第 58 卷 第 2 期/2021 年 1 月/激光与光电子学进展 研究论文

激光与光电子学进展 先进成像

一种适用于行星表面特征提取的实时 SIFT 算法

单宝彦[1,2,3]，朱振才[1*]，张永合[1,3]，邱成波[1,2,3]

[1]中国科学院微小卫星创新研究院，上海 201203；

[2]中国科学院大学，北京 100049；

[3]中国科学院微小卫星重点实验室，上海 201203

摘要 在行星探测任务中，针对尺度不变特征变换(SIFT)算法计算量大，无法同时满足对导航算法准确性和实时性要求的问题，提出了一种基于快速高斯模糊的并行化 SIFT 算法，即 FG-SIFT 算法。首先，将算法中构建高斯金字塔的二维高斯核函数分离成两个一维高斯函数，降低算法的计算复杂度。然后，对于每一维高斯函数，使用两个无限脉冲响应滤波器串联进行逼近，进一步减少计算量。最后，利用并行化处理的优势，设计算法各部分的并行化计算方案。仿真结果表明，FG-SIFT 算法的计算效率相较于原 SIFT 算法平均提高了 15 倍，相较于没有使用快速高斯模糊的 SIFT 算法，在图形处理器上的运行效率也有近 2 倍的提高，很大程度上减少了特征点提取的计算时长，提高了算法的实时性。

关键词 图像处理；尺度不变特征变换算法；快速高斯模糊；CUDA；实时性

中图分类号 TP391 **文献标志码** A **doi**：10.3788/LOP202158.0210020

A Real-Time SIFT Algorithm for Planetary Surface Feature Extraction

Shan Baoyan[1,2,3], Zhu Zhencai[1*], Zhang Yonghe[1,3], Qiu Chengbo[1,2,3]

[1]*Innovation Academy for Microsatellites of Chinese Academy of Sciences, Shanghai* 201203, *China*;

[2]*University of Chinese Academy of Sciences, Beijing* 100049, *China*;

[3]*Key Laboratory of Microsatellite, Chinese Academy of Sciences, Shanghai* 201203, *China*

Abstract In order to solve the problem that the scale invariant feature transform (SIFT) has a large amount of calculation and cannot meet the requirements of accuracy and real-time in the navigation algorithm, a parallel SIFT algorithm FG-SIFT based on fast Gaussian blur is proposed. First, the two-dimensional Gaussian kernel function, which constructs the Gaussian pyramid, is separated into two one-dimensional Gaussian functions to reduce the computational complexity. Then, two infinite impulse response filters are used in series to approximate each one-dimensional Gaussian kernel function to further reduce the computational complexity. Finally, using the advantage of parallel processing, the parallel computing scheme of each part of the algorithm is designed. Simulation results show that the computational efficiency of FG-SIFT algorithm is 15 times higher than that of the original SIFT algorithm, and the running efficiency of FG-SIFT algorithm on graphics processing unit is nearly 2 times higher than that of SIFT without fast Gaussian blur. This algorithm greatly reduces the calculation time of feature point extraction and improves the real-time performance.

Key words image processing; scale invariant feature transform algorithm; fast Gaussian blur; CUDA; real time

OCIS codes 100.3008; 100.2000; 110.2970

收稿日期：2020-06-18；**修回日期**：2020-07-09；**录用日期**：2020-10-13

基金项目：中国科学院战略性先导科技专项(A 类)空间科学背景型号项目(XDA15020305)

*E-mail：zczhu@hotmail.com

1 引 言

随着航天技术的发展，行星探测任务从地基空基望远镜观测逐渐延伸到航天器的近距离观测[1]，例如：美国发射的“勇气号”火星车任务和日本对近地小行星 1999 JU3 进行的“隼鸟 2 号”任务等。由于一般的行星与地球距离遥远，难以在地球与探测器之间建立实时有效的通信，因此，探测器需要具备自主导航的能力。这就需要探测器能够实时感知周围的环境，而视觉传感器具有采集信息量大、特征丰富以及成本较低的特点，使其在远距离探测中发挥着越来越重要的作用。在航天器进入低高度观测阶段，可利用视觉传感器所采集的图像信息，对相邻帧图像的重叠部分区域进行特征匹配，进而解算前后移动的相对位置关系，实现探测器自主光学导航的目的。同时，深空小天体距离遥远、表面环境复杂且未知，对于科学价值高的 C 类小行星，存在表面分化层颗粒细密、反照率低等特点。这不仅需要对图像进行实时在线处理，还对导航算法的精度和鲁棒性提出了较高的要求。

