Расширенное микширование аудиопотоков для гетерогенных компьютерных систем в телекоммуникациях

А.А. Колпаков[a], Ю.А. Кропотов[a]

[a] *Муромский институт (филиал) ВлГУ, 602264, ул. Орловская, 23, Муром, Владимирская область, Россия*

Аннотация

В работе рассмотрены вопросы расширенного микширования аудиопотоков для их обработки на графических процессорах, в которых комбинируются множество потоков путем использования двухпроходного рендера, что существенно снижает время переключения между буферами. Методом экспериментальных компьютерных сравнительных исследований было осуществлено оценивание производительности разработанного алгоритма. Целью работы является разработка эффективного алгоритма микширования аудиопотоков для обработки на графических процессорах. Результаты исследований показали, что применение разработанного алгоритма приводит к существенному увеличению производительности вычислений. Представленное решение предполагается реализовать в виде программного обеспечения многопроцессорных устройств в системах телекоммуникаций.

Ключевые слова: двухпроходной рендер; алгоритм повышения производительности; параллельные вычисления; гетерогенные вычислительные системы; графические процессоры; микширование аудио данных.

1. Введение

С развитием современных компьютерных сетей особенно актуальным становится разработка информационно-телекоммуникационных систем цифровой внутриобъектовой оперативно-командной, громкоговорящей и телефонной связи на их основе. Для обеспечения оптимальной производительности при проектировании таких систем оправдано использование панельных или планшетных компьютеров на основе готовых решений. Поскольку голосовое общение играет важную роль, режим голосовых конференций с несколькими участниками является в таких системах основным.

Простейший сценарий организации голосовой коммуникации заключается в том, что каждый источник звука шлет свой аудиопоток каждому приемнику независимо. Такой метод прост и удобен, но требует высокой пропускной способности сети, что не всегда можно обеспечить. Поэтому лучшим методом может служить микширование аудио, что означает комбинирование аудиопотоков от каждого источника в один. Исходя из возможности звуковых волн накладываться друг на друга, данный метод может обеспечивать приемлемое качество звука, при этом обеспечивая снижение загруженности сети. Однако, применение данного метода может существенно увеличить нагрузку на центральный процессор сервера, что может негативно сказываться на производительность системы в целом. Выходом из данной ситуации может стать применение графических процессоров для решения данной задачи [1].

2. Проблемы использования графических процессоров для аудио микширования

Хотя графические процессоры достаточно производительны, существует несколько проблем в использовании их для аудио микширования, которые связаны с архитектурой и ограниченностью функционала [2].

Первая проблема – это пропускная способность шины между графическим процессором и основной памятью, которая меньше, чем между основным процессором и основной памятью. Например, чипсет Intel 975X обеспечивает теоретическую пропускную способность для CPU 10.7 ГБ/с, а для GPU только 8 ГБ/с. Практика показывает, что отсутствие поддержки асинхронного ввода/вывода требует больших временных затрат для дополнительных операций, таких как блокировка/разблокировка буфера. Поскольку общие вычисления на GPU базируются на 3D рендеринге, скорость записи обычно выше, чем скорость чтения. Такая асимметрия делает процедуру считывания результата достаточно продолжительной [3, 4].

Во-вторых, общие вычисления на GPU базируются на 3D моделях, различные задачи требуют различных настроек GPU, таких как 3D модели, трансформирующие матрицы и программы шейдеров. Во время загрузки настроек вычислительные потоки GPU не задействованы. Хуже всего то, что GPU не сообщает CPU о завершении выполнения задания, поэтому CPU вынужден периодически проверять статус GPU. Это довольно времязатратная операция, так как она нарушает параллелизм между GPU и CPU.

В-третьих, недостатком GPU является производительность в логических операциях. Как известно, CPU отслеживает ветвления, GPU же работает по-другому: каждая ветка ветвления сначала выполняется, а потом уже выбирается нужный результат. Это делает распараллеливание легче, но требует больше ресурсов.

