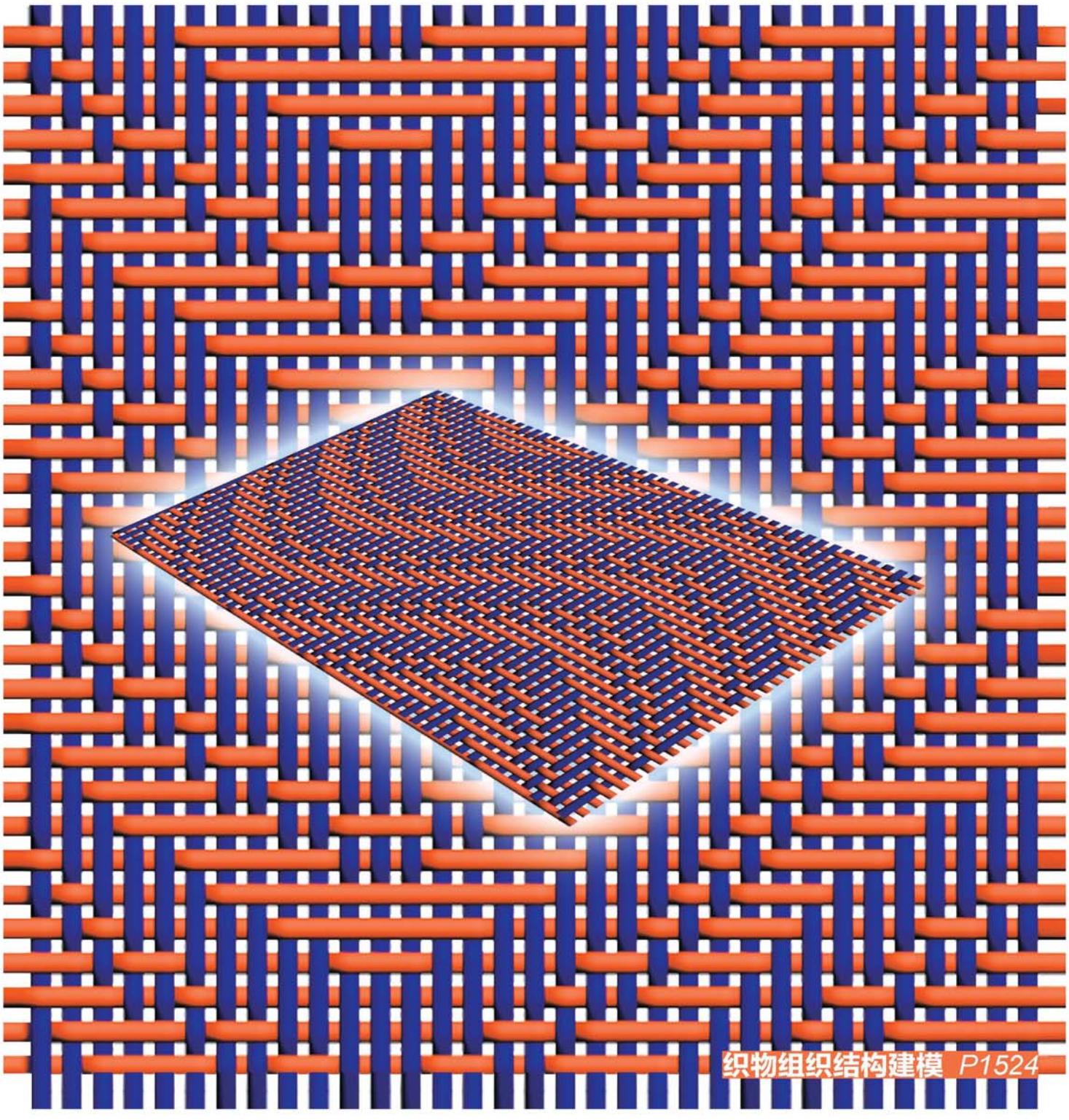
JOURNAL OF IMAGE AND GRAPHICS

主办：中国科学院遥感与数字地球研究所
中国图象图形学学会
北京应用物理与计算数学研究所

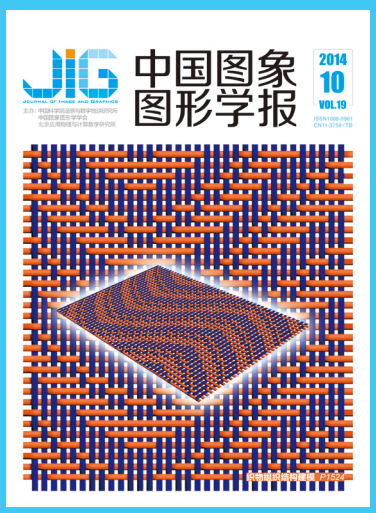
中国图象图形学报

2014
10
VOL.19

ISSN1006-8961
CN11-3758/TB



织物组织结构建模 P1524



第19卷第10期（总第222期）
2014年10月16日

中国精品科技期刊
中国国际影响力优秀学术期刊
中国科技核心期刊

中国图象图形学报

刊名题字：宋健　月刊（1996年创刊）

主管单位 中国科学院
主办单位 中国科学院遥感与数字地球研究所
中国图象图形学学会
北京应用物理与计算数学研究所

主　　编 李小文
编辑出版 《中国图象图形学报》编辑出版委员会
邮政信箱 北京9718信箱
邮　　编 100101
电子信箱 jig@irsa.ac.cn
电　　话 010-64807995
网　　址 www.cjig.cn

广告经营许可证 京朝工商广字第0361号
总 发 行 北京报刊发行局
订　　购 全国各地邮局
海外发行 中国国际图书贸易集团有限公司
（邮政信箱：北京399信箱　邮编：100048）
印刷装订 北京科信印刷有限公司

Journal of Image and Graphics

Title inscription: Song Jian　Monthly, Started in 1996

Superintended by Chinese Academy of Sciences
Sponsored by Institute of Remote Sensing and Digital Earth, CAS
China Society of Image and Graphics
Institute of Applied Physics and Computational Mathematics

Editor-in-Chief LI Xiaowen
Editor, Publisher Editorial and Publishing Board of Journal of Image and Graphics
P.O.Box 9718, Beijing, P.R.China
Zip code 100101
E-mail jig@irsa.ac.cn
Telephone 010-64807995
Website www.cjig.cn

Distributed by Beijing Bureau for Distribution of Newspapers and Journals
Domestic All Local Post Offices in China
Overseas China International Book Trading Corporation
(P.O.Box 399, Beijing 100048,P.R.China))
Printed by Beijing Kexin Printing Co., Ltd.