尺度不变特征变换（SIFT）算法，是一种自动检测图像中的关键点信息，并生成描述符的算法[2]。由于其自身对图像具有尺度不变性，而且对于仿射变换、噪声、光照变化等具有一定的鲁棒作用，使其特征提取匹配的精度远高于其他方法，符合行星探测任务中鲁棒性的需要。但是该算法的计算复杂度高，算法耗费时间长，不能满足实时性的要求。针对这一问题，国内外学者进行了很多研究。Zhao 等[3]在确定关键点的主方向时，充分考虑了距离关键点较近的像素的贡献，将检测点同尺度邻域内的其他 14 个点和相邻尺度的邻域进行比较，减少了计算的开销。Li 等[4]利用 Sobel 边缘检测器生成边缘群尺度空间，SIFT 检测器在边缘群尺度空间的约束下检测极值点。李鹤宇等[5]通过对灰度图像增强预处理，弥补了生成灰度图时信息丢失的问题，再用 DAISY 描述子替代 SIFT 描述子进行图像的特征点提取。Bay 等[6]利用积分图像和更小维度的描述子向量来简化算法的计算量，提升算法性能。这些方法虽然有效地降低了运算复杂度，但却损失了特征提取匹配的精度和算法的精确性。

因此，本文提出了一种使用快速高斯模糊的并行化 SIFT 算法 FG-SIFT。利用图形处理器（GPU）在并行计算和浮点计算方面的优势[7-9]，优化各计算步骤在 GPU 和中央处理器（CPU）端的分配、调度及数据传输，在保证一定匹配精度的情况下，提高算法的计算效率，以满足在航天器自主光学导航任务中对特征点匹配结果的准确性和实时性的要求。

2 SIFT 算法原理

SIFT 算法的实质是在不同尺度空间检测关键点，求取关键点的主方向并生成特征描述子。主要流程包括：尺度空间的构建、定位关键点并确定主方向、生成各个关键点的特征描述向量[2]。

1）构建高斯差分金字塔

尺度空间的构建由对图像的降采样处理和高斯模糊来实现。首先第 0 组图像由原图像扩大 2 倍得到，然后再对图像依次降采样得到一组从下到上依次减小的塔状模型，根据原始图像的大小，可以得到共计 o 组图像，然后对每一组图像进行不同尺度的高斯模糊。将高斯模糊后的图像记为 $L(x,y,\sigma)$，其中 x、y 用来标记像素点的位置，σ 是高斯函数的标准差。原图像记为 $I(x,y)$，那么高斯模糊后的图像可以表示为原图像 $I(x,y)$ 与高斯核函数 $G(x,y,\sigma)$ 的卷积，即

$$L(x,y,\sigma)=G(x,y,\sigma)*I(x,y), \tag{1}$$

其中，二维高斯核函数为

$$G(x,y,\sigma)=\frac{1}{2\pi\sigma}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)。 \tag{2}$$

改变高斯核 σ 的值，就可以得到一组不同尺度下的高斯图像，由此生成图像的高斯金字塔。在进行极值点检测的过程中，还需要构建高斯差分金字塔（DOG）。其函数表示为 $D(x,y,\sigma)$，是由高斯金字塔中一组图像的相邻层图像灰度值相减得到，即

$$D(x,y,\sigma)=[G(x,y,k\sigma)-G(x,y,\sigma)]*I(x,y)=L(x,y,k\sigma)-L(x,y,\sigma)。 \tag{3}$$

2）关键点精确定位

在构建好的 DOG 差分尺度空间进行局部极值点检测。对当前像素值与其周围像素及相邻上下两层共计 26 个像素的灰度值进行比较。若该像素的灰度值为极大或极小值，则该像素被确定为一个关键点的候选点。

由于上述方法选择的关键点是在离散空间的极值点，还需要通过线性插值的方法对关键点进行精

确定位。获取关键点精确位置的插值函数用 DOG 函数在尺度空间的二阶泰勒展开来表示

$$D(\boldsymbol{X})=D+\left(\frac{\partial D}{\partial \boldsymbol{X}}\right)^{\mathrm{T}}\boldsymbol{X}+\frac{1}{2}\boldsymbol{X}^{\mathrm{T}}\frac{\partial^{2}D}{\partial D^{2}}\boldsymbol{X},\quad(4)$$

