УДК 004.932.4

И. С. Шошин

Применение анизотропного фильтра Перона – Малика в задаче распознавания посадочной площадки

Рассмотрена реализация сглаживающего анизотропного фильтра Перона – Малика на графическом процессоре. Фильтр содержит встроенный детектор границ, благодаря чему сохраняет значимые контуры при сглаживании, однако вычислительная сложность не позволяет применять его в режиме реального времени. Предлагаемая схема была использована в качестве элемента алгоритма распознавания посадочного ориентира на цифровом дискретном изображении. При моделировании оптической системы посадки беспилотного летательного аппарата вертолетного типа работа модифицированного фильтра показала высокую производительность и качество обработки.

Ключевые слова: фильтр Перона – Малика, анизотропный фильтр, *GPGPU*, *OpenCL*, обработка изображений, сглаживание изображений, оптическая система посадки, посадочные ориентиры.

Введение

Распознавание посадочного ориентира – необходимое условие для осуществления самостоятельной посадки беспилотного летательного аппарата (БПЛА) вертолетного типа. Беспилотные вертолеты могут быть использованы для автоматизации спасательных, разведывательных и оборонительных задач. Использование информации о положении аппарата относительно обнаруженной посадочной площадки позволит скорректировать траекторию посадки и повысить точность ее выполнения. Корпус беспилотного устройства оборудован видеокамерой или системой видеокамер. В процессе обнаружения посадочных ориентиров изображение с камеры содержит шумы и избыточную для распознавания информацию. Предшествующий распознаванию этап предполагает предварительную обработку изображения, получаемого из оптической системы БПЛА. Один из приемов по удалению шума из входящего изображения – это выполнение операции сглаживания.

В настоящей статье рассмотрен фильтр, представленный Пероном и Маликом [1]. Выбор метода обусловлен наличием детектора границ, что позволяет сохранить значимые края на изображении при фильтрации шума. Это необходимо, например, для алгоритмов распознавания, основанных на контурном анализе.

Математическая модель

Согласно работе [1], запишем уравнение анизотропной диффузии

$$\begin{aligned} \boldsymbol{I}_t &= \operatorname{div}(c(x, y, t) * \nabla \boldsymbol{I}) = \\ &= c(x, y, t) * \Delta \boldsymbol{I} + \nabla c * \nabla \boldsymbol{I}, \end{aligned} \tag{1}$$

где div – оператор дивергенции;

$c(x, y, t)$ – величина коэффициента диффузии;

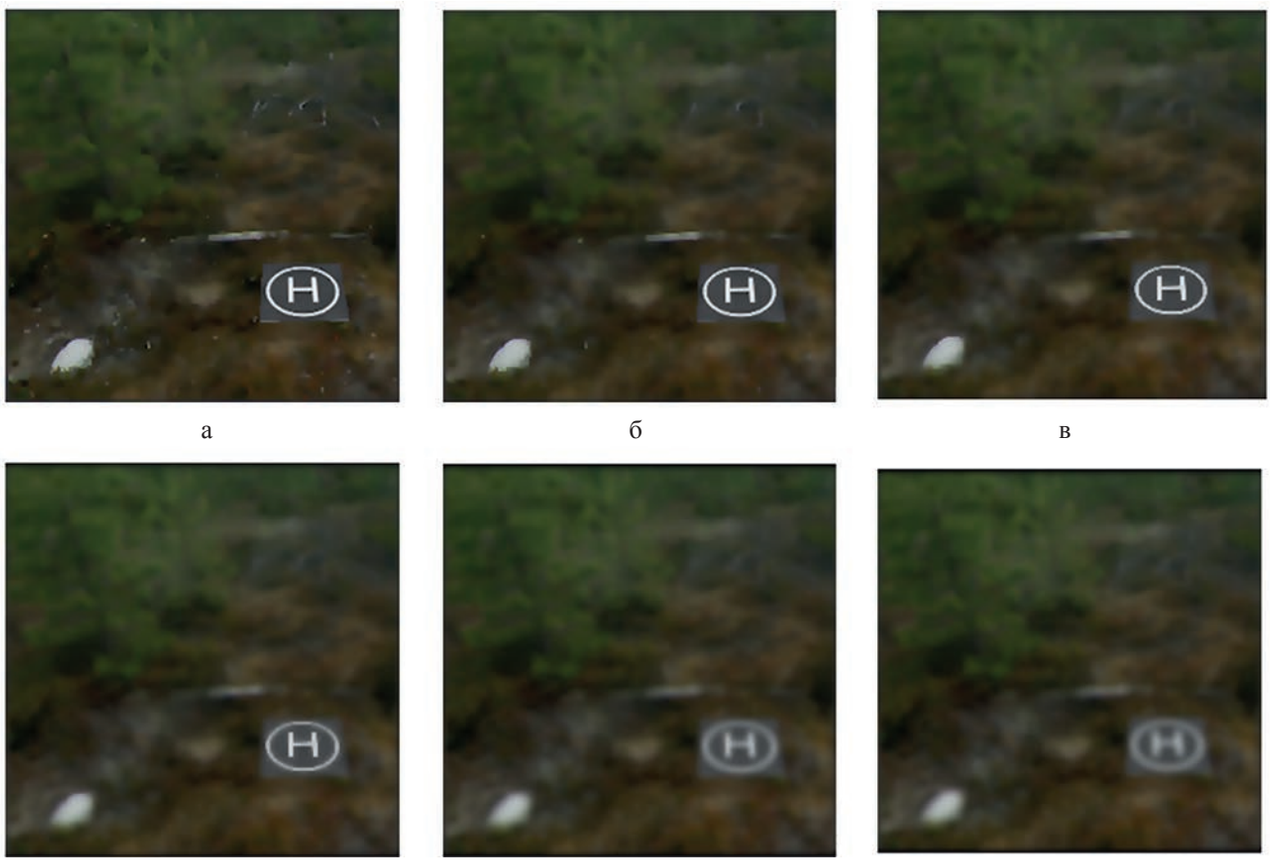
∇, Δ – градиент и лапласиан соответственно.

