ISSN 1816-0301 (print)
УДК 004

Поступила в редакцию 02.02.2018
Received 02.02.2018

Б. А. Залесский, Ф. С. Троцкий

Объединенный институт проблем информатики Национальной академии наук Беларуси, Минск, Беларусь

ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ ДЕТЕКТОРА ЭКСТРЕМАЛЬНЫХ ОСОБЫХ ТОЧЕК ИЗОБРАЖЕНИЙ

Аннотация. Рассматривается параллельная версия детектора особых (ключевых, характерных) точек-экстремумов, применяемых для описания, анализа и сравнения изображений с помощью локальных дескрипторов, которые вычисляются в окрестностях найденных точек. Для задания ориентации дескрипторов предлагается использовать локальные гистограммы ориентированного градиента. В версии, предназначенной для выполнения на программно-аппаратной архитектуре CUDA, учтена специфика графических процессоров фирмы NVIDIA, что позволило ускорить вычисление экстремальных особых точек на несколько порядков. Вычисление неориентированных экстремальных особых точек изображения FullHD-размера на бюджетной видеокарте занимает 5–6 мс, ориентированных – 11–12 мс.

Ключевые слова: изображения, особые точки, детектор, алгоритм, параллельная версия, локальный дескриптор, CUDA

Для цитирования. Залесский, Б. А. Параллельная версия детектора экстремальных особых точек изображений / Б. А. Залесский, Ф. С. Троцкий // Информатика. – 2018. − Т. 15, № 2. – С. 55–63.

B. A. Zalesky, Ph. S. Trotski

The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, Belarus

PARALLEL VERSION OF DETECTOR OF EXTREMAL KEY POINTS ON IMAGES

Abstract. The article presents a parallel version of the detector of extremal key points, which are used to describe, analyze and compare digital images by local descriptors. Local descriptors are determined in neighborhoods of the extremal key points. The orientation of the descriptors are found with aid of Histograms of Oriented Gradient. The specificity of the parallel architecture of NVIDIA graphics cards has been taken into account in the developed version, oriented to the implementation on CUDA. It accelerated the calculation of the extremal key points by several orders. Computation of the not oriented extremal key points for images of the FullHD-size on the budget graphics card takes 5–6 ms. The oriented extremal key points are computed within 11–12 ms.

Keywords: images, key points, detector, algorithm, parallel version, local descriptor, CUDA

For citation. Zalesky B. A., Trotski Ph. S. Parallel version of detector of extremal key points on images. *Informatics*, 2018, vol. 15, no. 2, pp. 55–63 (in Russian).

Введение. Многие актуальные задачи анализа, сравнения и распознавания изображений решаются с помощью сопоставления их локальных характеристик, вычисленных в окрестностях особых (ключевых) точек. Следуя сложившейся терминологии, будем называть алгоритмы, находящие особые точки, *детекторами*, а алгоритмы, вычисляющие локальные характеристики в каждой найденной особой точке, – *дескрипторами*. Характеристики особых точек сравниваются с помощью *алгоритмов поиска соответствий* (англоязычный термин matchers). При использовании особых точек для решения задач сравнения или распознавания изображений сначала применяется один из детекторов, затем дескриптор и, наконец, алгоритм поиска соответствий.

Одни из первых детекторов, успешно примененных для решения задач компьютерного зрения, были предложены в 1999 г. в алгоритмах SIFT и HoG [1, 2]. В первом алгоритме используется детектор ключевых точек, основанный на анализе гессиана разности сглаженных

в разной степени копий исходного изображения, а во втором ключевыми считаются все точки регулярной решетки. Спустя год был опубликован алгоритм SURF, детектор которого находит ключевые точки с помощью приближенного гессиана, вычисляемого с помощью интегрального изображения [3]. Использование интегрального изображения позволило значительно ускорить процесс вычисления особых точек по сравнению с SIFT и HoG.

Затем появились FAST [4], STAR [5], BRISK [6], KAZE [7], AKAZE [8] и другие алгоритмы, в состав которых входят различные версии детекторов особых точек. Сравнительный анализ детекторов выполнен в работе [9]. Здесь следует отметить появившийся несколько лет назад дескриптор LATCH [10], который строит информативные локальные характеристики, обеспечивающие надежное сравнение изображений, с помощью ранее предложенных детекторов.

