帧和帧之间、帧和关键帧之间、关键帧和关键帧之间 建立约束
- 回环检测建立约束

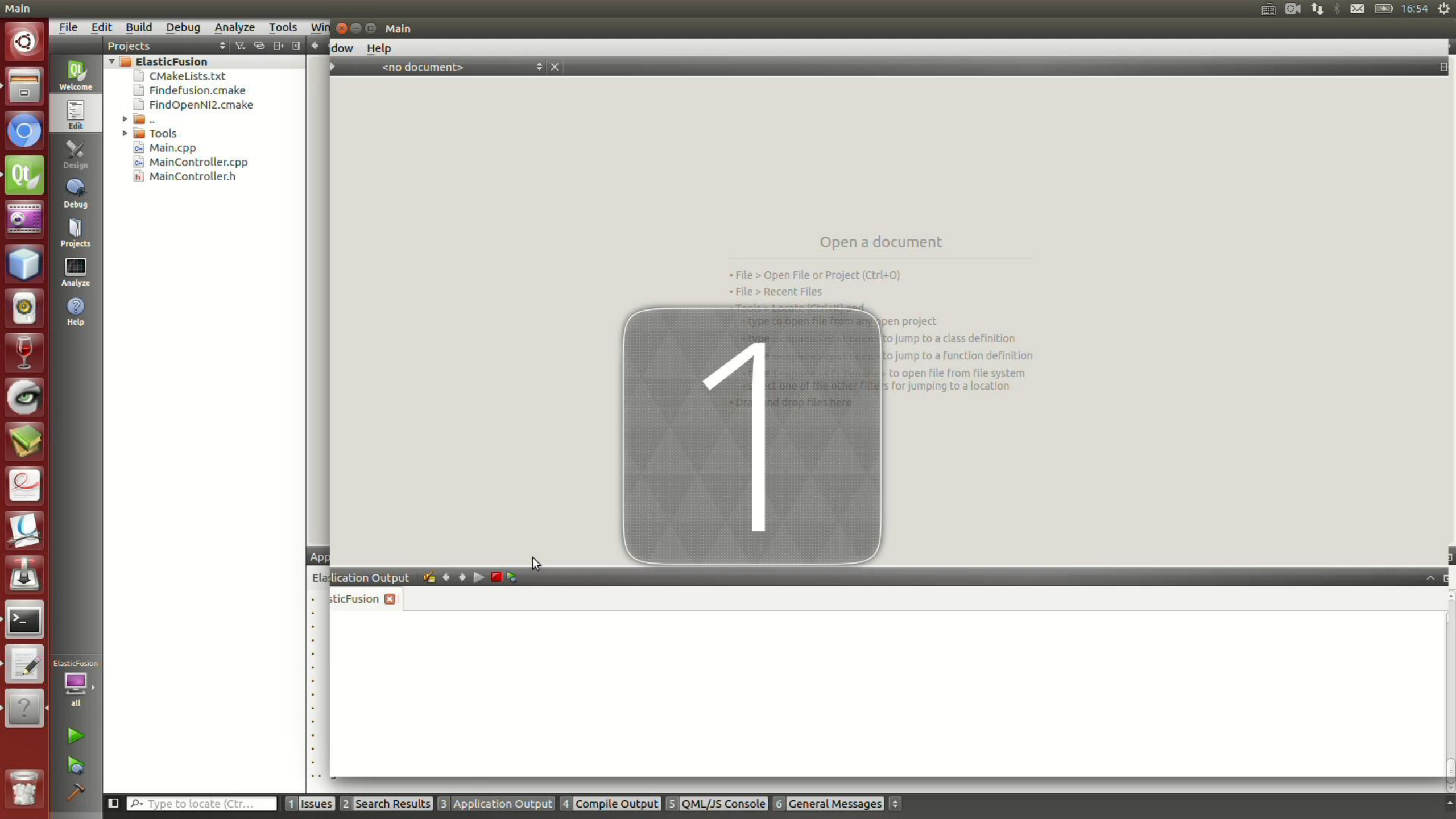




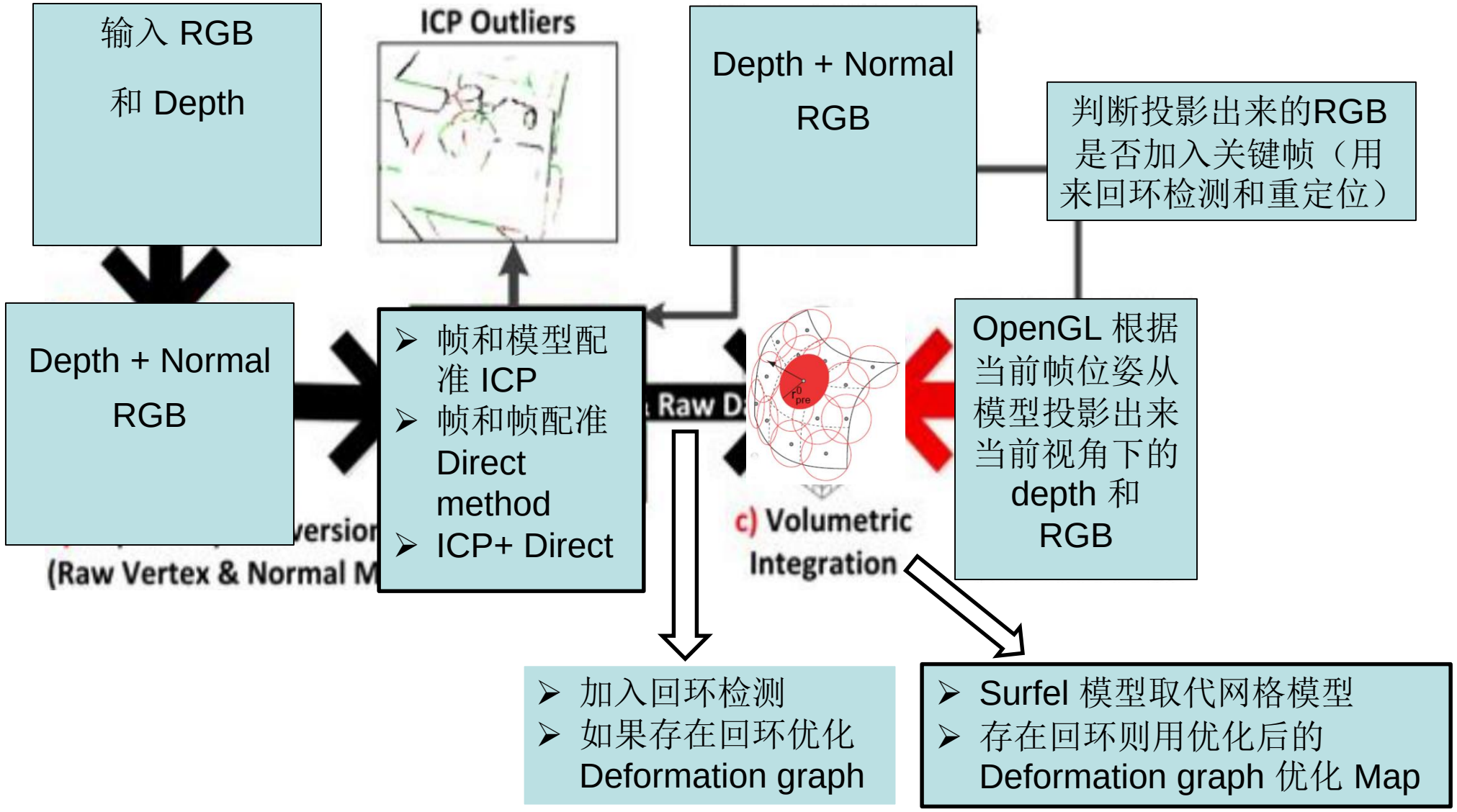
Thomas Whelan

ElasticFusion：

回环建立约束优化重建的 Map，而不是相机的轨迹


Main
File Edit Build Debug Analyze Tools Win
Projects
ElasticFusion
CMakeLists.txt
Findefusion.cmake
FindOpenNI2.cmake
..
Tools
Main.cpp
MainController.cpp
MainController.h
Welcome
Edit
Design
Debug
Projects
Analyze
Help
Main
dow Help
<no document>
Open a document
• File > Open File or Project (Ctrl+O)
• File > Recent Files
Application Output
sticFusion
ElasticFusion
all
Type to locate (Ctr...
1 Issues
2 Search Results
3 Application Output
4 Compile Output
5 QML/JS Console
6 General Messages

