РЕФЕРАТ

Выпускная квалификационная работа магистра: 66 с., 39 рисунков, 15 таблиц, 25 источников, одно приложение.

Презентация: 13 слайдов Microsoft PowerPoint.

НЕЛИНЕЙНАЯ ФИЛЬТРАЦИЯ ИЗОБРАЖЕНИЙ, РЯД ВОЛЬТЕРРА, ПОЛИНОМ КОЛМАГОРОВА-ГАБОРА, ПАРАМЕТРИЧЕСКАЯ ИДЕНТИФИКАЦИЯ, ТЕХНОЛОГИЯ CUDA

Объектом исследования являются изображения подвергнутые динамическим искажениям вида дефокусировки или смаза.

Цель работы – создание нелинейного фильтра с линейными неизвестными коэффициентами и исследовать качество применения нелинейных фильтров.

Разработана программная реализация построенного нелинейного фильтра. На тестовых изображениях проведён ряд экспериментов по использованию различных нелинейных фильтров; и получены сравнительные результаты.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Если известен оператор искажающей системы, задача построения фильтра для восстановления изображений сводится к нахождению некоторого приближения к обратному оператору. Часто оператор системы оказывается неизвестным или известен неточно, а вместо этого известными являются тестовые изображения или фрагменты на искаженном изображении, эталонные функции распределения яркости на которых могут быть заданы с использованием априорной информации. В этом случае параметры фильтра могут быть определены путем непосредственной идентификации инверсного тракта формирования изображений [1].

Концепция оптимальной линейной фильтрации до недавнего времени имела преобладающее значение. Подход, основанный на решении задачи идентификации линейных моделей, в том числе по малым фрагментам изображений, рассматривался в работах [2,3]. Опыт использования линейных моделей показывает, что при использовании моделей в классе КИХ-фильтров, размерность задачи идентификации при интенсивных искажениях должна быть большой, что приводит к ухудшению обусловленности задачи. Попытка улучшения обусловленности применением более грубой сетки отсчетов приводит к потере качества. Применение моделей БИХ-фильтров снимает проблему размерности, однако при этом возникает серьезная проблема обеспечения устойчивости.

Поэтому надежды на повышение качества восстановления изображений, не без оснований, связывают с построением нелинейных фильтров. В значительной мере это связано с тем, что реальные системы формирования изображений действительно чаще характеризуются нелинейными искажениями и априорной неопределенностью математического описания и информации, как о самой системе, так и помехах [4].

Для решения задачи идентификации нелинейных моделей разработано много подходов и методов [5]. Одним из конструктивных подходов является

использование моделей в виде последовательности Вольтерра. В частности, Винер показал, что функциональный ряд Вольтерра может быть использован для описания систем, в которых нелинейность не слишком существенна. Опираясь на этот результат предпринимаются попытки использования рядов Вольтерра для оценки и идентификации нелинейных систем [6]. Основная проблема, с которой приходится при этом сталкиваться, это сложность определения ядер ряда. Поэтому на этом пути пока не достигнуто значительных успехов.

Цель настоящей работы разработка и исследование процедур идентификации и последующей реализации линейных по параметрам нелинейных фильтров в варианте, приводящем к формальному описанию в виде степенного ряда.

1 АНАЛИЗ ЗАДАЧИ ПОСТРОЕНИЯ НЕЛИНЕЙНОГО ФИТЛЬТРА

В данной работе в качестве построения восстанавливающего фильтра была взята за основу задача идентификации. Смысл данной задачи заключается в построение оптимальной или подходящей в некотором смысле модели системы (объекта) по результатам наблюдений входных и выходных переменных этого объекта. То есть обратный искажающий оператор определяется путем идентификации параметров модели искажающей системы по специально сформированным для этой цели тестовым изображениям [9].

Линейная фильтрация очень широко используется при устранении шумов на изображениях. Линейные КИХ-фильтры достаточно эффективны в вычислительном отношении и просты в реализации. Однако в приложении к цифровым изображениям они обладают рядом существенных недостатков: размывают границы и могут уничтожать мелко детальные особенности изображения.

Эффект размывания границ может быть существенно снижен при использовании нелинейных фильтров [1]. Существуют и изучены следующие нелинейные фильтры для обработки изображений: Байесовская фильтрация [11]; Итерационные методы восстановления изображения [11]; Полиномиальный фильтр, характеризуемый дискретным функциональным рядом [7]; Медианная фильтрация [13]; Адаптивные фильтры [13]; Ранговая обработка изображений [13]; Пороговая фильтрация [14]; Фильтры экстремумов [14]. Но к сожалению, ни один из низ не предназначен для восстановления размытых и расфокусированных изображений.

1.1 Методы нелинейной фильтрации

Уже существует множество методов восстановления (реставрации) изображений. К наиболее распространенным из них относятся: метод

фильтрации Винера, инверсная фильтрация (метод преобразования Фурье), метод на основе статистического оценивания и др.

Исправленное изображение в случае инверсной фильтрации описывается в функцией:

$$\hat{F}_1(x,y) = F_1(x,y) + \frac{1}{4\pi^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{N(\omega_x,\omega_y)}{H_D(\omega_x,\omega_y)}e^{i[\omega_x x,\omega_y y]}d\omega_x d\omega_y ,$$

где $F_1(x,y)$ — идеальное изображение; $H_D(\mathrm{x},\mathrm{y})$ — импульсный отклик линейной искажающей системы; $N(\mathrm{x},\mathrm{y})$ — аддитивный шум.

Частотная характеристика фильтра при винеровской фильтрации имеет вид:

$$H_H(\omega_x,\omega_y) = \frac{H_D^*(\omega_x,\omega_y)}{\left|H_D(\omega_x,\omega_y)\right|^2 + W_N(\omega_x,\omega_y)},$$

где $H_D(\omega_x,\omega_y)$ — частотная характеристика искажающей системы; $W_N(\omega_x,\omega_y)$ — энергетический спектр шума [8].

Отличительной чертой данных фильтров является необходимость знания о так называемом искажающем параметре системы (или искажающей функции). Так в случае фильтрации Винера и инверсной фильтрации, необходимо знать частотную характеристику искажающей системы.