式中：$\boldsymbol{X}=(x,y,\sigma)^{\mathrm{T}}$。

对(4)式求导数并使方程等于零，可得到极值点的偏移量为

$$\hat{\boldsymbol{X}}=-\left(\frac{\partial^{2}D}{\partial \boldsymbol{X}^{2}}\right)^{-1}\frac{\partial D}{\partial \boldsymbol{X}}。\quad(5)$$

极值点的函数值可表示为

$$D(\hat{\boldsymbol{X}})=D+\frac{1}{2}\left(\frac{\partial D}{\partial \boldsymbol{X}}\right)^{\mathrm{T}}\hat{\boldsymbol{X}}。\quad(6)$$

经过有限次的迭代，求得关键点原位置加上极值点的偏移量，即为关键点的精确位置。

3) 关键点方向分配

为了满足关键点的旋转不变性，对关键点局部梯度做直方图统计，确定统计直方图角度的最大值作为关键点的主方向。同时，把大于统计直方图最大值 80%的方向作为关键点的辅方向。至此，获得了关键点的精确位置、尺度和主方向。

4) 生成描述符

确定关键点的主方向后，将坐标轴旋转至关键点的主方向上，以确保描述符的旋转不变性。在关键点的 4×4 邻域内，每个邻域作为一个种子点，统计 8 个方向的灰度梯度直方图。共计得到 4×4×8=128 维的特征向量，作为关键点的特征描述子。

3 基于快速高斯模糊的 SIFT 改进算法

为了增强特征点提取的鲁棒性，SIFT 算法需要在构造的多组高斯金字塔中进行特征点检测。而高斯金字塔的构建及高斯模糊的过程，是 SIFT 算法中耗时最长的一部分。本文算法通过分离二维高斯核及使用两个级联 IIR 滤波器对分离后的一维高斯函数进行逼近，代替原算法中的二维高斯模糊来构建差分金字塔，并在建立高斯差分金字塔及 SIFT 特征点检测时，使用统一计算设备架构(CUDA)来加速算法的实现过程。具体算法流程如图 1 所示，虚线右侧是原算法的实现流程，左侧为本文改进后的算法流程。

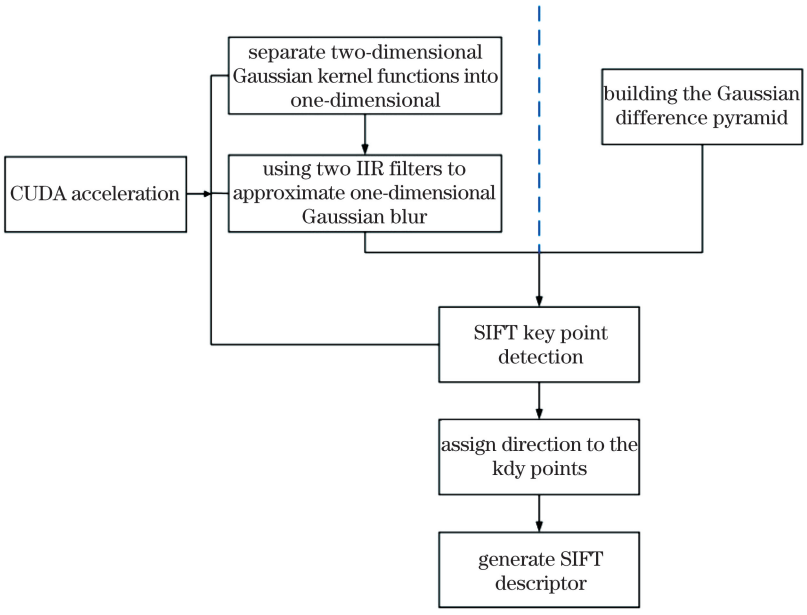

图 1　本文算法流程图

Fig. 1　Flow chart of proposed algorithm

3.1　分离二维高斯核函数

根据(1)式，选取半径大小为 3σ 的窗口进行高斯模糊计算，对于每个像素需要进行 36σ^{2} 次乘加运算。由此可得，对于大小为 $m\times n$ 的图像，进行一次二维高斯模糊运算所需的时间复杂度为 $o(m\times n\times\sigma^{2})$。注意到(2)式可以分解为

$$G(x,y,\sigma)=\frac{1}{2\pi\sigma^{2}}\exp\left(-\frac{x^{2}}{2\sigma^{2}}\right)\exp\left(-\frac{y^{2}}{2\sigma^{2}}\right)。\quad(7)$$

