第 38 卷 第 23 期
Vol.38 No.23

计 算 机 工 程
Computer Engineering

2012 年 12 月
December 2012

·人工智能及识别技术· 文章编号：1000—3428(2012)23—0198—05 文献标识码：A 中图分类号：TP391

基于 GPU 的实时三维点云数据配准研究

荆 锐[1,2]，赵旦谱[1]，台宪青[1]

(1. 中国科学院自动化研究所，北京 100190；2. 中国科学院研究生院信息科学与工程学院，北京 100080)

摘 要：在三维重建中，不同摄像机坐标系下点云配准耗时过多。为此，提出一种基于图形处理单元(GPU)的实时三维点云数据配准算法。利用投影映射法获取匹配点对，使用点到切平面距离最小化方法计算变换矩阵，通过 GPU 多线程并行处理大规模图像数据。实验结果表明，对于分别包含 307 200 个数据的 2 帧点云，在保持原有配准效果的基础上，该算法的最优耗时仅为基于 CPU 的最近邻迭代算法的 11.9%。

关键词：图形处理单元；3D 重建；摄像机跟踪；同时定位与地图构建；并行处理

Research on Real-time 3D Point Cloud Data Registration Based on GPU

JING Rui[1,2], ZHAO Dan-pu[1], TAI Xian-qing[1]

(1. Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China;

2. School of Information Science and Engineering, Graduate University of Chinese Academy of Sciences, Beijing 100080, China)

【Abstract】In 3D reconstruction, point cloud registration time of different camera coordinate system is too long. In order to sovle this problem, this paper proposes a real-time 3D point cloud data registration algorithm based on Graphic Processing Unit(GPU). The projection mapping method is used to find corresponding point pairs, the transformation matrix is obtained by minimizing the distance from the point to the tangent plane, and uses the GPU to multi-thread and parallelly process mass image data. Experimental results show that, in order to keep original registration effect, the optimal time-consuming of this algorithm is about 11.9% compared with Iterative Closest Point algorithm based on GPU(ICP-GPU) in containing 307 200 data 2 frame point cloud.

【Key words】Graphic Processing Unit(GPU); 3D reconstruction; camera tracking; Simultaneous Localization and Map Building(SLAM); parallel processing

DOI: 10.3969/j.issn.1000-3428.2012.23.049

1 概述

同时定位与地图构建(Simultaneous Localization and Map Building, SLAM)技术广泛应用于机器人导航，由于三维重建的实时性限制，目前仍以构建二维地图为主。随着实时深度摄像机[1]的迅速发展，大规模的三维点云可以通过摄像机透视变换实时获取，若能解决实时三维点云配准与三维实时渲染这 2 个大难点，则可实现实时三维重建。在深度摄像机获取的相邻帧深度图转换成的三维点云后，对相邻 2 帧点云进行实时配准，用于追踪摄像机的位置，将不同坐标系下的点云融合到一个全局坐标系中以构建全局三维地图。

最近邻迭代(Iterative Closest Point, ICP)算法[2]将 2 帧点云的距离最近点作为对应匹配点，建立对应点之间的距离误差函数，通过非线性方法来最小化误差函数，不断迭代从而逼近最佳变换。传统的 ICP 算法过于理想，错误匹配点对过多、迭代速度慢、耗时长，于是科研人员对传统的 ICP 算法进行了如下改进：

(1)匹配点对获取方法的改进

除了传统的最近距离法选择匹配点外，文献[3]还提出了投影映射法搜索匹配点对，极大地缩短了对应点匹配时间，但无法达到较高的精度。

(2)点对剔除算法的改进

除了距离约束剔除错误点对外，文献[4]系统分析研究了多种约束条件，得出结论：对于刚性变换，法向约束能剔除适中量的错误点对、刚性约束取出错误点对较少，匹配误差约束剔除了过多的点对。

(3)误差函数选择的改进

文献[5]改进了误差函数，提出使用点到对应点所在切平面的距离作为误差函数，从一定程度上匹配它们的局部几何特征，加快了 ICP 的收敛速度，且迭代次数少，不易陷入局部极值。