И наконец, набор инструкций GPU несовместим с CPU. Кроме того, время выполнения и длина кода лимитированы. Все это делает сложным перенос существующих алгоритмов на графические процессоры [5].

3. Структура разрабатываемого алгоритма

Базовый алгоритм аудио микширования состоит из пяти этапов. Первый шаг – это суммирования аудио семплов от различных источников, что может быть представлено следующей формулой [6,7]:

$$\overrightarrow{M}_t = \sum_{k=0}^{n} \vec{u}_{k,t}, \quad (1)$$

где $\vec{u}_{k,t}$ – вектор семпла, т.е. вектор отсчетов, полученных микрофоном k за время t;

$\overrightarrow{M}_t$ – итоговый вектор микширования.

Второй этап – это эхокомпенсация, которая в базовом виде заключается в исключении семпла i-го устройства из итогового вектора [8]. Данный этап представляется в виде

$$\overrightarrow{M}_{i,t} = \overrightarrow{M}_t - \vec{u}_{i,t}, \quad (2)$$

где $\overrightarrow{M}_{i,t}$ – итоговый вектор микширования для i-го устройства;

$u_{i,t}$ – семпл i-го устройства.

Для корректности итогового вектора $\overrightarrow{M}_{i,t}$ необходимо, чтобы его размерность была равна размерности входящих векторов. Однако, после проведения этапов 1 и 2 вектор $\overrightarrow{M}_{i,t}$ может быть переполнен, что приведет к нежелательным шумам [9]. Для того, чтобы использовать изначально вектор $\overrightarrow{M}_{i,t}$ больших размеров, необходимо проводить его сжатие для дальнейшего использования. Это делается на третьем этапе по формуле

$$\overrightarrow{M}_{i,t} = \overrightarrow{M}_{i,t} \times frac_{i,t}, \quad (3)$$

где $frac_{i,t}$ – коэффициент ослабления для i-го устройства. Этот коэффициент необходимо вычислять автоматически, исходя из максимального сжатого семпла. Поиск максимума среди сжатых семплов происходит на четвертом этапе, а корректировка коэффициента ослабления – на пятом.

Блок-схема базового алгоритма микширования представлена на рис. 1.