Для сохранения границ изображения от размытия коэффициент диффузии вычисляется в зависимости от направления максимального изменения яркости изображения в точке (x, y). В работе [1] авторы предложили следующие функции для вычисления коэффициента диффузии:

$$g(\nabla \boldsymbol{I}) = e^{\left(\left(\frac{\|\nabla \boldsymbol{I}\|}{K}\right)^2\right)}; \tag{2}$$

$$g(\nabla \boldsymbol{I}) = \frac{1}{1 + \left(\|\nabla \boldsymbol{I}\| / K\right)^2}. \tag{3}$$

Коэффициент K имеет фиксированную, экспериментально подбираемую величину, обозначающую порог чувствительности формулы к границам, который также может быть получен с помощью функции оценки уровня шума изображения. Низкое значение K приводит к незначительному размытию, высокое – сводит фильтр к линейному размытию. Варианты обработки изображения размером 256×256 пикселей, содержащего посадочный ориентир, 16 проходами фильтра Перона – Малика, использующего формулу (2) для получения коэффициента диффузии с разными значениями K, приведены на рис. 1.



а б в

г д е

Рис. 1. Результат воздействия диффузного коэффициента на процесс обработки изображения при *K*, равном: а – 8; б – 13; в – 21; г – 34; д – 55; е – 89

Последовательная реализация фильтра Перона – Малика

Последовательную реализацию алгоритма на основе дискретизации уравнения (3), представленной в работе [1], можно выразить с помощью псевдокода следующим образом:

