УДК 511

Реализация процедуры постобработки быстрого алгоритма геометрического кодирования цифровых изображений с применением архитектуры CUDA

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

В данной 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. Существуют реализации алгоритма геометрического кодирования, которые сравнимы и даже обгоняют известные алгоритмы по скорости вычислений, а также качеству получаемого результата. В статье описывается алгоритм постобработки предварительно полученных контуров цифровых изображений в части их утончения. Данная процедура играет важную роль в области компьютерного зрения, особенно в случаях, когда необходимо сохранить все детали изображения при его минимальном объеме. Также представлена оценка скорости работы алгоритма с учетом новой процедуры в сравнении с широко известной реализацией алгоритма Канни (Canny) с применением библиотеки компьютерного зрения OpenCV и параллельной архитектуры CUDA. Примеры, демонстрирующие работоспособность новой процедуры алгоритма, представлены.

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

In this paper some improvements of fast contour recognition algorithm are presented. The main idea of this algorithm is based on the calculation of the geometric characteristics of a two-dimensional surface in R^3, which encodes digital images. To obtain the contours, Minkowski fractal dimension calculation is used. In this paper a post-processing algorithm of contour thinning without loss of quality of entire contour picture is suggested. Also, an estimation of the speed of 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 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. После этого 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, приводящая к границам некоторой толщины. В большинстве задач компьютерного зрения уже достаточно таких границ, поэтому процедура их последующей постобработки не несет 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 данной статье такой алгоритм описан. Являясь процедурой постобработки, алгоритм должен получать на входе одноканальное изображение с утолщенными границами. Процесс получения утолщенных границ подробно описан в работе [2].

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

[2] *Чекунов Алексей Юрьевич* — выпусник каф. дифференциальной геометрии и приложений мех.-мат. ф-та МГУ, e-mail: chekunov.alexey@gmail.com.

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 подробности в [2]. Подробная 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.* Ниже 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 точки, 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 упорядочим 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 $x_i \in V(x), \quad 1 \leqslant i \leqslant 4$.

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

Шаг 2. Все черные точки, которые получили хотя бы одну сильную белую метку, превращаем в белые точки, то есть – 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 получаем *сильно утонченные контуры*, состоящие из сильных черных точек и *слабо утонченные контуры*, состоящие из сильных и слабых черных точек, см. рис.1.

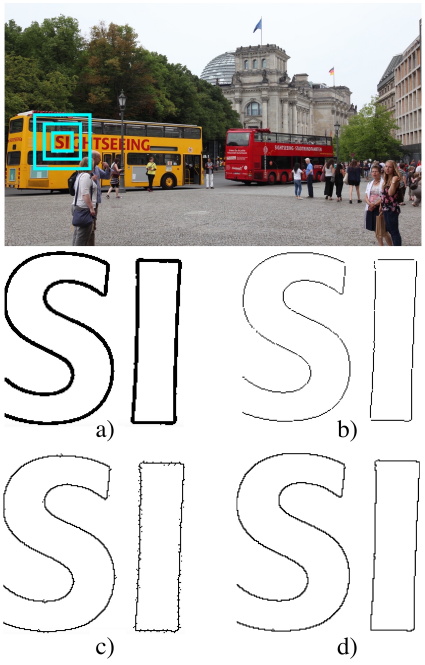


Рис. 1. Сверху — исходное изображение. Внизу: a — первичные контуры; b — множество сильных черных точек, образующее контуры толщиной в 1 пиксель, имеющие лишние разрывы; c — слабые утонченные контуры, содержащие ложные отростки; d — окончательные контуры после процедуры утончения

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.

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.

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. На рис. 2 представлен пример, демонстрирующий образование таких отростков.