Среди недостатков известных детекторов следует отметить их чувствительность к изменениям яркости и контраста изображений, а также нередкие случаи неравномерного распределения особых точек на изображениях. (Области изображения без особых точек не могут быть обнаружены или распознаны с помощью алгоритмов поиска соответствий.)

Ранее одним из авторов [11] были предложены детекторы особых точек, основанные на поиске локальных экстремумов функций – характеристик непреобразованных изображений. Предложенные детекторы менее чувствительны к изменениям яркости изображений, найденные ими особые точки распределены на изображении более равномерно.

В настоящей статье рассматривается один из предложенных в [11] экстремальных детекторов, основанный на вычислении локальных максимумов оконных дисперсий изображения, который для краткости будем обозначать σ^2-детектор. Формальное описание σ^2-детектора приведено в разд. 1. Данный детектор находит на изображении примерно столько же особых точек, сколько находят другие экстремальные моментные детекторы [11], однако он проще реализуется программно, а его вычисление занимает меньше времени.

Указанный σ^2-детектор может быть вычислен на CPU (central processing unit) персонального компьютера с помощью интегрального изображения [3], однако процессорная версия недостаточно быстра для работы с видеопоследовательностями размером FullHD и HD в режиме реального времени.

В настоящей работе предлагается параллельная версия алгоритма вычисления σ^2-детектора, предназначенная для выполнения на программно-аппаратной архитектуре CUDA. В ней учтена специфика графических процессоров GPU (graphics processing unit) фирмы NVIDIA, что позволило ускорить вычисление детектора на несколько порядков. Описывается также кратко CUDA-версия вычисления ориентации особых точек относительно изображения с помощью локальных гистограмм ориентированного градиента. Приводятся результаты сравнения быстродействия предложенной параллельной версии алгоритма вычисления σ^2-детектора с быстродействием других детекторов, в том числе реализованных на CUDA.

1. Определение σ^2-детектора и его вычисление на CPU. Для формального определения σ^2-детектора рассмотрим полутоновое изображение $\mathbf{I}=\{I(\mathbf{p})\}$, $\mathbf{p}\in S$, с множеством пикселов $S=\{(x,y)\}$, $x=0,\ ...,\ m-1$, $y=0,\ ...,\ n-1$, квадратные окрестности $O_k(\mathbf{p})$ с центром в пикселах $\mathbf{p}$ размером $k\times k$, а также локальные (оконные) дисперсии

$$\sigma^2=\sigma^2(\mathbf{p})=\beta_2(\mathbf{p})-m^2(\mathbf{p}),\ \ m(\mathbf{p})=k^{-2}\sum\nolimits_{\mathbf{g}\in O_k(\mathbf{p})}I(\mathbf{g})\ \text{ и }\ \beta_2(\mathbf{p})=k^{-2}\sum\nolimits_{\mathbf{g}\in O_k(\mathbf{p})}I^2(\mathbf{g}). \quad (1)$$

Экстремальный σ^2-детектор выделяет пиксел $\mathbf{p}$ изображения как особую точку, если существует окрестность $O_\ell(\mathbf{q})$ размером $\ell\times\ell$ (значения k и ℓ выбираются соответствующими размерам локальных признаков на изображении, в большинстве экспериментов выбирались $k=5, \ell=4$) с центром в $\mathbf{q}$, содержащая $\mathbf{p}$, для которой значение $\sigma^2(\mathbf{p})$ является максимумом значений σ^2 в этой окрестности, причем выполняются следующие условия:

Условие 1. $\sigma^2(\mathbf{p}) > \sigma^2(\mathbf{v})$ хотя бы для одного пиксела $\mathbf{v} \in O_\ell(\mathbf{q})$.

Условие 2. Решение $\mathbf{p}$ находится от центра окрестности $\mathbf{q}$ на расстоянии, не превосходящем наперед заданное число $\tau\left(0 < \tau < \frac{\ell}{2}\right)$.

Замечание. Условие 1 равносильно тому, что $\sigma^2(\mathbf{p})$ не является константой в окрестности $O_\ell(\mathbf{q})$, а условие 2 предотвращает попадание экстремальной особой точки на границу окрестности $O_\ell(\mathbf{q})$. При нарушении хотя бы одного из условий найденные точки могут оказаться неинформативными. Нетрудно показать, что отсутствие условия 1 приводит к тому, что все внутренние точки изображения постоянной яркости стали бы особыми, хотя на самом деле в этом случае они не обладают индивидуальными локальными признаками. При невыполнении условия 2 все внутренние точки изображения, яркости которого лежат на некоторой наклонной плоскости, являются особыми.