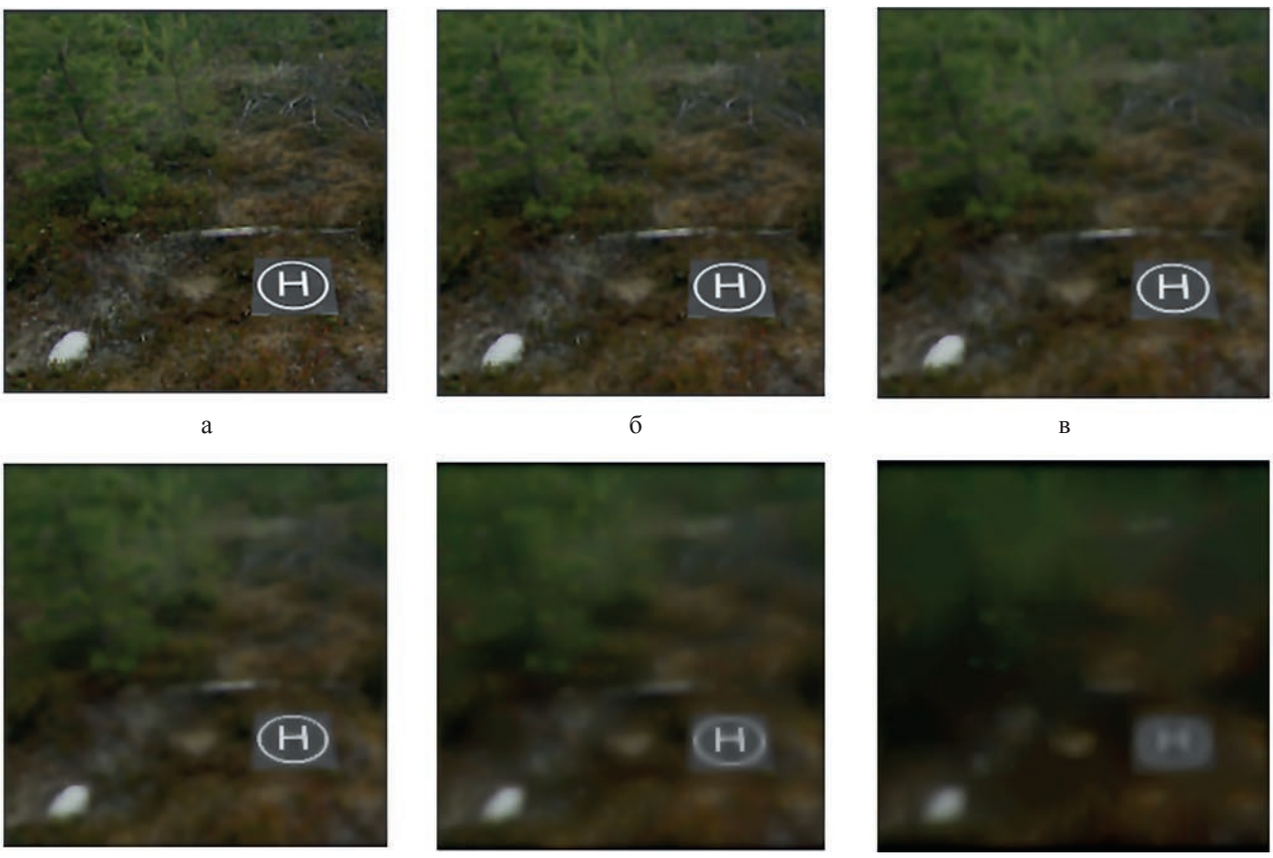
цикл *it* = [0, кол-во итераций]:
  цикл *x* = [1, ширина-1]:
    цикл *y* = [1, высота-1]:
      $dn, ds = I_{x, y-1} - I_{x, y}, I_{x, y+1} - I_{x, y}$
      $dw, de = I_{x-1, y} - I_{x, y}, I_{x+1, y} - I_{x, y}$
      $cn, cs = g(|dn|), g(|ds|)$
      $ce, cw = g(|de|), g(|dw|)$
      $I_{x, y} = p + 0.25 * (cn * dn + cs * ds + ce * de + cw * dw)$

Последовательная реализация такого алгоритма имеет сложность $O(w*h*t)$, где *w*, *h* – ширина и высота изображения соответственно; *t* – количество проходов фильтра (на практике будет содержать двухзначное число). При такой реализации алгоритм можно использовать только для постобработки, а не для систем реального времени. Эффект сглаживания при увеличении количества итераций можно наблюдать на рис. 2. Изображения обработаны фильтром Перона – Малика, использующим формулу (2) для получения коэффициента диффузии при $K = 34$.

Алгоритм представленной схемы поддается распараллеливанию, благодаря чему возможна его реализация на графическом процессоре, предоставляющем наиболее высокую производительность.

Параллельная реализация фильтра Перона – Малика на *GPU*

Для оперативного получения результата предлагается реализация работы фильтра на графическом процессоре. Для этих целей была выбрана программная платформа *OpenCL* [2], позволяющая выполнять параллельные вычисления на центральных и графических процессорах разных производителей (*Intel*, *AMD*, *NVidia*, *Texas Instruments* и др.).



а б в

г д е

Рис. 2. Результат обработки изображения размером 256×256 пикселей анизотропным фильтром Перона – Малика при *t*, равном:
а – 2; б – 4; в – 8; г – 16; д – 32; е – 64

Схема адаптированного алгоритма фильтрации Перона – Малика представлена на рис. 3.