1.2 Концепция задачи исследования

Рассмотрев принципы Винеровской фильтрации изображений, можно сделать вывод что в общем случае для размытых изображений график частотной характеристики восстанавливающей функции будет иметь вид, представленный на рисунке 1.1.

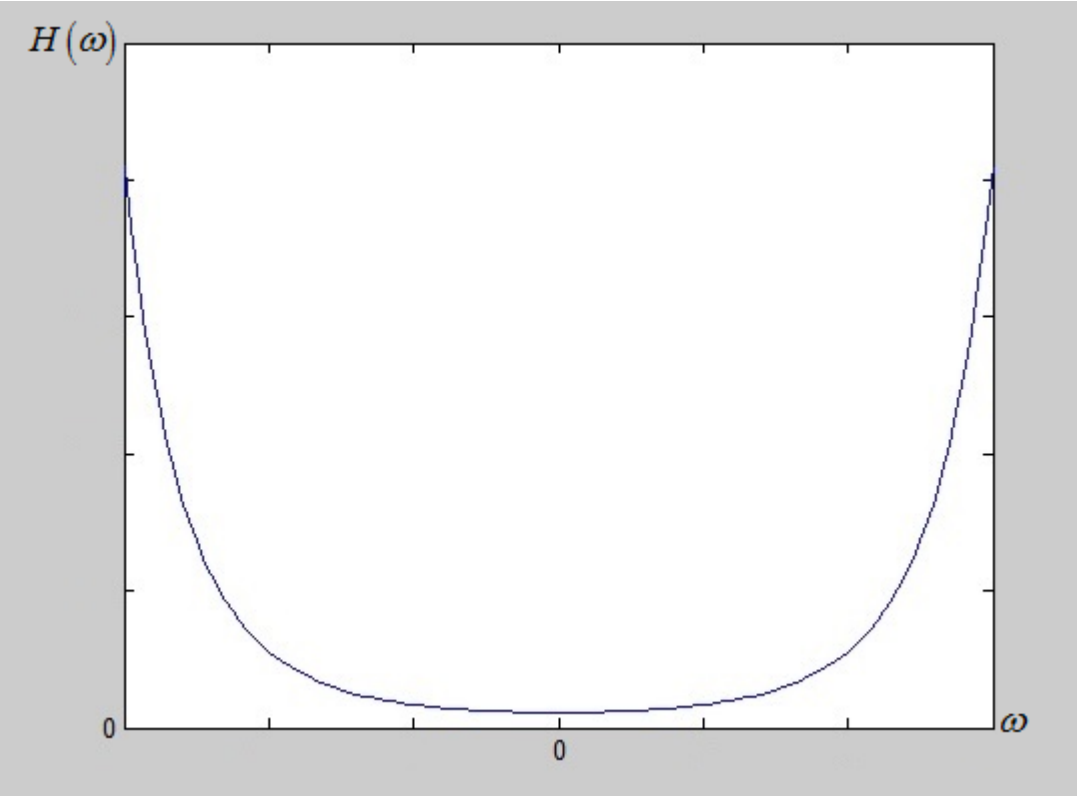

Рисунок 1.1 – Вид частотной характеристики восстанавливающей функции

Однако фильтр для данной частотной характеристики будет неустойчив из-за стремления значений к бесконечности по краям интервала аргументов. Так как основное влияние на искажение оказывают значения частотной характеристики искажающей функции не сильно удаленные от центра оси значений, то в качестве решение предлагается плавно свести эти значения к нулю, для придания фильтру устойчивости. График функции частотной характеристики искомого фильтра представлен на рисунке 1.2 – с неизвестными параметрами искажения и отклонения: a и b.

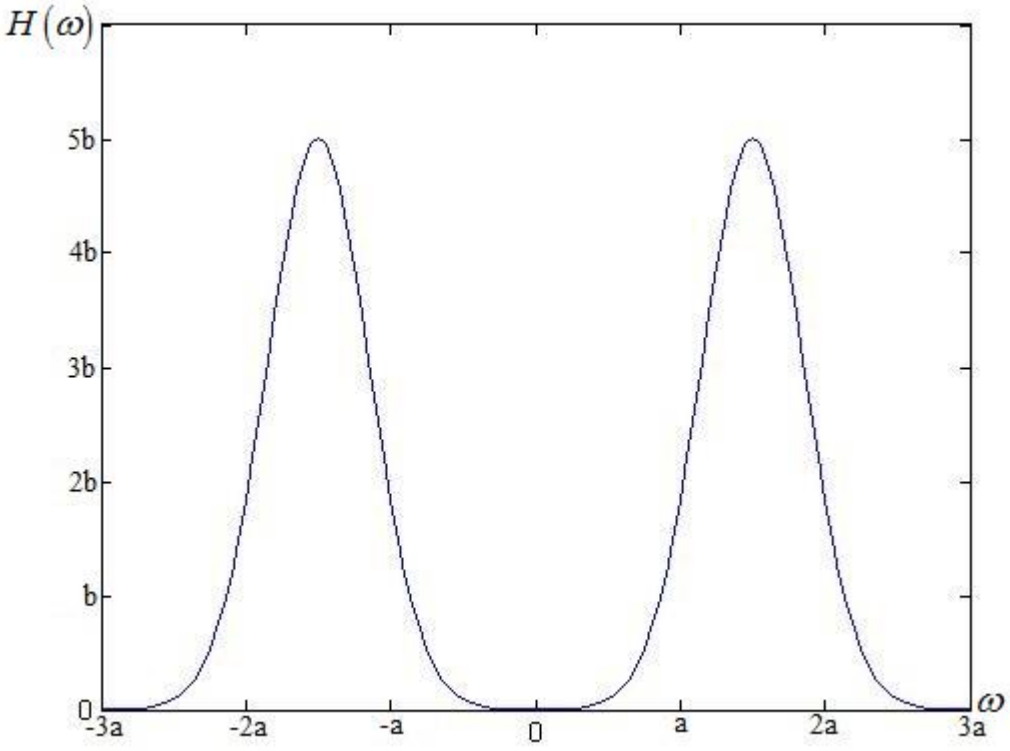

Рисунок 1.2 – Вид частотной характеристики устойчивой восстанавливающей функции

Для нахождения уравнения восстанавливающего фильтра, нам необходимо составить уравнение для приведенного вида частотной характеристики. Поэтапно были составлены и рассмотрены следующие функции:

$$y(x) = \left| xe^{-|x|} \right| \text{ и } y(x) = \left| xe^{-x^2} \right|. \quad (1.1)$$

Графики функций (1.1) представлены на рисунке 1.3 и рисунке 1.4.