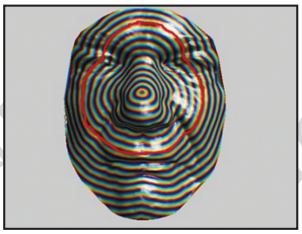
CN 11-3758/TB
ISSN 1006-8961
CODEN ZTTXFZ

国外发行代号 M1406
国内邮发代号 82-831
国内定价 50.00元

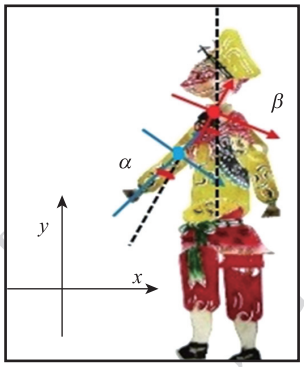
2014年10月 第19卷 第10期（总第222期）
中国图象图形学报
Zhongguo Tuxiang Tuxing Xuebao

目次

自适应梯度重建分水岭分割算法(第1430页)



基于特征点针对表情变化的3维人脸识别(第1459页)

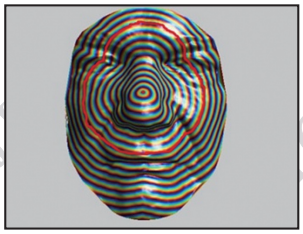


数字皮影的交互仿真和动画绘制(第1490页)

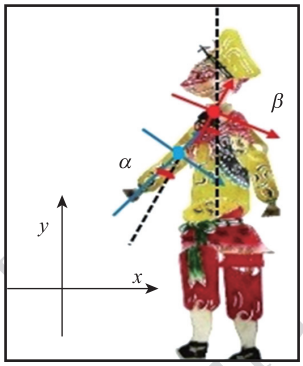
Volume 19, Number 10
Published Oct 16, 2014

CONTENTS
JOURNAL OF IMAGE AND GRAPHICS

Adaptive gradient reconstruction for watershed based image segmentation (P1430)



Expression-insensitive 3D face recognition method based on facial fiducial points(P1349)



Interaction and animation simulation of digital shadow play(P1490)

E-mail: jig@irsa.ac.cn
Website: www.cjig.cn
Tel: 010-64807995

中国图象图形学报
JOURNAL OF IMAGE AND GRAPHICS

中图法分类号：TP301.6　　**文献标识码**：A　　**文章编号**：1006-8961(2014)10-1481-09

论文引用格式：Chen B, Chen H P, Li X H. Near real time linear stereo cost aggregation on GPU[J]. Journal of Image and Graphics, 2014, 19(10): 1481-1489.[陈彬, 陈和平, 李晓卉. GPU 近实时线性双目立体代价聚合[J]. 中国图象图形学报, 2014, 19(10): 1481-1489.][DOI: 10.11834/jig.20141010]

GPU 近实时线性双目立体代价聚合

陈彬[1]，陈和平[2]，李晓卉[1]

1. 武汉科技大学信息科学与工程学院，武汉　430081；
2. 武汉科技大学计算机科学与技术学院，武汉　430074

摘　要：目的　近年来双目视觉领域的研究重点逐步转而关注其“实时化”策略的研究，而立体代价聚合是双目视觉中最为复杂且最为耗时的步骤，为此，提出一种基于 GPU 通用计算（GPGPU）技术的近实时双目立体代价聚合算法。**方法**　选用一种匹配精度接近于全局匹配算法的局部算法——线性立体匹配算法（linear stereo matching）作为代价聚合策略；结合线性代价聚合的原理，对其主要步骤（代价计算、均值滤波及系数求解等）的计算流程进行有针对性地并行优化。**结果**　对于相同的实验样本，用本文方法在 NVIDA GTX780 实验平台上能在更短的时间计算出代价矩阵，与原有的 CPU 实现方法相比，代价聚合的效率平均有了数十倍的提升。**结论**　实时双目立体代价聚合方法，为在个人通用 PC 平台上实时获取高质量双目视觉深度信息提供了一个高效可靠的途径。

关键词：双目视觉；代价聚合；GPU 通用计算；并行计算

Near real time linear stereo cost aggregation on GPU

Chen Bin[1], Chen Heping[2], Li Xiaohui[1]

1. *School of Information Science and Engineering, Wuhan University of Science and Technology, Wuhan* 430081, *China.*;
2. *School of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan* 430074, *China.*

Abstract: Objective Stereo vision depends on feasible approaches for real-time/hardware implementation. Cost aggregation, the most complex part of the stereo matching algorithm, substantially affects the overall running time. Therefore, this study proposes a novel parallelization strategy to map the stereo cost aggregation of graphics processing units (GPUs) using compute unified device architecture (CUDA). **Method** The linear stereo matching algorithm is selected as the stereo cost aggregation strategy in the proposed approach. Linear stereo matching with constant complexity can achieve more accurate disparity maps than global disparity optimization methods. Although its computation complexity is considerably less than that of most global approaches, linear stereo matching, even when optimized by some effective strategies, remains to demonstrate a performance that exceeds real-time or near real-time requirements for practical applications. The parallelization strategy introduced in this study is based on a separable filter with linear complexity in the filter window size and with proven efficiency on GPU platforms. The computation for each step (cost computation, mean filter, and coefficients computation) of the cost aggregation is reformulated, and the rational use of different types of GPU memory is ensured. This study proposes several parallelization optimizations to increase parallelism degree and data throughput. After being optimized by these parallelization optimizations, our approach ensures that the computation of each CUDA thread is independent of other threads and maximizes parallelism degree. These parallelization optimizations also reduce the complexity of each thread from

收稿日期：2014-01-13；**修回日期**：2014-06-05

基金项目：国家自然科学基金项目(61105070)

第一作者简介：陈彬(1981—　)，男，工程师，现为武汉科技大学控制理论与控制工程专业博士研究生，主要研究方向为计算机视觉。E-mail：chenbin@wust.edu.cn

中国图象图形学报
JOURNAL OF IMAGE AND GRAPHICS
Vol. 19, No. 10, Oct. 2014

the exponential relationship to the linear relationship with window radius and further improve the efficiency. The efficiency of the memory access and the data throughput are also dramatically improved in our final implementation, cached by texture or shared memories in certain circumstances. These experimental results show that the proposed strategy is effective and efficient. **Result** We dramatically accelerate the stereo cost aggregation on GPUs under the assistance of the outstanding parallel computation performance of GPUs. Compared with the original CPU implementation accelerated by the integral image technology, our CUDA implementation on a specific NVIDIA GTX780 GPU provides, on the same stereo image pairs, accurate cost matrix within a significantly shorter running time (less than 80 ms) and improves the average efficiency by tenfold. Our approach also outperforms other real-time or near real-time stereo cost aggregation implementations on GPUs. **Conclusion** The proposed approach outperforms the previous constant time stereo solutions and produces accurate results comparable with those of adaptive weight aggregation on GPUs with CUDA. It also provides an efficient and feasible method to obtain an accurate disparity map on general PC platforms in real time.