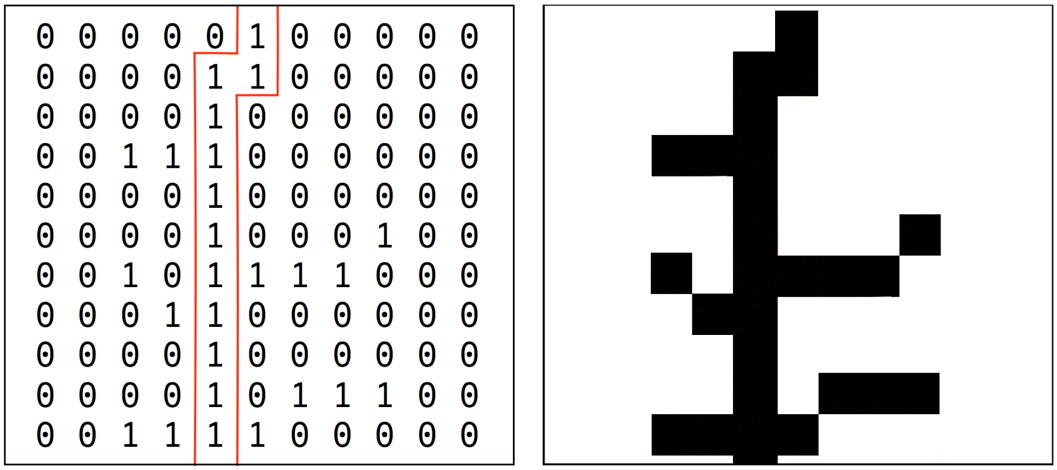


Рис. 2. Пример ложных границ изображения: a — пиксели слабо утонченного контура имеют значение 1, остальные пиксели – значение 0. Красным цветом показан истинный однопиксельный контур. Ложным отростками соответствуют остальные пиксели со значениями 1; b — результирующая картина слабо утонченных контуров с ложными отростками

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.

Назовем *промежуточными* те черные точки, которые являются слабыми, но не являются сильными. Поскольку сильно утонченные границы ложных отростков не содержат, то ложные отростки слабо утонченных контуров состоят из промежуточных точек. Опишем процедуру удаления тех промежуточных черных точек, которые образуют ложные отростки.

Шаг 3. Для каждой промежуточной точки x рассмотрим 8 соседей из ее окрестности — квадрата 3×3 (без ее самой как центральной точки). Посчитаем количество точек слабо утонченных контуров среди этих 8 соседей. Если оно меньше 2 – то есть 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 содержится либо ровно две промежуточных черных точки, смежных друг с другом по стороне (напомним, что наши точки – это пиксели, то есть квадратные ячейки), либо ровно три промежуточных черных точки, расположенные вдоль одной стороны упомянутого квадрата 3×3. Такой процедурой мы удалим концы отростков длины, превосходящей 1 пиксель (уменьшая их длину на 1), боковые точки на отростках, а также изолированные черные точки (см. рис. 3а). Будем повторять данный шаг алгоритма до тех пор, пока происходит хотя бы одно забеление черной точки. В итоге длина всех отростков станет равной 1.

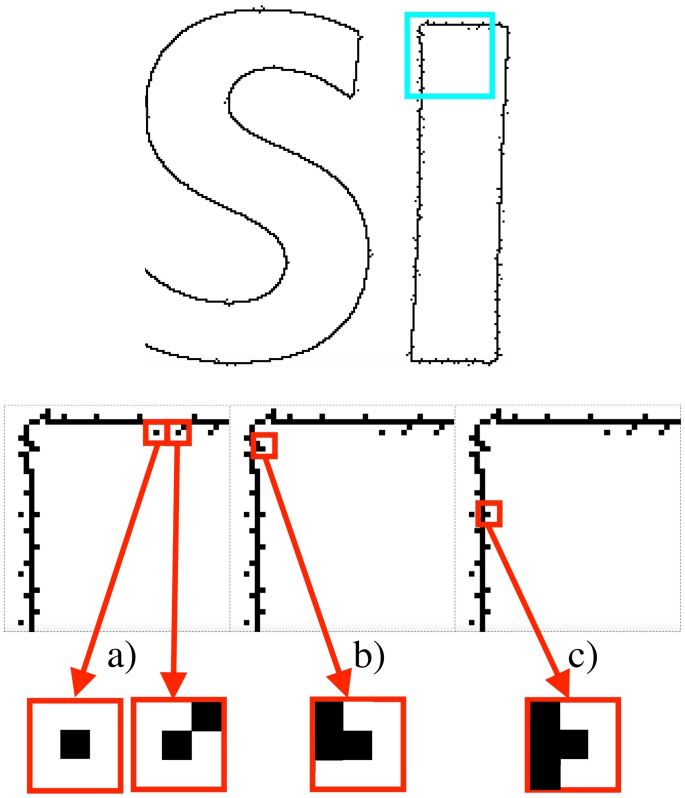


Рис. 3. Пример работы алгоритма удаления ложных отростков

Шаг 4. В завершении удалим ложные отростки длины 1. Рассмотрим оставшиеся промежуточные точки. Для каждой такой точки будем превращать соответствующий пиксель в белый в случае, когда в его выколотой окрестности 3×3 с центром в этом пикселе содержится (не считая этого пикселя) либо ровно два черных пикселя, смежных друг с другом по стороне (не по углу), либо ровно три черных пикселя, расположенных вдоль одной стороны квадрата (см. рис. 3*b-c*).

Поскольку контуры уже не содержат никакого квадрата 2×2, целиком состоящего из черных пикселей, то указанная процедура достаточна 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

В данном разделе сравнивается скорость работы реализаций алгоритма геометрического кодирования с процедурой утончения границ с алгоритмом выделения границ Canny (см. [5]), широко используемым в библиотеке компьютерного зрения *OpenCV* (оба с применением $CUDA$).

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 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. [1-2]).

Все вычисления были произведены на стационарном персональном компьютере со следующей кон-

фигурацией:

- 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 100 цифровых изображений с различными разрешениями:

- 1000 x 1000;
- 2000 x 2000;
- 4000 x 4000;
- 6000 x 6000.

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.

Таблица 1

Размер изображения	Алгоритм Canny	Алгоритм GC	Прирост скорости
1000 x 1000	154	139	1.1
2000 x 2000	494	389	1.27
4000 x 4000	1873	1281	1.46
6000 x 6000	4072	2751	1.48

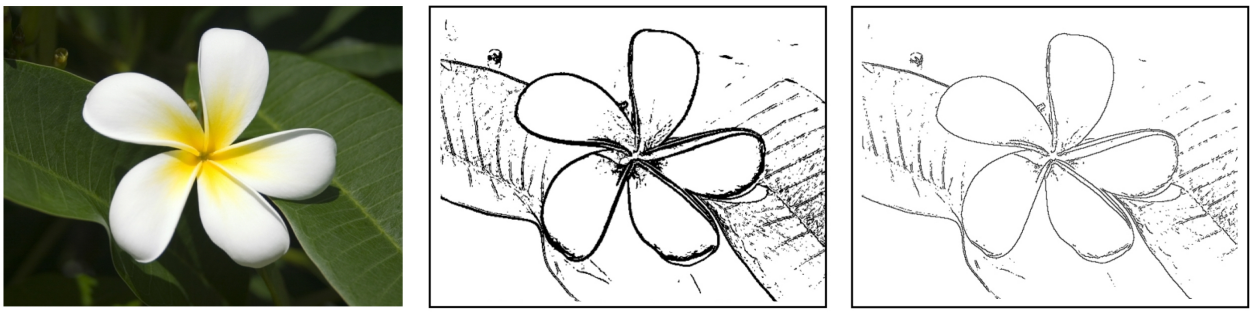
Для работоспособности данного метода необходимо, чтобы библиотека *OpenCV* была установлена с поддержкой *CUDA*.

В таблице указано среднее время обработки 100 изображений в миллисекундах без учета 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 производительности для реализации новой версии алгоритма в оттенках серого, что является дополнительным преимуществом в эпоху развития изображений высокого разрешения.

Демонстрация качества работы алгоритма

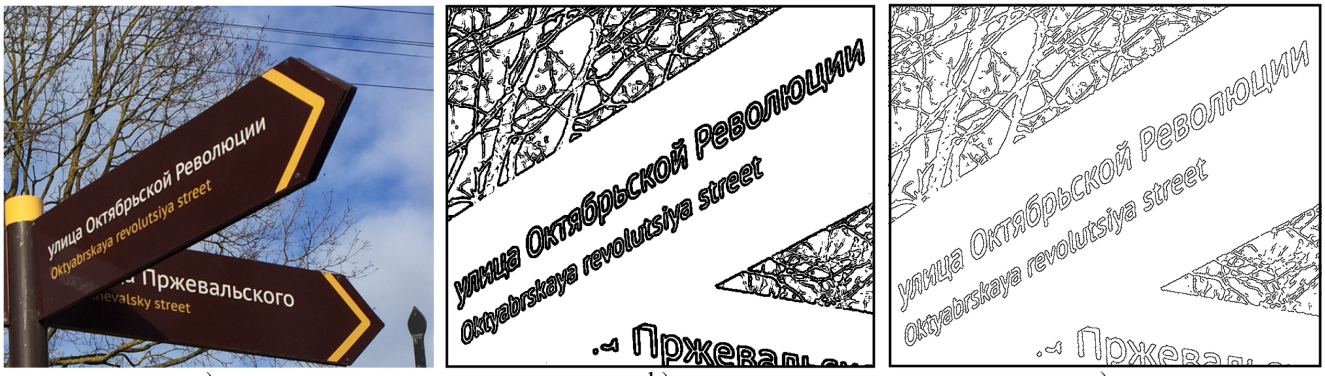
Приведем результаты работы алгоритма геометрического кодирования после процедуры утончения выделенных границ. В качестве основной функции выберем однородную функцию h_3. На рис. 4 продемонстрирована процедура утончения границ цветка.