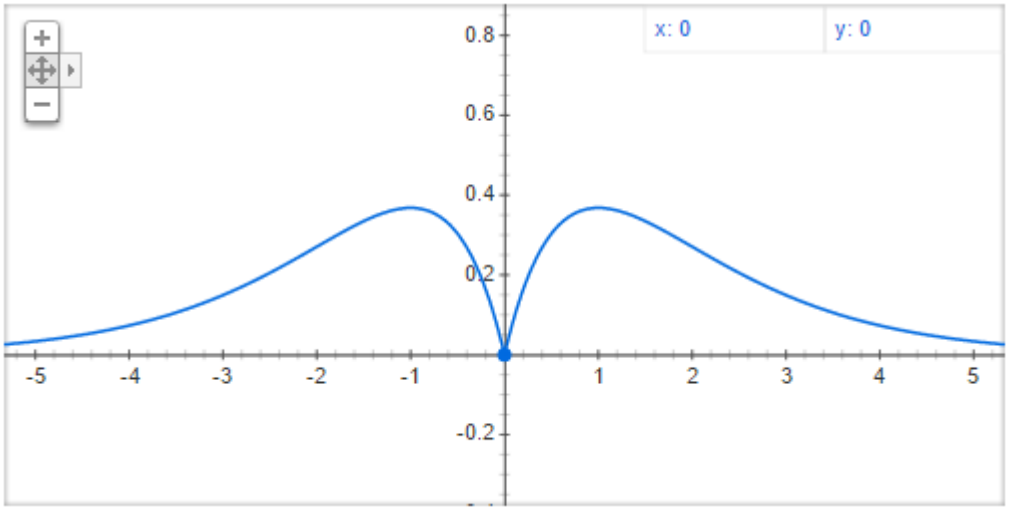

Рисунок 1.3 – График функции $y(x)=\left|xe^{-|x|}\right|$

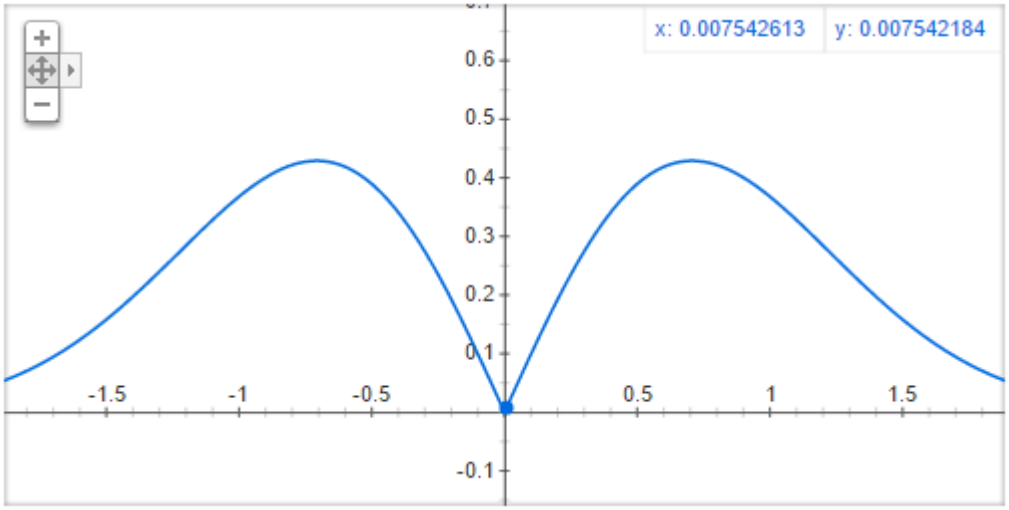

Рисунок 1.4 – График функции $y(x)=\left|xe^{-x^2}\right|$

Взяв за основу функцию Гаусса получена в общем виде функция, удовлетворяющая виду частотной характеристики устойчивой восстанавливающей функции (рисунок 1.2):

$$H(\omega)=ae^{-(c\omega-f)^2}+be^{-(d\omega+g)^2}. \qquad (1.2)$$

Так же рассмотрено преобразование функции от двух аргументов, так как изображение – это двумерная координатная плоскость со значениями яркости в точках (пиксели):

$$H(\omega_1,\omega_2) = ae^{-\left(\sqrt{b\omega_1^2+c\omega_1^2}-d\right)^2}. \qquad (1.3)$$

Графики функции (1.2) представлены на рисунке 1.5.