作者简介：荆 锐(1987－)，女，硕士研究生，主研方向：机器视觉，模式识别；赵旦谱，博士后；台宪青，研究员、博士

收稿日期：2012-03-02 **修回日期：**2012-03-28 **E-mail：**rui.jing@ia.ac.cn

(4)综合改进

文献[6]将点到切平面误差函数与投影映射算法结合起来，兼顾了点到投影的速度优势与点到切平面算法的精度优势，达到了快速、高效的目的。

图像处理器单元[7](Graphic Processing Unit, GPU)是显卡的主要处理单元，可与 CPU 高速交换数据，且可并行运算数据，适合处理大规模数据的运算。相对于传统基于 OpenGL 和 DirectX 的 GPGPU 编程，统一计算架构[8](Compute Unified Device Architecture, CUDA)提供了更直观的编程模型和优化原则，可在 Visual Studio 上使用 C 语言进行编程，代码简单、适用性广泛，且程序可在 CPU 与 GPU 之间切换运行。

本文参考文献[9-10]算法，提出一种基于 GPU 的实时三维点云数据配准。通过投影映射法选择对应匹配点对，利用距离和法向约束剔除错误点对，将所有源点到对应目标点所在切平面的距离作为误差函数，并通过自适应迭代法计算变换矩阵，大多数处理过程都使用 CUDA 多线程处理，将数据均匀分配到 GPU 的多个线程并行处理。

2　投影映射法与点到切平面误差函数

由于机器人中的摄像机一直处于运动状态，为了将摄像机不同位置下，获取的点云数据融合到全局坐标系中，需要计算每一帧摄像机坐标系 O_k 与全局坐标系 O_g 的刚体变换矩阵，用 $\boldsymbol{T}_{k,g}=[\boldsymbol{R}_{k,g}\ |\boldsymbol{t}_{k,g}]$ 表述变换矩阵，其中，$\boldsymbol{T}_{k,g}$ 为 3×4 矩阵；$\boldsymbol{R}_{k,g}$ (3×3)是旋转矩阵；$\boldsymbol{t}_{k,g}$ (1×3)是平移矩阵。

ICP 算法本质上是使用基于最小二乘法的最优配准方法来估计参数矩阵，包括：(1)初始矩阵估计；(2)获取匹配点对；(3)计算变换矩阵；(4)迭代进行步骤(2)、步骤(3)，直到满足正确配准的收敛精度要求。

2.1　基于投影映射法的匹配点对获取

与最近距离法相比，投影映射法匹配时间短，同时由于本文算法的实时性，高帧频确保了帧间位移很小，弥补了投影映射法匹配精度低的缺点。

使用文献[11]方法获取相邻帧间的匹配点对，对于点云空间集 V_k 中任一点 s，坐标为向量 $\boldsymbol{v}_k(u,v)$，按照下式获取前一帧点云 V_{k-1} 中对应的匹配点 d，坐标为向量 $\boldsymbol{v}_{k-1}(u',v')$：

$$\begin{cases}\boldsymbol{v}_{k,g}(u,v)=\boldsymbol{R}_{k,g}^{z-1}\boldsymbol{v}_k(u,v)+\boldsymbol{t}_{k,g}^{z-1}\\ \boldsymbol{n}_{k,g}(u,v)=\boldsymbol{R}_{k,g}^{z-1}\boldsymbol{n}_k(u,v)\end{cases}\tag{1}$$

$$\boldsymbol{v}_{cp}(u',v')=\boldsymbol{R}_{k-1,g}^{-1}\cdot[\boldsymbol{v}_{k,g}(u,v)-\boldsymbol{t}_{k-1,g}]\tag{2}$$