a) b) c)

Рис. 4. a — оригинальное изображение; b — GC до утончения; c — GC после утончения

Наиболее часто утончение границ используется для улучшения читаемости букв, цифр и других мелких деталей, которые невозможно распознать с помощью предварительно выделенных границ. На рис. 5-6 представлены примеры распознавания текста на указателе и дорожного знака в реальной ситуации, причем процедура утончения в этом случае улучшает читаемость деталей.



a) b) c)

Рис. 5. a — оригинальное изображение; b — GC до утончения; c — GC после утончения

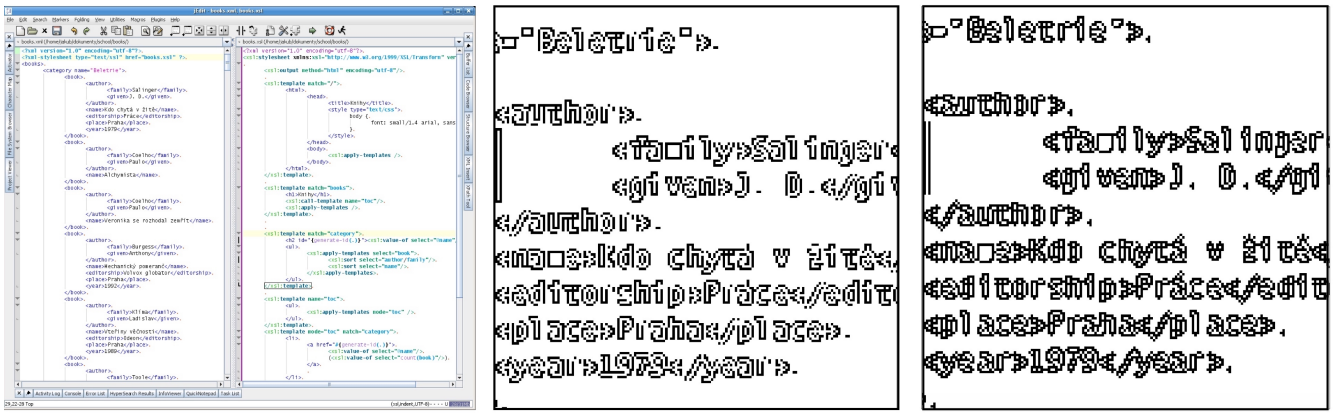
a) b) c)

Рис. 6. *a* — оригинальное изображение; *b* — GC до утончения; *c* — GC после утончения

Обратим внимание, что результат работы алгоритма геометрического кодирования после процедуры утончения сравним по качеству с результатом работы детектора границ Canny (см. рис. 7-8).

a) b) c)

Рис. 7. *a* — оригинальное изображение; *b* — Детектор границ Canny; *c* — GC после утончения



a) b) c)

Рис. 8. *a* — оригинальное изображение; *b* — Детектор границ Canny; *c* — GC после утончения

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 данной статье, позволяет применять данный алгоритм при распознавании мелких деталей изображения, а также текста. Расчеты показали, что предлагаемый в статье алгоритм особенно предпочтителен на изображениях большого разрешения, обгоняя по скорости вычислений популярную реализацию алгоритма Canny. Этот факт делает возможным применение данного алгоритма в промыш-

ленности 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 Recognition. 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 Transactions on Pattern Analysis and Machine Intelligence. 1986. **8**. 679–714.