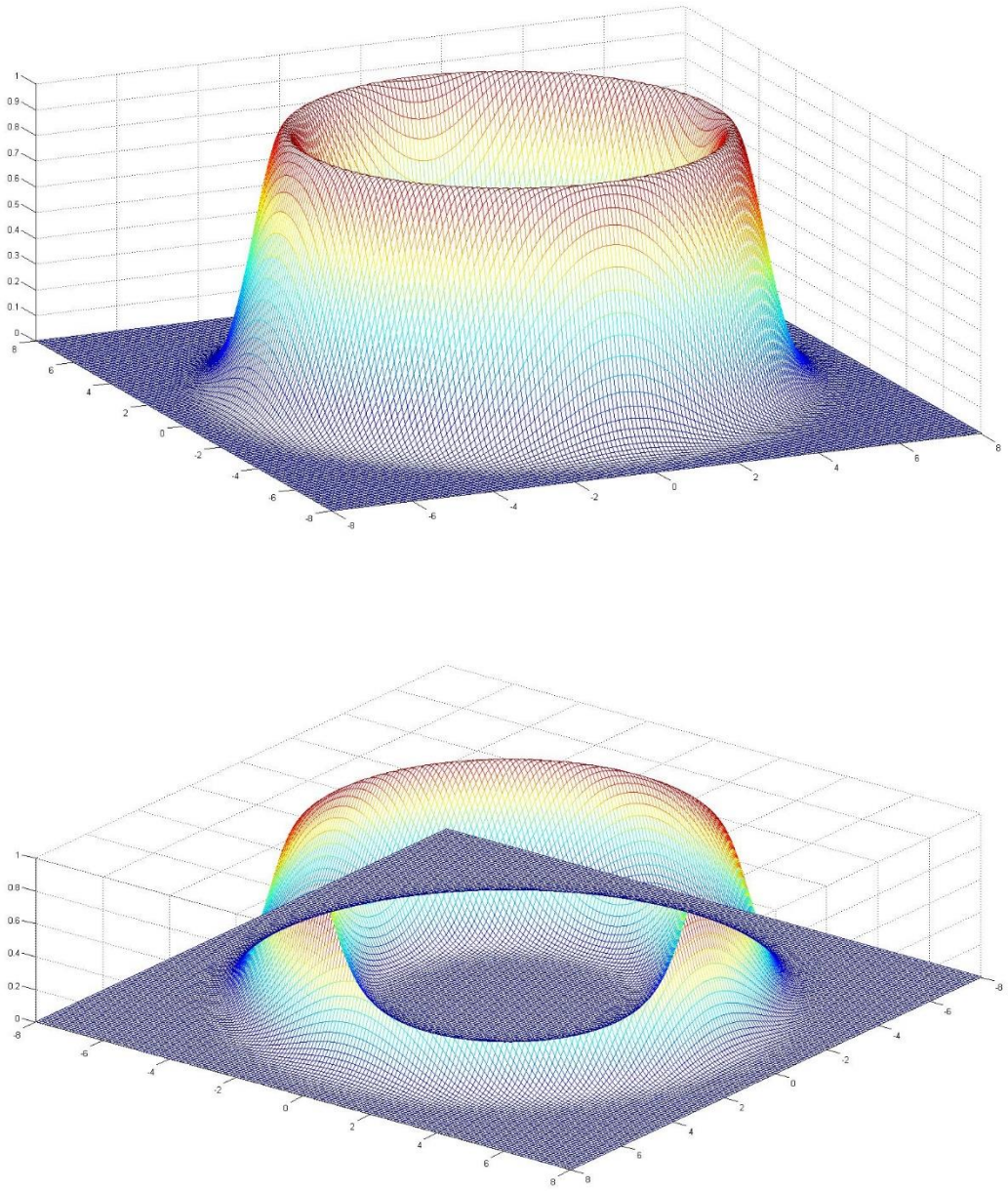

Рисунок 1.5 – Вид частотной характеристики устойчивой восстанавливающей функции для трехмерного случая

Для получения уравнения функции восстанавливающего фильтра общего вида при известной частотной характеристике необходимо произвести

преобразование Фурье над частотной характеристикой [8]. Рассмотрим двумерный случай:

$$f(n)=\int_{-\infty}^{+\infty}\left(ae^{-(c\omega-f)^2}+be^{-(d\omega+g)^2}\right)e^{i\omega n}d\omega. \quad (1.4)$$

Рассмотрим одно слагаемое:

$$f_1(n)=a\int_{-\infty}^{+\infty}e^{-(c\omega-f)^2}e^{i\omega n}d\omega. \quad (1.5)$$

Воспользуемся свойством преобразования Фурье:

$$e^{ian}f(n)\Leftrightarrow F(x-a), \quad (1.6)$$

тогда:

$$f_1(n)=ae^{in\frac{f}{c}}\int_{-\infty}^{\infty}e^{-c\omega^2}e^{i\omega n}d\omega. \quad (1.7)$$

Еще одно свойство преобразования Фурье (показывает, что функция Гаусса совпадает со своим изображением):

$$\exp\left(-an^2\right)\Leftrightarrow\frac{1}{\sqrt{2a}}\exp\left(\frac{-x^2}{4a}\right), \quad (1.8)$$

тогда:

$$f_1(n)=ae^{in\frac{f}{c}}\frac{1}{\sqrt{2c}}e^{-\frac{n^2}{4c}}=\frac{a}{\sqrt{2c}}e^{\frac{n}{c}\left(if-\frac{n}{4}\right)}. \quad (1.9)$$

Добавив второе слагаемое и переобозначив константы получим: $f(n)=ae^{icn-fn^2}+be^{-idn-gn^2}$, где a,b,c,d,f,g – неизвестные параметры. Неизвестные параметры входят в уравнение нелинейным образом, что очень сильно усложнят поиск конкретного восстанавливающего фильтра. А для трёхмерного случая: $f(n_1,n_2)=\iint ae^{-\left(\sqrt{b\omega_1^2+c\omega_2^2}-d\right)^2}e^{i\omega_1 n_1+i\omega_2 n_2}d\omega_1 d\omega_2$ – аналитически получить функцию восстанавливающего фильтра общего вида и вовсе не удалось. Учитывая эти два положения взят в рассмотрение ряд Вольтерра – как способ представления интеграла в виде набора линейных функций, что могло бы облегчить задачу. Далее в работе рассматривается построение

восстанавливающего фильтра в виде дискретного варианта ряда Вольтерра, обычно называемого полиномом Колмагорова-Габора.

2 РАЗРАБОТКА НЕЛИНЕЙНЫХ ФИЛЬТРОВ

2.1 Постановка задачи

На искаженном изображении определим опорную область (маску) D с дискретными отсчетами $x(n_1,n_2)\in D,\quad n_1=\overline{1,N_1},\quad n_2=\overline{1,N_2}$. Пусть $x(k_1,k_2)\in D$ отсчет из этой опорной области, на формирование которого оказывают влияние все отсчеты из этой же области (случай, когда $x(k_1,k_2)\notin D$ мы не рассматриваем). С использованием информации о степени влияния всех отсчетов из области D мы хотим построить нелинейный фильтр для формирования, соответствующего ему «неискаженного» отсчета $y(k_1,k_2)$. Нелинейный фильтр будем строить в виде ряда Вольтерра в дискретном варианте, обычно называемого полиномом Колмогорова-Габора:

$$y(k_1,k_2)=c_0+\sum_{n_1,n_2=1}^{N_1,N_2}c_{n_1,n_2}x(n_1,n_2)+\sum_{n_1,n_2=1}^{N_1,N_2}\sum_{m_1,m_2=1}^{N_1,N_2}c_{n_1,n_2;m_1,m_2}x(n_1,n_2)x(m_1,m_2)+\ldots \quad (2.1)$$

где $c_0, c_{n_1,n_2}, c_{n_1,n_2,m_1,m_2},\ldots$ - коэффициенты полинома. Нелинейная модель (2.1) является линейной по параметрам. При этом часто в эту модель вводят дополнительные нелинейные функции входных переменных без существенного усложнения структуры модели.