则(1)式可以写为

$$L(x,y,\sigma)=\sum_{i=x-3\sigma}^{i=x+3\sigma}\sum_{j=y-3\sigma}^{j=y+3\sigma}I(i,j,\sigma)\times G(i-x,j-y,\sigma)=\sum_{i=x-3\sigma}^{i=x+3\sigma}\frac{1}{\sqrt{2\pi}\sigma}\exp\left[-\frac{(i-x)^2}{2\sigma^2}\right]*\sum_{j=y-3\sigma}^{j=y+3\sigma}\frac{1}{\sqrt{2\pi}\sigma}\exp\left[-\frac{(j-y)^2}{2\sigma^2}\right]\times I(i,j,\sigma)。\tag{8}$$

记 $g(p,\sigma)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{p^2}{2\sigma^2}\right)$ 则(8)式可以写为

$$L(x,y,\sigma)=\sum_{i=x-3\sigma}^{i=x+3\sigma}g(i-x,\sigma)*\sum_{j=y-3\sigma}^{j=y+3\sigma}g(j-y,\sigma)\times I(i,j,\sigma)。\tag{9}$$

记 $h(x,y)=\sum_{j=y-3\sigma}^{j=y+3\sigma}g(j-y,\sigma)I(x,j)$，则可得到

$$L(x,y,\sigma)=\sum_{i=x-3\sigma}^{i=x+3\sigma}g(i-x,\sigma)h(i,y)。\tag{10}$$

由此，对图像进行二维高斯模糊可以分解为两次一维高斯模糊。由(10)式可知，第一次一维高斯模糊的时间复杂度为 $o(m\times n\times\sigma)$，第二次高斯模糊的复杂度也为 $o(m\times n\times\sigma)$。因此，对于分解为两次一维高斯模糊后的算法，时间复杂度为 $o(m\times n\times\sigma)$。

3.2　一维快速高斯模糊

对于分离出的一维高斯函数，可将其近似并分解为两个递归滤波器串联。

$$\begin{cases}L_{\mathrm{mid}}(x)=I(x)+a_0I(x)+a_1I(x-1)+b_1L_{\mathrm{mid}}(x-1)+b_2L_{\mathrm{mid}}(x-2)\\L(x)=L_{\mathrm{mid}}(x)+a'_0I(x)+a'_1I(x+1)+b_1L(x+1)+b_2L(x+2)\end{cases},\tag{11}$$

式中：a_0、a_1、a'_0、a'_1、b_1、b_2 为系数。与 σ 的关系分别为

$$\begin{cases}a_0=\dfrac{(1-\lambda)^2}{1+2\alpha\lambda-b_2}\\a_1=a_0\lambda(\alpha-1)\\a'_0=a_0\lambda(\alpha+1)\\a'_1=-a_0b_2\\b_1=-2\lambda\\b_2=\exp(-\alpha)\end{cases},\tag{12}$$

式中：$\alpha=\frac{\exp(0.726^2)}{\sigma}$；$\lambda=\exp(-\alpha)$。由此，一维高斯模糊可近似为两次处理，第一次从前至后，称为前向滤波，第二次从后至前，称为后向滤波。

一维快速高斯模糊可复用其前后两次输出的结果进行计算，使得算法的计算时间复杂度与窗口半径无关，即运算复杂度与 σ 取值无关，所以算法的复杂度为 $o(m\times n)$。

4　仿真设计及结果分析

4.1　快速高斯模糊的并行化实现方案

由(11)式可知，为实现半径无关的高斯模糊算法，一维高斯模糊中的前向滤波和后向滤波分别依赖于前后两次的输出结果。因此，对于每次一维高斯模糊的计算，图像的每一行或列应当被分配到同一个线程中。由于图像一般是以行主序存储，那么如果进行横向一维高斯模糊，则会导致同一个 warp 中每个线程所读取的内存不连续。所以，为了保证 warp 在同一时刻读取连续的内存，本节选择实现纵向一维高斯模糊，之后再转置图像，进行第二次纵向快速高斯模糊。在单个线程内，每次前向滤波的迭代，其取像素地址自增图像宽度；每次后向滤波的迭代，其取像素自减图像宽度。并且对于每个线程 id，即 blockIdx.x * blockDim.x + threadIdx.x，都代表连续的图像横坐标。高斯核函数实现如图 2 所示。