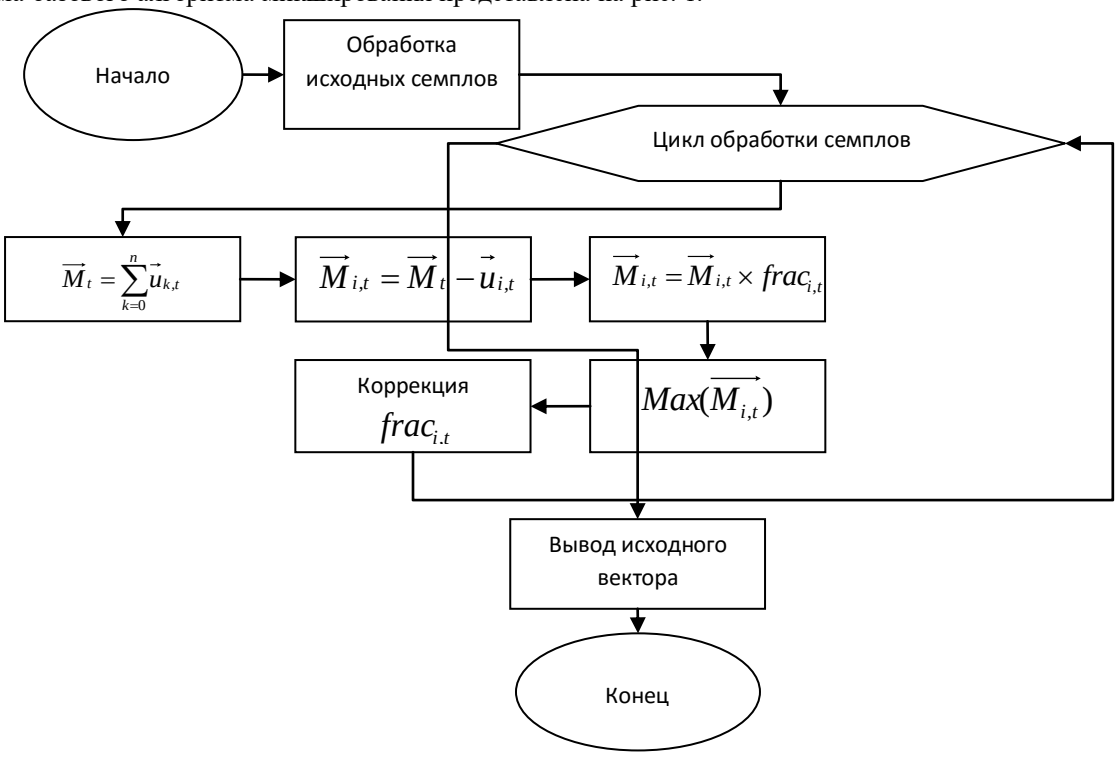

Рис. 1. Блок-схема базового алгоритма микширования.

Для лучшего описания алгоритма, возможности GPU представляются в виде f – функции текстуры X и координат пикселя, представленной формулой

$$f(X,P) = \{y_i \mid \forall p_i \in P, y_i = f(X, p_i)\}, \quad (4)$$

где y_i – проекция пикселя p_i на текстуру X,

P – проекция 3D-модели.

Для каждого пикселя в проекции 3D модели P будет рассчитана функция (4) и затем результат будет записан в буфер рендера.

Из формулы (4) видно, что образец вычислений на GPU может быть представлен как $A=(X,P,f)$.

Пусть *n* обозначает общее количество источников звука в сессии, а *L* – длину одного аудио семпла. Каждый из трех первых шагов микширования (накопление семплов, эхокомпенсация и сжатие) выдает *n* последовательностей по *L* байт. В то же время два последних шага (поиск максимального семпла и адаптация коэффициента затухания) выдают только n целых чисел. Поскольку первые три шага могут быть выполнены в рамках одной проекции для вычислений на GPU, они могут быть скомбинированы в один шаг, как показано ниже

$$\vec{m}_i = \left(\sum_{j=0}^{n-1} \vec{u}_j - \vec{u}_i \right) \times frac_i . \quad (5)$$

Поскольку четвертый шаг использует другое измерение, отличное от первых трех, он не может быть объединен в рамках формулы (5). Блок-схема расширенного алгоритма микширования представлена на рис. 2.