Предполагается, что наряду с исходным искаженным изображением имеется тестовое (обучающее) изображение или некоторый его фрагмент. Такой «неискаженный» фрагмент может быть задан на искаженном изображении как желаемая функция распределения яркости в некоторой области с использованием априорной информации о геометрической форме и спектральной интенсивности известных объектов (например, с использованием технологии «узнаваемые цвета»).

Пусть проведены измерения всех отсчетов яркости в N опорных областях D искаженного изображения. Из соответствующих этим опорным областям N отсчетов $y(n_1,n_2)$ на тестовом изображении, составим вектор Y

размерности $N\times1$. Если в каждой из этих опорных областей $D(n_1, n_2)$ число отсчетов одинаково, то число слагаемых в правой части (2.1) также одинаково. Если при этом параметры модели (коэффициенты полинома) в указанных N опорных областях изображения можно считать постоянными, в соответствии с (1) можно записать матричное соотношение

$$\mathbf{Y} = \mathbf{X}\mathbf{c} + \xi, \tag{2.2}$$

где $\mathbf{X}$ - матрица $N\times M$, каждая строка которой составлена из отсчётов изображения или их комбинаций вида в соответствующей области D, а M равно числу слагаемых в правой части, $\mathbf{c}$ - $M\times1$-вектор неизвестных параметров, ξ - $N\times1$-вектор, компонентами которого являются ошибки измерений, аппроксимации и др.

Задача состоит в построении оценки $\hat{\mathbf{c}}$ вектора параметров $\mathbf{c}$ по доступным для наблюдения вектору $\mathbf{Y}$ и матрице $\mathbf{X}$ при неизвестном векторе ошибок ξ. Нетрудно заметить, что вычислительная сложность сформулированной задачи идентификации существенным образом зависит от размерности модели (2.1). В частности, при возрастании интенсивности искажений число слагаемых в правой части, а, следовательно, размерность M быстро возрастают. Рост размерности наряду с вычислительными проблемами может приводить к снижению качества модели.

В настоящей работе исследуются различные варианты снижения размерности модели за счет учета симметрии искажений, а также исключения произведений отсчетов, приводящих к почти линейной зависимости векторов-столбцов матрицы $\mathbf{X}$. Оценка качества моделей осуществляется путем сравнения исходных неискаженных изображений с изображениями, полученными в результате обработки искаженных изображений, полученных путем моделирования.

2.2 Описание алгоритма

В большинстве случаев искажения типа дефокусировки в некоторой локальной пространственной области обладают радиальной симметрией. Обусловлено это формой пятна размытия, имеющего место при различных аберрациях оптических систем. Этот эффект используется [1] для существенного снижения размерности модели (2.1). В частности, сгруппировать значения отсчетов $x_i(n_1, n_2)$ при одинаковых коэффициентах полинома, находящихся (в силу симметрии) на одинаковых расстояниях r от центральной точки опорной области $D(n_1, n_2)$:

$$x(r) = \frac{1}{m}\sum_{i=1}^{m} x_i(n_1, n_2, r), \tag{2.3}$$

где n_1, n_2 – все возможные значения координат отсчетов на расстоянии r (для которых $n_1 = |n_1|$, $n_2 = |n_2|$, а m - число таких отсчетов.

На рисунке 2.1 приведен пример опорной области 7×7, на которой точками на окружностях показаны отсчеты, подлежащие объединению. В данном случае модель (2.1) можно представить в виде:

$$y(k_1, k_2) = c_0 + \sum_{i=1}^{R} c_i x(r_i) + \sum_{i=1}^{R}\sum_{j=1}^{R} c_{i,j} x(r_i) x(r_j) + \ldots \tag{2.4}$$

В этой модели могут присутствовать слагаемые, приводящие к почти линейной зависимости векторов-столбцов матрицы **X**. В частности, нетрудно заметить, что третья и четвертая (от центра) окружности находятся на малом расстоянии друг от друга, поэтому соответствующие им усредненные значения $x(r)$ будут почти совпадающими. Такие значения отсчетов целесообразно сгруппировать между собой.

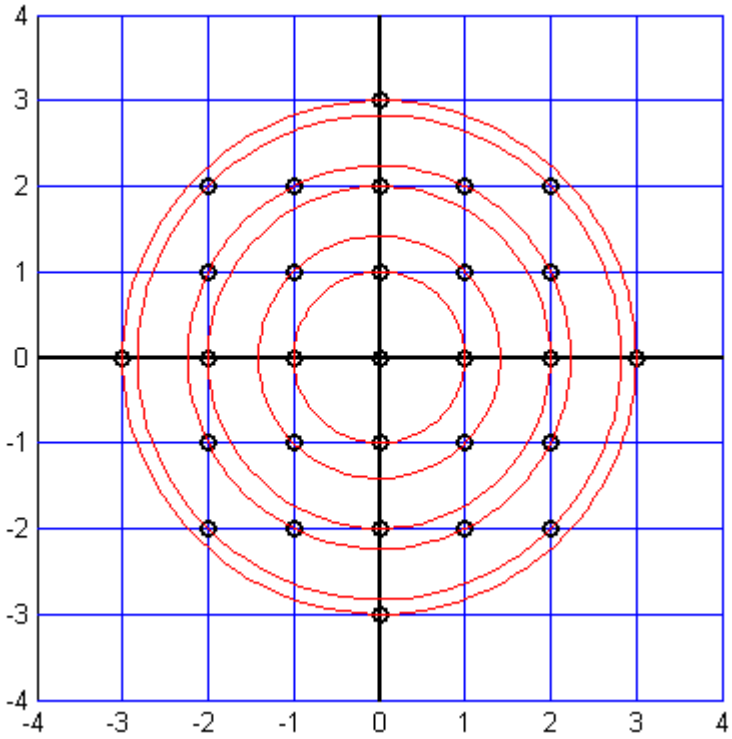

Рисунок 2.1 – Пример опорной области 7×7

Дальнейшее упрощение модели (2.4) может состоять в исключении слагаемых, содержащих произведения отсчетов с разными индексами. В данном случае также целесообразно в первую очередь исключать слагаемые, соответствующие усредненным значениям $x(r)$, находящимся на близких окружностях. В некоторых случаях характер искажений допускает использование простой модели, в которой все слагаемые, содержащие произведения отсчетов с разными индексами исключены:

$$y(k_1,k_2)=c_0+\sum_{i=1}^{R}c_{1,i}x(r_i)+\sum_{i=1}^{R}c_{2,i}x^2(r_i)+\ldots \qquad (2.5)$$

В разделе 4 приводятся результаты сравнительных экспериментов при использовании моделей различной размерности.