由于每次一维高斯模糊需要进行两次滤波运算，为了减少访存带来的延迟，可使用共享内存存放

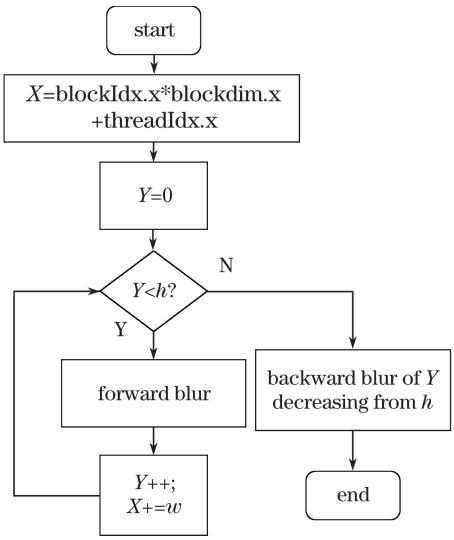

图 2　高斯核函数实现流程

Fig. 2　Gaussian kernel function implementation process

中间结果。在灰度图中，每个像素需占用一个字节，且需要存储一次中间结果。由此，对于本文所使用的 GPU，在使用共享内存存储中间结果时，最大可支持对高度为 1536 pixel 的图像进行处理。由于图像转置之后需要再进行一次一维高斯模糊，所以最大可支持 1536 pixel×1536 pixel、深度为 8 bit 的灰度图进行处理。由(11)式可知，在进行后向模糊时，也可以将中间结果存入输出缓存，避免对图像大小的限制，但是访问全局内存会带来延迟问题。因此，本文采取根据图像大小选取中间结果存放位置的方案。为了避免 Bank Conflict，对共享内存中的临时数据采取如表 1 所示的布局。

表 1　block 内共享内存布局

Table 1　Shared memory layout in block

Bank0	Bank1	Bank2	Bank3	Bank4	…
(0,0)→(0,7)	(1,0)→(1,7)	(2,0)→(2,7)	(3,0)→(3,7)	(4,0)→(4,7)	…
(0,8)→(0,15)	(1,8)→(1,15)	(2,8)→(2,15)	(3,8)→(3,15)	(4,8)→(4,15)	…
…	…	…	…	…	…

4.2　关键点提取的并行化实现方案

以 Group 为单位进行关键点提取运算。对于每个 Group，分别对每张图像进行关键点初筛。由于需要对每个点周围像素以及上下两层图像对应的像素进行比较，因此需要对全局内存进行随机访问。对于每张图像，仍然采取按照列分配线程的方式进行运算，其流程如图 3 所示。

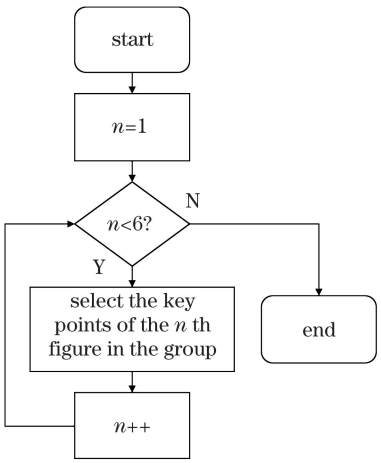

图 3　关键点提取并行化流程

Fig. 3　Key point extraction parallelization process

对于每个初筛结果，分配一个线程进行关键点的精确定位和筛选，并同时传入以初筛结果个数为依据分配的显存，用以存储精确筛选的特征点。对于舍弃的特征点，存入坐标(−1，−1)。为每个精确特征点分配一个线程计算主方向和特征向量，并传入合适大小的显存区域，存放结构如下：{(x，y)｜描述子}。最后，将得到的结果拷贝回主机内存，在 CPU 上进行特征点匹配。

4.3　快速高斯模糊与高斯模糊结果对比

为了保证结果的精确性，本节对比了基于 CUDA 实现的快速高斯模糊和原高斯模糊算法的计算结果，分别取 $\sigma\in\{\sigma\,|\,\sigma=\sigma_0 2^{\frac{s}{S}+n}, S=5, s=0, n=1,2,\cdots,6\}$，统计每个 σ 取值时两种算法计算结果的平均误差和最大误差，其统计结果如表 2 所示。

表 2　不同 σ 值的误差统计

Table 2　Error statistics of different σ values

Number	σ value	Average error /%	Maximum error /%
1	1.903	0.810	7.059
2	2.263	0.487	7.059
3	2.693	0.333	7.451
4	3.200	0.823	7.059
5	3.805	0.506	7.843
6	4.525	0.493	8.627