Key words: stereo vision; cost aggregation; general purpose GPU; parallel computing

0 引 言

双目视觉研究在计算机视觉领域是一个受到广泛关注的研究课题,而立体匹配又是该研究课题中一个至关重要且极具挑战性的研究热点[1-3]。双目视觉的立体匹配算法旨在从处于同一水平方向并经过严格校准、对齐的左右两幅图像中查找相同的物理点以确定这对图像中各个像素点的相互对应关系,并根据这些对应关系计算出每一个物理点距离拍摄位置的深度信息。文献[1-2,4]对双目视觉的立体匹配原理、步骤以及已有的算法作了详细介绍和分析。经由双目视觉匹配计算出的深度信息在机器人、自动驾驶及3维重建等研究领域有着广泛的应用。

目前,大部分双目立体匹配的代价聚合算法可以划分成两大类,即局部算法和全局算法。局部代价聚合算法一般是指在一个指定大小的窗口内设法提高信噪比进而聚合得到代价矩阵。而全局代价聚合算法则是在整个图像或图像的某个子集如(sim-global matching[5-6])内使指定的代价函数趋于极小值进而得出最终的代价矩阵。一般而言,局部算法的处理速度远快于全局算法,而大部分全局算法的最终结果要优于局部算法[2]。但随着局部代价聚合算法的发展,这种情形开始发生转变:一些先进的局部算法(如 fast bilateral stereo matching[7]及 Adaptive weight stereo matching[8-10])也能获得接近于全局算法的精确结果。即便如此,复杂的聚合过程依旧无法使这些局部算法在获得精确的代价矩阵的同时满足实时性的要求。Stefano 在其综述文献[1]中预测双目视觉立体匹配领域的研究重点将关注如何实时实现整个匹配过程。

目前,双目视觉立体匹配的实时方案一般是基于 FPGA(field programmable gate array)这一硬件结构设计并实现的[11-13],如:Gehrig 等人在文献[13]中描述了一种基于 FPGA 芯片的实时 Semi-global 匹配算法,能够每秒处理 25 帧大小为 640×480 像素且深度范围为 128 的双目图像。限于 CPU 的处理能力,面向个人通用 PC 平台的实时双目立体匹配算法的发展却较为缓慢。本文旨在利用 GPU 超强的并行计算能力,在通用 PC 平台上加速代价矩阵聚合过程使其满足实时性的要求。

综合考虑各种局部代价聚合算法的特点,选取线性代价聚合算法(linear stereo aggregation[14])作为代价聚合策略。线性代价聚合算法是基于 Ju 和 Kang 在复杂度为 $O(1)$ 的 Bilateral 滤波器[15]基础上提出的一个线性模型[16],通过该线性模型,整个线性代价聚合的算法复杂度为常数。与其他的局部聚合算法相比,线性代价聚合算法不仅能大大降低运行时间,还能利用彩色图像在多种深度选择策略的帮助下获得高品质的深度信息。为了充分结合 GPU 的并行运算能力和线性代价聚合算法的优异性能,提出了一种并行化加速策略,使其得以在 GPU 平台上高效地实现,为在 PC 平台上实时实现双目立体匹配提供了一条切实可行的途径。

1 双目立体代价聚合与 CUDA

1.1 线性立体代价聚合

双目立体匹配一般包含以下几个步骤:1)双目图像对的预处理;2)立体代价聚合;3)深度选择;

4)深度图的后处理。其中立体代价聚合是一个双目立体匹配算法的核心,其计算过程的时间耗费也最大。

线性立体代价聚合算法的结果是一个大小为 $W\times H\times D$(W,H 分别为图像的宽度和高度,D 为深度的最大范围)的代价矩阵 $\boldsymbol{C}$。代价矩阵 $\boldsymbol{C}$ 中每个元素表示的是左右两幅图像($\boldsymbol{I}_1$ 为左图像,$\boldsymbol{I}_2$ 为右图像)中对应像素点在指定深度 d 下的相关程度,即

$$\boldsymbol{C}(\boldsymbol{p},d) = \bar{\boldsymbol{a}}_p\boldsymbol{I}_{1,2}(\boldsymbol{p},\boldsymbol{q}) + \bar{\boldsymbol{b}}_q \tag{1}$$

式中,$\boldsymbol{I}_{1,2}(\boldsymbol{p},\boldsymbol{q}) = [\boldsymbol{I}_1(\boldsymbol{p}),\boldsymbol{I}_2(\boldsymbol{q})]^{\mathrm{T}}$ 是由左图像 $\boldsymbol{p}$ 点的值 $\boldsymbol{I}_1(\boldsymbol{p})$ 与右图像 $\boldsymbol{q}$ 点的值 $\boldsymbol{I}_2(\boldsymbol{q})$ 构成的一个 2×1 或 6×1 向量(当左右图像为灰度图时该向量的大小为 2×1,为 RGB 彩色图时则为 6×1)。$\boldsymbol{q}$ 为在深度 $\boldsymbol{d}$ 的情况下右图像中与左图像点 $\boldsymbol{p}$ 相对应的点,且 $\boldsymbol{q}=\boldsymbol{e}_{bm}(\boldsymbol{p},d)=[\boldsymbol{p}_x-d,\boldsymbol{p}_y]$。$\bar{\boldsymbol{a}}_p$、$\bar{\boldsymbol{b}}_q$ 分别为在以 $\boldsymbol{p}$ 点为中心的窗口 $\boldsymbol{W}_p$ 内系数 $\boldsymbol{a}_k$ 与 b_k 的均值。

若左图像中的 $\boldsymbol{p}$ 点与右图像中的 $\boldsymbol{q}$ 点对应于同一物理点,为了确保 $\boldsymbol{C}(\boldsymbol{p},d)$ 为 $\boldsymbol{C}(\boldsymbol{p},1,\cdots,D)$ 中的最小值,根据文献[16-17],系数 $\boldsymbol{a}_k$ 与 b_k 可定义为

$$\boldsymbol{a}_k = (COV_{p\in W_k}\boldsymbol{I}_{1,2}(\boldsymbol{p},\boldsymbol{q}) + \delta\boldsymbol{E})^{-1}\times\left(\frac{1}{N}\sum_{\boldsymbol{p}\in\boldsymbol{W}_k}\boldsymbol{I}_{1,2}(\boldsymbol{p},\boldsymbol{q})\boldsymbol{e}(\boldsymbol{p},\boldsymbol{q}) - \boldsymbol{U}_k\bar{\boldsymbol{e}}_k\right) \tag{2}$$

$$b_k = \bar{e}_k - \boldsymbol{a}_k^{\mathrm{T}}\boldsymbol{U}_k \tag{3}$$

式中,$COV_{\boldsymbol{p}\in\boldsymbol{W}_k}\boldsymbol{I}_{1,2}(\boldsymbol{p},\boldsymbol{q})$ 表示位于窗口 $\boldsymbol{W}_k$(窗口内元素的个数为 N)内所有向量 $\boldsymbol{I}_{1,2}(\boldsymbol{p},\boldsymbol{q})$ 的协方差矩阵,其大小为 2×2 或 6×6。$\boldsymbol{E}$ 为与协方差矩阵大小相同的单位矩阵。$\boldsymbol{e}$ 为匹配代价函数值,在此选用的代价函数为绝对差之和,即 $\boldsymbol{e}(\boldsymbol{p},\boldsymbol{q}) = \sum_{\{R,G,B\}}|\boldsymbol{I}_1(\boldsymbol{p})-\boldsymbol{I}_2(\boldsymbol{q})|$;而 $\bar{e}_k$ 为 $e(\boldsymbol{p},\boldsymbol{q})$ 在窗口 $\boldsymbol{W}_k$ 内的均值。U_k 是窗口 $\boldsymbol{W}_k$ 内所有向量 $\boldsymbol{I}_{1,2}(\boldsymbol{p},\boldsymbol{q})$ 的均值。δ 为一个经验参数,δ 越大图像的边缘则越平滑[14]。分析式(2)(3)可知,系数 $\boldsymbol{a}_k$ 为 2×1 或 6×1 的向量,而系数 b_k 则为一标量。

