переделать логотип, так как он недостаточно передавал смысл приложения. Вскоре пришла идея использовать микрофон на фоне оранжевого круга с графиком амплитуды звуковой волны (рис. 6.2). Но оказалось, что график амплитуды создавал ненужный «шум» на заднем фоне и иконка все так же плохо передавала основной смысл приложения. Тогда на основе этого логотипа был сделан логотип на рисунке 6.3. Здесь уже избавились от «шумных» деталей и пришли к использованию карандаша, который символизирует заметки и что в них можно что-нибудь записать, в качестве ножки микрофона. Данное решение показалось достаточно интересным и поэтому именно этот логотип решили использовать как иконку приложения.

Рис. 6.1

Рис. 6.2

Рис. 6.3

Дальнейшие перспективы: публикация мобильного приложения в магазине приложений AppStore, добавление дополнительных функций, такие как синхронизация с облачными сервисами и возможность распознавания голоса, улучшение графического пользовательского интерфейса.

iOS-приложение с голосовыми заметками Notevox является простым в использовании и позволяет создавать и воспроизводить напоминания удобным для пользователя способом.

Исследованиеподдержанопроектом CERES. Centers of Excellence for young RESearchers (Reg.no. 544137-TEMPUS-1-2013-SK-JPHES),

CERES Co-funded by the Tempus Programme of the European Union

Список использованных источников:

1. Apple Developer:
   a. CoreDataProgrammingGuide [Электронный ресурс] — Режим доступа: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/index.html#//apple_ref/doc/uid/TP40001075-CH2-SW1 — Дата доступа: 19.01.2017
   b. AVFoundation [Электронный ресурс] — Режим доступа: https://developer.apple.com/reference/avfoundation?language=objc — Дата доступа: 29.01.2017
   c. Programming with Objective-C [Электронный ресурс] — Режим доступа: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html#//apple_ref/doc/uid/TP40011210-CH1-SW1 — Дата доступа: 15.01.2017
2. Wikipedia:
   a. Поток выполения [Электронный ресурс] — Режим доступа: https://ru.wikipedia.org/wiki/Поток_выполнения— Дата доступа: 03.04.2017
   b. GrandCentralDispatch[Электронный ресурс] — Режим доступа: https://ru.wikipedia.org/wiki/Grand_Central_Dispatch— Дата доступа: 03.04.2017
3. CocoaPods[Электронный ресурс] — Режим доступа: Guideshttps://guides.cocoapods.org— Дата доступа: 13.03.2017
4. [Электронный ресурс] — Режим доступа: https://github.com/MortimerGoro/MGSwipeTableCell— Дата доступа: 13.01.2017
5. Apple inc. «Apple Human Interface Guidelines»
6. Stanford CS193p Developing Applications for iOS Fall 2013-14 (Objective-C)
7. Stanford CS193p Developing Applications for iOS Winter 2017 (Swift)

ФРАКТАЛЬНОЕ СЖАТИЕ ИЗОБРАЖЕНИЙ И ЕГО УСКОРЕНИЕ С ПОМОЩЬЮ GPU

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Шевченя А.В.

Саттарова П.С – м. т. н., ассистент

«Информация — это не материя и не энергия, информация — это информация» - сказал выдающийся математик и философ Норберт Винер [1]. Сегодня информация является одним из важнейших ресурсов. Согласно исследованиям, проведенным в 2011 году группой исследователей из Университета Южной Калифорнии под руководством Мартина Гилберта, в 2007 году человечество было способно хранить около 290 эксабайт информации. В то же время, объём передаваемой информации различных видов составлял 2 зетабайта [2]. Эти цифры позволяют оценить масштабность мировых процессов передачи данных различных видов. Вполне обоснованно встает вопрос оптимального способа хранения информации и передачи ее с наименьшими затратами. В ходе различных исследований [3] было выяснено, что значительная часть информации, которая хранится в электронном виде, обладает той или иной избыточностью и может быть преобразована таким образом, чтобы занимать меньший объём памяти. Получается, что хранение информации в исходном виде неэффективно. В связи с этим разрабатываются различные алгоритмы сжатия данных, позволяющие сократить необходимый для хранения объем памяти. В данной работе речь пойдет об алгоритме фрактального сжатия изображений и его ускорении за счет использования GPU.