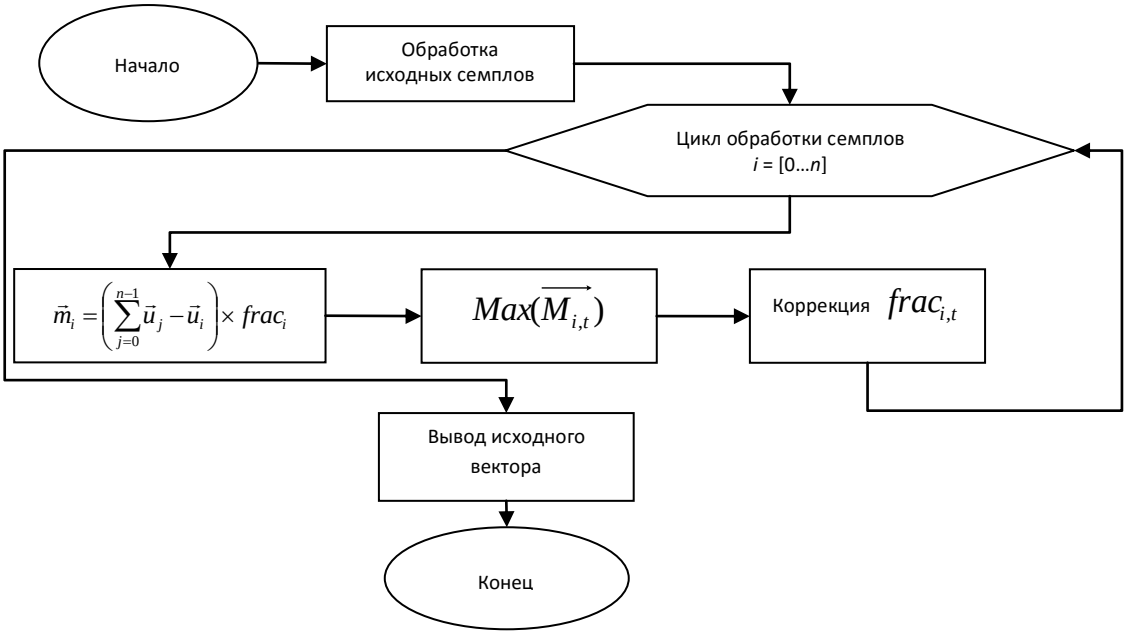

Рис. 2. Блок-схема расширенного алгоритма микширования.

Обычно вычисления, проекции которых не совпадают и не имеют пересечений, могут быть объединены по общим признакам. Поэтому если эти вычисления обозначить как *A1=(Xl,PI,f1)* и *A2=(X2,P2,f2)*, то их можно объединить, как показано ниже.

$$X = \langle X_1, X_2 \rangle, P = P_1 \cup P_2,\ f(\langle X_1, X_2 \rangle, p) = \begin{cases} f_1(X_1, p), p \in P_1, \\ f_2(X_2, p), p \in P_2. \end{cases} \quad (6)$$

Как видно из (6) основным алгоритмом является проверка координаты каждого пикселя для выбора функции выполнения модуля. Так как GPU выполняет все ветви до выбора нужной, каждая ветвь будет выполнена для каждого пиксела, что займет много времени.

В разрабатываемом алгоритме представлен альтернативный метод выполнения большого количества функций, который выводит в один буфер рендера выходные данные различной длины с помощью многопотокового рендеринга. Здесь основным правилом является перемещение проекции путем модифицирования проекционной матрицы, чтобы вычислительная площадь каждой функции ограничивалась необходимой областью, а не всем буфером.

В зависимости от пикселя на него может приходиться разный объем информации без потери универсальности. Обозначим через *w* общее число пикселей, необходимое для хранения *L* байт. Таким образом, буфер рендера может быть представлен как (*w+1*)*n*. 3D модель разрабатываемого алгоритма представляет собой прямоугольник, который лежит на плоскости *Z*. Он имеет размеры: *2w/*(*w+1*) единиц по ширине и 2 единицы по высоте. Координаты вершин – (-1,-1,0), (1-2/(*w*+1),-1,0), (-1,1,0), (1-2/(*w*+1),1,0).

При первом проходе рендеринга в качестве матрицы проецирования выбирается единичная матрица. Это обеспечивает проекции совпадение с 3D моделью. После преобразования видимой области, в этом проходе для выполнения первых трех шагов микширования применяется формула (5). Преобразования 3D-модели, производимые в первых трех шагах, представлены на рис. 3.

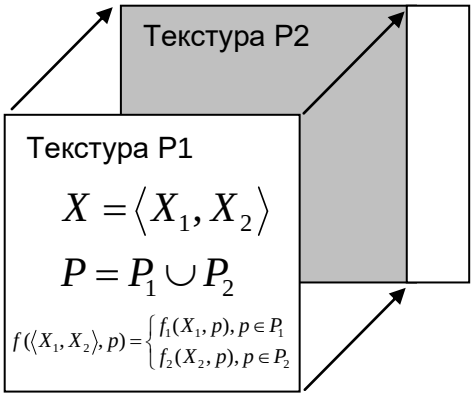

Рис. 3. Первый проход алгоритма.