1.2　CUDA 通用并行计算架构

与传统 CPU 的 SISD(single instruction, multiple threads)架构不同,现代 GPU 是基于 SIMT(single instruction, multiple threads)并行计算架构而设计的计算机显示设备。得益于 SIMT 并行计算架构,GPU 有着优异的并行计算能力,因此,GPU 被逐渐应用到大型的通用计算领域,并由此衍生出了一个研究热点,即 GPU 通用计算(GPGPU)。借助于 GPGPU 的发展,近些年来 GPU 也开始广泛应用于图像处理及计算机视觉领域。CUDA(NVidia's compute unified device architecture)是由 NVIDIA 公司针对其 GPU 硬件芯片提出的一个 GPU 通用计算的平台和编程模型。

从硬件角度,CUDA 将 GPU 看成是一个由多个 SM(stream multiprocessor)构成的集合,而每个 SM 又是由多个 SP(stream processor)组成。SP 是 CUDA 模型中的最小硬件计算单元,类似于 CPU 的 ALU(arithmetic logic unit)。从软件角度看,CUDA 的最小调度单元为线程(thread),即 CUDA 可以将一个线程调度分配到某个 SP 上使之运行。与 CPU 的线程相比,CUDA 的线程是轻量化的线程,其调度开销几乎为 0。

利用 CUDA 设计的并行算法性能很大程度上依赖于对各种不同存储器的合理使用。CUDA 编程模型将 GPU 设备的存储器划分为全局存储器、共享存储器以及纹理存储器(texture memory)。其中全局存储器是容量最大且使用频率最高的一种存储器,但在 3 类存储器中其访问效率最低。与全局存储器不同,共享存储器位于 GPU 芯片之内,故其访问延迟远低于全局存储器而与寄存器的访问效率相当。纹理存储器可以看成是 GPU 芯片与全局存储器之间的一种特殊的快速只读缓存。关于 GPU 的 3 类存储器的详细说明及存储特点可参考文献[18]。

2　线性立体代价聚合的 GPU 并行化

通过 1.1 节以及式(1)—(3)可知,线性立体代价聚合算法是在某一指定大小的窗口内对该窗口中心像素点的匹配代价进行计算,其运算复杂度与窗口半径呈平方关系。文献[14]利用积分图[19]将代价聚合的算法复杂度优化为常数,即与窗口半径无关(故而文献[14]中图 2 所示算法运行时间与横轴平行)。这种优化策略在 CPU 平台上对聚合效率的提升效果十分明显,但积分图却不能发挥 GPU 并行计算系统的优势:首先,积分图的计算过程彼此依赖性太强,从而大大降低其并行度从而无法发挥并行计算的优势;此外,对积分图的访问频繁发生在效率较低的全局存储器中且是非对齐的,这将大大降低

中国图象图形学报
JOURNAL OF IMAGE AND GRAPHICS
Vol. 19, No. 10, Oct. 2014

算法的数据吞吐率。离开积分图的优化,GPU 聚合过程的算法复杂度将退化为与窗口半径呈平方关系。为此,针对特定的并行计算系统以及并行设计模型,有必要为整个代价聚合过程设计与之匹配的并行化计算策略。

2.1 立体代价聚合并行算法流程

根据式(1)—(3),线性立体代价聚合如算法 1。

算法 1 线性立体代价聚合

输入:左图像 $\boldsymbol{I}_1$,右图像 $\boldsymbol{I}_2$,深度范围 D,参数 δ。

输出:代价矩阵 $\boldsymbol{C}$。

$\bar{\boldsymbol{I}}_1 = Mean\_Mul(\boldsymbol{I}_1, 1)$, $\bar{\boldsymbol{I}}_2 = Mean\_Mul(\boldsymbol{I}_2, 1)$;

$\boldsymbol{e} = SumAbsDiff\_Mul(\boldsymbol{I}_1, \boldsymbol{I}_2, D)$;

$\bar{\boldsymbol{e}} = Mean\_Mul(\boldsymbol{e}, D)$;

$[\boldsymbol{a}, \boldsymbol{b}] = Coeffs\_Mul(\boldsymbol{I}_1, \boldsymbol{I}_2, \bar{\boldsymbol{I}}_1, \bar{\boldsymbol{I}}_2, \boldsymbol{e}, \bar{\boldsymbol{e}}, \delta)$;

$\bar{\boldsymbol{a}} = Mean\_Mul(\boldsymbol{a}, D)$, $\bar{\boldsymbol{b}} = Mean\_Mul(\boldsymbol{b}, D)$;

$\boldsymbol{C} = CostMatrix(\boldsymbol{I}_1, \boldsymbol{I}_2, \bar{\boldsymbol{a}}, \bar{\boldsymbol{b}})$;

return $\boldsymbol{C}$。

算法 1 中包含 4 个主要的步骤,即代价函数计算(*SumAbsDiff_Mul*),均值滤波(*Mean_Mul*),系数求解(*Coeffs_Mul*)及代价矩阵计算(*CostMatrix*)。

对应于算法 1,*SumAbsDiff_Mul* 负责同时计算所有深度(D)下左图像 $\boldsymbol{I}_1$ 与右图像 $\boldsymbol{I}_2$ 相对应点的代价函数值。该步骤的输出 $\boldsymbol{e}$ 是一个 $W \times H \times D$ 的 3 维矩阵,其中每一个元素 $\boldsymbol{e}(\boldsymbol{p}, \boldsymbol{q}) = \sum_{\{R,G,B\}} |\boldsymbol{I}_1(\boldsymbol{p}) - \boldsymbol{I}_2(\boldsymbol{q})|$。为了进一步提高效率,均值滤波 $Mean\_Mul(I, n)$ 经过 2.3 节的设计可以同时对 n 个大小相同且顺序存储在 $\boldsymbol{I}$ 里的图像或矩阵进行均值滤波。而 *Coeffs_Mul* 步骤则是在之前结果的基础上利用式(2)(3)计算出系数 $\boldsymbol{a}$ 和 $\boldsymbol{b}$。系数求解是本算法的核心亦是耗时最长的步骤。最后,*CostMatrix* 是在已知系数 $\bar{\boldsymbol{a}}$ 和 $\bar{\boldsymbol{b}}$ 的情况下,根据式(1)计算出最终的代价矩阵 $\boldsymbol{C}$。

2.2 并行化代价函数计算

代价函数的结果矩阵 $\boldsymbol{e}$ 中每个元素的计算只与左、右图像相应点的值有关。换言之,代价函数每个元素的计算过程是独立的,因此,可以将一个元素的计算过程映射到一个 CUDA 线程上,从而可保证各个 CUDA 线程之间是互不依赖的。结合 NVIDIA 在文献[18]中的建议,将整个计算过程划分为 $(W/32) \times (H/8)$ 个线程块,每个线程块中包含 32×8 共 256 个线程。基于这样的划分,块 $[b_x, b_y]$ 中线程 $[t_x, t_y]$ 计算点 $p = [32 \times b_x + t_x, 8 \times b_y + t_y]$ 的代价函数,且每个 CUDA 线程的工作流程可由算法 2 加以描述。

