第 36 卷 第 8 期 计 算 机 工 程 2010 年 4 月
Vol.36 No. 8 Computer Engineering April 2010
·图形图像处理· 文章编号：1000—3428(2010)08—0219—03 文献标识码：A 中图分类号：TP312

基于 CUDA 的尺度不变特征变换快速算法

田 文，徐 帆，王宏远，周 波
(华中科技大学电子与信息工程系，武汉 430074)

摘 要：针对尺度不变特征变换(SIFT)算法耗时多限制其应用范围的缺点，提出一种基于统一计算设备架构(CUDA)的尺度不变特征变换快速算法，分析其并行特性，在图像处理单元(GPU)的线程和内存模型方面对算法进行优化。实验证明，相对于 CPU，算法速度提升了 30~50 倍，对 640×480 图像的处理速度达到每秒 24 帧，满足实时应用的需求。

关键词：尺度不变特征变换；特征提取与匹配；图像处理单元；统一计算设备架构

Fast Scale Invariant Feature Transform Algorithm Based on CUDA

TIAN Wen, XU Fan, WANG Hong-yuan, ZHOU Bo
(Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan 430074)

【Abstract】 Aiming at the shortage of Scale Invariant Feature Transform(SIFT) algorithm in time consumption, this paper proposes a fast SIFT algorithm based on Compute Unified Device Architecture(CUDA), and analyzes its parallelism. It is further optimized according to the detailed analysis on the thread and memory model of the graphic hardware by using the power of Graphics Processing Unit(GPU). The GPU implementation runs 30~50 times faster than the CPU implementation in the experiments. It achieves 24 frames per second processing speed on 640×480 images, and is suitable for the real-time application.

【Key words】 Scale Invariant Feature Transform(SIFT); feature extraction and match; Graphics Processing Unit(GPU); Compute Unified Device Architecture(CUDA)

1 概述

图像特征点的提取与匹配是计算机视觉和模式识别领域研究得最多、最基础也是最困难的问题。很多相关应用如多视图三维重建[1]都把特征点提取与匹配作为一个基本步骤，包括全景图拼接、多视图重建及从视频中提取对象等。尺度不变特征变换(Scale Invariant Feature Transform, SIFT)算法[2]是一种相似不变算法，在图像经过尺度缩放和角度旋转后仍然能够提取稳定的特征，是该领域应用最广泛也是输出质量最好的算法之一。但 SIFT 算法计算复杂耗时多的缺点限制了其在更多领域拓展应用，如交互实时性十分重要的网络在线应用。针对图形处理单元(Graphic Processing Unit, GPU)和多核 CPU 对 SIFT 算法进行实现与优化成为近年来的研究热点。以常用的分辨率为 640×480 的图像为例，文献[3]基于 OpenGL 的算法实现在 NVIDIA 7900GTX 显卡上达到 10 FPS 的执行速度；文献[4]基于 OpenMP 算法实现在 Intel 8 核 CPU 上执行速度达到了 45 FPS，并对算法的可伸缩性进行了详细的分析与实验。

本文提出一种基于 NVIDIA 的统一计算设备架构(Compute Unified Device Architecture, CUDA)的 SIFT 快速算法。计算设备选择 GPU 是因为 SIFT 算法中相当一部分计算属于图像处理，而图像处理尤其是卷积滤波这种单指令多数据应用正是 GPU 的优势所在。相对于传统基于 OpenGL 和 DirectX 的 GPGPU 编程，CUDA 提供了更直观的编程模型和优化原则，并且运用 CUDA 编写的代码可以很好地运行在多核 CPU 上，代码的适用性更广泛。实验证明，本文算法能够对 640×480 的图像实现实时的特征提取与匹配，处理速度达到每秒 24 帧。实验中通过对比 SIFT 算法的开源实现，对算法的实现细节和实验结果进行了分析。