Каждый рабочий поток выполняет одну и ту же версию ядра программы, но с разными входными данными. Общее количество таких рабочих элементов зависит от модели процессора. Если изображение имеет слишком большой размер, который превосходит максимально возможное количество рабочих потоков по двум направлениям, изображение разбивается на несколько частей, а запуск параллельной фильтрации выполняется в несколько итераций для каждой из них. Массив пикселей всего изображения (при условии, что размер массива не превосходит максимально допустимый объем используемой памяти устройства) передается в глобальную память графического процессора посредством прямого доступа к памяти (*DMA*). При этом область памяти доступна как для чтения, так и для записи. Благодаря этому нет задержек между обработкой частей изображения, связанных с передачей данных между *GPU* и *CPU* на данных этапах. Достаточно лишь передать смещение относительно нача-

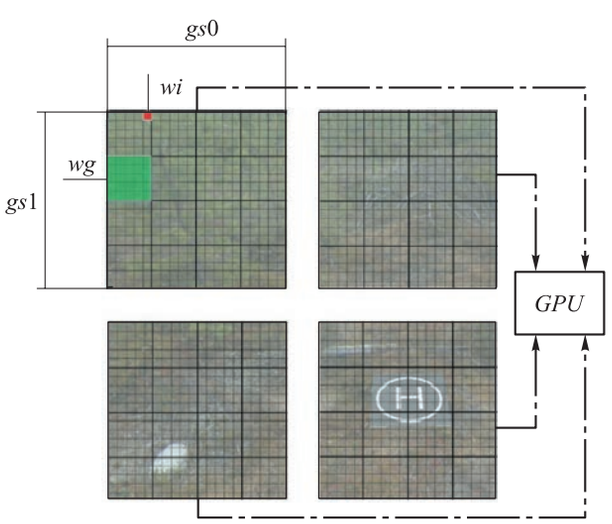


Рис. 3. Схема реализации фильтра Перона – Малика на *GPU*:
*gs*0, *gs*1 – максимальный размер рабочих потоков в каждом измерении; *wi* – рабочий поток, исполняющий ядро программы; *wg* – группа рабочих потоков

ла при выполнении параллельной фильтрации очередного сегмента изображения.

Приведем код ядра программы, выполняемый на *GPU*:

```
/*!
\param pixels  пиксели изображения
\param w  ширина изображения
\param h  высота изображения
\param offset_x  смещение по x
\param offset_y  смещение по y
*/
__kernel void pm (__global uint* pixels
                  __private int w,
                  __private int h,
                  __private int offset_x,
                  __private int offset_y)
{
// глобальные индексы рабочего потока + смещения по изображению
   const int x = offset_x + get_global_id(0);
   const int y = offset_y + get_global_id(1);
// предотвращение выхода за пределы отведенной памяти под массив пикселей
   if(x < w && y < h)
   {
// вычисления, как и в последовательной реализации, но без циклов
// ...
   }
}
```

Код, выполняемый на *CPU*, осуществляет запуск параллельных вычислений:

```
// ...
// разбиение изображения на сегменты
int parts_x = ceil(idata->w / (float)max_work_group_size);
int parts_y = ceil(idata->h / (float)max_work_group_size);
int offset_x = 0, offset_y = 0;
// кол-во проходов фильтра
for(int it = 0; it < pdata->iterations; ++it) {
// смещение по X
for(int px = 0; px < parts_x; ++px) {
    offset_x = px * work_size[0];
    clSetKernelArg (kernel, idx_offset_x, sizeof (int), (void *)&offset_x);
// смещение по Y
for(int py = 0; py < parts_y; ++py) {
    offset_y = py * work_size[1];
    clSetKernelArg (kernel, idx_offset_y, sizeof (int), (void *)&offset_y);
    clFinish(queue);

// выполнение параллельной фильтрации для сегмента изображения
CheckError(clEnqueueNDRangeKernel(queue, kernel, 2, nullptr, work_size,
      nullptr, 0, nullptr, &event));
} } }
// ...
```

Сравним производительность фильтра при использовании последовательного алгоритма Перона – Малика, *OpenCL* реализации на центральном и графическом процессорах. В таблице представлено время работы каждой из реализаций фильтра для указанных входных данных.