Фрактальное сжатие изображений — алгоритм сжатия изображений с потерями, основанный на применении к ним систем итерируемых функций. Алгоритм обладает высокой вычислительной сложностью, однако при этом для некоторых видов изображений (реальные изображения природы) позволяет получить высокую степень сжатия (может достигать 100:1 [4]) при сохранение приемлемого качества. Идея сжатия основана на использовании системы итерируемых функций (СИФ, Iterated Function System или IFS), представляющей собой сжимающееся отображение, многократное применение которого приводит объект в «стабильное» состояние. Возможность использования теории СИФ к проблеме сжатия изображения была исследована М. Барнсли [5]. Однако подход Барнсли не позволял решать необходимую задачу, т. к. рассматривал подобие полного изображения с его частями. Основной изъян его подхода заключался в том, произвольные изображения, в отличие от фракталов, не самоподобны. Решение нашёл А. Жакен в 1992 году. Его идея заключалась в том, что самоподобие будет искаться не между целым изображением и его частями, а между различными частями изображения. Упрощенно алгоритм можно описать следующей последовательностью действий:

1. Изображение делится на небольшие неперекрывающиеся блоки-квадраты — ранговые блоки.
2. Строится пул всех возможных перекрывающихся блоков с линейными размерами в два раза больше размера рангового - доменные блоки.
3. Для каждого рангового блока по очереди «применяются» доменные блоки и ищется подходящее преобразование, при котором разница между ранговым и доменным блоком будет минимальна.
4. Наиболее подходящие доменный блок и соответствующее преобразование ставятся в соответствие ранговому блоку.

Для доменных блоков обычно применяются следующие аффинные преобразования: поворот на 90, 180, 270 градусов, отражение по горизонтали или вертикали, отражение относительно диагоналей. Помимо этого, доменный блок подвергается сжатию до размеров рангового блока путем усреднения значений яркости блоков по 4 пикселя.

Сами по себе все вышеописанные преобразования не гарантируют, что отображение будет сжимающимся. Для достижения этого необходимо вычислить коэффициенты преобразования пикселей доменного блока в пиксели рангового таким образом, чтобы среднеквадратичная ошибка была минимальной, а коэффициенты удовлетворяли определённым условиям.

Процесс декодирования в свою очередь относительно прост. Задается изображение с любым содержанием (чаще всего это изображение, равномерно заполненное серым цветом), к его доменным блокам применяются описанные выше преобразования. Гарантируется, что в результате нескольких последовательных итераций полученное изображение приближается к исходному.

Для оценки качества полученного изображения используем метрику SSIM (Structure SIMularity, индекс структурного сходства). Отличительной особенностью метрики является то, что метод учитывает «восприятие ошибки» благодаря учёту структурного изменения информации. Идея заключается в том, что пиксели имеют сильную взаимосвязь, особенно когда они близки пространственно [11].

Для реализации алгоритмы были выбраны язык программирования Python и фреймворк OpenCL. Выбор пал на эти технологии из-за простоты работы с ними.

Сначала алгоритм был реализован только на Python. Данная реализация была использована в качестве эталонной и в последующих сравнениях производительности её результаты работы принимались как базовые.

Затем был реализован алгоритм с использованием OpenCL. Оценка производительности осуществлялась для двух вариантов алгоритма: с ограничением параметров преобразования пикселей (Параллельный алгоритм 1) и без ограничения (Параллельный алгоритм 2).

Как итог реализации алгоритма, были получены следующие результаты:

Изображение	Последовательный алгоритм	Параллельный алгоритм 1	Параллельный алгоритм 2
256x256 Lena	6003	74	35
512x512 Nature	>36000	1083	359

Таблица 1 - Результат работы алгоритма, с

Стоит отметить, что не было получено точного результата измерений для изображения Nature, т.к. последовательный алгоритм работал более 10 часов, а затем был принудительно остановлен.