ElasticFusion 算法流程



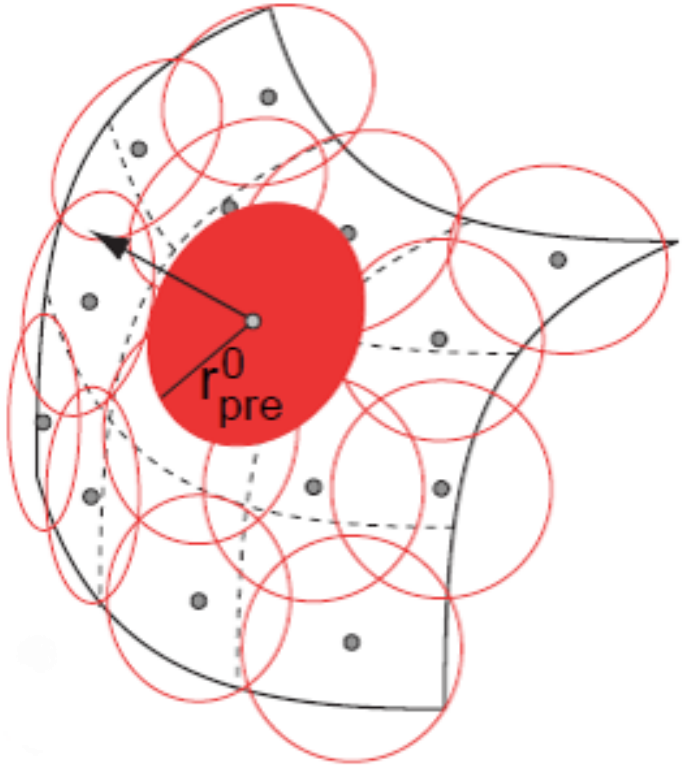
ElasticFusion Surfel 模型

表示形式：

- 中心点坐标 $(x\ y\ z)$
- 面片的半径 r
- 法向量 n
- 颜色 $(R\ G\ B)$
- 时间 τ

模型融合

- 使用点表示模型方便使用 OpenGL
- 点的融合、更新、模型投影全部使用 OpenGL，速度快



Camera Tracking（ICP + Direct）

ICP 算法和 KinectFusion 中是一样的

$$E_{icp} = \sum_{k} \left\| \left(\mathbf{v}^{k} - \exp(\hat{\xi}) \mathbf{T} \mathbf{v}_{n}^{k} \right) \cdot \mathbf{n}^{k} \right\|^{2}, \qquad (14)$$

直接法 SVO和 LSD 很相似 @王京

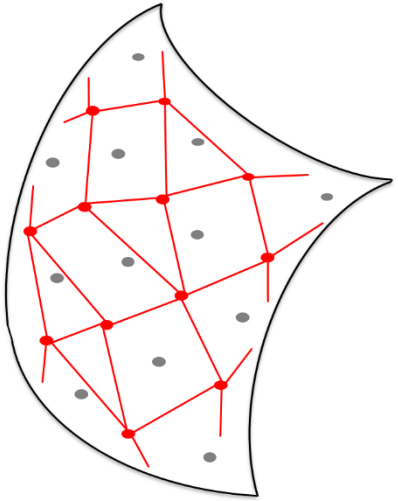
$$E_{rgbd} = \sum_{\mathbf{p} \in \mathcal{L}} \left\| I_{n}(\mathbf{p}) - I_{n-1} \left(\Pi_{n-1} (\exp(\hat{\xi}) \mathbf{T} V_{n}(\mathbf{p})) \right) \right\|^{2} \qquad (20)$$