2 基于 CUDA 的 SIFT 算法优化与实现

2.1 尺度空间构建

构建尺度空间包含 3 个步骤：高斯滤波，图像缩放和 DoG (Difference of Gaussian)计算。首先将初始图像上传到全局内存中，随后绑定为纹理内存。与全局内存必须采用接合式的内存访问才能保持高性能不同，纹理内存能够享受 GPU 内部针对二维数据访问优化的纹理缓存，从而在随机存取的环境下保持较高的性能，另外，纹理内存还能享受硬件的双线性插值，从而完成需要的图像缩放操作。

3 个步骤中最耗时的是高斯滤波。以文献[2]推荐的算法参数 DoG 等级 $S=3$ 为例，每个组(Octave)内生成 $S+3=6$ 幅高斯图像，除去原图像共需要进行 5 次高斯滤波。每次高斯滤波之前，将在 CPU 端计算好的分离一维高斯核上传到全局内存或常量内存中，再先后启动 2 个核函数，在核函数中读取高斯核和图像数据完成卷积滤波。

在实际应用中，相对于 GPU 强大的浮点计算性能，内存的带宽往往会成为瓶颈，这是 GPU 硬件上内存带宽增长滞后于计算性能增长的客观事实决定的。对于标准差为 σ、窗宽为 $6\sigma+1$ 的一维高斯滤波，每个像素的滤波需要执行 $3\sigma+1$ 次内存访问读取高斯核，$6\sigma+1$ 次内存访问读取邻近像素值，共 $9\sigma+2$ 次内存读取，还需要执行 $6\sigma+1$ 次 MAD 指令完成卷积滤波计算。上述分析表明算法性能严重地被显存带宽限制。

作者简介：田 文(1983－)，女，博士研究生，主研方向：计算机视觉，三维重建；徐 帆，博士；王宏远，教授、博士生导师；周 波，讲师

收稿日期：2009-09-25 **E-mail：**twbigs@gmail.com

参考文献[5]中的算法，在计算过程中同步生成高斯核，可减少内存读取并增加算法的计算密集度。本文运用了一种类似前向差分的方法，对于高斯函数：

$$G_0(x)=\frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-\frac{x^2}{2\sigma^2}}$$

以卷积步长 $\delta=1$ 求商，有

$$G_1(x)=\frac{G_0(x+\delta)}{G_0(x)}=\mathrm{e}^{-\frac{\delta^2}{2\sigma^2}}\mathrm{e}^{-\frac{\delta x}{\sigma^2}}=\mathrm{e}^{-\frac{1}{2\sigma^2}}\mathrm{e}^{-\frac{x}{\sigma^2}}$$

$$G_2(x)=\frac{G_1(x+\delta)}{G_1(x)}=\mathrm{e}^{-\frac{1}{\sigma^2}}$$

从上式可知 $G_2(x)$ 是一个常量，在计算过程中不需要更新。通过简单的乘法增量计算可以获得其他的高斯核系数：

$$G_0(x)=G_0(x-1)G_1(x-1)$$

$$G_1(x)=G_1(x-1)G_2(x-1)$$

行滤波 CUDA 代码片段如下：

```
//CPU 端代码
float g0_initial = 1.0f / (sqrt(2.0f * 3.141 59f) * sigma);
float g1_initial = exp(-0.5 f / (sigma * sigma));
//GPU 端核函数代码
__global__
void KernelSmoothImageRow_Fly(float* output,
  float g0_initial, float g1_initial, int halfSize)
{ //计算当前线程在图像上的坐标
  const int idxX= __mul24(blockIdx.x, blockDim.x) + threadIdx.x;
  const int idxY=__mul24(blockIdx.y, blockDim.y) + threadIdx.y;
  const int idx =__mul24(idxY, gridDim.x) + idxX;
        float g0, g1, g2;
  g0 = g0_initial;
  g1 = g1_initial;
  g2 = g1 * g1;
  //高斯窗口中心点
  float res = tex2D(texInput, idxX, idxY) * g0;
  //中心点两侧的高斯核系数增量计算
  for (int i=1; i<halfSize; ++i)
  {       g0 *= g1;
          g1 *= g2;
res += tex2D(texInput, idxX+i, idxY) * g0;
          res += tex2D(texInput, idxX-i, idxY) * g0;    }
          output[idx] = res;}
```