对于所有图像，本文使用的快速高斯模糊算法与原高斯模糊算法之间的误差分布如图 4 所示。

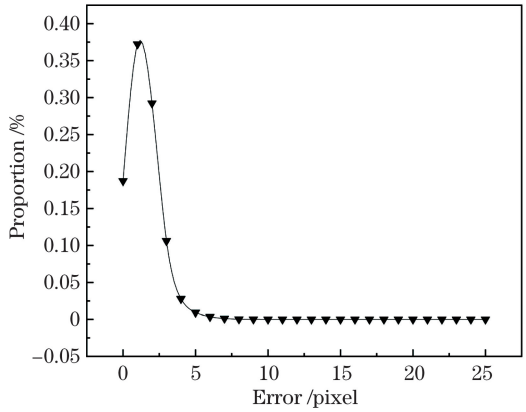

图 4　误差分布图

Fig. 4　Error distribution

由图 4 可知，基于 CUDA 的快速高斯模糊算法相较于原高斯模糊算法，其计算结果误差集中分布于 5 pixel 以下，误差极小。

4.4　仿真结果及性能分析

目前，比较好的 SIFT 改进算法有加速稳健特征(SURF)[6]和 SiftGPU[10]。区别于 SIFT 利用高斯差分金字塔进行极值点检测，SURF 主要使用

Hessian 矩阵行列式近似值图像来检测关键点，并采用盒型滤波器和积分图来简化计算，使得计算速度有了 3 倍的提升。SiftGPU[10]是当前使用最为广泛的 SIFT 算法的 GPU 版本，可采用 CUDA 或 OpenGL 对 SIFT 算法进行加速。为了对本文算法的精确性和实时性进行评估，本节将选取四组不同旋转角度及四组不同像素尺寸的模拟小行星实物影像作为测试图像进行仿真实验，并与开源库 OpenCv 提供的 SIFT 算法、文献[6]中的 SURF 算法、文献[10]中的 SiftGPU 算法进行对比。为了保证实验结果的准确性，所有算法的验证过程均使用统一硬件平台，主要配置为 AMD Ryzen7 3800X + RTX2070。图 5 展示了在模拟小行星旋转角度为 24°时，四种算法对相邻图像序列进行匹配，并使用随机采样一致(RANSAC)算法剔除误匹配后的结果。

表 3 统计了在选取的四组不同旋转角度测试图

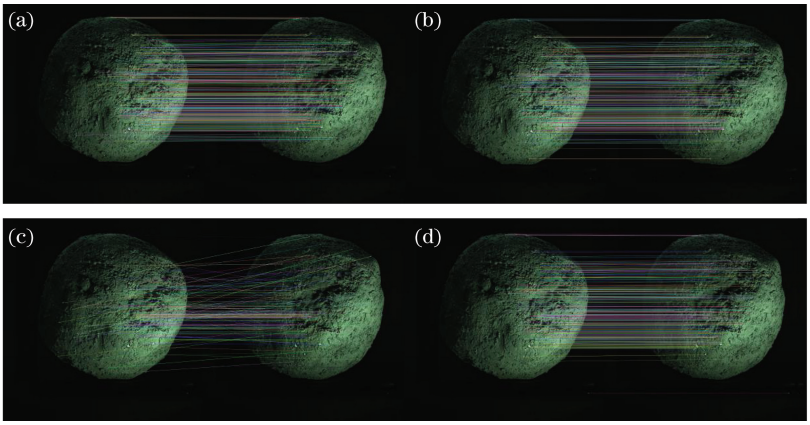

图 5 四种算法的特征点匹配结果。(a) SIFT 算法；(b) SiftGPU 算法；(c) SURF 算法；(d) FG-SIFT 算法

Fig. 5 Feature point matching results of four algorithms. (a) SIFT algorithm; (b) SiftGPU algorithm; (c) SURF algorithm; (d) FG-SIFT algorithm

表 3 四种算法匹配效果

Table 3 Matching effect of four algorithms

Algorithm	Image	Group 1	Group 2	Group 3	Group 4
	Real angle /(°)	4.27	8.82	14.60	24.00
OpenCv_SIFT	Total matches	6938	6906	3344	830
	Effective matches	6798	6536	2867	428
	Matching accuracy /%	98.0	94.6	85.7	51.6
	Calculated angle /(°)	4.34	9.03	14.31	24.31
	Angel error /%	1.64	2.38	1.99	1.29
SiftGPU	Total matches	6766	6408	3147	734
	Effective matches	6601	6014	2715	379
	Matching accuracy /%	97.6	93.9	86.3	51.6
	Calculated angle /(°)	4.21	9.03	14.91	24.29
	Angel error /%	1.41	2.38	2.12	1.21
SURF	Total matches	11844	12582	7608	1578
	Effective matches	7644	5939	1902	161
	Matching accuracy /%	64.5	47.2	25.0	10.2
	Calculated angle /(°)	4.54	7.50	10.44	27.61
	Angel error /%	6.32	14.97	28.49	15.04
FG-SIFT	Total matches	6916	6326	2580	726
	Effective matches	6481	5964	2180	379
	Matching accuracy /%	93.7	94.3	84.5	52.2
	Calculated angle /(°)	4.36	8.59	14.26	24.70
	Angel error /%	2.11	2.61	2.33	2.92

