Реализация некоторых алгоритмов обработки изображений с использованием технологии CUDA на графических устройствах

Н.Н. Богословский
Томский государственный университет

Обработки цифровых изображений в настоящее время – очень актуальная проблема. Часто обрабатывается большой объем цифровых изображений, а многие алгоритмы обработки цифровых изображений являются вычислительно трудоемкими. В работе рассматриваются два алгоритма обработки изображений: алгоритм медианной фильтрации и алгоритм увеличения изображения с использованием бикубической интерполяции. Алгоритмы реализованы с использованием технологии NVIDIA CUDA на графических процессорах.

Введение

При работе с цифровыми изображениями часто возникает задача удаления шумов, так как цифровое изображение вместе с полезным сигналом содержит, как правило, различные шумы. Один из видов шумов – это импульсный шум. Под импульсным шумом понимается искажение сигналов большими импульсными выбросами произвольных значений и малой длительности. Импульсные помехи в цифровом сигнале могут возникать из-за неправильного или ложного срабатывания одного пикселя цифровой матрицы, поврежденного участка памяти (при отсутствии контроля ошибок) или помех в каналах передачи сообщений [1]. Два основных типа импульсных шумов – это шумы типа «соль-перец» и случайные шумы. На изображении с шумом типа «соль-перец» поврежденные пиксели могут принимать только максимальное и минимальное значение в заданном динамическом диапазоне[1].

Существует множество методов удаления шумов на цифровых изображениях, обзор таких методов, например, приведен в работе [2]. Наиболее популярный нелинейный фильтр удаления импульсного шу-

[1]Динамический диапазон – характеристика устройства или системы, предназначенной для преобразования, передачи или хранения некой величины (мощности, силы, напряжения, звукового давления и т.д.), представляющая логарифм отношения максимального и минимального возможных значений величины входного параметра устройства (системы).

ма – это медианный фильтр, так как он достаточно хорошо удаляет шумы [1] и обладает хорошей вычислительной эффективностью [3].

Часто при работе с цифровыми изображениями возникает задача масштабирования изображения. Наиболее сложной при этом является задача увеличения изображения, при решении которой необходимо восстанавливать и дополнять промежуточные пиксели в изображении. Один из способов основан на использовании бикубической интерполяции.

Хотя все эти методы обладают достаточно большой вычислительной эффективностью, тем не менее остро стоит вопрос о повышении быстродействия, особенно если необходимо применять данные методы в системах реального времени, таких как дорожно-транспортные системы наблюдений, охранные системы наблюдений и др.

Одним из методов повышения скорости работы программ и алгоритмов является использование технологии GPGPU (General-purpose graphics processing units или «GPU общего назначения») [4]. Данная технология была предложена в качестве инструмента для параллельных вычислений компанией NVIDIA. В настоящее время она рассматривается в качестве эффективного средства разработки приложений для решения различных задач научно-исследовательского характера. Одна из реализаций GPGPU представлена технологией CUDA [5, 6]. CUDA (Compute Unified Device Architecture) – архитектура и программная модель для параллельных вычислений, которая позволяет производить расчёты на GPU NVIDIA со значительным ускорением [5, 6].

Медианный фильтр

Пусть $x_{i,j}$ для $(i,j) \in A = \{1,\ldots,M\} \times \{1,\ldots,N\}$ – значение уровня серого цвета в исходном цифровом изображении X размера $M \times N$ в пикселе с номером (i,j). $[s_{\min}, s_{\max}]$ – это динамический диапазон, т.е. $s_{\min} \le x_{i,j} \le s_{\max}$ для всех .

Обозначим через Y цифровое изображение с импульсным шумом. В классической модели импульсного шума типа «соль-перец» значение уровня серого цвета определяется следующим образом:

$$y_{i,j} = \begin{cases} s_{\min}, \text{ с вероятностью } p, \\ s_{\max}, \text{ с вероятностью } q, \\ x_{i,j}, \text{ с вероятностью } 1-p-q, \end{cases}$$

где $r = p + q$ определяет уровень шума. $(i, j) \in A$.

Пусть $S_{i,j}^{w}$ – это окно размером $w \times w$ с цетром в точке (i, j), т.е.

$$S_{i,j}^{w} = \{(k,l) : |k - i| \le w, |l - j| \le w\}.$$

Тогда алгоритм медианного фильтра можно представить следующим способом:

1) Фиксируем точку $(i, j) \in A$.
2) Формируем последовательность $K_{i,j} = \{x_{k,l} \in Y; (k,l) \in S_{i,j}^{w}\}$, состоящую из значений уровня серого цвета изображения Y.
3) Находим медиану[1] $x_{i,j}^{med}$ последовательности $K_{i,j}$ и полагаем значение в точке (i, j) фильтрованного изображения, равное $x_{i,j}^{med}$.
4) Если не прошли все точки изображения, то переходим к пункту 1.