На рис. 4 приведен график зависимости времени обработки от размера изображения

Производительность фильтра в зависимости от реализации и входных данных

Реализация	Вычислительное устройство	Количество проходов	Размер изображения, px	Время обработки, с
CPU	*Intel(R) Core(TM) i5-4278U CPU 2.60 GHz*	16	256×256	0,3320
CPU OpenCL	*Intel(R) Core(TM) i5-4278U CPU 2.60 GHz*	16	256×256	0,0030
GPU OpenCL	*Intel(R) HD 5100*	16	256×256	0,0006
CPU	*Intel(R) Core(TM) i5-4278U CPU 2.60 GHz*	16	1920×1920	18,1870
CPU OpenCL	*Intel(R) Core(TM) i5-4278U CPU 2.60 GHz*	16	1920×1920	2,4910
GPU OpenCL	*Intel(R) HD 5100*	16	1920×1920	0,4650
CPU	*Intel(R) Core(TM) i5-4278U CPU 2.60 GHz*	64	1920×1920	68,8880
CPU OpenCL	*Intel(R) Core(TM) i5-4278U CPU 2.60 GHz*	64	1920×1920	10,1390
GPU OpenCL	*Intel(R) HD 5100*	64	1920×1920	1,4950

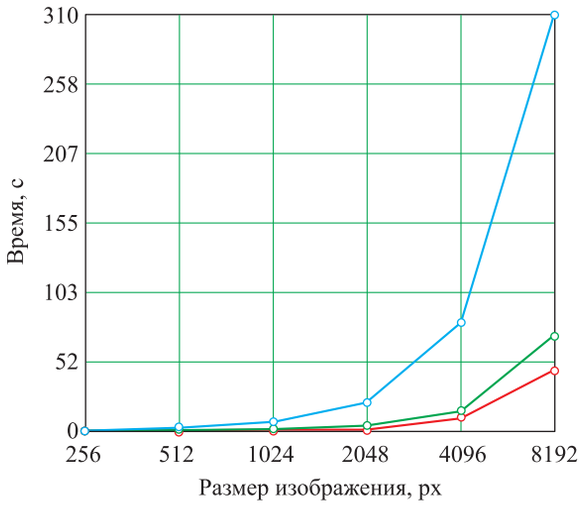


Рис. 4. График зависимости времени обработки от размера изображения:
– *CPU*; – *CPU OpenCL*; – *GPU OpenCL*

при фиксированном количестве проходов, равном 16. Обработка изображений осуществляется на следующих вычислительных устройствах: *Intel*(*R*) *Core*(*TM*) *i*5-4278*U CPU* 2.60 *GHz* и *Intel*(*R*) *HD* 5100.

Как видно из приведенных данных (см. таблицу и рис. 4), реализация на графическом процессоре представляет наиболее производительный результат выполнения обработки изображения анизотропным фильтром Перона – Малика.

Симулятор оптической системы посадки

В работе [3] представлена модульная среда отладки алгоритма распознавания системы посадки беспилотного вертолета (рис. 5), позволяющая моделировать сигнал, получаемый с камеры БПЛА, а также ряд возможных искажений: перспективных искажений, наличие перекрытий, смены освещения и др.

В среде реализован современный подход к моделированию освещения на основе окружения. Инструмент позволяет оценить работоспособность разрабатываемого алгоритма распознавания, управлять параметрами загружаемых в виде расширений фильтров и наблюдать за результатом их воздействия на процесс обработки изображения. Среда удобна для поиска экспериментальных величин алгоритмов, как, например, коэффициент чувствительности формул (2) и (3) к границам в зависимости от окружения, размера изображения и количества проходов. Реализованный с помощью