其中，g 代表全局坐标系；z 代表第 z 次迭代；$\boldsymbol{n}_{k,g}(u,v)$ 是对应的法向量。

式(1)利用第 $z-1$ 次迭代所得的刚体变换矩阵 $\boldsymbol{R}_{k,g}^{z-1}$，将 O_k 坐标系下的任意顶点坐标向量 $\boldsymbol{v}_k(u,v)$ 及法向量 $\boldsymbol{n}_k(u,v)$ 全部转换到全局坐标系 O_g 下，即顶点坐标向量 $\boldsymbol{v}_{k,g}(u,v)$ 与法向量 $\boldsymbol{n}_{k,g}(u,v)$；式(2)利用前一帧变换矩阵(旋转逆矩阵 $\boldsymbol{R}_{k-1,g}^{-1}$ 与平移矩阵 $\boldsymbol{t}_{k-1,g}$)将 O_g 下任意点向量 $\boldsymbol{v}_{k,g}(u,v)$ 映射到 O_{k-1} 坐标系下，然后通过摄像机反透视投影获得对应匹配点 d 的图像坐标 (u',v')。

2.2　点到切平面距离误差函数

点到对应匹配点切平面的误差函数如图 1 所示，计算源点到对应点所在切平面的距离，选取当前帧点云 V_k 中的 s 点与其在前一帧点云 V_{k-1} 中的对应点 d 所在切平面的距离 l 作为误差，误差函数描述如下：

$$E=\arg\min\sum_{D_i(u,v)>0}\left\|\boldsymbol{n}_{k-1,g}(u',v')^{\mathrm{T}}\ (\boldsymbol{T}_{k,g}\cdot\boldsymbol{v}_k\ (u,v)-\boldsymbol{v}_{k-1,g}(u',v'))\right\|^2\tag{3}$$

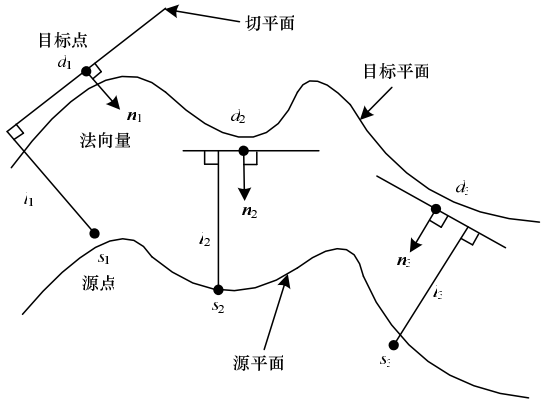


图 1　点到对应匹配点切平面的误差函数

关于最小化目标函数的方法，参考文献[11]所述，假设存在更新矩阵 $\boldsymbol{T}_{inc}^z$，满足 $\boldsymbol{T}_{k,g}^z=\boldsymbol{T}_{inc}^z\cdot\boldsymbol{T}_{k,g}^{z-1}$，$\boldsymbol{T}_{inc}^z$ 定义如下：

$$\boldsymbol{T}_{inc}^z=[\boldsymbol{R}_{inc}^z|\boldsymbol{t}_{inc}^z]=\begin{bmatrix}1&\alpha&-\gamma&t_x\\-\alpha&1&\beta&t_y\\\gamma&-\beta&1&t_z\end{bmatrix}\tag{4}$$

更新矩阵 $\boldsymbol{T}_{inc}^z$ 可用向量 $\boldsymbol{x}=(\beta,\gamma,\alpha,t_x,t_y,t_z)^{\mathrm{T}}$ 表示，使用迭代点变换最小化误差函数。式(4)顶点到世界坐标系的转换可转化为下式：

$$\boldsymbol{T}_{k,g}^z\boldsymbol{v}_k\ (u,v)=\boldsymbol{R}_{k,g}^z\cdot\boldsymbol{v}_k\ (u,v)+\boldsymbol{t}_{k,g}^z=\boldsymbol{G}(u,v)\boldsymbol{x}+\boldsymbol{v}_k\ (u,v)\tag{5}$$

其中，矩阵 $\boldsymbol{G}(u,v)$ (3×6)由顶点 $\boldsymbol{v}_{k,g}(u,v)$ 的反对称矩阵组成，$\boldsymbol{G}(u,v)=[[\boldsymbol{v}_{k,g}\ (u,v)]_\times|\boldsymbol{I}_{3\times3}]$。