Вместо условия 1 можно использовать другие условия, предотвращающие появление неинформативных особых точек, например условие, чтобы $\sigma^2(\mathbf{p})$ был строгим и поэтому единственным максимумом в $O_\ell(\mathbf{q})$, и т. д.

Формально экстремальные особые точки образуют множество

$$E = E_{k,\ell,\tau}(\mathbf{I}) = \tag{2}$$
$$= \left\{ \mathbf{p} \in S \mid \exists \mathbf{q} \in S : \mathbf{p} \in O_\ell(\mathbf{q}), \|\mathbf{p}-\mathbf{q}\| \le \tau, \forall \mathbf{u} \in O_\ell(\mathbf{q}), \sigma^2(\mathbf{p}) \ge \sigma^2(\mathbf{u}), \exists \mathbf{v} \in O_\ell(\mathbf{q}) : \sigma^2(\mathbf{p}) > \sigma^2(\mathbf{v}) \right\}.$$

В некоторых случаях при практическом применении предложенного σ^2-детектора множество особых точек E приходится прореживать – оставлять из нескольких расположенных близко равноценных особых точек одну, например, случайным образом или по какому-либо дополнительному критерию. Процедура прореживания особых точек не занимает много времени и используется при работе с большинством известных детекторов.

Предложенный σ^2-детектор может быть вычислен на CPU с помощью двух интегральных изображений, одно из которых построено по яркостям, а второе – по квадратам яркостей исходного изображения.

Для вычисления оконной дисперсии на CPU можно также использовать метод «бегущей строки» для быстрого вычисления оконных сумм, но даже в этом случае время вычисления процессорной версии детектора будет слишком большим для работы с видеопоследовательностями.

В большинстве случаев локальные характеристики особых точек, вычисляемые дескрипторами, неинвариантны к повороту изображения, поэтому для каждой найденной точки нужно задавать каким-либо образом ее ориентацию, которая используется при построении характеристик. Несмотря на то что, по сути, ориентация особой точки является одной из ее характеристик, многие программные реализации детекторов определяют не только координаты таких точек, но и направления, относительно которых потом дескрипторы вычисляют характеристики. Ниже кратко описывается быстрая реализация на CUDA алгоритма процедур нахождения ориентации особых точек на основе поиска максимума локальных гистограмм ориентированного градиента.

2. Вычисление σ^2-детектора на GPU. При описании параллельной версии алгоритма вычисления σ^2-детектора на CUDA будет удобнее индексировать все прямоугольные окна на изображении не центральным пикселом, а координатами их левого верхнего угла. Для этого будем использовать вместо обозначения $O_k(\mathbf{p})$ обозначение $O_{\mathbf{p}}$ или $O_{(x,y)}$.

Сначала изложим кратко суть алгоритма, после чего приведем пошаговое описание его реализации.

Стандартные операции выделения памяти процессора, копирования входных данных в память GPU и результатов из памяти GPU, очистки памяти процессора и GPU не будут включены в состав алгоритма, хотя в разд. 4 приведены отдельно их длительности. Следует пом-

нить, что указанные операции являются относительно медленными – они занимают в общей сложности несколько миллисекунд, что сравнимо с выполнением нескольких шагов алгоритма, поэтому при его многократном применении, например, для работы с видеопоследовательностями желательно избегать повторяющихся выделений и очисток памяти и использовать однажды определенные массивы.

При организации параллельных вычислений на GPU одна из основных задач – минимизация числа конфликтов одновременного доступа к видеопамяти.

Идея предложенного алгоритма состоит в представлении сумм значений яркостей (квадратов яркостей) изображения в окне $O_{\mathbf{p}}$ детектора в виде сумм значений яркостей (квадратов яркостей) его столбцов, а затем сложения полученных сумм. Такой подход имеет смысл в силу того, что сумма значений одного столбца может использоваться при подсчете сумм яркостей (и квадратов яркостей) нескольких окон $O_{\mathbf{p}}$. Для уменьшения числа конфликтов одновременного доступа к видеопамяти изображение разбивается на горизонтальные полосы, лежащие друг от друга на расстоянии k, равном высоте окна, и сначала считаются суммы значений яркостей $k\times 1$-столбцов изображения, первые элементы которых начинаются на выделенных горизонтальных полосах (рис. 1).