改进的算法通过增加 $6\sigma+1$ 次乘法计算减少了 $3\sigma+1$ 次内存读取，一定程度上缓解了内存带宽的压力。实测性能从基于常量内存实现的每秒 59×10^7 像素提升到每秒 74×10^7 像素，算法效率提升了 25%。

2.2 特征点提取

判断一个像素点是否是 DoG 极值点分 3 步实现：

(1)与该像素点所在 DoG 图像中的 8 个邻近像素进行比较，同时检查是否达到对比度的阈值。

(2)与尺度空间内相邻 DoG 图像中的 18 个邻近像素进行比较。

(3)检查边缘效应。

经过第(1)步的判断只会剩下 5%左右的像素点，这样会尽量减少后续代码段的线程分支，从而保持较高的算法效率。

生成特征点列表通过 GPU 的数据流缩减实现，通过一个标准并行算法前缀求和来实现；本文算法中调用了 CUDPP (http://www.gpgpu.org/developer/cudpp/)中数据流缩减标准实现。

2.3 特征点方向计算

首先将特征点列表上的每个特征点映射为一个线程块，每个线程对应高斯图像一个邻域像素点，计算梯度方向和大小并累加到分组数为 36 的直方图上，完成直方图统计。特征点的梯度直方图放在线程块的共享内存中，共享内存是 GPU 芯片内部的高速缓存，带宽及访问延迟都大大优于全局内存。每个线程都会访问直方图，且直方图统计属于随机的散布式访问，很难优化，因此，运用 CUDA 高效的共享内存十分关键。最后由一个线程遍历存储在共享内存区的直方图，求得统计值最大的方向输出。这里存在 2 个实现细节问题：(1)每个特征点的尺度可能不一样，对应的统计窗口大小也会不一样，而 CUDA 要求每个线程块的大小一致且有大小限制。以一个组内高斯图像尺度最大为 $\sigma=3.2$ 为例，统计窗口的面积应为 $(9\sigma+1)^2\approx888$ ；但 CUDA 每个线程块最大的线程数为 512，无法实现像素与线程的一一对应。本文将线程块大小固定为 16×16，并在每个线程内计算采样坐标，实现在统计窗口内的均匀采样。(2)文献[2]的策略是在计算特征点方向和描述子之前计算并保存梯度方向和大小，供后续步骤计算时读取。但这个很合理的 CPU 算法优化流程并不适合 GPU：首先，GPU 的内存空间有限，额外保存梯度的方向和大小会使算法的内存占用量增加 1 倍；单独的计算梯度会使整个 GPU 算法增加一个核函数，额外的核函数载入时间也会影响算法效率。本文算法将计算梯度作为计算特征点方向的一个步骤，组合在同一个核函数中，这样做的缺点是在计算特征描述子时还需计算一遍梯度。

本文算法将直方图存放在共享内存中，使用 CUDA 提供的 atomicAdd 函数实现了共享内存上的累加原子操作，因此，不会造成线程间的内存同步问题。

2.4 特征描述子计算

取特征点对应尺度高斯图像上 16×16 的邻域，每 4×4 的邻域统计梯度方向形成一个分组数为 8 的直方图，从而获得长为 4×4×8=128 的特征描述子。统计梯度方向时会将特征点的方向计算在内，以获得特征的旋转不变性。最终输出的特征描述子经过正规化处理会获得图像亮度变化的不变性。每个特征点对应一个 16×16 的线程块。线程块内的每个线程对应一个采样点，计算梯度信息累加到存放在共享内存区的描述子上，最后将特征描述子从共享内存拷贝到全局内存输出。

2.5 SIFT 特征匹配

SIFT 特征描述子模为 1，相关系数为描述子的点积，全部的相关系数可以通过矩阵乘法计算：考虑图像 I 和图像 J 之间的特征匹配问题，图像 I 中有 m 个特征点，图像 J 中有 n 个；将图像 I 中所有特征的特征描述子整理为 $m\times128$ 的矩阵 $\boldsymbol{A}$，图像 J 中所有特征的特征描述子整理为 $n\times128$ 的矩阵 $\boldsymbol{B}$；计算 $\boldsymbol{AB}^{\mathrm{T}}$ 即可获得两图间特征的相关系数矩阵。