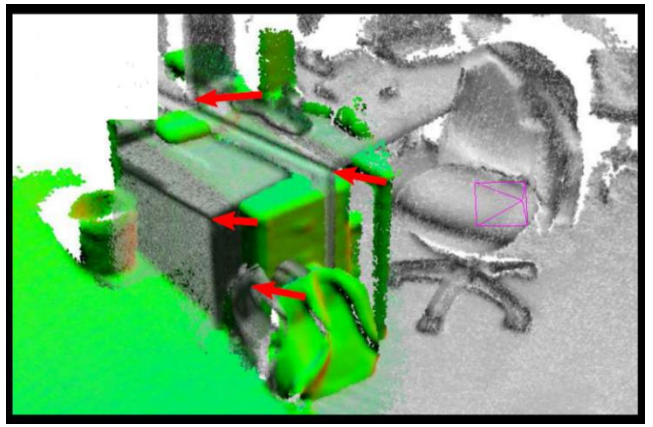
结合了二者

$$E = E_{icp} + w_{rgbd} E_{rgbd} \qquad (26)$$

ElasticFusion Graph Nodes 建立连接

- Graph Nodes 从重建好的 Surfel 中均匀抽样初始化
- Graph Nodes 按照 时间关系（而不是空间关系）建立连接
- Map 中其它点和 Nodes 先按照时间关系搜索最近的点，再在最近点周围按照空间关系找最近的 Nodes 相连接





Deformation Graph 结构和优化：

红色的点是抽取的 Nodes，每个 Node 有待优化参数 G_R^n 和 $\mathcal{G}_{\mathbf{t}}^n$，其中待优化的参数是 仿射变换

根据 Local Loop Closure 和 Global Loop Closure 建立的约束，优化 Nodes 中参数

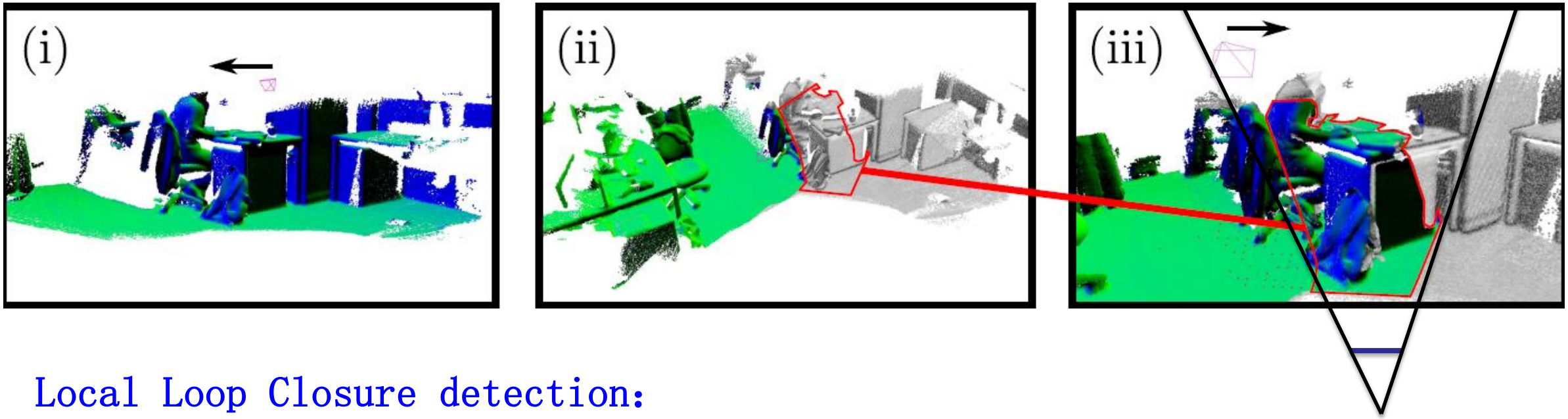
优化之后再将优化的参数作用于全部的点：

位置坐标更新：$\hat{\mathcal{M}}_{\mathbf{p}}^s = \phi(\mathcal{M}^s) = \sum_{n \in \mathcal{I}(\mathcal{M}^s, \mathcal{G})} w^n(\mathcal{M}^s) \left[\mathcal{G}_{\mathbf{R}}^n(\mathcal{M}_{\mathbf{p}}^s - \mathcal{G}_{\mathbf{g}}^n) + \mathcal{G}_{\mathbf{g}}^n + \mathcal{G}_{\mathbf{t}}^n\right]$

法向量更新：$\hat{\mathcal{M}}_{\mathbf{n}}^s = \sum_{n \in \mathcal{I}(\mathcal{M}^s, \mathcal{G})} w^n(\mathcal{M}^s) {\mathcal{G}_{\mathbf{R}}^n}^{-1^\top} \mathcal{M}_{\mathbf{n}}^s,$

Node 权值：$w^n(\mathcal{M}^s) = (1 - \left\|\mathcal{M}_{\mathbf{p}}^s - \mathcal{G}_{\mathbf{g}}^n\right\|_2 / d_{max})^2.$