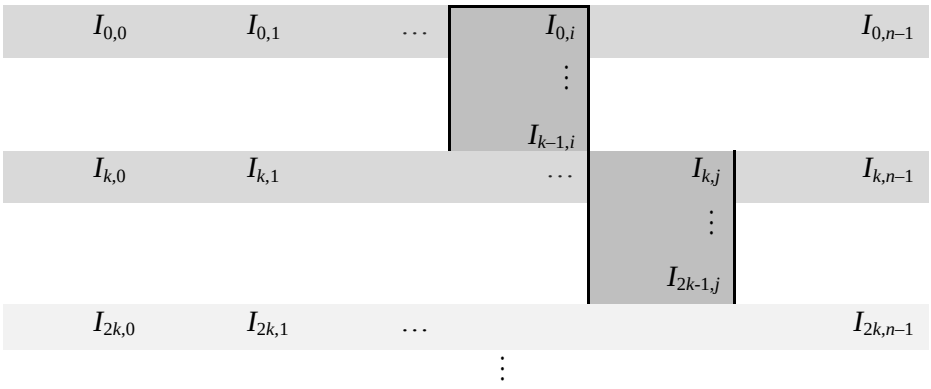

Рис. 1. Схема вычисления сумм значений яркостей в столбцах (изображены два столбца с начальными элементами на направляющих, индексы i, j – номера столбцов, k – номер строки изображения, n – длина изображения)

Затем методом бегущей строки вычисляются суммы всех остальных столбцов высотой k, не лежащих на выделенных полосах. Если обозначить через $c_{(x,y)} = \sum_{j=0}^{k-1} I_{(x,y+j)}$ (через $C_{(x,y)} = \sum_{j=0}^{k-1} I^2_{(x,y+j)}$) сумму (сумму квадратов) значений яркостей $k\times 1$-столбца изображения с верхним пикселом $\mathbf{p} = (x, y)$, то получим выражения

$$c_{(x,y+1)} = c_{(x,y)} + I_{(x,y+k)} - I_{(x,y)},$$

$$C_{(x,y+1)} = C_{(x,y)} + I^2_{(x,y+k)} - I^2_{(x,y)}. \qquad (3)$$

Значения $c_{(x,y)}$ и $C_{(x,y)}$ записываются в массивы, размер которых равен размеру исходного изображения. Изображения c и C транспонируются, и для них повторяются предыдущие вычисления, в которых $I_{(x,y)}$ заменено на $c_{(x,y)}$, а $I^2_{(x,y)}$ – на $C_{(x,y)}$.

Далее вычисляется транспонированная матрица Σ^T с элементами

$$\tilde{\sigma}^2_{(x,y)} = \sum_{u=0}^{k-1} C_{(x,u)} - \left(\sum_{u=0}^{k-1} c_{(x,u)} \right)^2 \Big/ k^2, \tag{4}$$

которая отличается от матрицы выборочных оконных дисперсий на множитель k^2 (очевидно, что координаты (x, y) максимумов двух матриц совпадают).

Для вычисления множества экстремальных ключевых точек в общем случае необходимо найти максимумы дисперсий внутри всех $\ell \times \ell$-квадратов $O_{(x,y)}$, а затем проверить условие близости координат найденных локальных максимумов к центрам окрестностей, присутствующее в формуле (2).

Оптимизация вычисления на GPU локальных экстремумов во всех пикселах достаточна сложна, а сами вычисления весьма ресурсоемки, поэтому для ускорения алгоритма и упрощения его реализации предлагается разбить пикселы изображения на непересекающиеся квадраты. Внутри этих квадратов требуется найти пикселы локальных максимумов и выбрать в качестве особых точек те, которые удалены от центров квадратов на расстояние, не превосходящее заданное. После этого необходимо сместить все квадраты, разбивающие S, на половину их стороны по вертикали и горизонтали и найти остальные особые точки внутри них.