算法 2 代价函数的计算

输入:左图像 $\boldsymbol{I}_1$,右图像 $\boldsymbol{I}_2$。

输出:绝对差之和矩阵 $\boldsymbol{e}$。

$x = 32 \times b_x + t_x, y = 8 \times b_y + t_y$;

for each d in $[1, D]$ do

$$\boldsymbol{e}(\boldsymbol{p}, \boldsymbol{q}) = \sum_{\{R,G,B\}} |\boldsymbol{I}_1(x, y) - \boldsymbol{I}_2(x - d, y)|;$$

end for

return $\boldsymbol{e}$。

分析算法 2 可知,计算矩阵 $\boldsymbol{e}$ 需要从全局存储器中将右图像 $\boldsymbol{I}_2$ 的所有像素读入寄存器 D 次。为了提高效率,处于同一个线程块的线程在计算前将需要处理的右图像数据预先读入共享存储器,如此将能大大降低存储器访问时延。为了进一步提高代价函数的计算效率,将 PTX(parallel thread execution)[18] 指令引入到计算过程中以替代算法 2 中关于 $\boldsymbol{e}(\boldsymbol{p}, \boldsymbol{q})$ 的计算。

2.3 并行化均值滤波

通过算法 1 可知,整个线性立体代价聚合过程至少需要 $3 \times D + 2$ 次均值滤波的操作。因此,均值滤波器的效率对整个代价聚合算法的效率起着至关重要的作用。依据均值滤波的原理,结合 CUDA 模型,最为直观的并行化策略是利用一个 CUDA 线程处理均值滤波结果中的一个元素。这种并行策略亦能保证各个线程之间的计算过程是独立的,其效率也远高于传统的 CPU 串行算法。但进一步分析可知:如果窗口的大小为 N,则该并行策略计算每个输出需要访问位于该窗口内的所有 N 个元素。综合来看,输出整个滤波结果需要访问待滤波数据 N 次,如此频繁的全局存储器的访问会制约整个并行计算的效率。

根据可分离的卷积滤波技术[20],均值滤波可以表示为

$$\bar{\boldsymbol{I}} = \frac{1}{N}\boldsymbol{I} * \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{pmatrix}_{R \times R} \tag{4}$$

$$\bar{\boldsymbol{I}} = \frac{1}{N}\boldsymbol{I} * \left[\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} * (1 \quad \cdots \quad 1) \right] =$$

$$\left[\frac{1}{\sqrt{N}}\boldsymbol{I}*\begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix}\right]*\frac{1}{\sqrt{N}}(1\quad\cdots\quad 1)\qquad(5)$$

式中，R 为滤波窗口的直径，即有 $N=R\times R$。式(5)表明，均值滤波可以划分成 X 轴与 Y 轴两个方向的独立卷积运算。如此整个均值滤波将最多只需访问每个待滤波元素 $2R$ 次，相比于 $N=R^2$ 访问次数，对全局存储器的访问次数大大降低，算法复杂度也由 $O(R^2)$ 降低为 $O(R)$。其中 X 轴与 Y 轴方向每个 CUDA 线程的卷积过程如算法 3，算法 4。

算法 3　X 轴方向卷积计算

输入：源图像或矩阵 $\boldsymbol{I}$，宽度 W，窗口半径 r。

输出：中间结果 $\bar{\boldsymbol{I}}_t$。

$y=32\times b_x+t_x$;

$k=\sum_{i=1}^{r+1}I(i,y)$;

for each x in$[1,W]$ do

　$\bar{I}_t(x,y)=k/(2\times r+1)$;

　$k+=I(x+1+r,y),k-=I(x-r,y)$;

end for

return $\bar{\boldsymbol{I}}_t$。

算法 4　Y 轴方向卷积计算

输入：X 轴计算结果 $\bar{\boldsymbol{I}}_t$，高度 H，窗口半径 r。

输出：均值滤波结果 $\bar{\boldsymbol{I}}$。

$x=32\times b_x+t_x$;

$k=\sum_{i=1}^{r+1}\bar{I}_t(x,i)$;

for each y in$[1,H]$ do

　$\bar{\boldsymbol{I}}(x,y)=k/(2\times r+1)$;

　$k+=\bar{I}_t(x,y+1+r),k-=\bar{I}_t(x,y-r)$;

end for

return $\bar{\boldsymbol{I}}$。

在对单幅图像或单个矩阵进行均值滤波时，算法 3 和算法 4 的 CUDA 线程块由 32 个线程组成，每个线程处理一整行或一整列数据。为了进一步提高效率，使滤波算法可以同时处理多幅图像或矩阵，可将一维的线程块扩展成 32×8 的 2 维线程块，即一个线程块包含 256 个线程，处于相同列的线程处理同一幅图像或同一个矩阵，而不同列的线程处理不同的图像或矩阵。如此每个线程利用 $n/8$ 次循环就能处理连续存储的 n 幅大小相等的图像或矩阵。

此外，Y 轴方向的卷积运算对全局存储器的访问是严格对齐的，而 X 轴方向的卷积运算则是非对齐的。因此，为了进一步提高效率，算法 3 每个线程在运算前可将需要处理的数据预先读入到共享存储器中以消除非对齐访问对效率的负面影响。随后的实验也表明，以上各种并行优化策略是切实有效的。

2.4　并行化系数求解

为了阐述方便，将系数 $\boldsymbol{a}$ 的计算过程划分为两个部分，即：$\boldsymbol{a}1_k=COV_{p\in\boldsymbol{W}_k}\boldsymbol{I}_{1,2}(\boldsymbol{p},\boldsymbol{q})+\delta\boldsymbol{E}$，$\boldsymbol{a}2_k=(1/N)\sum_{p\in\boldsymbol{W}_k}\boldsymbol{I}_{1,2}(p,q)e(p,q)-\boldsymbol{U}_k\bar{\boldsymbol{e}}_k$，则系数 $\boldsymbol{a}_k=a1_k^{-1}a2_k$。与 2.3 节类似，依据式(2)进行直接计算，其性能瓶颈为对窗口内元素的频繁访问。为了降低对全局存储器的访问频率，对其计算过程进行改进。

设 $\boldsymbol{\sigma}_X(i)=\boldsymbol{X}(i)-\bar{\boldsymbol{X}},\boldsymbol{\xi}_{X,Y}(i,j)=\boldsymbol{\sigma}_X(i)\boldsymbol{\sigma}_Y(j)^{\mathrm{T}}$ 则有

$$\boldsymbol{a}1_k=COV_{p\in\boldsymbol{W}_k}\boldsymbol{I}_{1,2}(p,q)+\delta\boldsymbol{E}=\frac{1}{N}\begin{pmatrix}\sum_{p\in\boldsymbol{W}_k}\boldsymbol{\sigma}_{\boldsymbol{I}_1}(\boldsymbol{p})\boldsymbol{\sigma}_{\boldsymbol{I}_1}(\boldsymbol{p})^{\mathrm{T}} & \sum_{p\in\boldsymbol{W}_k}\boldsymbol{\sigma}_{\boldsymbol{I}_1}(\boldsymbol{p})\boldsymbol{\sigma}_{\boldsymbol{I}_2}(\boldsymbol{q})^{\mathrm{T}}\\ \sum_{p\in\boldsymbol{W}_k}\boldsymbol{\sigma}_{\boldsymbol{I}_2}(\boldsymbol{q})\boldsymbol{\sigma}_{\boldsymbol{I}_1}(\boldsymbol{p})^{\mathrm{T}} & \sum_{p\in\boldsymbol{W}_k}\boldsymbol{\sigma}_{\boldsymbol{I}_2}(\boldsymbol{q})\boldsymbol{\sigma}_{\boldsymbol{I}_2}(\boldsymbol{q})^{\mathrm{T}}\end{pmatrix}+\frac{1}{N}\begin{pmatrix}N\delta\boldsymbol{E}' & 0\\ 0 & N\delta\boldsymbol{E}'\end{pmatrix}=\frac{1}{N}\sum_{p\in\boldsymbol{W}_k}\begin{pmatrix}\boldsymbol{\xi}_{\boldsymbol{I}_1,\boldsymbol{I}_1}(\boldsymbol{p},\boldsymbol{p})+\delta\boldsymbol{E}' & \boldsymbol{\xi}_{\boldsymbol{I}_1,\boldsymbol{I}_2}(\boldsymbol{p},\boldsymbol{q})\\ \boldsymbol{\xi}_{\boldsymbol{I}_2,\boldsymbol{I}_1}(\boldsymbol{q},\boldsymbol{p}) & \boldsymbol{\xi}_{\boldsymbol{I}_2,\boldsymbol{I}_2}(\boldsymbol{q},\boldsymbol{q})+\delta\boldsymbol{E}'\end{pmatrix}\qquad(6)$$

式中，$\boldsymbol{E}'$ 为 1×1 或 3×3 的单位矩阵。$\boldsymbol{a}2_k$ 可表示为

$$\boldsymbol{a}2_k=\frac{1}{N}\sum_{p\in\boldsymbol{W}_k}\boldsymbol{I}_{1,2}(\boldsymbol{p},\boldsymbol{q})e(\boldsymbol{p},\boldsymbol{q})-U_k\bar{\boldsymbol{e}}_k=\frac{1}{N}\sum_{p\in\boldsymbol{W}_k}\begin{pmatrix}\boldsymbol{I}_1(\boldsymbol{p})e(\boldsymbol{p},\boldsymbol{q})-\bar{\boldsymbol{I}}_1(k)\bar{\boldsymbol{e}}_k\\ \boldsymbol{I}_2(\boldsymbol{p})e(\boldsymbol{p},\boldsymbol{q})-\bar{\boldsymbol{I}}_2(k)\bar{\boldsymbol{e}}_k\end{pmatrix}\qquad(7)$$

经过变换后，式(6)(7)清楚地表明 $\boldsymbol{a}1_k$ 的计算可以看做是对一个具有 4 个特殊通道的图像进行均值滤波，而 $\boldsymbol{a}2_k$ 亦可看做是对一个具有 2 个特殊通道的图像进行均值滤波。据此思路及 2.3 节的方法，$\boldsymbol{a}1_k$ 的并行计算过程可由算法 5 和算法 6 加以描述，如此系数求解的复杂度也可由与窗口半径呈平方关系降低为线性关系（*Inf* 为矩阵求逆运算）。为了简化计算过程，系数 $\boldsymbol{a}$、$\boldsymbol{b}$ 的完整计算过程已被融入到算法 5 和算法 6 中。此外 $\boldsymbol{a}2_k$ 的计算过程与

$\boldsymbol{a}1_k$ 类似,在此就不再另作说明。

算法 5　*Coeffs_Mul X* 轴方向计算

输入:左图像 $\boldsymbol{I}_1$,右图像 $\boldsymbol{I}_2$,宽度 W,窗口半径 r,
左图像均值 $\bar{\boldsymbol{I}}_1$,右图像均值 $\bar{\boldsymbol{I}}_2$,
深度 d,经验参数 δ。

输出:中间结果 ***Coef***。

$y=32\times b_x+t_x$;

$q_y=p_y=y$;

$$\boldsymbol{K}=\sum_{p_x=1}^{r+1}\begin{pmatrix}\boldsymbol{\xi}_{I_1,I_1}(\boldsymbol{p},\boldsymbol{p})+\delta\boldsymbol{E}' & \boldsymbol{\xi}_{I_1,I_2}(\boldsymbol{p},\boldsymbol{q})\\ \boldsymbol{\xi}_{I_2,I_1}(\boldsymbol{q},\boldsymbol{p}) & \boldsymbol{\xi}_{I_2,I_2}(\boldsymbol{q},\boldsymbol{q})+\delta\boldsymbol{E}'\end{pmatrix};$$

for each x in$[1,W]$ do

$\boldsymbol{Coef}(x,y,d)=\boldsymbol{K}/(2\times r+1)$

$p_x=x+1+r, q_x=p_x-d$;

$$\boldsymbol{K}+=\begin{pmatrix}\boldsymbol{\xi}_{I_1,I_1}(\boldsymbol{p},\boldsymbol{p})+\delta\boldsymbol{E}' & \boldsymbol{\xi}_{I_1,I_2}(\boldsymbol{p},\boldsymbol{q})\\ \boldsymbol{\xi}_{I_2,I_1}(\boldsymbol{q},\boldsymbol{p}) & \boldsymbol{\xi}_{I_2,I_2}(\boldsymbol{q},\boldsymbol{q})+\delta\boldsymbol{E}'\end{pmatrix};$$

$p_x=x-r, q_x=p_x-d$;

$$\boldsymbol{K}-=\begin{pmatrix}\boldsymbol{\xi}_{I_1,I_1}(\boldsymbol{p},\boldsymbol{p})+\delta\boldsymbol{E}' & \boldsymbol{\xi}_{I_1,I_2}(\boldsymbol{p},\boldsymbol{q})\\ \boldsymbol{\xi}_{I_2,I_1}(\boldsymbol{q},\boldsymbol{p}) & \boldsymbol{\xi}_{I_2,I_2}(\boldsymbol{q},\boldsymbol{q})+\delta\boldsymbol{E}'\end{pmatrix};$$

end for

return ***Coef***。

X 轴计算的中间结果 ***Coef*** 是一个大小为 $W\times H\times D$ 的 3 维矩阵,而 ***Coef*** 的每个元素则是一个 2×2 或 6×6 的对称矩阵,因此可以采用上三角矩阵或下三角矩阵存储 ***Coef***,如此可将存储空间和计算量同时降低一半。

算法 6　*Coeffs_Mul Y* 轴方向计算

输入:中间结果 ***Coef***,
高度 H,窗口半径 r, 深度 d。

输出:系数矩阵 $\boldsymbol{a},\boldsymbol{b}$。

$x=32\times b_x+t_x$;

$$\boldsymbol{K}=\sum_{i=1}^{r+1}\boldsymbol{Coef}(x,i,d);$$

for each y in$[1,H]$ do

$\boldsymbol{a}(x,y)=Inf(\boldsymbol{K}/(2\times r+1))\times\boldsymbol{a}2(x,y)$;

$$\boldsymbol{b}(x,y)=\bar{\boldsymbol{e}}(x,y)-\boldsymbol{a}(x,y)^{\mathrm{T}}\times\begin{pmatrix}\bar{\boldsymbol{I}}_1(x,y)\\ \bar{\boldsymbol{I}}_2(x-d,y)\end{pmatrix};$$