Результаты сжатия с помощью данного метода, а также с помощью метода JPEG:

Изображение	Исходный размер	JPEG	Последовательный алгоритм	Параллельный алгоритм 1	Параллельный алгоритм 2
256x256 Lena	196662	11667	31312	30198	31332
512x512 Nature	786486	60165	N/A	181104	182973

Таблица *1* - Результаты сжатия, байт

Качество изображений, полученных при декомпрессии сжатых изображений, разнится в зависимости от изображения, однако остается на достаточно высоком уровне. Оценка проводилась как визуально, так и с помощью метрики SSIM. Результаты метрики:

Метод	256x256 Lena	512x512 Nature
Параллельный алгоритм 1	0.962	0,978
Параллельный алгоритм 2	0.954	0.979
Последовательный алгоритм	0.961	0.977

Таблица *2* - Результаты метрики SSIM

В результате работы получено программное обеспечение, позволяющее ускорить работу алгоритма в 170 раз. Однако полученные результаты всё же не позволяют говорить о том, что данный метод можно использовать в повседневной деятельности, где необходима возможность постоянно сжимать и восстанавливать изображения. Так же не удалось достичь степени сжатия на уровне JPEG. Выделяя направления для улучшения, можно отметить, что реализация на языке более низкого уровня (C/C++) позволила бы выполнять преобразования быстрее, т.к. Python является относительно медленным в плане выполнения языком. Однако негативным фактором в данном случае является многословность запуска, вызова и использования OpenCL в этих языках. Также улучшениям может подвергнуться часть, написанная с помощью OpenCL. В частности, можно попытаться использовать __local память или текстурную память. Помимо этого, улучшению можно подвергнуть и сам алгоритм. К примеру, большое количество исследований в области фрактального сжатия проводились в рамках изучения наилучших стратегий поиска доменных блоков. Существует несколько вариантов поиска, которые могут в различной степени ускорить алгоритм.

Список использованных источников:

1. Винер, Н. Кибернетика, или управление и связь в животном и машине; или Кибернетика и общество/ 2-е издание. - М.: Наука; Главная редакция изданий для зарубежных стран, 1983. - 344 с.
2. Hilbert, M. & Lopez, P. The world’s technological capacity to store, communicate, and compute information. / M. Hilbert // Science. - 2011. №332(6025). - P. 60–65.
3. Berger, T. Rate distortion theory: A mathematical basis for data compression / Berger T. – NJ: Prentice-Hall, Englewood Cliffs, 1971
4. Алгоритмысжатияданных [Электронныйресурс]. - Электронныеданные. - Режимдоступа: https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2#.D0.90.D0.BB.D0.B3.D0.BE.D1.80.D0.B8.D1.82.D0.BC.D1.8B.D1.81.D0.B6.D0.B0.D1.82.D0.B8.D1.8F_.D0.B4.D0.B0.D0.BD.D0.BD. D1.8B.D1.85
5. Barnsley, M. Fractals Everywhere / Michael Barnsley – San Diego: Academic Press, 1988
6. Fisher, Y. Fractal Image Compression – Theory and Application / Y. Fisher – N.Y.: Springer Verlag, 1994. – 341 p.
7. Kominek, J. Advances in Fractal Compression for Multimedia Applications / J.Kominek // Multimedia Systems. - 1997. - №4. - P. 255-270.
8. Encarta [Электронный ресурс]. - Электронные данные. - Режим доступа: http://www.britannica.com/topic/Encarta
9. Python [Электронный ресурс]. - Электронные данные. - Режим доступа: https://ru.wikipedia.org/wiki/Python
10. OpenCL [Электронный ресурс]. - Электронные данные. - Режим доступа: https://ru.wikipedia.org/wiki/OpenCL
11. SSIM [Электронный ресурс]. - Электронные данные. - Режим доступа: https://ru.wikipedia.org/wiki/SSIM

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДЛЯ АНАЛИЗА СОВМЕСТИМОСТИ И ПОИСКА АНАЛОГОВ ЛЕКАРСТВЕННЫХ СРЕДСТВ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь