УДК 511

РЕАЛИЗАЦИЯ ПРОЦЕДУРЫ ПОСТОБРАБОТКИ БЫСТРОГО АЛГОРИТМА ГЕОМЕТРИЧЕСКОГО КОДИРОВАНИЯ ЦИФРОВЫХ ИЗОБРАЖЕНИЙ С ПРИМЕНЕНИЕМ АРХИТЕКТУРЫ CUDA

Г. В. Носовский[1], **А. Ю. Чекунов**[2]

В статье представлено улучшение алгоритма распознавания контуров. Идея алгоритма основана на вычислении геометрических характеристик двумерной поверхности в R^3, кодирующей данное изображение. Для получения результирующей картины контуров вычисляется фрактальная размерность Минковского по всему изображению. Описывается алгоритм постобработки предварительно полученных контуров цифровых изображений в части их утончения. Также представлена оценка скорости работы алгоритма с учетом новой процедуры в сравнении с широко известной реализацией алгоритма Канни (Canny), использующей библиотеку компьютерного зрения OpenCV и параллельную архитектуру CUDA. Приведены примеры, демонстрирующие работоспособность новой процедуры алгоритма.

Ключевые слова: распознавание образов, геометрическое кодирование, кодирующая поверхность, контурный анализ, распознавание контуров, компьютерное зрение, обработка изображений, склейка изображений.

[1] *Носовский Глеб Владимирович* — канд. физ.-мат. наук, доцент каф. дифференциальной геометрии и приложений мех.-мат. ф-та МГУ, e-mail: gleb.nosovskiy@gmail.com.

[2] *Чекунов Алексей Юрьевич* — e-mail: chekunov.alexey@gmail.com.

Nosovskii Gleb Vladimirovich — Candidate of Physical and Mathematical Sciences, Associate Professor, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Differential Geometry and Applications.

Chekunov Aleksei Yurievich.

The paper presents some improvements of fast contour recognition algorithm. The main idea of this algorithm is based on calculation of geometric characteristics of a two-dimensional surface in R^3, which encodes digital images. To obtain the contours, Minkowski fractal dimension calculation is used. A post-processing algorithm of contour thinning without loss of quality of entire contour picture is proposed in the paper. An estimate of the speed of the new algorithm is presented in comparison with the well-known implementation of the Canny algorithm using the OpenCV computer vision library and the parallel architecture of CUDA. Examples demonstrating the efficiency of the new algorithm are presented.

Key words: pattern recognition, geometrical coding, coding surface, contour analysis, edge detection, computer vision, image processing, image stitching.

Введение. Одной из важнейших задач современной компьютерной геометрии является быстрая обработка цифровых изображений. На практике первичный анализ изображений часто начинается с распознавания контуров (границ) расположенных на них объектов. Недавно в работе [1] был предложен новый дифференциально-геометрический подход к решению данной проблемы. В работе [2] был подробно описан алгоритм геометрического кодирования, позволяющий быстро выделять границы цифровых изображений. В качестве пороговой фильтрации использовалась техника вычисления фрактальной размерности Минковского (см. [3]) по всему изображению, предъявляющая границы некоторой толщины. В большинстве задач компьютерного зрения уже достаточно таких границ, поэтому процедура их последующей постобработки не носит необходимый характер. Однако в некоторых случаях важно, чтобы выделенные границы достигали толщины в один пиксель. В работе [2] было подмечено, что даже при стремлении размерности Минковского к единице границы остаются утолщенными, при этом часть деталей может быть потеряна. Поэтому важно разработать алгоритм, способный без потери качества утончать предварительно выделенные границы.

В настоящей работе такой алгоритм описан. Будучи процедурой постобработки, алгоритм должен получать на входе одноканальное изображение с утолщенными границами. Процесс получения утолщенных границ подробно описан в [2].

Утончение границ цифровых изображений. Напомним, что алгоритм геометрического кодирования основан на вычислении некоторой функции распознавания DF (Detection Function) типа f или h (см. [1]). Функция выбирается из набора специально разработанных геометрических характеристик на кодирующей поверхности (см. [1]). Кодирующая поверхность параметризуется координатами на плоскости изображения, поэтому мы можем рассматривать функцию DF как функцию плоскости IP (Image Plane). После вычисления функции DF точки границ определяются как точки, в которых DF превышает определенный порог T: $C = \{x \in IP : DF(x) > T\}$. Порог T вычисляется автоматически с использованием целевого значения фрактальной размерности Минковского MD для результирующих контуров. Значение MD задается в программе на основе предварительных расчетов (см. [2]). Реализация данного процесса с использованием параллельной архитектуры CUDA описана в работе [4].

Контуры, полученные с помощью описанного выше алгоритма, обычно имеют толщину в несколько пикселей. Будем называть их *первичными контурами*. Далее будет описана вычислительная процедура, которая уменьшает толщину первичных контуров до одного пикселя.

Прежде всего введем необходимые определения. Будем называть пиксели, принадлежащие первичным контурам, *черными точками*, а не принадлежащие — *белыми точками*.

Сопоставим каждой черной точке x квадрат $V(x)$ размера 2×2 таким образом, чтобы точка x являлась левым верхним углом этого квадрата. (С тем же успехом можно было бы сопоставить каждой черной точке другой содержащий ее квадрат 2×2, однако результат будет практически тем же самым.) Естественно, квадраты, соответствующие различным черным точкам, могут пересекаться друг с другом.

Описание алгоритма утончения контуров.

Шаг 1. Для каждой черной точки x расположим четыре значения функции DF в квадрате $V(x)$ в порядке убывания: $DF(x_1) \geqslant DF(x_2) \geqslant DF(x_3) \geqslant DF(x_4)$, где $x_i \in V(x), 1 \leqslant i \leqslant 4$.

Присвоим *сильную белую метку* точке x_4 и *слабую белую метку* точке x_3.

Шаг 2. Все черные точки, которые получили хотя бы одну сильную белую метку, превращаем в белые точки, т.е. исключаем их из окончательных контуров. Черные точки, которые получили только слабые белые метки, помечаем как *слабые черные точки*. Все остальные черные точки, которые вообще не получили меток, помечаем как *сильные черные точки*. В итоге имеем *сильно утонченные контуры*, состоящие из сильных черных точек, и *слабо утонченные контуры*, состоящие из сильных и слабых черных точек (см. рис. 1).

Как сильно, так и слабо утонченные контуры почти всюду имеют толщину в один пиксель, поскольку по построению они не могут содержать в себе никакого квадрата 2×2, целиком состоящего из черных точек.

Рис. 1. Слева — исходное изображение, справа: a — первичные контуры; b — сильно утонченные контуры, имеющие лишние разрывы; c — слабо утонченные контуры, содержащие ложные отростки; d — окончательные контуры после процедуры утончения

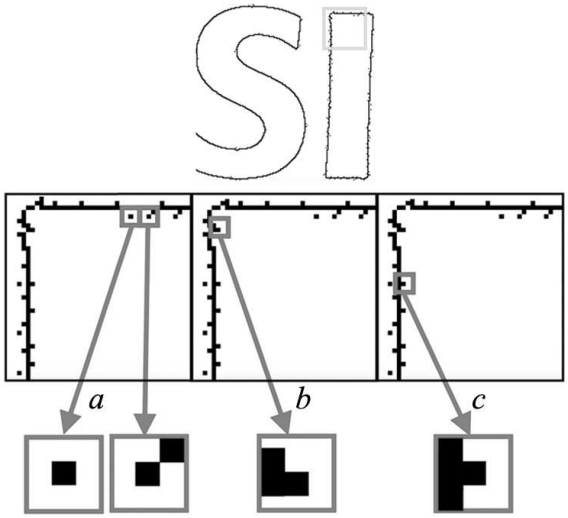

Рис. 2. Пример работы алгоритма удаления ложных отростков

Сильно утонченные контуры состоят из пикселей, в которых значение функции DF достигает локального максимума. В случае, когда этот максимум нестрогий, из нескольких соседних пикселей с одинаковым локально максимальным значением функции DF будет оставлен лишь один, который и войдет в сильно уточненный контур. Это значит, что сильно утонченный контур будет проходить по оси первичного контура там, где значение функции DF максимально. Однако сильно утонченные контуры имеют существенный недостаток: они, как правило, содержат множество мелких разрывов (см. рис. 1, b).

Слабо утонченные контуры получаются непрерывными в дискретном смысле. Их недостаток состоит в том, что они часто образуют *ложные отростки* длиной в несколько пикселей, ответвляющиеся от оси первичного контура.

Разрывы в сильно утонченных контурах можно было бы устранить с помощью специального алгоритма заклеивания дырок. Однако с вычислительной точки зрения это более затратно, чем исправить слабо утонченные контуры, удалив из них ложные отростки. Опишем процедуру удаления черных точек, которые образуют ложные отростки.

Шаг 3. Для каждой черной точки x слабо утонченных контуров рассмотрим 8 соседей из ее окрестности — квадрата 3×3 (без нее самой как центральной точки). Посчитаем количество точек слабо утонченных контуров среди этих 8 соседей. Если оно меньше 2, т.е. равно 0 или 1, то исключаем точку x из контура и помечаем ее как белую. Кроме того, исключаем точку x из контура и помечаем ее как белую также в случаях, когда среди ее указанных 8 соседей содержатся либо ровно две черные точки, смежные друг с другом по стороне (напомним, что наши точки — это пиксели, т.е. квадратные ячейки), либо ровно три черные точки, расположенные вдоль одной стороны упомянутого квадрата 3×3. Такой процедурой мы удалим концы отростков длины, превосходящей 1 пиксель (см. рис. 2, a, справа), однопиксельные отростки (см. рис. 2, b, c), а также изолированные черные точки (см. рис. 2, a, слева). На практике ложные отростки имеют длину 1–2 пикселя, поэтому после применения данного алгоритма длина всех отростков станет равной 1.

Шаг 4. В завершение удалим ложные отростки длины 1. Рассмотрим оставшиеся черные точки слабо утонченных контуров. Для каждой такой точки повторно применим алгоритм шага 3.

Поскольку контуры уже не содержат никакого квадрата 2×2, целиком состоящего из черных пикселей, то указанной процедуры достаточно для удаления однопиксельных отростков.

После выполнения всех шагов получаем окончательные утонченные контуры. Они будут проходить по оси первичных контуров, там, где значение функции DF максимально, и не содержать лишних разрывов. Пример таких контуров представлен на рис. 1, *d*.

Оценка скорости работы алгоритма. Сравним скорости работы реализаций алгоритма геометрического кодирования с процедурой утончения границ и алгоритма выделения границ Canny (см. [5]), широко используемого в библиотеке компьютерного зрения OpenCV (оба алгоритма используют CUDA).

Заметим, что аналогичное сравнение реализаций алгоритмов уже описывалось в работе [4], однако алгоритм геометрического кодирования применялся без процедуры утончения.

Ввиду того что CUDA-реализация алгоритма Canny обрабатывает только изображения в оттенках серого, сравнение проводилось с версией, работающей только с изображениями в оттенках серого, однако в работе [4] показано, что алгоритм геометрического кодирования реализован и для обработки цветных цифровых изображений с применением архитектуры CUDA. В качестве функции DF выделения границ алгоритма геометрического кодирования была выбрана наиболее быстрая однородная функция h_3:

$$h_3(k_1, k_2) = \frac{(k_1 + k_2)^2}{k_1 k_2},$$

где $k_1, k_2 > 0$ являются собственными значениями первой фундаментальной формы (метрического тензора) в соответствующей точке кодирующей поверхности (см. [1]).

Все вычисления были произведены на стационарном персональном компьютере со следующей конфигурацией:

CPU: Intel(R) Core(TM) i7 CPU 9600K (3.6 GHz), 32 GB RAM;

GPU: NVIDIA Geforce GTX 1050 Ti;

OpenCV version: 4.5.0.