Во втором проходе рендеринга проекция смещается влево на *L* пикселей. В то же время программа шейдера переключается в режим поиска максимального семпла. В этом проходе может быть записана только одна колонка, поскольку большинство частей проекции лежат вне буфера рендера и будут автоматически игнорированы GPU. Так как отсечение проекции было выполнено в начале рендера, этот метод вызывает функцию только для корректных пикселей, а не для всего буфера. Действия, производимые на втором проходе алгоритма, представлены на рис. 4.

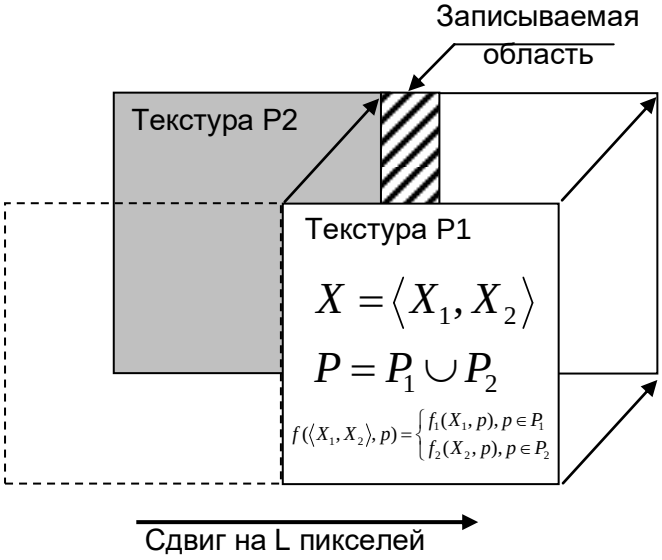

Рис. 4. Второй проход алгоритма.

4. Использование алгоритма с одной текстурой

Как замечено выше, оба прохода алгоритма в качестве входных данных требуют n последовательностей семплов каждый. Для каждой последовательности должна быть выделена своя уникальная текстура. Всего требуется n текстур размерностью L. Такая многотекстурная технология не подходит для аудио микширования. Во-первых, для каждого источника звука требуется своя текстура. Поэтому могут потребоваться дополнительные проходы. Во-вторых, загрузка множества маленьких текстур гораздо медленнее, чем загрузка одной большой.

В разрабатываемом алгоритме предлагается загрузка единой текстуры. В качестве примера представлен первый проход алгоритма. Входные данные содержат n семплов последовательностей размером L байт и n коэффициентов затухания. Используется два текстурных буфера формата RGBA. Текстура T1 имеет размерность [L/4]*n и хранит все семплы последовательностей в линию. Текстура T2 имеет размерность 1*n и хранит коэффициенты затухания. Координаты всех текстур приведены в таблице 1.

Таблица 1. Координаты текстур, используемых в разработанном алгоритме

Вершины	Координаты текстуры T1	Координаты текстуры T2
(-1, -1, 0)	(1/2w, 1)	(0.5, 1)
(1-2/(w+1), -1, 0)	(1, 1)	(0.5, 1)
(-1, 1, 0)	(1/2w, 0)	(0.5, 0)
(1-2/(w+1), 1, 0)	(1, 0)	(0.5, 0)

Аудио микширование производится независимо для каждого пикселя. Текстурные координаты пикселя (x,y) рассчитываются путем интерполяции текстурных координат вершин. Исходя из таблицы 1, текстурная координата для

T1 или pt.t0 должна быть (X,Y), а для T2 или pt.t1 – (0.5,Y). Pt.t0 указывает на семпл, для которого производятся текущие преобразования микширования, данный семпл назовем «точкой прицеливания». Для исключения появления эха другая текстурная координата ptCur ограничена по доступу T1 вместо pt.t0. Составляющая x текстуры ptCur идентична текстуре pt.t0, а составляющая y рассчитывается из циклической переменной, которая обозначает каждый источник звука. В цикле регистр положения v используется, чтобы пропустить «точку прицеливания». Наконец, коэффициенты затухания считываются из текстуры pt.t1. Поскольку коэффициент хранится в первом байте, выборка производится только по синей составляющей.