2.3 Параллельная реализация

Для разработки массивно-многопоточного алгоритма применения восстанавливающего фильтра к искаженному изображению в данной работе

используется технология CUDA. Данная технология имеет наилучшую эффективность, когда каждый отдельный поток вычисляет одну и туже функцию. Более подробное описание технологии описано в разделе 3.5.

Применение полученного фильтра к искаженному изображению совершается поэтапно для каждого слагаемого модели. Например, для модели (2.5): сначала параллельно для каждого отсчета производится расчёт линейных членов ряда, затем квадратичных и так далее. Данный подход может помочь получить наилучшую эффективность использования технологии CUDA для восстановления больших изображений.

3 ОПИСАНИЕ ПРОГРАММНОГО КОМПЛЕКСА

3.1 Внешнее описание программного комплекса

Назначение: программный комплекс предназначен для восстановления размытых изображений. Он позволяет задавать шаблоны размытых изображений для идентификации параметров восстанавливающего фильтра и восстанавливать размытые изображения используя найденные параметры фильтра.

На рисунке 3.1 представлена диаграмма PDOM (Problem Domain Object Model – объектная модель предметной области) – описывает предметную область поставленной задачи в терминах объектов и взаимосвязей между ними [18]. В таблицах 3.1-3.3 представлены описание частей диаграммы.

Рисунок 3.1

Таблица 3.1 – Словарь данных

Термин	**Описание**
Входные изображения	Набор изображений необходимый для работы системы, туда входят: изображения характеризующие шаблон искажения (исходное и искаженное изображения), изображение для восстановления.
Выходное изображение	Восстановленное изображение.
Статистика	Количественные характеристики по результату восстановления изображения.

Таблица 3.2 – Описание атрибутов

Название класса	**Название атрибута**	**Описание**
Входные изображения	Тип	Характеризует предназначение изображения для системы (исходное изображение для шаблона, изображение для восстановления и т.п.
Система	Параметры	Параметры характеризующие алгоритм работы системы
Статистика	Тип	Тип выводимых статистических данных по работе системы

Таблица 3.3 – Описание зависимостей

Название зависимости	**Класс 1**	**Класс 2**	**Кратность**	**Описание**
Загружаются	Входные изображения	Система	*..1	В систему загружаются необходимые изображения
Предоставляет	Система	Статистика	1..1	Система выводит статистику результата работы
Выводит	Система	Выходное изображение	1..1	Система выводит обработанное изображение

3.2 Функциональная спецификация

На рисунке 3.2 представлены диаграммы вариантов использования программного комплекса. В таблице 3.4 представлено описание актеров системы. А в таблице 3.5 детально описаны варианты использования программного комплекса.

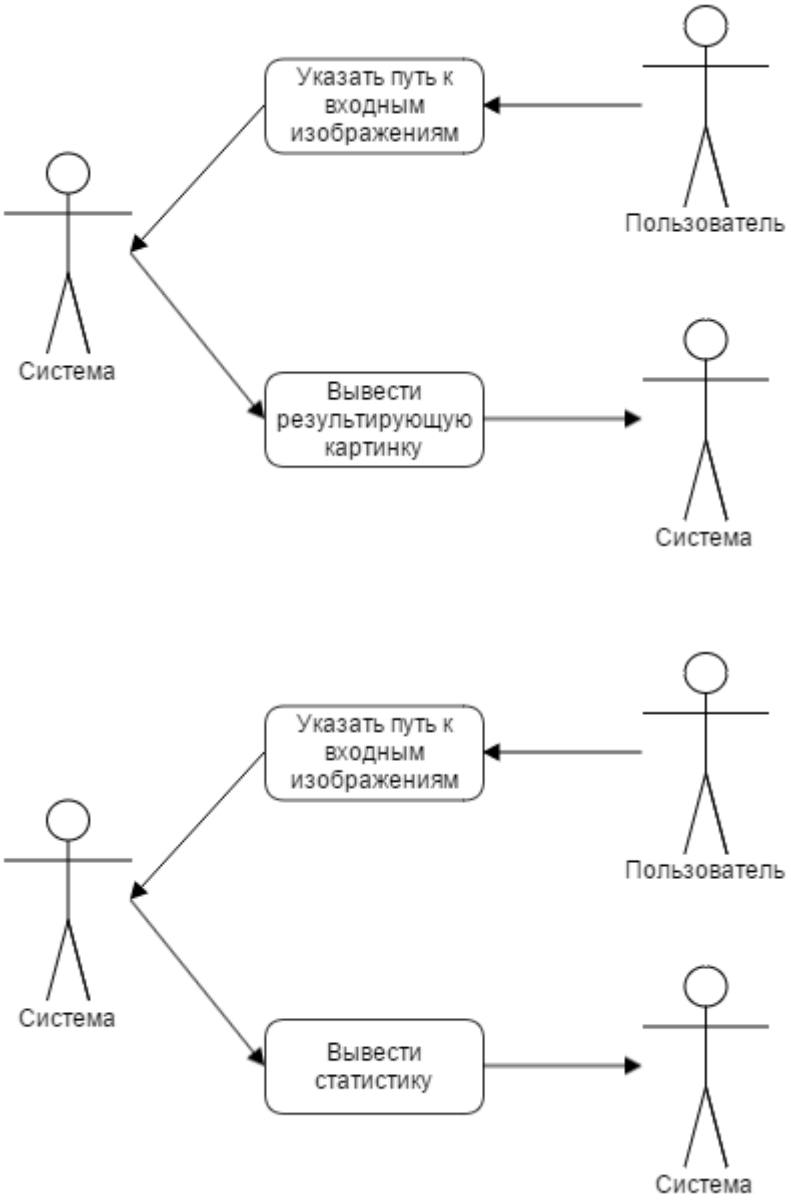

Рисунок 3.2

Таблица 3.4

Название	**Описание**
Система	Основной вычислительный модуль, имеющий возможность считать входные изображения, обработать их и вывести результат обработки, а также предоставить статистику по процессу обработки изображений.
Хранилище	Место хранения входных изображений для обработки системой.
Экран	Устройство вывода результатов работы системы.