Спецификации GPU:

Device: Geforce GTX 1050 Ti;

CUDA Driver Version: 11.4;

CUDA Capability version number: 6.1;

Total amount of global memory: 4096 MB;

CUDA Cores: 768;

Warp size: 32.

В качестве эксперимента было обработано по 1000 цифровых изображений с разрешениями 100 x 100, 1000 x 1000, 2000 x 2000 и 4000 x 4000.

Запуск реализации алгоритма Canny с применением CUDA осуществляется с помощью следующего метода класса cv::cuda::CannyEdgeDetector библиотеки OpenCV:

virtual void cv::cuda::CannyEdgeDetector::detect (InputArray image, OutputArray edges).

Описание параметров метода:

image — оригинальное изображение (цветное или в оттенках серого);

edges — одноканальное изображение с результирующей картиной контуров.

Описание параметров класса cv::cuda::CannyEdgeDetector:

thresh1 — нижний порог для процедуры постобработки контуров;

thresh2 — верхний порог для процедуры постобработки контуров;

aperSize — размерность маски для нахождения производных;

L2grad — флаг, устанавливающий возможность выбора более точной нормы L_2, если L2grad = true, или стандартной нормы L_1, если L2grad = false.

В наших вычисления используются следующие значения описанных выше параметров:

thresh1 = 100,0; thresh2 = 150,0; aperSize = 3; L2grad = false.

Для работоспособности данного метода необходимо, чтобы библиотека OpenCV была установлена с поддержкой CUDA.

В таблице указано среднее время обработки 1000 изображений в миллисекундах с учетом этапов загрузки и выгрузки данных. Также указывается коэффициент, равный отношению скорости работы алгоритма Canny к скорости работы алгоритма геометрического кодирования (GC) с процедурой постобработки контуров.

Заметим, что чем больше размер обрабатываемого изображения, тем существеннее прирост

Размер изображения	Алгоритм Canny	Алгоритм GC	Прирост скорости
100 x 100	316	285	1.1
1000 x 1000	3331	2906	1.15
2000 x 2000	11735	9409	1.25
4000 x 4000	40883	31607	1.29

производительности для реализации новой версии алгоритма в оттенках серого, что является дополнительным преимуществом в эпоху развития изображений высокого разрешения.

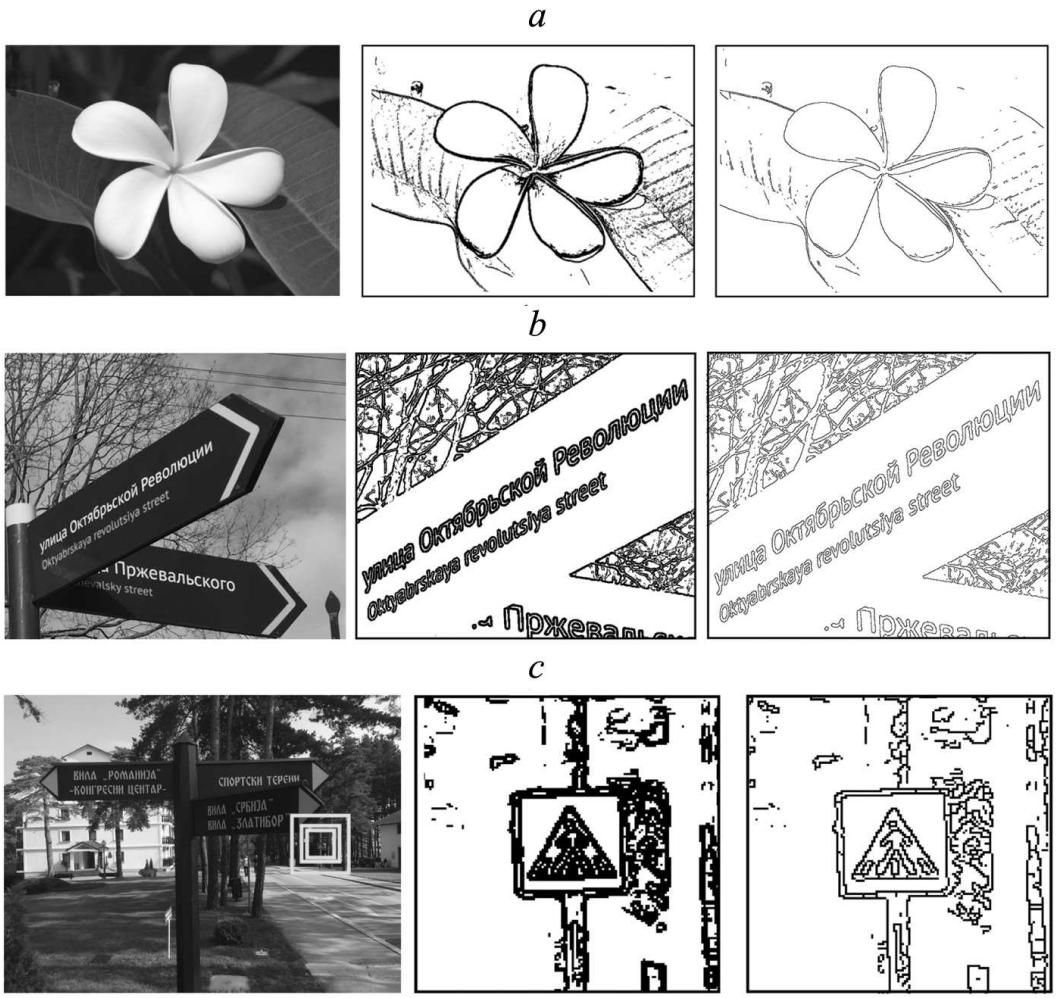

Рис. 3. Слева направо: оригинальное изображение → GC до утончения → GC после утончения

Приведем результаты работы алгоритма геометрического кодирования после процедуры утончения выделенных границ. В качестве основной функции выберем однородную функцию h_3. На рис. 3, *a* продемонстрирована процедура утончения границ цветка.

Наиболее часто утончение границ используется для улучшения читаемости букв, цифр и других мелких деталей, которые невозможно распознать с помощью предварительно выделенных границ. На рис. 3, *b*, *c* представлены примеры распознавания текста на указателе и дорожного знака в реальной ситуации, причем процедура утончения в этом случае улучшает читаемость деталей.

Заключение. Алгоритм геометрического кодирования имеет большой потенциал развития. Основные преимущества: работа с цветными изображениями, хорошая распараллеливаемость, адаптивный подбор параметров, быстрота работы, качество получаемого результата. Процедура постобработки (утончения), описанная в настоящей работе, позволяет применять данный алгоритм при распознавании мелких деталей изображения, а также текста. Расчеты показали, что предлагаемый

алгоритм особенно предпочтителен для изображений большого разрешения, поскольку обгоняет по скорости вычислений популярную реализацию алгоритма Canny. Этот факт предоставляет возможность применения данного алгоритма в промышленности и приложениях.

Работа выполнена при поддержке гранта РНФ (проект № 21–11–00355) в МГУ имени М. В. Ломоносова.

СПИСОК ЛИТЕРАТУРЫ

1. *Носовский Г.В.* Геометрическое кодирование цветных изображений // Вестн. Моск. ун-та. Матем. Механ. 2018. № 1. 3–11.
2. *Носовский Г.В., Чекунов А.Ю., Подлипаев С.А.* Быстрый алгоритм геометрического кодирования цифровых изображений // Вестн. Моск. ун-та. Матем. Механ. 2017. № 6. 20–27.
3. *Li J.Q., Sun D.C.* An improved box-counting method for image fractal dimension estimation // Pattern Recogn. 2009. **42**, N 11. 2460–2469.
4. *Чекунов А.Ю.* Реализация быстрого алгоритма геометрического кодирования цифровых изображений с применением архитектуры CUDA // Вестн. Моск. ун-та. Матем. Механ. 2018. № 6. 18–28.
5. *Canny J.* A computational approach to edge detection // IEEE Trans. Pattern Anal. and Mach. Intell. 1986. **8**. 679–714.

Поступила в редакцию
20.06.2022