5. Экспериментальное исследование разработанного алгоритма

Тестовые входные данные для микширования представляют собой последовательности по 320 семплов. Все семплы сгенерированы случайно. В качестве тестового стенда использован компьютер с процессором Intel Core i3-4130, 4 ГБ оперативной памяти, графическая карта NVIDIA GeForce GT730. Варьировалось количество последовательностей *M*. Результаты для базового и разработанного алгоритмов на выходе идентичны. Результаты экспериментального исследования зависимости времени выполнения микширования *t* от количество последовательностей *M* приведены на рис. 5.

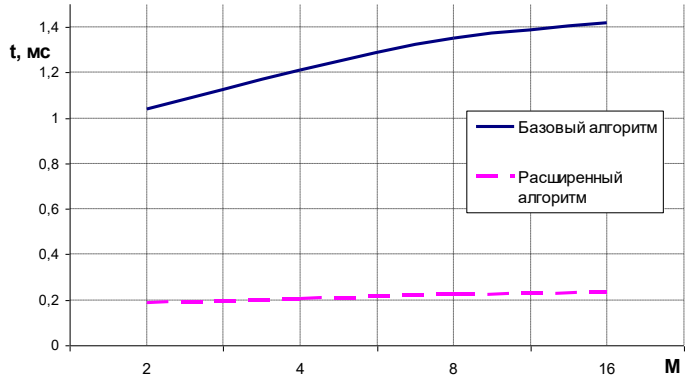

Рис. 5. Результаты экспериментального исследования зависимости времени выполнения микширования *t* от количество последовательностей *M*.

Как видно из результатов тестирования, представленных на рис. 5, применение расширенного алгоритма аудио микширования позволяет увеличить производительность компьютерной системы в 5-6 раз [10, 11].

В таблице 2 приведены результаты измерения среднего времени работы алгоритма *t* для 8 случайных последовательностей по 320 семплов. В ходе измерения было проведено 1000 циклов обработки, для каждого из которых генерировались новые случайные последовательности семплов. Для полученных результатов рассчитана дисперсия выходных последовательностей, значения которой также приведены в таблице 2.

Таблица 2. Результаты исследования алгоритма расширенного микширования аудиопотоков для 8 последовательностей семплов

Применяемый алгоритм	Среднее время работы алгоритма, мс	Дисперсия выходных последовательностей
Базовый алгоритм	1,351	3,757
Разработанный алгоритм	$2{,}226 \times 10^{-1}$	$1{,}426 \times 10^{-3}$

Как видно из результатов тестирования, представленных в таблице 2, дисперсия выходных последовательностей для базового алгоритма существенно выше, чем для разработанного [12,13]. Это связано с тем, что для вычислений в графическом процессоре применяются 32-битные числа, тогда как для вычислений на центральном процессоре – 64-разрядные. Применение разработанного алгоритма в гетерогенной компьютерной системе уменьшает время на обработку данных до $0{,}2226 \times 10^{-3}$ с вместо $1{,}351 \times 10^{-3}$ с – временем обработки данных базовым алгоритмом.

6. Заключение

В данной работе представлен алгоритм расширенного микширования аудиопотоков для вычислений на графических процессорах. Главное его достоинство – это комбинирование множества этапов микширования путем использования двухпроходного рендера, что существенно снижает время переключения между буферами. Использование для расчетов одной текстуры повышает эффективность операций ввода/вывода. Хотя операции ввода/вывода занимают приблизительно половину времени вычислений, экспериментальные исследования разработанного алгоритма показали увеличение производительности до 6 раз.