将式(5)代入式(3)，误差函数可转化为：

$$E=\arg\min\sum_{D_i(u,v)>0}\left\|\boldsymbol{n}_{k-1,g}(u',v')^{\mathrm{T}}\ (G(u,v)\boldsymbol{x}+\boldsymbol{v}_{k,g}\ (u,v)-\boldsymbol{v}_{k-1}\ (u',v')\right\|^2\tag{6}$$

$$\sum(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})\boldsymbol{x}=\sum\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}\tag{7}$$

其中：

$$\boldsymbol{A}=\boldsymbol{n}_{k-1,g}(u',v')^{\mathrm{T}}\cdot\boldsymbol{G}(u,v)$$

$$\boldsymbol{b}=\boldsymbol{n}_{k-1,g}(u',v')^{\mathrm{T}}\ [\boldsymbol{v}_{k-1}(u',v')-\boldsymbol{v}_{k,g}(u,v)]$$

通过式(6)对向量 $\boldsymbol{x}$ 求导，最终满足式(7)的 $\boldsymbol{x}$ 时，误差函数最小。

3　基于 GPU 的 ICP 算法

3.1　CUDA 并行处理数据的方法

CUDA 程序流程如图 2 所示，过程包括：(1)数据从 CPU 存储器传递至 GPU 存储器。(2)CPU 函数驱动 GPU。(3)GPU 并行运行每一个 kernel 函数。(4)GPU 将结果传回 CPU 存储器。

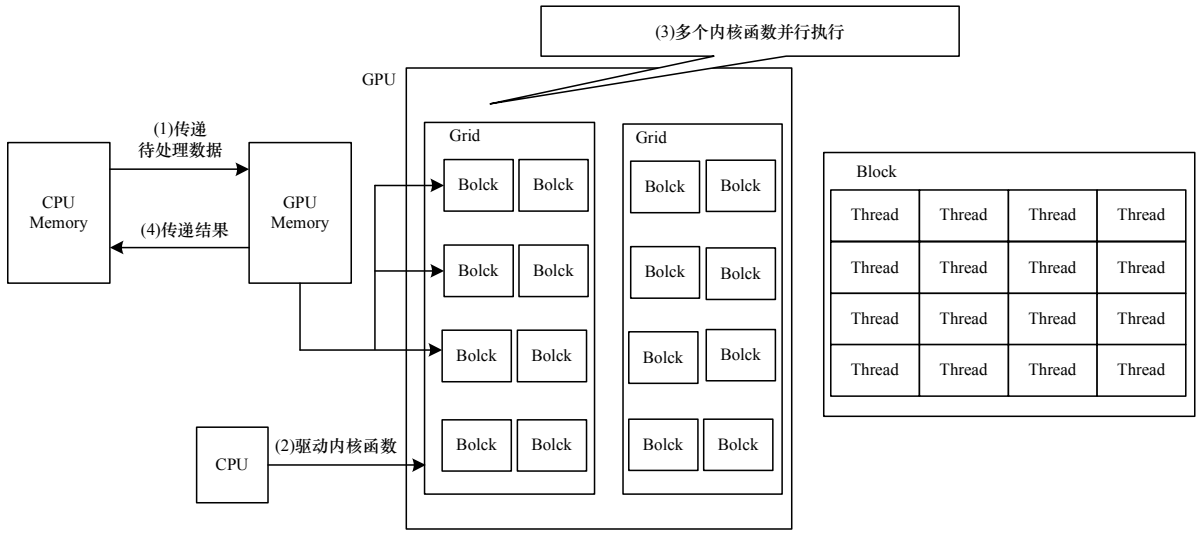


图 2 CUDA 程序流程

kernel 函数是 CUDA 内核函数，可放入 GPU 的多个线程中并行执行，工作原理[12]是：被设计成并行数据处理的内核 kernel 的 CUDA 程序，发送到 GPU 上的栅格 Grid 上执行，Grid 中包含若干个块 Block，每个块 Block 执行若干线程。一个具体的配置构成 kernel 程序的执行环境，它在程序启动的时候被指定。