CUDA 提供了基础数值线性代数运算库 BLAS 的 GPU 实现 CUBLAS。本文首先使用 cublasSgemm 函数完成矩阵乘法计算，随后开启 m 个线程，在核函数中为图像 I 中的每个特征点搜索最近邻匹配点。

3 实验结果

本文的实验平台为 NVIDIA GTX260 GPU，内建 192 个 1.242 GHz 的处理器。实验中 SIFT 算法参数设置为：每个组的 DoG 层数为 3，组的个数根据图像大小由算法自动选择，实验 1 图像大小为 640×480，组数为 5，实验 2 图像大小为 2 048×1 536，组数为 7。本文算法 SiftCUDA 对比 CPU 参考

实 现 Siftpp(http://vision.ucla.edu/~vedaldi/code/siftpp/siftpp.html)和 GPU 参考实现 SiftGPU(http://www.cs.unc.edu/~ ccwu/siftgpu/)的实验结果如表 1 所示。

表 1 算法性能实验结果

性能	实验 1(640×480)			实验 2(2 048×1 536)		
	Siftpp	SiftGPU	SiftCUDA	Siftpp	SiftGPU	SiftCUDA
图像数据传输/ms		27.00	0.46		249.00	4.54
构建尺度空间/ms	537	30.96	9.32	6 559	117.70	45.38
提取特征点/ms	26	9.77	12.09	330	25.67	40.60
计算方向/ms		10.50	7.26		16.33	22.51
计算描述子/ms	597	10.97	10.62	7 784	34.96	36.82
特征匹配/ms			1.57			128.16
时间总计/ms	1 160.00	89.20	41.32	14 673.00	443.66	278.01
每秒帧数	0.86	11.21	24.20	0.07	2.25	3.60
特征点数	323	311	355	5 230	5 092	5 915

实验结果表明 SiftCUDA 及 SiftGPU 这 2 个 GPU 实现在性能上远远超过了基于 CPU 的 Siftpp。而 SiftCUDA 相比另外 2 种实现在图像数据传输和构建尺度空间上存在较大优势；SiftCUDA 实测的 CPU 到 GPU 数据传输速度达到了 2.7 GB/s，在图像数据传输方面也优势明显；构建尺度空间方面得益于 CUDA 直接的内存管理机制，也取得了一定的性能改善。SiftCUDA 在 640×480 的图像上达到了每秒 24 帧的速度，能够满足实时应用的需要。

在算法质量方面，图 1 是匹配阈值设置为 1.6 时的实验 1 的特征匹配结果，左侧展示了两图间特征点的匹配对应情况，右侧展示了两图像中匹配点的尺度及方向信息。

图 1 SIFT 特征匹配结果

本文还在 2 组标准数据(http://www.robots.ox.ac.uk/~vgg/research/affine/ index.html)上测试了特征的可重现性，实验结果见表 2、表 3，SiftCUDA 与参考实现 Siftpp 在可重现性上的误差在可接受范围内。

表 2 Bark 图像序列可重现性实验结果

尺度变化	1.2	1.8	2.5	3.0	4.0
Siftpp 的可重现性/(%)	75.8	77.1	86.0	84.9	87.3
SiftCUDA 的可重现性/(%)	75.9	78.2	86.1	87.1	92.3

表 3 Graf 图像序列可重现性实验结果

视角变化/(º)	20	30	40	50	60
Siftpp 的可重现性/(%)	76.50	60.10	25.20	0.45	0
SiftCUDA 的可重现性/(%)	77.90	68.30	26.00	0.45	0

4 结束语

本文提出一种 SIFT 的快速算法，基于 CUDA 的算法实现充分运用了 GPU 的计算能力，实验测试性能达到了 CPU 实现的 30 倍~50 倍，能够满足实时应用的需求。

参考文献

[1] 方 磊, 王宏远, 徐 帆, 等. 可并行迭代式图像序列射影重建策略[J]. 计算机工程, 2008, 34(9): 16-19.