Таблица 3.5

№	*Имя*	*Описание*
1	**Название**	**Указать путь к входным изображениям**
	Описание	Система считывает набор входных изображений из указанного пути.
	Цель	Считать и обработать входные данные для анализа.
	Актеры	
	1	Система (инициатор)
	2	Пользователь (участник)
	Предусловия	Пользователь создал директорию с необходимыми входными изображениями и указал системе путь.
	Постусловия	Система обрабатывает выходные изображения.
	Поток событий	
	1	Система считывает входные изображения из указанного хранилища.
	2	Система обрабатывает считанные входные данные.
	Альтернативы	
	Исключения	
	1	Путь к входным изображениям указан не верно.
	2	По указанному пути недостаточно данных для анализа.
	3	По указанному пути входные данные в неверном формате.
2	**Название**	**Вывести статистику**
	Описание	Система обработала данные, накопила статистику и выводит ее на экран.
	Цель	Обработать данные и вывести статистику для дальнейшего анализа пользователю.
	Актеры	
	1	Система (инициатор)
	2	Система (участник)

Продолжение таблицы 3.5

	Предусловия	Система обработала входные данные и накопила статистику по процессу обработки.
	Постусловия	Система вывела отчет о статистике работы на экран.
	Поток событий	
	1	Система обрабатывает входные данные.
	2	Система накапливает статистику по обработке входных данных.
	3	Система выводит на экран накопленную статистику по обработке входных данных.
	Альтернативы	
	Исключения	
3	**Название**	**Вывести результирующую картинку**
	Описание	Система обработала входные данные, сформировала результат и выводит его.
	Цель	Обработать входные данные и вывести результат работы системы.
	Актеры	
	1	Система (инициатор)
	2	Система (участник)
	Предусловия	Система обработала входные данные и сформировала результат.
	Постусловия	Система вывела результат работы на экран.
	Поток событий	
	1	Система обрабатывает входные данные.
	2	Система формирует результат работы.
	3	Система выводит на экран результат работы.
	Альтернативы	
	Исключения	

Функциональные требования:

- Система позволяет пользователю указать директорию с входными изображениями, над которыми будет осуществляться работы системы.
- Система по завершению работы выводит пользователю восстановленное изображение.

Нефункциональные требования:

- Требования производительности.
  - Система должна справляться с нагрузками при работе с изображениями размером не более 2 Гб.
  - Система должна работать с большими данными менее часа.
- Требования надежности.
  - Система должна удостовериться в полной правильности входных данных на начальном этапе работы.
  - Система должна сообщать об ошибках, содержащих описание действий, которые их повлекли, и возможные варианты их исправления.
- Требования доступности.
  - Система не имеет ограничений по доступу.
- Требования к обработке ошибок.
  - В случае возникновения ошибок система должна вывести информативное сообщение о случившейся проблеме.
  - В случае возникновения ошибок система прекращает работу.
- Требования к интерфейсам.
  - Система предоставляет текстовый интерфейс для взаимодействия.
  - Текстовый интерфейс должен выводить простые и понятные сообщения о статусе работы системы.

- Система предоставляет вывод результата работы в окно графического интерфейса.

- Требования ограничения.
  - Для работы с системой требуется использовать компьютер с графическим процессором, поддерживающим технологию CUDA.

3.2.1 Спецификация качества

Критерии качества:

- Надежность: устойчивость.
- Эффективность: временная эффективность.

Примитивы качества:

- Устойчивость – система уведомляет пользователя о произошедших ошибках в процессе работы и продолжает свою дальнейшую работу.
- Временная эффективность – система работает над одним входным изображением не более, чем 5 минут.

3.3 Архитектура

3.3.1 Контекстная диаграмма

На рисунке 3.3 представлена контекстная диаграмма, которая представляет собой самое общее описание системы и ее взаимодействие с внешней средой.

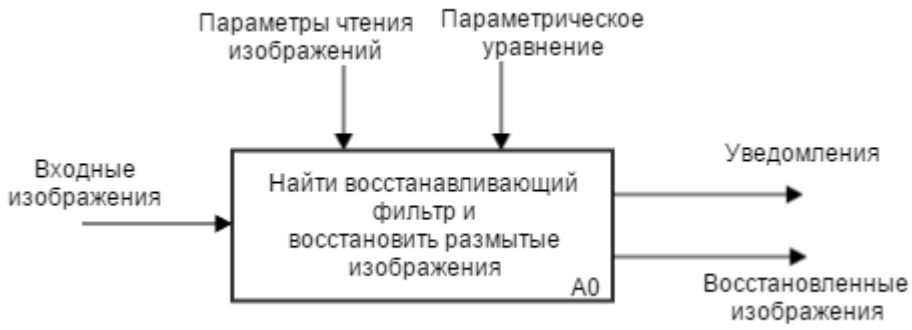

Рисунок 3.3

Внешние входы системы:

- Входные изображения – изображения, необходимые для построения восстанавливающего фильтра и изображения для восстановления.

Внешние выходы системы:

- Уведомления – текстовая информация на экране отображающая ход выполнения работы программы и найденные количественные характеристики восстанавливающего фильтра и процесса восстановления.
- Восстановленные изображения – результат работы программы.

3.3.2 Иерархия диаграмм

На рисунке 3.4 представлена полная контекстная диаграмма уровня A0. А на рисунках 3.5-3.7 контекстные диаграммы декомпозиций всех дочерних узлов.