本文算法设置每个块执行 32×8=256 个线程，则 GPU 中块 Block 的数量：

$$grid.x \times grid.y = \frac{640}{32} \times \frac{480}{8} = 1\,200$$

并行运行 256 个 kernel 线程。

3.2 基于 GPU 的 ICP 算法

将第 k 帧点云 V_k 与第 $k-1$ 帧点云 V_{k-1} 进行配准，计算刚性变换矩阵，最大限度的重合相同点。实现步骤如下：

Step1 三维顶点坐标及法向计算

硬件获取的每一帧深度图 D_k 经双边滤波器滤波后，利用已知的深度摄像机内参矩阵 $\boldsymbol{K}$ 反透视变换重建出所有点的三维坐标，可获取帧率 30 f/s 的三维点云数据。使用顶点地图计算法向量，叉乘中心点指向相邻顶点的向量，获得与两向量垂直的单位向量，即法向量。

顶点及法向计算的伪代码如下：

for 所有点 $D_k(u,v) \in$ 深度图像 D_k in parallel do
if $D_k(u,v)>0$ then
$V_k(u,v) \leftarrow K\cdot[u,v,D_k(u,v)]^T$ 透视投影
else
$V_k(u,v) \leftarrow$ Nan
if $D_k(u,v)>0$ and $D_k(u+1,v)>0$ and $D_k(u,v+1)>0$ then
$N_k(u,v) \leftarrow \text{cross}([V_k(u+1,v)-V_k(u,v)],[V_k(u,v+1)-V_k(u,v)])$

Step2 初始矩阵设置，z=0。

迭代初始矩阵在 ICP 算法中极为重要，直接影响到最终的迭代结果，若参数设置不合适，容易陷入局部最优。本文相邻 2 帧点云变化很小，可设置初始矩阵 $\boldsymbol{T}_{k,g}^{z=0}$ 等于前一帧变换矩阵 $\boldsymbol{T}_{k-1,g}$，第 1 次迭代顶点映射满足下式：

$$\boldsymbol{V}_{k,g}^{z=0}(p) \leftarrow \boldsymbol{T}_{k-1}\boldsymbol{V}_k(p) \tag{8}$$

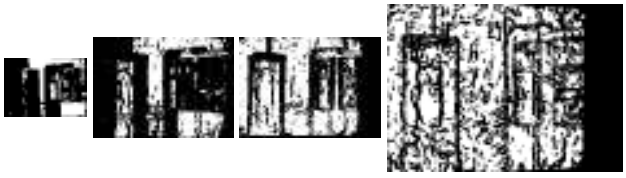
其中，z 表示迭代次数；k 代表帧数；g 代表全局变量；p 表示三维空间中的任意顶点。

Step3 匹配点对的获取

2.1 节描述投影映射法获取匹配点对过程，按式(1)、式(2)获取对应的匹配点，使用距离和法向约束剔除错误的匹配点对 C，该步骤伪代码如下：

for 所有顶点 $p \in V_k$ in parallel do
if p!=Nan then
$v_{k,g} \leftarrow T^{z-1}V_k(p)$, $n_{k,g} \leftarrow R^{z-1}N_k(p)$
$p' \leftarrow V_{k,g}^{z}(p)$投影映射
if $p' \in V_{k-1}$ then
$v \leftarrow T_{k-1}V_{k-1}(p')$, $n \leftarrow R_{k-1}N_{k-1}(p')$
if $\| v-v_{k,g} \|$ <distance threshold and
$\text{abs}(n\cdot n_{k,g})$<normal threshold then
找到对应点$[p,p'] \in C$

图 3 显示 Step3 在第 0 次、第 3 次、第 6 次、第 9 次迭代时获取的对应点对，其中，白色区域代表该顶点存在对应点；黑色区域代表该顶点未找到对应点。随着迭代次数的增加，存在对应点的顶点越来越多，配准效果越好。当 z=9 时，除了边缘点与不重叠区域，所有顶点均能找到对应匹配点。