Так как выборка обрабатываемых значений, попадающих в окно $S_{i,j}^{w}$, и нахождение медианы для всех точек изображения проводятся независимо, то такую фильтрацию можно проводить параллельно для всех точек.

При реализации параллельной обработки данных на GPU необходимо написать функцию, называемую ядром, которая будет выполняться всеми нитями параллельно. При реализации параллельного алгоритма медианной фильтрации в ядре были реализованы шаги 2 и 3 описанного выше алгоритма и, соответственно, каждая нить обрабатывала свой собственный пиксель исходного изображения с его окрестностью. На видеокартах используется несколько видов различной памяти, поэтому алгоритм был реализован с использованием глобальной и текстурной памяти. Результаты вычислений и времени выполнения на GPU для этих двух реализаций приводятся ниже.

Масштабирование изображения с использованием бикубической интерполяции

Пусть $x_{i,j}$ для $(i, j) \in A = \{1, \ldots, M\} \times \{1, \ldots, N\}$ – значение уровня серого цвета в исходном цифровом изображении X размера $M \times N$ в

[1]Медиана – одна из характеристик среднего для числового набора. По определению это такое число, что количество различных чисел набора, меньших данного, равно количеству различных чисел набора, больших данного.

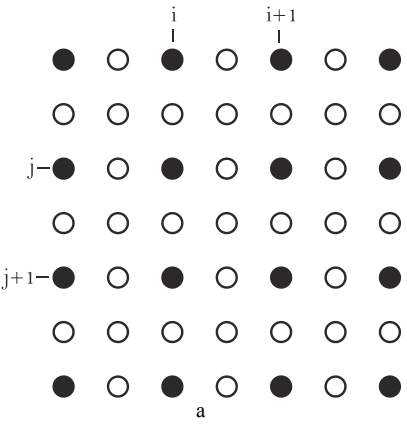
пикселе с номером (i, j). Допустим, необходимо увеличить изображение в два раза. Тогда, как показано на рис. 1, необходимо дополнительно рассчитать значения уровня серого цвета в точках, обозначенных незакрашенными кружками, расположенных между точками исходного изображения, обозначенных черными кружками.

Для расчета неизвестных значений использовалась бикубическая интерполяция. Сначала рассчитывались неизвестные значения по горизонтали с использованием кубической интерполяции по формуле (1). Затем с использованием аналогичной формулы проводились расчеты по вертикали. При расчетах в граничных точках использовались фиктивные точки.

$$p(t) = \frac{1}{2}\begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix}\begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix}\begin{bmatrix} x_{i-1,j} \\ x_{i,j} \\ x_{i+1,j} \\ x_{i+2,j} \end{bmatrix}. \qquad (1)$$

Расчет в разных точках при горизонтальной и вертикальной интерполяции проводится независимо, поэтому можно эффективно реализовать параллельную интерполяцию с использованием GPU. Сначала проводилась параллельно горизонтальная интерполяция, а после того как были вычислены все точки в нечетных строках – параллельно вертикальная интерполяция.

Для повышения эффективности параллельного алгоритма применялась поэтапная загрузка картинки из памяти CPU на GPU. Для этого изображение разбивалось на m полос, как показано на рис. 2. При таком разбиении горизонтальную интерполяцию можно осуществлять в разных полосах независимо. Для вертикальной же интерполяции, например в полосе 2, необходимо, чтобы уже была проведена горизонтальная интерполяция в 1-й и 3-й полосе, для того чтобы можно было провести интерполяцию в граничных узлах. В цикле, который выполнялся на CPU, было реализовано поэтапное копирование изображения с CPU на GPU, запуск ядер вертикальной и горизонтальной интерполяций на GPU осуществлялся в различных потоках в конкурирующем режиме с синхронизацией для правильного выполнения вертикальной интерполяции.



а

б

Рис. 1. Увеличение изображения в два раза с использованием бикубической интерполяции:

а — черными показаны точки, которые необходимо вычислить;

б — иллюстрация кубической интерполяции по 4 точкам

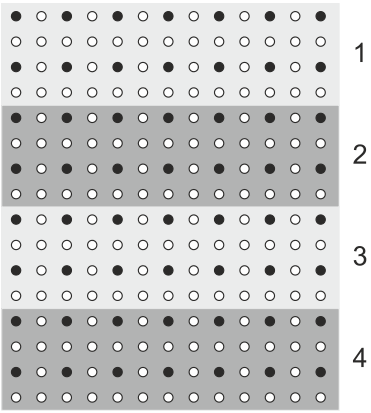


Рис. 2. Разбиение исходного изображения на несколько полос для поэтапного копирования и выполнения ядер

Результаты

Для проверки правильности параллельного алгоритма и программы, реализующей медианную фильтрацию, была сгенерирована тестовая картинка с шумом типа «соль-перец». На рис. 3 слева представлено изображение с шумом, а справа – отфильтрованное изображение.