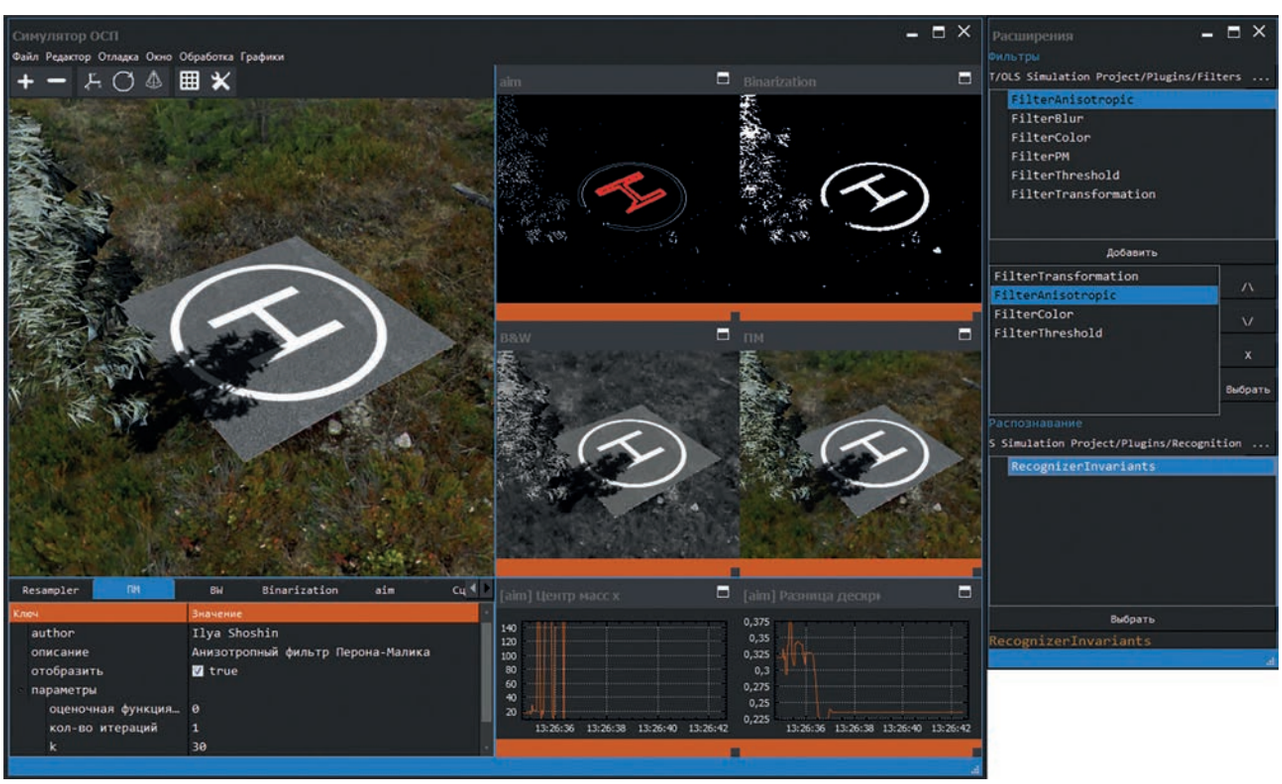


Рис. 5. Скриншот модульной среды отладки алгоритма распознавания системы посадки беспилотного вертолета

программной платформы *OpenCL* фильтр Перона – Малика был успешно интегрирован в среду в виде динамической библиотеки (*.dll*).

Заключение

Анизотропный фильтр Перона – Малика сглаживает входящее изображение, сохраняя значимые границы. Алгоритм подлежит распараллеливанию, благодаря чему была предложена адаптация алгоритма для выполнения параллельной фильтрации на графическом процессоре с помощью вычислительной платформы *OpenCL*. Проведенный анализ продемонстрировал высокую производительность реализации фильтра на *GPU* и возможность использования в среде моделирования оптической системы посадки для решения задачи распознавания посадочных ориентиров.

Список литературы

1. *Perona P., Malic J.* Scale-space and edge detection using anisotropic diffusion // IEEE Transactions on pattern analysis and machine intelligence. July 1990. Vol. 12. No. 7. 11 p.

2. *Nakamura T., Lizuka T., Asahara A., Miki S.* The OpenCL programming book. Fixstars Corporation, 2010. 324 p.

3. *Шошин И. С.* Модульная среда отладки алгоритма распознавания для системы посадки беспилотного вертолета // Союз машиностроителей России. Будущее машиностроения России: сб. докл., 5–8 октября 2016 г. М.: Изд-во МГТУ им. Н. Э. Баумана, 2016. С. 740–744.

Поступила 13.01.17

Шошин Илья Сергеевич – инженер-программист АО «Государственный научно-исследовательский институт приборостроения», г. Москва.

Область научных интересов: распознавание образов, *3D*-моделирование.

Employing the Perona – Malik anisotropic filter for the problem of landing site detection

The article deals with implementing a Perona – Malik anisotropic smoothing filter on a *GPU*. The filter features a built-in edge detector and because of this is able to preserve meaningful contours during smoothing, but its computational complexity prevents using it in real-time mode. We used the scheme suggested as an element of a landing guide detection algorithm operating on a discrete digital image. The modified filter demonstrated high performance and processing quality during simulating an optical landing system for a copter-type unmanned aerial vehicle.

Keywords: Perona – Malik filter, anisotropic filter, *GPGPU*, *OpenCL*, image processing, image smoothing, optical landing system, landing guides.

Shoshin Ilya Sergeevich – Programming Engineer, State Research Institute of Instrument Engineering Joint-Stock Company, Moscow.

Science research interests: image recognition, *3D*-modelling.