$\boldsymbol{K}+=\boldsymbol{Coef}(x,y+1+r,d)$;

$\boldsymbol{K}-=\boldsymbol{Coef}(x,y-r,d)$;

end for

return $\boldsymbol{a},\boldsymbol{b}$。

此外,类似于算法 4、算法 5 对全局存储空间的访问也是非对齐的,但限于共享存储器的大小,在算法 5 中利用共享存储器优化存储器访问效率的策略将不再可行。然而依旧可以采用纹理存储器一定程度地改善算法 5 的存储器访问效率,提高数据吞吐率。为了能同时计算所有深度的系数 $\boldsymbol{a}$ 和 $\boldsymbol{b}$,算法 5 和算法 6 的 CUDA 线程块组织方式从 1 维结构(32)扩展成 2 维结构(32×8),即处于相同列的 32 个线程处理一个深度,而不同列的线程处理不同深度的计算。如此,算法 5 和算法 6 的每个线程在经过 $D/32$ 次循环后就能计算出所有深度情况下的系数 $\boldsymbol{a}$ 和 $\boldsymbol{b}$。

2.5　并行化代价矩阵计算

在计算出系数 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的情况下,依据式(2)计算代价矩阵的过程则较为直观。代价矩阵的计算过程是将一个 CUDA 线程块设计成由 32×8 个线程构成的 2 维结构,每一个线程块负责代价矩阵 $\boldsymbol{C}$ 的一个 32×8 区域的计算任务,每个线程的工作流程如算法 7。

算法 7　代价矩阵的计算

输入:左图像 $\boldsymbol{I}_1$,右图像 $\boldsymbol{I}_2$,系数的均值 $\bar{a}$ 和 $\bar{b}$。

输出:代价矩阵 $\boldsymbol{C}$。

$x=32\times b_x+t_x, y=8\times b_y+t_y$;

for each d in$[1,D]$ do

$\boldsymbol{C}(x,y,d)=\bar{\boldsymbol{a}}(x,y)^{\mathrm{T}}\times[\boldsymbol{I}_1(x,y),\boldsymbol{I}_2(x,y)]+\bar{\boldsymbol{b}}(x,y)$;

end for

return $\boldsymbol{C}$。

3　实验结果与分析

为说明 GPU 代价聚合算法的有效性及其高效性,通过 3 次实验从不同角度加以阐述。3 次实验的平台均为 NVIDIA GTX780 显卡,它具有 2 304 个 CUDA 处理器核心,频率为 863 MHz,并且具有 3 G DDR5/384 位显存。实验样本来自于数据库[2]的 Tsukuba(384 × 288,最大深度为 16)和 Teddy(450 × 375,最大深度为 64)两对双目视觉图像对,这两张图像也是其他双目匹配算法广泛使用的样本。

首先,为了直观地说明本文所阐述的 GPU 代价聚合算法的效率,将 Tsukuba 与 Teddy 处理的各个分步骤的时间以及总的运行时间列举在表 1

中。为获得较好的效果，本次实验根据经验选取窗口半径 $r=5$，经验参数 $\delta=10^{-2.75}$。与 CPU 版本的线性代价聚合算法相比[14]，对 Teddy 图像对的处理 CPU 需要几十秒，而本文算法仅仅需要 78.56 ms，效率有了数百倍的提升。与其他 GPU 算法相比，FBS 聚合算法的 GPU 实现[21]与 Semi-Global 聚合算法[22]都需要大于 40 ms 来处理 Tsukuba，而本文的线性 GPU 代价聚合算法的运行时间不到 16 ms。由此可见，本文所描述的线性 GPU 代价聚合算法经过各种并行策略优化后，无论是与线性代价聚合算法的 CPU 实现还是与其他 GPU 代价聚合算法相比，运行时间都有大幅度地减少。

表 1　GPU 代价聚合时间

Table 1　Running time of GPU cost aggregation

图像名称	步骤	时间/ms	总时间/ms
Tsukuba	$\boldsymbol{I}_1$，$\boldsymbol{I}_2$ 均值滤波	1.79	15.86
	$\boldsymbol{a}$，$\boldsymbol{b}$ 的求解	9.34	
	$\boldsymbol{a}$，$\boldsymbol{b}$ 均值滤波	3.80	
	代价矩阵求解	0.93	
Teddy	$\boldsymbol{I}_1$、$\boldsymbol{I}_2$ 均值滤波	2.15	78.56
	$\boldsymbol{a}$、$\boldsymbol{b}$ 的求解	55.95	
	$\boldsymbol{a}$、$\boldsymbol{b}$ 均值滤波	15.87	
	代价矩阵求解	4.59	

接下来，为了说明本文阐述的 GPU 代价聚合算法的性能，分析了数据处理速率（结果元素的个数与运行时间的比值）与窗口半径和最大深度之间的关系（图 1 和图 2 分别表示了该算法的数据处理速率与窗口半径和最大深度范围之间的关系）。

如第 2 节所述，由于积分图优化技术不适合于 GPU 并行计算平台，故而线性代价聚合算法的复杂度将退化为与窗口半径呈平方关系而非文献[14]中图 2 所示的常数关系，但图 1 显示本文最终算法的数据处理速率随窗口半径的增加缓慢降低，且两者的关系近似于线性关系，这表明经过第 2 节的并行优化不仅能利用 GPU 并行计算平台显著提高双目视觉的代价聚合效率，同时如 2.3 节和 2.4 节所述将算法的复杂度降低到与窗口半径呈线性关

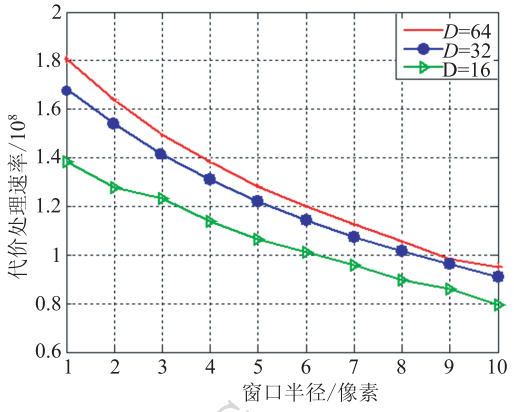


图 1　数据处理速率与窗口半径的关系（Teddy）

Fig. 1　Relationship between cost processing speeds and window radii

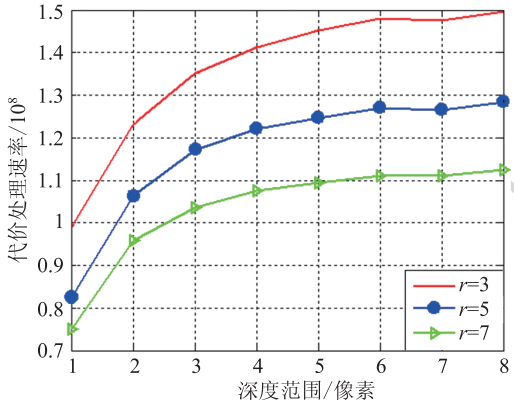


图 2　数据处理速率与深度范围的关系（Teddy）

Fig. 2　Relationship between cost processing speeds and disparity ranges

系。图 2 则表明算法的数据处理速率随着最大深度的增加而增加（实际深度为横坐标值乘以 8），且在不同半径的情况下其变化趋势是相同的。随着深度范围的增加，需要处理的元素个数也在增加，算法的效率也慢慢接近 GPU 平台的硬件处理能力极限，故而该曲线在末端会越来越平滑直至与 X 轴平行。