(a)z=0 (b)z=3 (c)z=6 (d)z=9

图 3 迭代过程中获取的对应点对

Step4 估计配准矩阵

在 2.2 节中，将源点到目标点切平面距离的平方和作为误差函数，根据式(8)最小化误差函数，估计变换矩阵。最小化点到切平面法的伪代码如下：

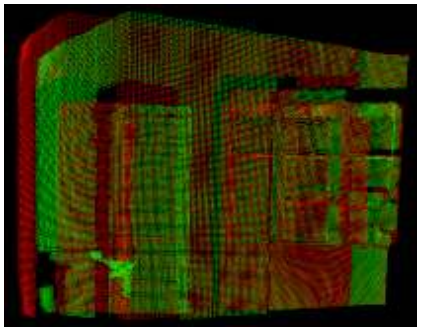
for 所有对应点对$[p,p'] \in C$ in parallel do
$n \leftarrow N_{k-1}(p')$
$d \leftarrow V_{k-1}(p')$

$s \leftarrow V_k(p)$
$b \leftarrow n \cdot (d-s)$
$A \leftarrow [s \times n, n, b]^T$
$AA \leftarrow$ sum of $A^T A$
$bb \leftarrow$ sum of $A \cdot b$
$x \leftarrow SVD(AA \cdot x = bb)$
$T_{inc} \leftarrow x$
$T_z \leftarrow T_{inc} \cdot T_{z-1}$

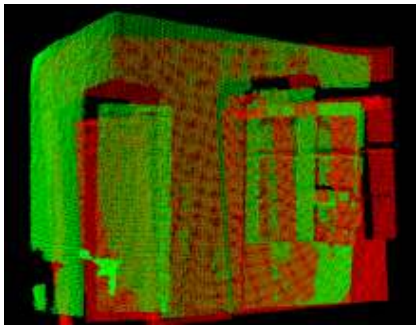
上述代码中，第 7 行、第 8 行需要计算所有相关点对所获取的数值和，从第 7 行起均在 CPU 中执行程序。

Step5 重复迭代 $z=z+1$

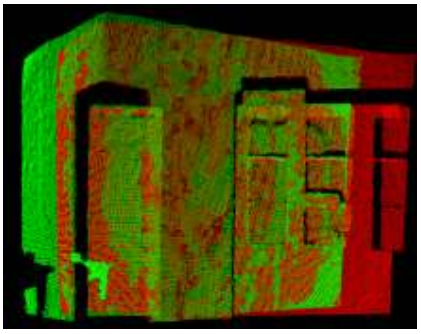
迭代过程的配准效果如图 4 所示。



(a)z=0



(b)z=6



(c)z=9

图 4　迭代过程的配准效果

如果更新矩阵 $\boldsymbol{x}$ 满足 $\|\boldsymbol{x}(1{:}3)\|<Rotate\ threshold$ 与 $\|\boldsymbol{x}(4{:}6)\|<Trans\ threshold$，即 2 次迭代之间矩阵参数变化很小，满足了一定收敛率，那么终止迭代，并确定最终变换矩阵 $\boldsymbol{T}_k$ 等于最后一次迭代矩阵 $\boldsymbol{T}_{zmax}$。

本文采用多尺度变换配准，将原始顶点分为 3 层，每层行列数为上一层的一半，相应的顶点数为上一层的 1/4，从而达到减少计算量的目的。图 4 显示相应的配准效果，第 0 次、第 1 次迭代(z=0−1)属于第 3 层，顶点个数为 160×120；第 3 次~第 8 次迭代(z=3−8)属于第 2 层，顶点个数为 320×240；第 9 次迭代(z=9)属于第 1 层，顶点个数为 640×480。随着迭代次数的增加，配准效果越来越好，最终获得正确的摄像机位置坐标，将不同点云统一到世界坐标系下。