像下，不同算法的匹配结果。其中，使用有效的匹配数量在总匹配数量中的占比来表示匹配结果的准确度，使用匹配的特征点解算出的模拟行星旋转角度与每组图像的真实转角之间的误差作为匹配精度。由表 3 对比结果可知：原 SIFT 算法在四组不同的旋转角度下，匹配结果的准确度和精度均为最高；SiftGPU 由于仅对 SIFT 算法进行了并行化，并未对算法计算原理进行改变，所以匹配结果与 SIFT 相仿；SURF 算法在每组匹配中均获得了数目最多的匹配数量，但是有效的匹配数量随着旋转角度的增大而快速减少，导致匹配结果的准确度远低于其他算法。且 SURF 算法解算出的角度误差较大；FG-SIFT 是对原 SIFT 算法进行了改进，匹配效果略有下降，但仍保持了较高的匹配精度和准确度。

为了进一步验证本文算法在计算效率上的提升，图 6 统计了在对四组不同像素尺寸图像进行特征点提取时，不同算法的运算时间和加速比。其中，在计算效率的对比中，SURF 算法使用开源库 OpenCv 提供的 GPU 版本，在后文中使用 SURF_GPU 论述。图 7 统计了在只构建高斯差分金字塔的部分，OpenCv_SIFT、FG-SIFT 与 SiftGPU 的运算时间和加速比。

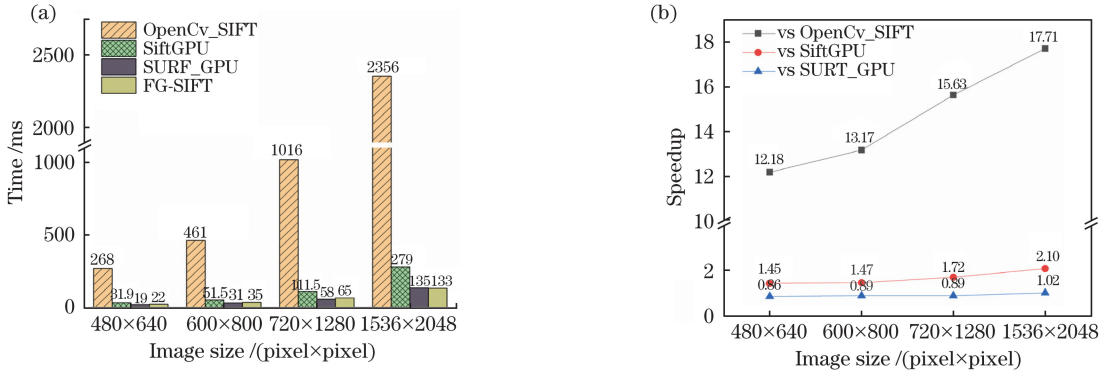

图 6　特征点提取耗时对比。(a)耗时统计；(b)算法加速比

Fig. 6　Time-consuming comparison of feature point extraction. (a) Time-consuming statistics; (b) algorithm speedup ratio

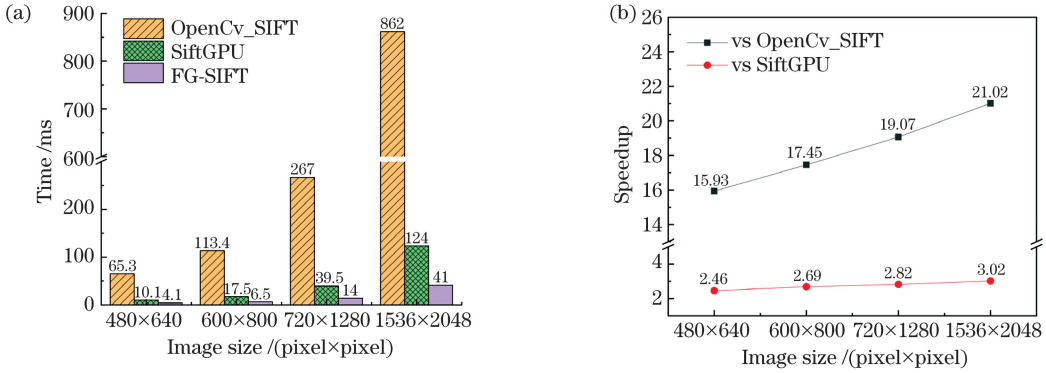

图 7　构建高斯金字塔耗时对比。(a)耗时统计；(b)算法加速比

Fig. 7　Time-consuming comparison of building Gaussian pyramid. (a) Time-consuming statistics; (b) algorithm speedup ratio