Local Loop Closure

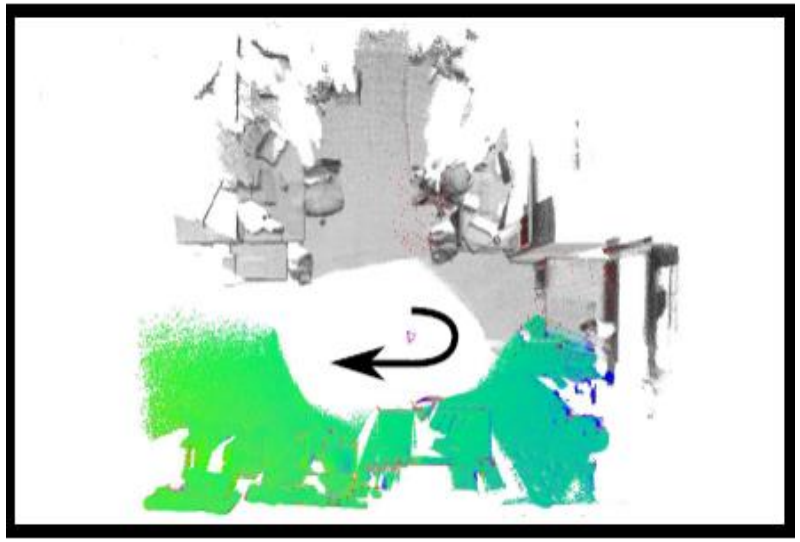


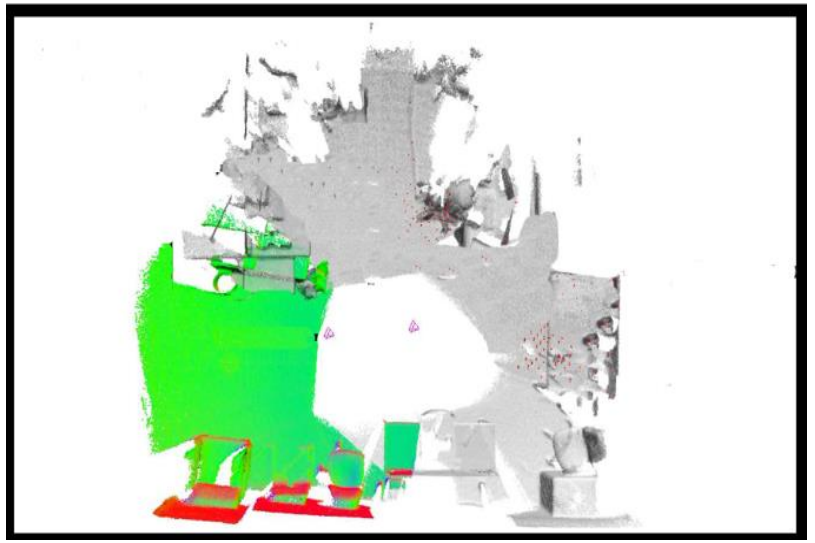
Local Loop Closure detection：

- 跟踪算法求解位姿，根据位姿将点云融合
- 按照时间将重建好的点划分成 ACTIVE 和 INACTIVE
- 根据求解得到的位姿，从 ACTIVE 和 INACTIVE 分别投影得到两幅点云
- 配准两幅点云，如果可以配准上，则说明存在 Local Loop Closure

ACTIVE 和 INACTIVE 分别在当前帧图像做投影获得两幅点云，两幅点云之间计算位姿变换

Global Loop Closure





有了 Local loop closure 为什么还要用 Global loop closure?