最后，以 Tsukuba 为例，取 $\delta=10^{-2.75}$，窗口半径分别为 3 和 5 时，计算得到的代价矩阵通过简单的 WTA（winner-takes-all）深度选择算法分别得到其深度图，如图 3 所示。比较图 3（b）（c），不难发现窗口半径取值越大其深度图的质量越好，但是物体边缘越模糊且算法的数据处理速率越低（如图 1 所示），故而在实际应用时应综合考虑合理选取窗口半径。

中国图象图形学报
JOURNAL OF IMAGE AND GRAPHICS
Vol. 19, No. 10, Oct. 2014

(a) Tsukuba 左图像　(b) 窗口半径 $r=3$　(c) 窗口半径 $r=5$

图 3　Tsukuba 深度图

Fig. 3　Disparity map of Tsukuba

4　结　论

本文提出了一种有效可行的 GPU 并行化策略以利用 GPU 实现线性双目代价聚合算法。实验结果表明,本文所阐述的 GPU 并行线性代价聚合算法将双目视觉的代价匹配过程的运行时间从数十秒乃至数百秒减少到几十毫秒,效率上有了数百倍的提升,这为基于个人通用 PC 平台的实时双目视觉应用提供了一个高效可靠的途径。

此外,对比文献[14],第 3 节实验所呈现的深度图较之高品质的深度图还有一定的差距,这是因为本文仅仅关注的是线性代价聚合部分,在计算深度图时未对图像和代价矩阵做预处理和后处理,即便如此,图 3 的效果还是比一些经过了预处理和后处理的经典匹配算法的质量要好[1]。这也充分说明了线性代价聚合算法的优越性。下一步工作将结合线性代价聚合算法的特点,为之设计适合于 GPU 平台的预处理和后处理方法,以期在 GPU 平台上建立完整的基于线性代价聚合的双目视觉匹配处理流程,使之能应用于实时双目视觉系统。

参考文献(References)

[1] Mattoccia S. Stereo vision: algorithms and applications [EB/OL]. [2013-11-20]. http://www.vision.deis.unibo.it/smatt/.

[2] Scharstein D, Szeliski R. Middlebury stereo vision [DB/OL]. [2013-11-20]. http://vision.middlebury.edu/stereo.

[3] Scharstein D, Szeliski R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms [J]. International Journal of Computer Vision, 2002, 47(1-3): 7-42. [DOI: 10.1023/A:1014573219977]

[4] Szeliski R. Computer Vision: Algorithms and Applications [M]. Berlin Heidelberg: Springer, 2011: 467-499. [DOI: 10.1007/978-1-84882-935-0]

[5] Hirschmuller H. Accurate and efficient stereo processing by semi-global matching and mutual information [C]//Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. San Diego, CA: IEEE, 2005(2): 807-814. [DOI: 10.1109/CVPR.2005.56]

[6] Hirschmuller H. Stereo processing by semi- global matching and mutual information [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2008, 30(2): 328-341. [DOI: 10.1109/TPAMI.2007.1166]

[7] Mattoccia S, Giardino S, Gambini A. Accurate and efficient cost aggregation strategy for stereo correspondence based on approximated joint bilateral filtering [C]// Proceedings of Computer Vision -ACCV 2009. Berlin Heidelberg: Springer, 2010: 371-380. [DOI: 10.1007/978-3-642-12304-7_35]

[8] Hosni A, Bleyer M, Gelautz M, et al. Local stereo matching using geodesic support weights [C]// Proceedings of IEEE International Conference on Image Processing. Cairo, Egypt: IEEE, 2009: 2093-2096. [DOI: 10.1109/ICIP.2009.5414478]

[9] Wang L, Liao M, Gong M, et al. High-quality real-time stereo using adaptive cost aggregation and dynamic programming [C]// Proceedings of 3D Data Processing, Visualization, and Transmission. Chapel Hill: IEEE, 2006: 798-805. [DOI: 10.1109/3DPVT.2006.75]

[10] Yoon K J, Kweon I S. Adaptive support-weight approach for correspondence search [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2006, 28(4): 650-656. [DOI: 10.1109/TPAMI.2006.70]

[11] Darabiha A, Rose J, Maclean J W. Video-rate stereo depth measurement on programmable hardware [C]// Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. Los Alamitos, California: IEEE, 2003(1): 203-210. [DOI: 10.1109/CVPR.2003.1211355]

[12] Park S, Jeong H. Real-time stereo vision FPGA chip with low error rate [C]//Proceedings of International Conference on Multi-

media and Ubiquitous Engineering. Seoul, Korea: IEEE, 2007: 751-756. [DOI: 10.1109/MUE.2007.180]

[13] Gehrig S K, Eberli F, Meyer T. A real-time low-power stereo vision engine using semi-global matching[J]. Computer Vision Systems, 2009: 134 -143. [DOI: 10.1007/978-3-642-04667-4_14]

[14] De-Maeztu L, Mattoccia S, Villanueva A, et al. Linear stereo matching[C]//Proceedings of IEEE International Conference on Computer Vision. Barcelona, Spain: IEEE, 2011: 1708-1715. [DOI: 10.1109/ICCV.2011.6126434]

[15] Porikli F. Constant time O (1) bilateral filtering[C]//Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. Anchorage, AK: IEEE, 2008: 1-8. [DOI: 10.1109/CVPR.2008.4587843]

[16] Ju M H, Kang H B. Constant time stereo matching[C]// Proceedings of Machine Vision and Image Processing Conference. Dublin, Ireland: IEEE, 2009: 13-17. [DOI: 10.1109/IMVIP.2009.10]

[17] He K, Sun J, Tang X. Guided image filtering[C]// Proceedings of Computer Vision-ECCV. Berlin Heidelberg: Springer, 2010: 1-14. [DOI: 10.1007/978-3-642-15549-9_1]

[18] NVIDIA. CUDA C programming guide [EB/ OL]. [2013-11-20]. http://www.NVIDIA.com.

[19] Crow F C. Summed-area tables for texture mapping[C]//Proceedings of ACM SIGGRAPH Computer Graphics. New York, NY: ACM, 1984, 18(3): 207-212. [DOI: 10.1145/800031.808600]

[20] Podlozhnyuk V. Image convolution with CUDA [J]. NVIDIA Corporation white paper, 2007, 2097(3): 9-14.

[21] Mattoccia S, Viti M, Ries F. Near real-time fast bilateral stereo on the GPU[C]//Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. Colorado Springs, CO: IEEE, 2011: 136-143. [DOI: 10.1109/CVPRW.2011.5981835]

[22] Chen B, Chen H. A realization of semi-global matching stereo algorithm on GPU for real-time application[C]//Seventh International Symposium on Multispectral Image Processing and Pattern Recognition. Guilin, China: SPIE, 2011, 8004: 1-9. [DOI: 10.1117/12.902015]