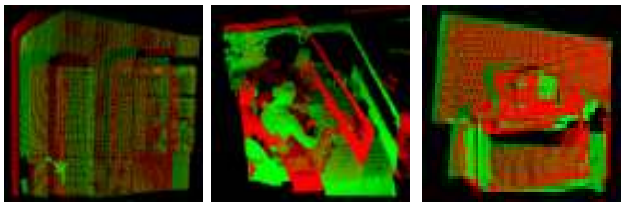
4　实验结果与分析

4.1　实验平台搭建

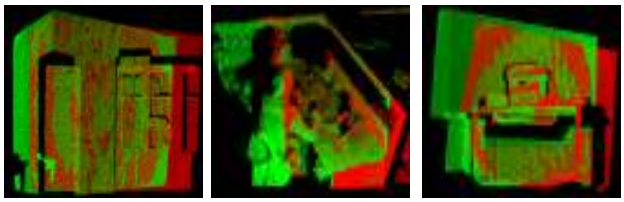
本文算法是在普通 PC 机上完成的，配置如下：CPU 为 Intel(R) Core(TM)2，频率为 2.5 GHz，内存为 4 GB；显卡为 NVIDIA GeForce 560 Ti，核心频率为 1.9 MHz，显存频率为 4 GHz，显存为 1 GB；深度摄像机为微软 Xbox Kinect[13]，分辨率为 640×480 像素，帧频 30 f/s。

4.2　实验结果

利用本文算法进行 3 种场景的点云配准，实验结果如图 5 所示，3 种场景都取得了较好的效果。



(a)配准前的 2 帧点云



(b)配准后的 2 帧点云

图 5　点云配准前后的对比

4.3　对比实验

对于不同数目的点云分别采用基于 CPU 的最近邻迭代算法(Iterative Closest Point Algorithm Based on CPU, ICP-CPU)算法和 ICP-GPU 算法进行配准，各阶段耗时比较结果如表 1 所示。从对比结果可以看出，点云包含顶点个数越少，ICP 算法耗时越短，但由于相应的分辨率降低，效果有所下降。对包含不同数目的点云，使用 ICP-GPU 算法在各个阶段都节省了相当多的时间，在点云个数为 307 200 与 76 800 时，ICP-GPU 算法的执行时间仅是 ICP-CPU 算法执行时间的 11.9%与 11.1%；在点云个数为 19 200 时，由于大量时间用于 CPU 与 GPU 频繁交换数据，导致效果不如前 2 个；由此可知，点云数目越多，基于 GPU 的 ICP 算法越占优势；反之，点云过少，GPU 加速效果越不明显。对于相同的点云个数，ICP-GPU 算法在各个阶段所加速的效果不同，其中，估计变换矩阵步骤的加速效果最差，这是由于相关点对求和与矩阵求解均在 CPU 上执行，未能充分利用 GPU 多线程资源。对分别包含 307 200 个数据的 2 帧点云，使用不同的 CUDA 线程数运行 ICP-GPU 算法进行配准，各阶段所用时间如表 2 所示。

表 1 ICP-GPU 与 ICP-CPU 算法的耗时比较

点云个数	算法	法向计算时间/ms	更新点云时间/ms	匹配点对时间/ms	变换矩阵时间/ms	总耗时/ms
307 200	ICP-GPU	5.13	0.50	1.40	0.32	9.09
	ICP-CPU	40.09	9.83	15.11	1.95	76.32
76 800	ICP-GPU	3.46	0.38	0.61	0.30	5.73
	ICP-CPU	33.09	2.63	3.99	1.02	51.23
19 200	ICP-GPU	3.09	0.57	0.65	1.17	6.22
	ICP-CPU	10.25	1.01	1.41	1.11	16.98

表 2 不同 CUDA 线程下 ICP-GPU 算法的耗时对比

线程总数	法向计算时间/ms	更新点云时间/ms	匹配点对时间/ms	变换矩阵时间/ms	总耗时/ms
256=32×8	5.12	0.50	1.28	0.32	8.94
240=80×3	5.11	0.50	1.38	0.32	9.04
240=40×6	5.13	0.50	1.40	0.32	9.09
64=16×4	5.20	0.60	1.01	0.33	9.39
768=32×24	5.13	0.53	1.30	0.37	9.30