从图 6 可以看出，处理一张 720 pixel×1280 pixel 的图像时，使用原 SIFT 算法需要耗时 1016 ms，无法满足对算法实时性的要求。FG-SIFT 与 SURF_GPU 算法耗时相当，均为 60 ms 左右，表现最优。SiftGPU 虽然相比于原 SIFT 算法，在计算速度上有一定提升，但是与另外两种算法相比，还有一定差距。

总体上，使用本文基于快速高斯模糊的 FG-SIFT 算法，与原 SIFT 算法相比平均运算效率有了 15 倍左右的提升，与没有使用快速高斯模糊的 SiftGPU 方法相比，运算效率也提高了接近 2 倍。结合图 7 的统计结果可知，在只进行构建高斯差分金字塔的部分，相比于 SiftGPU 方法，计算效率平均提升了 2.9 倍。可见快速高斯模糊在提升整体算法的计算效率上表现出了较大的优越性。

5 结 论

本文针对传统 SIFT 特征提取算法计算量大、实时性不高、无法满足在深空探测场景中的应用需求的问题，提出了一种基于快速高斯模糊的 SIFT 改进算法。并利用 GPU 并行化处理的优势，设计构建高斯金字塔和特征点提取部分的并行化方案，基于 CUDA 平台进行性能校验。仿真结果表明，改进的 SIFT 算法在特征提取计算效率上比原算法提高了近 15 倍，并且仍然保证了合理的精度，相对于未使用快速高斯模糊的 SiftGPU 算法也有接近 2 倍的速度提升，可同时满足在行星探测任务中对算法准确性和实时性的需求。

参 考 文 献

[1] Xiang Y J, Li F H, Liu Z. Application of SIFT in vision navigation system of planet surface rover[J]. Transducer and Microsystem Technologies, 2011, 30(2): 129-131.
向永嘉, 黎福海, 刘泽. SIFT 在行星表面探测器视觉导航系统中的应用[J]. 传感器与微系统, 2011, 30(2): 129-131.

[2] Lowe D G. Distinctive image features from scale-invariant keypoints [J]. International Journal of Computer Vision, 2004, 60(2):91-110.

[3] Zhao J, Liu H, Feng Y, et al. BE-SIFT: a more brief and efficient SIFT image matching algorithm for computer vision[C]// IEEE International Conference on Computer & Information Technology Ubiquitous Computing & Communications Dependable. IEEE, 2015: 568-574.

[4] Li Y, Liu L S, Wang L H, et al. Fast SIFT algorithm based on Sobel edge detector [C]//2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet), April 21-23, 2012, Yichang, China. New York: IEEE Press, 2012: 1820-1823.

[5] Li H Y, Wang Q. A real-time SIFT feature extraction algorithm [J]. Journal of Astronautics, 2017, 38(8): 865-871.
李鹤宇, 王青. 一种具有实时性的 SIFT 特征提取算法[J]. 宇航学报, 2017, 38(8): 865-871.

[6] Bay H, Ess A, Tuytelaars T, et al. Speeded-up robust features (SURF) [J]. Computer Vision and Image Understanding, 2008, 110(3): 346-359.

[7] Wang B L, Zhu Z L, Meng L. CUDA-based acceleration algorithm of SIFT feature extraction[J]. Journal of Northeastern University (Natural Science), 2013, 34(2): 200-204.
王蓓蕾, 朱志良, 孟琭. 基于 CUDA 加速的 SIFT 特征提取[J]. 东北大学学报(自然科学版), 2013, 34(2): 200-204.

[8] Du C Y, Yuan J L, Dong J S, et al. GPU based parallel optimization for real time panoramic video stitching[J]. Pattern Recognition Letters, 2018, 133(5):62-69.

[9] Acharya K A, Venkatesh Babu R, Vadhiyar S S. A real-time implementation of SIFT using GPU [J]. Journal of Real-Time Image Processing, 2018, 14(2): 267-277.

[10] Wu C C. SiftGPU: a GPU implementation of scale invariant feature transform (SIFT)[EB/OL]. [2020-05-11]. https://github.com/stanley0614/SiftGPU.