- 相机移动距离长时轨迹 drifts 太大，用 ACTIVE 和 INACTIVE 点投影得到的点云配准不上

方法：

- 采用回环检测算法 Randomized Ferns

通过对 回环检测/重定位 Randomized Ferns

图像编码：

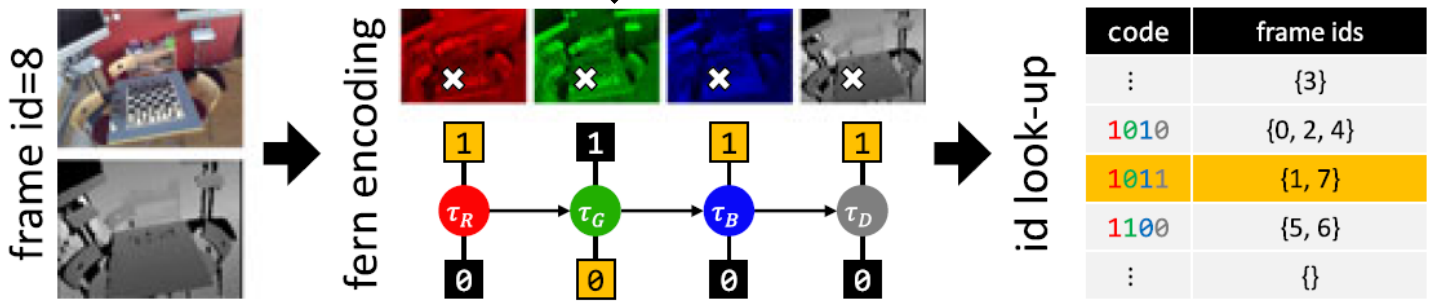
m blocks

每个 block n ferns

每个 fern

每个 fern 像素点 x 图像通道 c 阈值 τ

$$C=\{F_k\}_{k=1}^{m} \rightarrow F=\{f_i\}_{i=1}^{n} \rightarrow f(I,\theta,\tau)=\begin{cases}1 & if\ I(\theta)\geq\tau\\ 0 & if\ I(\theta)<\tau\end{cases} \qquad \theta=\{c,x\}$$



code	frame ids
⋮	{3}
1010	{0, 2, 4}
1011	{1, 7}
1100	{5, 6}
⋮	{}

离线 随机抽取 Block位置坐标、图像通道、阈值

帧和帧之间的相似度：

- 判断是否加入关键帧
- 回环检测/重定位寻找关键帧

新输入图像

计算 Block 码值

根据 Block 码值计算和 list 表中帧之间的相似度

相似度判别是否加入关键帧或者做重定位

Loop Closure 建立约束

Local Loop Closure 建立约束

$$Q_p = (P_{cur}P(u,\ D_t^a);\ P_{INA}P(u,\ D_t^a);\ t_{cur};\ t_{INA}) = (Q_s^P;\ Q_d^P;\ t_s;\ t_d)$$

其中：

P_{cur}：当前帧相对于上一帧位姿变换

$p(u,\ D_t^a)$：从上一帧模型投影点抽样得到的部分点

$P_{cur}p(u,\ D_t^a)$：转换成当前帧点坐标

P_{INA}：上一帧和 INACTIVE 点之间的位姿变换

$P_{INA} = P_{cur}P_{cur\to INACTIVE}$

$P_{cur\to INACTIVE}$：当前帧点转到 INACTIVE 的位姿变换

Global Loop Closure 建立约束

$$Q_p = (P_{cur}P(u,\ D_t^a);\ P_{Fern}P(u,\ D_t^a);\ t_{cur};\ t_{Fern}) = (Q_s^P;\ Q_d^P;\ t_s;\ t_d)$$

其中：

P_{cur}：当前帧相对于上一帧位姿投影出来的点间的位姿变换

$p(u,\ D_t^a)$：从上一帧模型投影点抽样得重点内容到的部分点

P_{Fern}：检测到的 Fern 相对于上一帧位姿投影出来的点间的位姿变换

$tcur$：当前帧的时刻

t_{Fern}：回环检测到的图像的时刻

Loop Closure 建立的约束优化 nodes 参数

优化过程中保持 R 是单位正交的

$$E_{rot} = \sum_{l} \left\| \mathcal{G}_{\mathbf{R}}^{l\top} \mathcal{G}_{\mathbf{R}}^{l} - \mathbf{I} \right\|_F^2, \qquad (11)$$

优化过程中，保持相邻的 node 间参数的连续性

$$E_{reg} = \sum_{l} \sum_{n \in \mathcal{N}(\mathcal{G}^l)} \left\| \mathcal{G}_{\mathbf{R}}^{l}(\mathcal{G}_{\mathbf{g}}^{n} - \mathcal{G}_{\mathbf{g}}^{l}) + \mathcal{G}_{\mathbf{g}}^{l} + \mathcal{G}_{\mathbf{t}}^{l} - (\mathcal{G}_{\mathbf{g}}^{n} + \mathcal{G}_{\mathbf{t}}^{n}) \right\|_2^2 \qquad (12)$$