Литература

[1] Lindholm, E. NVIDIA Tesla: A unified graphics and computing architecture / E. Lindholm, J. Nickolls, S. Oberman, J. Montrym// IEEE Micro – 2008. – Vol. 28(2). – P. 39-55.

[2] Luebke, D. GPGPU: general purpose computation on graphics hardware / D. Luebke, M. Harris, J. Kruger, T. Purcell, N. Govindaraju, I. Buck, C. Woolley, A. Lefohn// In SIGGRAPH '04: ACM SIG-GRAPH – 2004 Course Notes – New York, NY, USA. – 2004. – p. 33. DOI: 10.1145/1103900.1103933

[3] Колпаков, А. А. Аспекты оценки увеличения производительности вычислений при распараллеливании процессоров вычислительных систем / А.А. Колпаков // Методы и устройства передачи и обработки информации. – 2011. – №1(13). – С. 124-127.

[4] Колпаков, А. А. Теоретическая оценка роста производительности вычислительной системы при использовании нескольких вычислительных устройств / А.А. Колпаков // В мире научных открытий. –2012. – №1. – С. 206-209.

[5] Колпаков, А. А. Оптимизация генетических алгоритмов при использовании вычислений на графических процессорах на примере задачи нулевых битовых векторов / А.А. Колпаков // Информационные системы и технологии. –2013. – №2(76). – С. 22-28.

[6] Кропотов, Ю. А. Экспериментальные исследования закона распределения вероятности амплитуд сигналов систем передачи речевой информации / Ю.А. Кропотов // Проектирование и технология электронных средств. –2006. – Т.4. – С. 37-42.

[7] Кропотов, Ю. А. Статистические параметры сигналов при проектировании оперативно-командных телекоммуникационных систем / Ю.А. Кропотов // В мире научных открытий. –2010. – №6-1. – С. 39-44.

[8] Кропотов, Ю. А. Аппроксимация закона распределения вероятности отсчетов сигналов акустических помех / Ю.А. Кропотов, А.А. Быков // Радиотехнические и телекоммуникационные системы. –2011. – №2. – С. 61-63.

[9] Кропотов, Ю. А. О корреляционном оценивании параметров моделей акустических эхо-сигналов / В.А. Ермолаев, Ю.А. Кропотов // Вопросы радиоэлектроники. –2010. – Т.1 №1. – С. 46-50.

[10] Кропотов, Ю. А. Модели, алгоритмы системы автоматизированного мониторинга и управления экологической безопасности промышленных производств / Ю.А. Кропотов, А. Ю. Проскуряков, А. А. Белов, А.А. Колпаков // Системы управления, связи и безопасности. –2015. – №2. – С. 184-197.

[11] Кропотов, Ю. А. Методы проектирования телекоммуникационных информационно–управляющих систем аудиообмена в сложной помеховой обстановке / Ю.А. Кропотов, А. А. Белов, А. Ю. Проскуряков, А.А. Колпаков // Системы управления, связи и безопасности. –2015. – №2. – С. 165-183.

[12] Kropotov, Ju.A. Identification of Models for Discrete Linear Systems with Variable, Slowly Varying Parameters / V.A. Ermolaev, V.T. Eremenko, O.E. Karasev, Ju.A. Kropotov //Journal of Communications Technology and Electronics. – 2010. – vol. 55, № 1. – P. 52-57.

[13] Kropotov, Ju.A. Algorithms for processing acoustic signals in telecommunication systems by local parametric methods of analysis [Electronic resource]/ V.A. Ermolaev, Ju.A. Kropotov // 2015 International Siberian Conference on Control and Communications (SIBCON) – Proceedings. – 2015. – Access mode: http://ieeexplore.ieee.org/document/7147109/