Запишем алгоритм вычислений на GPU:

Шаг 1. Вычислить и сохранить в 1D-массиве суммы $c_{(x,y)}$ значений яркостей и суммы $C_{(x,y)}$ квадратов яркостей столбцов, первые элементы которых лежат на горизонтальных направляющих с y-координатами вида $y = \mu k$, где $\mu = 0, \ldots, \lceil m/k \rceil - 1$.

Шаг 2. Вычислить суммы всех остальных столбцов $c_{(x,y)}$ и $C_{(x,y)}$ методом бегущей строки по формуле (3) и сохранить результат в 1D-массивах, представляющих собой построчную запись матриц **c** и **C**.

Шаг 3. Транспонировать матрицы **c** и **C**, используя их 1D-представление. (Смысл шагов 3 и 4 объясняется в разд. 3.)

Шаг 4. Вычислить и сохранить в 1D-массиве значения $\tilde{\sigma}^2_{(x,y)}$ на горизонтальных направляющих с y-координатами вида $y = \mu k$, где $\mu = 0, \ldots, \lceil n/k \rceil - 1$, используя 1D-представления матриц $\mathbf{c}^T$ и $\mathbf{C}^T$.

Шаг 5. Вычислить методом бегущей строки и сохранить в 1D-массиве значения $\tilde{\sigma}^2_{(x,y)}$.

Шаг 6. Найти координаты $(x_{\max}, y_{\max})$ локальных максимумов $\tilde{\sigma}^2_{(x,y)}$ в окрестностях $O_{\ell,(x,y)}$ методом редукции.

Шаг 7. Сохранить в качестве особых точек те $(x_{\max}, y_{\max})$, для которых $\left\| (x_{\max}, y_{\max}) - (x + \ell/2, y + \ell/2) \right\| < \tau$, где τ – наперед заданный порог, удовлетворяющий неравенству $0 < \tau < \frac{\ell}{2}$.

Шаг 8. Вычислить градиент **G(I)** изображения **I**, используя один из известных дискретных дифференциальных операторов, например Собеля, Превитт, Робертса или Щара.

Шаг 9. Построить гистограмму оконного ориентированного градиента каждой особой точки и задать в качестве ее ориентации номер координаты максимального значения гистограммы.

3. Особенности реализации алгоритма на CUDA. Как известно, архитектура параллельных вычислений CUDA, разработанная фирмой NVIDIA, предназначена для проведения массивных параллельных вычислений на графических процессорах. Вычисления на графическом процессоре осуществляются путем запуска ядра GPU, при вызове которого указывается

число требуемых блоков, а также число нитей, используемых в каждом из задействованных блоков. Каждый задействованный блок выполняется на одном потоковом мультипроцессоре (SM). Количество одновременно выполняющихся операций (а следовательно, и время вычислений) зависит от числа мультипроцессоров и их характеристик. На бюджетных видеокартах имеется от 5 до 20 SM. Например, на GeForce GTX 1050 и 1060 имеется 10 SM, а на GTX1080 – 20.

Для эффективного параллельного выполнения операций нитями каждого блока требуется избегать конфликтов одновременного доступа нитей блока к ячейкам памяти GPU с учетом специфики обращения к ней варпами. Вышесказанное объясняет использование метода бегущей строки для вычисления детектора, а также способа вычисления среднего, дисперсии и ориентированного градиента по столбцам на шагах 1–4 алгоритма. Проведенные эксперименты показали, что вычисление оконной суммы значений яркости изображений размера FullHD на видеокарте GeForce GTX 1050 методом бегущей строки при выполнении суммирования по столбцам занимает 0,22 мс, а при суммировании по строкам – 2,21 мс. Операция транспонирования матрицы размером FullHD на той же видеокарте заняла 0,57 мс.

Таким образом, предлагаемый способ вычисления оконных сумм изображения суммированием значений $c_{(x,y)}$, $C_{(x,y)}$ методом бегущей строки по столбцам, затем транспонированием матрицы полученных сумм и, наконец, вычислением результата повторным суммированием транспонированной матрицы по столбцам потребовал в два раза меньше времени, чем суммирование промежуточных значений, полученных на шаге 2 алгоритма, по строке (без транспонирования).

Вычисление максимальных значений $\tilde{\sigma}^2_{(x,y)}$ в окрестностях $O_{\ell,(x,y)}$ выполнялось методом редукции [12, с. 75].

4. Результаты экспериментов. Для оценки быстродействия предложенной параллельной версии алгоритма вычисления экстремального σ^2-детектора были проведены эксперименты с полутоновыми изображениями разных размеров на персональном компьютере, оборудованном видеокартой NVIDIA GeForce GTX 1050, имеющей 2 Гбайта видеопамяти, и лэптопе с видеокартой NVIDIA GeForce GTX 1060 с 6 Гбайтами видеопамяти.

В таблице приведены средние длительности 50 повторных вычислений экстремальных особых точек с использованием для хранения исходных изображений неблокированной памяти, выделенной функцией malloc, и блокированной, выделенной функцией cudaHostAlloc (cHA).

Результаты экспериментов

Размер полутонового изображения	Количество найденных точек	Время вычисления, мс							
		NVIDIA GeForce GTX 1050				NVIDIA GeForce GTX 1060			
		с ориентацией		без ориентации		с ориентацией		без ориентации	
		malloc	cHA	malloc	cHA	malloc	cHA	malloc	cHA
1920×1080	10979	17,90	17,22	7,65	7,03	11,41	10,84	5,97	5,31
1600×850	7362	11,88	11,33	5,12	4,61	8,01	7,67	4,23	3,93
1280×720	4787	8,09	7,69	3,50	3,12	5,37	5,03	2,84	2,48
720×480	1613	3,55	3,59	1,53	1,51	2,56	2,34	1,37	1,19

Время выделения памяти на видеокарте не включено в результаты вычислений, так как при повторном использовании алгоритма целесообразно использовать однажды выделенную память. Время копирования данных из памяти компьютера в видеопамять GPU NVIDIA GeForce GTX 1060 изображения размером 1920×1080 составляет 0,67 мс для блокированной памяти и 1,22 мс – для неблокированной. Время копирования результатов вычислений – найденных характеристик – составляет примерно 0,15 мс.

Ниже приведены результаты сравнения предложенного детектора экстремальных особых точек и известного детектора SURF (реализованного на CUDA в OpenCV 3.3), дающего наиболее информативные особые точки на рассмотренных изображениях в сравнении с алгоритмами ORB, SIFT и AKAZE. Оба алгоритма применялись для установления соответствий между особыми точками пар изображений, найденными предложенным экстремальным детектором

и CUDA-версией детектора SURF. Для оценки быстродействия использовались только те пары изображений, которые были корректно совмещены друг с другом на основе особых точек, найденных обоими алгоритмами.

Нахождение экстремальных точек FullHD-изображения размером 1920×1080 на CPU Intel Core i7-6700 с использованием метода бегущей строки для быстрого вычисления оконных сумм заняло более 400 мс. Для изображений размером 720×480 это время равнялось 54 мс.

Приведем результаты вычислительных экспериментов для GPU NVIDIA GeForce GTX 1050. Вычисление особых точек на полутоновых изображениях природного ландшафта (рис. 2) размером FullHD с помощью программной реализации экстремального детектора на CUDA заняло около 17 мс (для каждого изображения), а с помощью CUDA-версии детектора SURF, реализованной в OpenCV 3.3, – примерно 54 мс. Для изображений лесного ландшафта размером 720×480 время вычисления особых точек с помощью реализации предложенного алгоритма и CUDA-версии детектора SURF заняло 4,5 и 13 мс соответственно.

Вычисление особых точек изображения размером 1920×1080 на видеокарте NVIDIA GeForce GTX 1060 предложенным детектором и детектором CUDA-версии SURF занимает соответственно 12,3 и 33,8 мс.

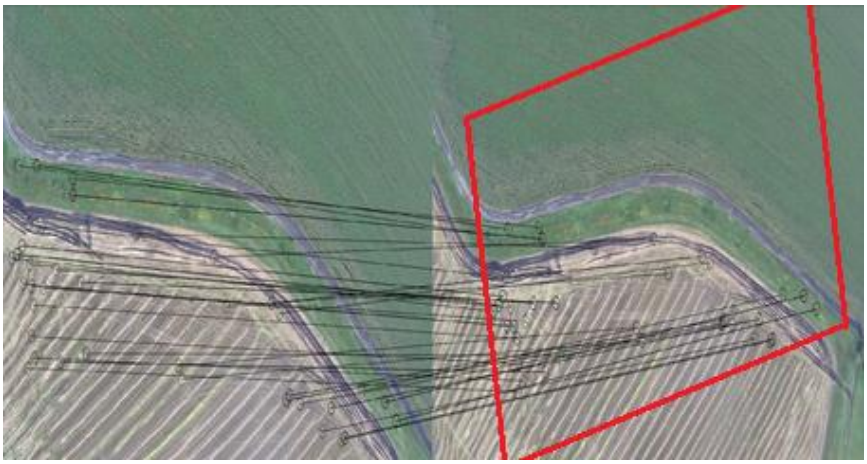

Рис. 2. Соответствие изображений, установленное с помощью экстремальных особых точек, найденных σ^2-детектором

На рис. 3 приведены соответствия двух изображений, установленные предложенным детектором. Примечательно, что известные алгоритмы SIFT, SURF, ORB и AKAZE не нашли на меньшем изображении ни одной особой точки.

а) *б)*

Рис. 3. Экстремальные особые точки, вычисленные σ^2-детектором, и найденные соответствия; в качестве шаблона использовалось меньшее изображение (*а*) и большее изображение (*б*)

Заключение. Особенностью предложенного детектора является возможность его применения для слабоконтрастных изображений, когда известные детекторы, основанные на вычислении градиента, не находят достаточного количества особых точек. Особые точки, вычисленные экстремальным детектором, в большинстве случаев распределяются по изображению более равномерно (по сравнению с точками, найденными другими детекторами).

Предложенная параллельная версия экстремального детектора особых точек позволяет проводить вычисления в режиме реального времени на бюджетных видеокартах, таких, например, как NVIDIA GeForce GTX 1050. Во многих случаях данная версия оказалась не только устойчивее, но и в несколько раз быстрее CUDA-версий известных детекторов. Например, вычисление особых точек вместе с их ориентацией на изображении размером 720×480 с помощью экстремального детектора занимает всего 2,3÷2,6 мс, что дает возможность выполнить другие функции обработки изображений, не выходя из режима реального времени.

Вычисления в предложенной параллельной реализации организованы так, чтобы уменьшить, насколько возможно, число конфликтов одновременного доступа к ячейкам видеопамяти.

Список использованных источников

1. Lowe, D. Object recognition from local scale invariant features / D. Lowe // Proc. of the 7th IEEE Intern. Conf. on Computer Vision (ICCV). – Corfu, 1999. – P. 1150–1157.

2. Dalah, N. Histograms of oriented gradients for human detection / N. Dalah, B. Triggs // Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR 2005). – San Diego, 2005. – Vol. 1. – P. 886–893.

3. Bay, H. SURF: speeded up robust features / H. Bay, T. Tuytelaars, L. Van Gool // Proc. of the 9th European Conf. on Computer Vision (ECCV). – Austria, Graz, 2006. – Vol. 3951, pt. 1. – P. 404–417.

4. Rosten, E. Faster and better: a machine learning approach to corner detection / E. Rosten, T. Drummond // IEEE Transactions on Pattern Analysis and Machine Intelligence. – 2010. – Vol. 32, no. 1. – P. 105–119.

5. Agrawal, M. CenSurE: center surround extremas for realtime feature detection and matching / M. Agrawal, K. Konolige, M. R. Blas // Lecture Notes in Computer Science. – 2008. – Vol. 5305. – P. 102–115.

6. Leutenegger, S. BRISK: binary robust invariant scalable keypoints / S. Leutenegger, M. Chli, R. Y. Siegwart // Proc. of the 13th IEEE Intern. Conf. on Computer Vision (ICCV). – Barcelona, 2011. – P. 2548–2555.

7. Alcantarilla, P. KAZE features / P. Alcantarilla, A. Bartoli, J. Davison // Proc. of the 12th European Conf. on Computer Vision (ECCV). – Firenze, 2012. – P. 214–227.

8. Alcantarilla, P. Fast explicit diffusion for accelerated features in nonlinear scale spaces / P. Alcantarilla, J. Nuevo, A. Bartoli // Proc. of the 24th British Machine Vision Conf. (BMVC). – Bristol, 2013. – P. 11–21.

9. Comparative assessment of point feature detectors and descriptors in the context of robot navigation / A. Schmidt [et al.] // J. of Automation, Mobile Robotics and Intelligent Systems. – 2013. – Vol. 7, no. 1. – P. 11–20.

10. Levi, G. Learned arrangements of three patch codes / G. Levi, T. Hassner // IEEE Winter Conf. on Applications of Computer Vision (WACV). – NY, USA, 2016. – P. 33–42.

11. Залесский, Б. А. Детекторы экстремальных особых точек на изображениях / Б.А. Залесский // Доклады Национальной академии наук Беларуси. – 2017. – № 5(61). – С. 37–41.

12. Сандерс, Д. Технология CUDA в примерах: введение в программирование графических процессоров / Д. Сандерс, Э. Кэндрот. – М. : ДМК Пресс, 2013. – 234 с.

References

1. Lowe D. Object recognition from local scale invariant features. *Proceedings of the 7th IEEE International Conference on Computer Vision (ICCV).* Corfu, 1999, pp. 1150–1157.

2. Dalah N., Triggs B. Histograms of oriented gradients for human detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005).* San Diego, 2005, vol. 1, pp. 886–893.

3. Bay H., Tuytelaars T., Van Gool L. SURF: speeded up robust features. *Proceedings of the 9th European Conference on Computer Vision (ECCV).* Austria, Graz, 2006, vol. 3951, pt. 1, pp. 404–417.

4. Rosten E., Drummond T. Faster and better: a machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010, vol. 32, no. 1, pp. 105–119.

5. Agrawal M., Konolige K., Blas M. R. CenSurE: center surround extremas for realtime feature detection and matching. *Lecture Notes in Computer Science*, 2008, vol. 5305, pp. 102–115.

6. Leutenegger S., Chli M., Siegwart R. Y. BRISK: binary robust invariant scalable keypoints. *Proceedings of the 13th IEEE International Conference on Computer Vision (ICCV).* Barcelona, 2011, pp. 2548–2555.

7. Alcantarilla P., Bartoli A., Davison J. KAZE features. *Proceedings of the 12th European Conference on Computer Vision (ECCV).* Firenze, 2012, pp. 214–227.

8. Alcantarilla P., Nuevo J., Bartoli A. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *Proceedings of the 24th British Machine Vision Conference (BMVC).* Bristol, 2013, p. 11–21. doi: 10.5244/C.27.13

9. Schmidt A., Kraft M., Fularz M., Domagała Z. Comparative assessment of point feature detectors and descriptors in the context of robot navigation. *Journal of Automation, Mobile Robotics and Intelligent Systems*, 2013, vol. 7, no. 1, pp. 11–20.

10. Levi G., Hassner T. Learned arrangements of three patch codes. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, NY, USA, March, 2016, p. 33–42. doi: 10.1109/WACV.2016.7477723

11. Zalesskiy B. A. Detektoryi ekstremalnyih osobyih tochek na izobrazheniyah [Detectors of exteremal key points on images]. Dokladyi Natsionalnoy akademii nauk Belarusi [*Reports of the National Academy of Sciences of Belarus*], 2017, no. 5(61), pp. 37–41 (in Russian).

12. Sanders D., Kendrot E. Tehnologiya CUDA v primerah: vvedenie v programmirovanie graficheskih protsessorov. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Moscow, DMK Press Publ., 2013, 234 p. (in Russian).

Информация об авторах

Залесский Борис Андреевич – доктор физико-математических наук, заведующий лабораторией обработки и распознавания изображений, Объединенный институт проблем информатики Национальной академии наук Беларуси (ул. Сурганова, 6, 220012, Минск, Республика Беларусь). E-mail: zalesky@newman.bas-net.by

Троцкий Филипп Сергеевич – младший научный сотрудник, Объединенный институт проблем информатики Национальной академии наук Беларуси (ул. Сурганова, 6, 220012, Минск, Республика Беларусь). E-mail: trotskiphilipp@gmail.com

Information about the authors

Boris A. Zalesky – Dr. Sc. (Physics and Mathematics), Head of Laboratory of Image Processing and Recognition, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus (6, Surganov Str., 220012, Minsk, Republic of Belarus). E-mail: zalesky@newman.bas-net.by

Philip S. Trotski – Junior researcher, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus (6, Surganov Str., 220012, Minsk, Republic of Belarus). E-mail: trotskiphilipp@gmail.com