将 source 点向 destination 点做优化，将两次重建好的点对齐

$$E_{con} = \sum_{p} \| \phi(\mathcal{Q}_{\mathbf{s}}^{p}) - \mathcal{Q}_{\mathbf{d}}^{p} \|_2^2 \qquad (13)$$

固定 destination 点，优化前后 destination 点不能变化太大，改变 source 点坐标，向 destination 点对齐。

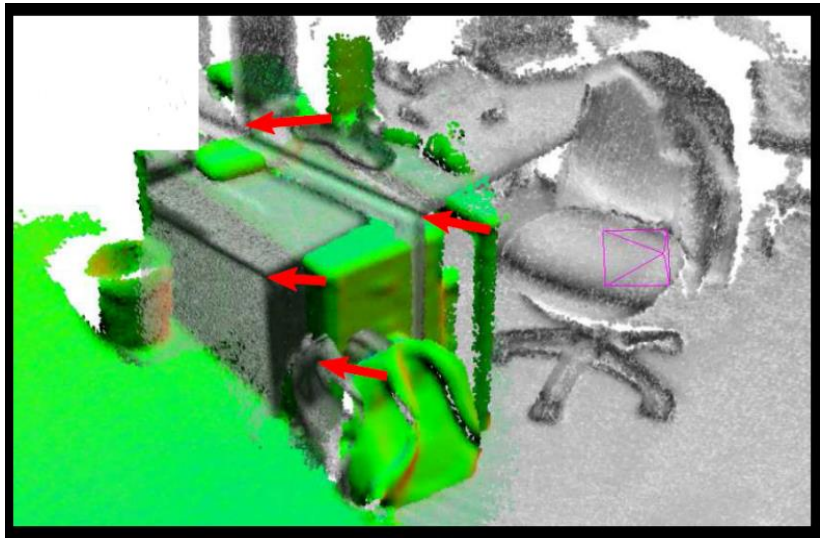
$$E_{pin} = \sum_{p} \| \phi(\mathcal{Q}_{\mathbf{d}}^{p}) - \mathcal{Q}_{\mathbf{d}}^{p} \|_2^2 \qquad (14)$$

Gauss-Newton 求解：

$$E_{def} = w_{rot} E_{rot} + w_{reg} E_{reg} + w_{con} E_{con} + w_{con} E_{pin} \qquad (15)$$

自己的理解：

存在回环时，将 ACTIVE点向 INACTIVE点对齐的过程



Surfel 融合

点融合、更新：

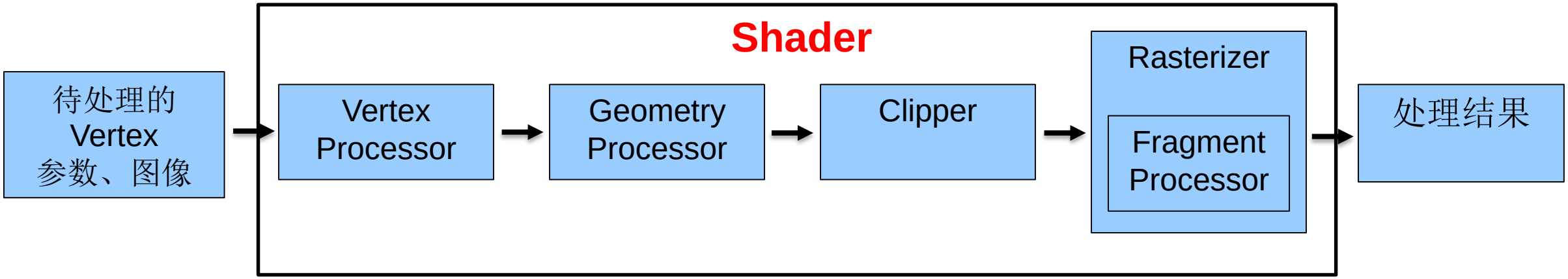
- 跟踪算法求解位姿，根据位姿将点云融合

模型点向图像做投影：

- 重建好的模型根据当前帧相机位姿投影得到深度图像和RGB图像

简单理解：

一次处理一个 Vertex，处理的时候 Shader 中可以传一系列参数，Shader 构成独立的处理单元。



KinectFusion 介绍：

http://blog.csdn.net/fuxingyin/article/details/51417822

ElasticFusion 介绍：

http://blog.csdn.net/fuxingyin/article/details/51433793