[2] Lowe D G. Distinctive Image Features from Scale-invariant Keypoints[J]. International Journal of Computer Vision, 2004, 60(2): 91-110.

[3] Sinha S, Frahm J M, Pollefeys M, et al. Feature Tracking and Matching in Video Using Programmable Graphics Hardware[Z]. (2007-11-11). http://www.springerlink.com/content/5rv615p24360/.

[4] Zhang Qi, Chen Yurong, Zhang Yimin, et al. SIFT Implementation and Optimization for Multi-core Systems[C]//Proc. of IEEE International Symposium on Parallel and Distributed Processing. Miami, USA: IEEE Press, 2008: 1-8.

[5] Turkowski K. Incremental Computation of the Gaussian[M]// Nguyen H. GPU Gem 3. New Jersey, SA, USA: Addison Wesley Professional, 2007.

编辑 张正兴

(上接第 218 页)

效区域，同样也会提取那少量有效区域，从而必然产生误差。本文利用鱼眼图像有效区域的领域信息区别上述两者，提高了准确性，可以得到更理想的结果。

如何更快地进行区域生长是以后的工作方向之一，因为该部分耗费大量时间，所以如果能提高该部分效率，那么整个算法的效率将会大大提高。另外，如何充分利用经过区域生长后的二值图像也是以后的研究方向之一。本文只使用了二值图像的边缘信息，信息利用率还需进一步研究。

参考文献

[1] Szeliski R, Shum Heung-Yeung. Creating Full View Panoramic Image Mosaics and Environment Maps[C]//Proceedings of SIGGPAOH'97. Los Angeles, USA: IEEE Press, 1997: 251-258.

[2] 张 诚, 汪嘉业. 对鱼眼照片场景实现三位重建和虚拟浏览[J]. 计算机辅助设计与图形学学报, 2004, 16(1): 79-83.

[3] 黄有度, 苏化朋. 一种鱼眼图像到透视投影图像的变换模型[J]. 系统仿真学报, 2005, 17(1): 29-32.

[4] Ying Xianghua, Hu Zhanyi, Zha Hongbin. Fisheye Lenses Calibration Using Straight-line Spherical Perspective Projection Constraint[C]//Proc. of the 7th Asian Conf. on Computer Vision. Hyderabad, India: [s. n.], 2006: 591-600.

[5] 英向华, 胡占义. 一种基于球面投影约束的鱼眼镜头校正方法[J]. 计算机学报, 2003, 26(12): 1702-1708.

[6] Mundhenk T N, Michael J R, Liao Xiaoqun, et al. Techniques for Fisheye Lens Calibration Using A Minimal Number of Measurements[C]//Proc. of the SPIE Intelligent Robotics and Computer Vision Conference. Boston, Massachusetts, USA: [s. n.], 2000: 8-9.

[7] 蒋加伏, 谭 蓉, 杨鼎强. 基于轮廓特征和小波变换的图像拼接[J]. 计算机工程, 2009, 35(3): 225-226.

[8] 唐 俊, 赵为民, 谷 峰. 基于鱼眼图像的全景漫游模型[J]. 微机发展, 2003, 13(2): 69-70.

[9] 王大宇, 崔汉国, 陈 军. 鱼眼图像轮廓提取及校正研究[J]. 计算机工程与设计, 2007, 28(12): 2878-2882.

[10] 汪嘉业, 杨兴强, 张彩明. 基于鱼眼镜头拍摄的图像生成漫游模型[J]. 系统仿真学报, 2001, 11(13): 66-68.

[11] 汤红忠, 黄辉先, 张东波, 等. 基于 B 样条曲线模型的彩色图像分割[J]. 计算机工程, 2009, 35(2): 214-215.

[12] 运筹学教材编写组. 运筹学[M]. 3 版. 北京: 清华大学出版社, 2005.

编辑 张正兴