通过比较，采用 32×8=256 个线程时实时性最好，运行时间是 CPU-ICP 算法的 11.9%，加速了 8.4 倍。CUDA 线程数大致为 256 时，算法时间最短；线程数距 256 越近，CUDA 利用率越高，算法时间越短；同时，由于 CUDA 适合处理高度密集型数据，分配线程时应兼顾考虑数据的二维空间分布。Kinect 摄像头获取深度图像帧频为 30 f/s，每一帧周期为 T_{kinect}=33 ms，基于 GPU 的点云配准算法时间 T_{icp} 占用时间不到 10 ms，ICP 算法只占用深度图像周期的 1/3，实时性较好，分析其原因为：(1)充分利用 CUDA 多线程并行处理。(2)投影映射法节省匹配点对获取时间。(3)点到切平面的误差函数的选择减少了迭代次数。(4)实时性与投影映射法相辅相成，确保了帧间差距小，投影映射法获取匹配点对的正确率增加，投影映射法解决了匹配点对获取耗时长的问题，进一步提高了实时性。

综上所述，本文算法可用于各类传感器(激光、红外、超声波、结构光等)，常见点云包含 640×480 =307 200 个点，算法的最优耗时仅 ICP-CPU 算法的 11.9%，基本满足所有距离传感器的实时性要求。

5 结束语

本文提出一种基于 GPU 的实时三维点云数据配准算法。分析投影映射法，并与点到切平面距离误差函数相结合，兼顾时间与精度的可行性，将不同摄像机坐标系下的点云统一到同一世界坐标系下，并跟踪到了摄像机位置，为机器人视觉导航中实时三维重建与机器人自定位打下基础。实验结果表明，该算法实时性好、配准精度高，满足了实时三维重建的要求，可用于机器人同时定位和构建三维地图。然而，该算法仅适用于刚性变换点云，对于柔性变换及运动目标的三维重建将是今后的研究方向。

参考文献

[1] 向学勤, 潘志庚, 童 晶. 深度相机在计算机视觉与图形学上的应用研究[J]. 计算机科学与探索, 2011, 5(6): 481-489.

[2] Besl P J, McKay N D. A Method for Registration of 3D Shapes[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1992, 14(2): 239-256.

[3] Blais G, Levine M. Registering Multiview Range Data to Create 3D Computer Object[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1995, 17(8): 820-824.

[4] Liu Yonghuai. Constraints for Closest Point Finding[J]. Pattern Recognition Letters, 2008, 29(7): 841-851.

[5] Chen Yang, Medioni G. Object Modeling by Registration of Multiple Range Images[J]. Image and Vision Computing, 1992, 10(3): 145-155.

[6] Park S Y, Subbarao M. An Accurate and Fast Point-to-plane Registration Techique[J]. Pattern Recognition Letters, 2003, 24(16): 2967-2976.

[7] 吴恩华, 柳有权. 基于图形处理器(GPU)的通用计算[J]. 计算机辅助设计与图形学报, 2004, 16(5): 601-602.

[8] Hubert N. The New Book GPU Gems3[EB/OL]. (2010-11-21). http://developer.download.nvidia.com/shaderlibrary/webpages/shader_library.html.

[9] Izadi S, Kim D, Hilliges O, et al. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera[C]//Proc. of ACM Symposium on User Interface Software and Technology. Santa Barbara, USA: [s. n.], 2011.

[10] Newcombe R A. KinectFusion: Real-time Dense Surface Mapping and Tracking[C]//Proc. of IEEE International Symposium on Mixed and Augmented Reality. Basel, Switzerland: [s. n.], 2011.

[11] Rusinkiewicz S, Levoy M. Efficient Variants of the ICP Algorithm[C]//Proc. of the 3rd International Conference on 3D Digital Imaging and Modeling. Quebec, Canada: [s. n.], 2001.

[12] Sanders J, Edward K. CUDA by Example: An Introduction to General-purpose GPU Programming[M]. [S. l.]: Addison-Wesley Professional, 2010.

[13] 阿 信. 微软官方博客揭秘 Kinect 工作原理[EB/OL]. (2011-01-21). http://www.digg360.com/2011/01/how_you_become_the_controller.

编辑 刘 冰