Расчеты с использованием параллельного алгоритма проводились на компьютере под управлением ОС Windows 7 с видеокартой NVIDIA GTS 450 и на компьютере под управлением ОС Linux с видеокартами Tesla 1060 и Tesla 2070. В табл. 1 приведены результаты расчета на различных видеокартах и для изображений различной размерности.



а б

Рис. 3. Изображение с шумом типа «соль-перец» – *а*; *б* – отфильтрованное изображение с использованием параллельного медианного фильтра

Как видно из таблицы, максимальное ускорение при использовании окна фильтрации $w = 3$ получилось при расчетах с использованием текстурной памяти на видеокарте Tesla 2070. При использовании окна фильтрации размера 3 отличие в ускорении в зависимости от используемой памяти незначительно, более того, на видеокарте GTS 450 при использовании текстурной памяти происходит увеличение времени счета и как следствие ускорение падает.

Таблица 1. Время расчета и ускорение медианного алгоритма подавления шума

	CPU	GTS 450		Tesla 1060		Tesla 2070	
Размерность изображения (тип памяти)	Время на компьютере с Windows/Linux	Время, ms	Ускорение	Время, ms	Ускорение	Время, ms	Ускорение
Размер окна фильтрации $w = 3$							
512x512 (global)	95.71/ 94.98	3.65	26.2	3.04	31.24	2.43	39.08
512x512 (texture)	95.71/ 94.98	5.87	16.3	2.87	33.09	2.45	39.09
1600x1200 (global)	640.51/5 61.69	25.35	25.26	18.24	30.79	8.82	63.68
1600x1200 (texture)	640.51/5 61.69	28.00	22.86	18.92	29.68	8.69	64.63
Размер окна фильтрации $w = 5$							
512x512 (global)	506.44/ 364.26	4.32	117.04	3.65	99.79	2.11	172.31
512x512 (texture)	506.44/ 364.26	6.13	82.60	2.96	123.06	1.81	200.36
1600x1200 (global)	3395/ 2295	27.85	121.87	23.33	98.37	11.31	202.8
1600x1200 (texture)	3395/ 2295	26.06	130.24	17.59	130.47	8.76	261.9

Таблица 2. Время расчета и ускорение алгоритма увеличения изображения в два раза с использованием бикубической интерполяции

	CPU	GTS 450		Tesla 1060		Tesla 2070	
Размерность изображения (тип памяти)	Время на компьютере с Windows/Linux	Время, ms	Ускорение	Время, ms	Ускорение	Время, ms	Ускорение
Размер окна фильтрации $w = 3$							
512x512	35,04/ 29,49	1,32	26,54	1,08	27,23	1,04	28,35
1024x1024	211,25/ 191,09	6,74	31,34	2,05	44,31	3,16	60,47
2048x2048	900,38/ 790,13	25,43	35,40	13,48	66,79	8,72	90,61
4096x4096	4256,87/ 3904.92	104,38	40,78	60,68	70,15	33,78	115,59

При использовании окна фильтрации размера 5 для всех видеокарт ускорение дополнительно увеличивается в 4–5 раз, так как вычислительная сложность алгоритма повышается и более выражено проявляется эффект многопоточного выполнения программы на GPU.

В табл. 2 приведены время и ускорение для алгоритма увеличения изображения в два раза с использованием бикубической интерполяции. Наибольшего ускорения удалось достичь на самой производительной видеокарте Tesla 2070, ускорение составило 115.59. При использовании стандартной видеокарты GTS 450 тоже удалось достичь хороших результатов. Наибольшее ускорение на этой видеокарте удалось получить при увеличении изображения размерности 4096x4096.

Заключение

В ходе выполнения данной работы были реализованы два алгоритма с использованием технологии многопоточного программирования CUDA для графических видеокарт. Показано, что реализация алгоритмов для графических видеокарт и их использование оправданы, так как позволяют получить значительное ускорение времени выполнения расчетов. Использование специализированных графических процессоров, адаптированных для вычислительных алгоритмов, таких как Tesla 1060 и Tesla 2070, позволяет получить существенное ускорение по сравнению с последовательной программой.

Литература

1. **Bovik A.** Handbook of Image and Video Processing. – Academic Press, 2000.
2. **Astola J., Kuosmanen P.** Fundamentals of Nonlinear Digital Filtering. – Boca Raton, CRC, 1997.
3. **Huang T.S., Yang G.J., Tang G.Y.** Fast two-dimensional median filtering algorithm // IEEE Transactions on Acoustics, Speech, and Signal Processing. – 1979. – Vol. 1. – P. 13–18.
4. GPGPU.org: General-Purpose computation on Graphics Processing Units [Электронный ресурс]. – Электрон. дан. – URL: http://gpgpu.org/ (дата обращения: 25.10.2011).
5. CUDA [Электронный ресурс]. – Электрон. дан. – URL: http://www.nvidia.com/object/cuda_home_new.html (дата обращения: 25.10.2011).
6. CUDA: GPGPU.org [Электронный ресурс]. – Электрон. дан. – URL: http://gpgpu.org/developer/cuda (дата обращения: 25.10.2011).