Реализация фильтра самоподобия для графических процессоров

А.Н. Борисов[1], В.В. Мясников[1]

[1]*Самарский национальный исследовательский университет им. академика С.П. Королева, Московское шоссе 34а, Самара, Россия, 443086*

Аннотация

В статье рассматривается реализация одного из вариантов фильтра самоподобия, известного под названием foveated self-similarity, для графических процессоров. Данный вариант фильтра самоподобия использует особенности человеческого зрения при измерении степени различия фрагментов изображения. Фильтр был реализован с использованием NVIDIA CUDA 11 и OpenMP. Приводится сравнение производительности реализаций для центрального и графического процессоров для изображений различных размеров. Согласно полученным результатам, графический процессор NVIDIA GeForce RTX 3080 оказывается в 23 раза быстрее центрального процессора AMD Ryzen 3700X при фильтрации изображения размером 4096x4096 (4,6 с против 105,8 с).

Ключевые слова

Фильтр самоподобия, фильтрация шума, CUDA, nonlocal means, foveated self-similarity

1. Введение

Устранение шума – одна из типовых задач обработки изображений, для решения которой было предложено немало методов и алгоритмов, одним из которых является фильтр самоподобия [1]. Данный фильтр устраняет шум через усреднение пикселей, обладающих сходными окрестностями. При этом данный фильтр хорошо сохраняет границы и детали изображения.

Фильтр самоподобия построен на предположении о том, что пиксели со сходными окрестностями будут иметь близкие значения яркости, следовательно, шум может быть убран через усреднение значений яркости пикселей с «подобными» окрестностями (отсюда термин «самоподобие»). Очевидно, что для использования данного фильтра необходима функция $d(\mathbf{x},\mathbf{y})$, измеряющая «сходство» между окрестностями пикселей на позициях $\mathbf{x} = (m_1, n_1)$ и $\mathbf{y} = (m_2, n_2)$. В оригинальной работе в качестве такой функции используется поэлементное произведение евклидовой нормы разности окрестностей пикселей **x** и **y** на гауссово ядро.

В работе [2] предложен следующий способ вычисления расстояния:

$$d(\mathbf{x},\mathbf{y}) = \left\| FOV(W_{\mathbf{x}}) - FOV(W_{\mathbf{y}}) \right\| = \left\| W_{\mathbf{x}}^{FOV} - W_{\mathbf{y}}^{FOV} \right\|, \quad (1)$$

$$W_{\mathbf{x}}^{FOV}(\mathbf{u}) = [F_{\mathbf{x}+\mathbf{u}} * K_{\mathbf{u}}](0), \quad (2)$$

где $FOV(W)$ – оператор, имитирующий особенности поля зрения (foveation operator), $W_{\mathbf{x}}$ – некоторая окрестность пикселя **x**, $F_{\mathbf{x}+\mathbf{u}}$ – окрестность пикселя **x+u**, $K_{\boldsymbol{u}}$ – гауссово ядро с параметрами, зависящими от текущей позиции $\boldsymbol{u}$. Ядра $K_{\boldsymbol{u}}$ конструируются таким образом, что степень размытия $W_{\boldsymbol{x}}^{FOV}$ увеличивается от центра к краям, тем самым симулируя особенности реального зрения (полная процедура описана в [2]). Данное определение $d(\boldsymbol{x},\boldsymbol{y})$ позволяет существенно улучшить свойства фильтра –повысить качество фильтрации шума при лучшем, чем в [1], сохранении мелких деталей. Данный вариант фильтра самоподобия носит название Foveated Self-Similarity Filter (FOV-SSIM).

Со времени появления фильтра самоподобия было написано несколько реализаций данного фильтра для графических процессоров [3]-[4], демонстрирующих существенный прирост производительности. Однако, к настоящему моменту нет известных реализаций FOV-SSIM.

2. Реализация алгоритма

Структура фильтра такова, что каждый пиксель может обрабатываться независимо, следовательно, финальный этап алгоритма может быть неограниченно распараллелен. Однако вычисление дистанции между окрестностями пикселей требует ряда операций свертки с разными ядрами. Простым решением данной проблемы является предварительное вычисление свертки всего изображения с каждым ядром. Данный подход существенно увеличивает потребление памяти, что на самом деле не является существенной проблемой – изображение может быть разделено на тайлы меньшего размера, которые будут обработаны независимо, а затем собраны воедино.

Обработка изображения состоит из двух этапов. Первый этап – это свертка изображения с ядрами, количество которых зависит от параметров алгоритма. Промежуточные результаты свертки хранятся в разделяемой памяти графического процессора, для ускорения расчета. Помимо этого, для перекрытия чтения следующей порции данных и обработки текущей порции используется асинхронное копирование, а финальная фаза суммирования производится с помощью варп-синхронной редукции. Подробное описание можно найти в [5]. Результат первого этапа затем используется при вычислении расстояния между окрестностями пикселей.

3. Результаты экспериментов

Эксперименты проводились для случайных изображений различного размера. Во всех экспериментах размер окрестности пикселя равен 7x7, область поиска сходных окрестностей имеет размер 21x21, ядра свертки – гауссовы ядра размером 7x7. Результаты приведены в Таблице 1.

Таблица 1
Время обработки изображения

Размер изображения	Время ЦП, мс	Время ГП, мс	Ускорение
64x64	136	3	45,33
256x256	694	20	34,70
1024x1024	7 076	291	24,32
4096x4096	105 816	4 587	23,07

4. Литература

[1] Buades, A. A review of image denoising algorithms, with a new one / A. Buades, B. Coll, J. Morel // Multiscale Modeling and Simulation. – 2005. – Vol. 4(2). – P. 490-530. DOI: 10.1137/ 040616024.

[2] Foi, A. Foveated self-similarity in nonlocal image filtering / A. Foi, G. Boracchi // Proc. of SPIE. – 2012. – Vol. 8291. DOI: 10.1117/12.912217.

[3] de Fontes, F.P.X. Real-time ultrasound image denoising / F.P.X. de Fontes, G.A. Barroso, P. Hellier // Journal of Real-Time Image Processing. – 2011. – Vol. 6. – P. 15-22.

[4] Granata, D. Noise removal from remote sensed images by non local means with OpenCL algorithm / D. Granata, A. Palombo, F. Santini, U. Amato // Remote Sensing. – Vol. 12(3). DOI: 10.3390/rs12030414.

[5] CUDA Toolkit Documentation v11.2.0 [Электронный ресурс]. – Режим доступа: https://docs.nvidia.com/cuda/index.html (20.01.2021).