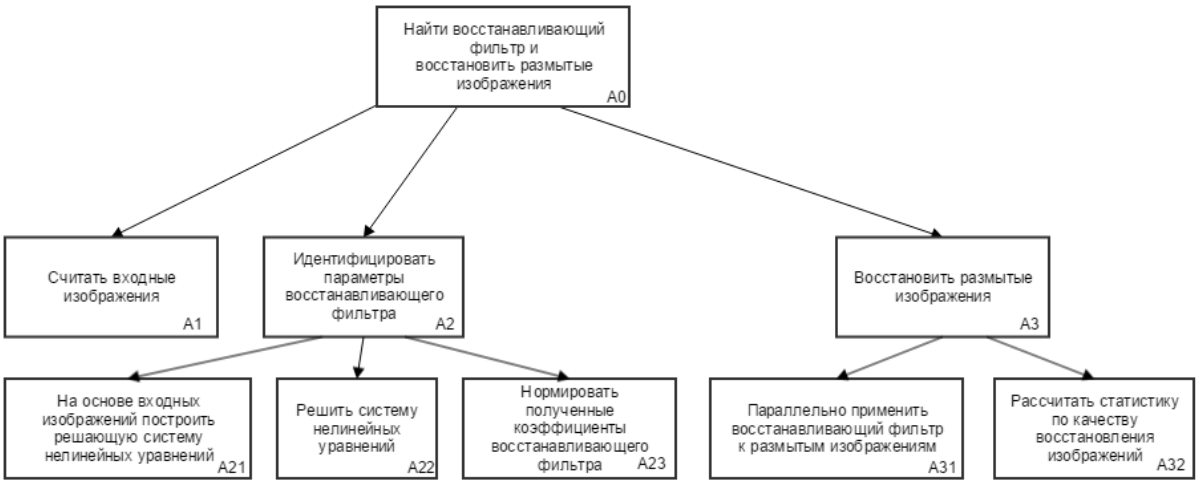

Рисунок 3.4

Рисунок 3.5 – Диаграмма первого уровня

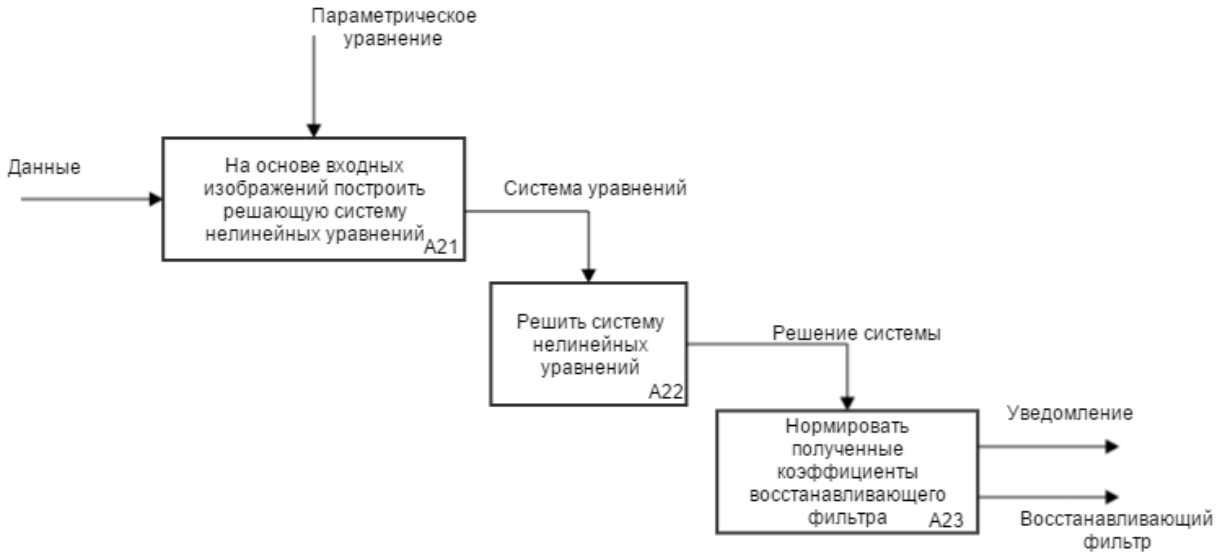

Рисунок 3.6 – Диаграмма второго уровня A2

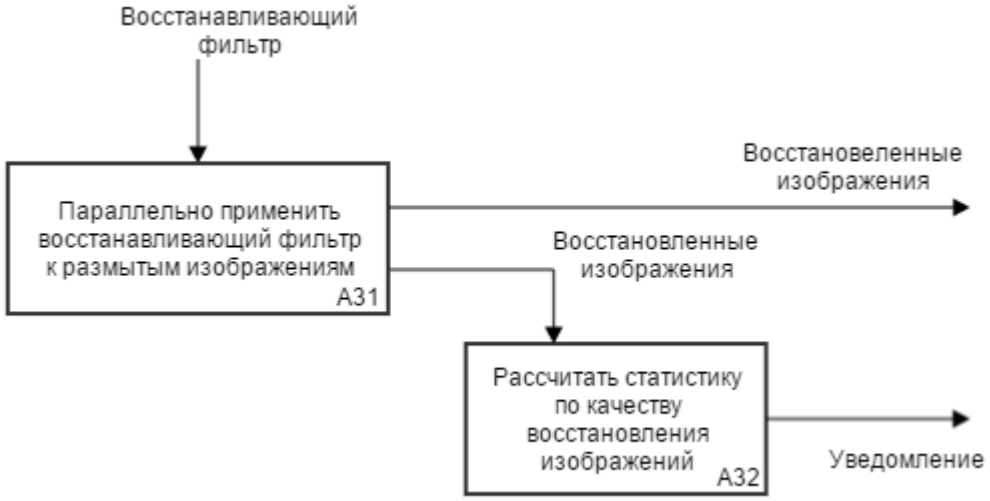

Рисунок 3.7 – Диаграмма второго уровня A3

3.3.3 Расчёт коэффициентов декомпозиции и сбалансированности для каждой диаграммы

Необходимо стремиться к тому, чтобы количество блоков на диаграммах нижних уровней было бы ниже количества блоков на родительских диаграммах, т.е. с увеличение уровня убывал бы коэффициент декомпозиции:

$$K_d = \frac{N}{L},$$

где N – количество блоков на диаграмме, L – уровень декомпозиции диаграммы.

Коэффициент сбалансированности:

$$K_b = \left| \frac{\sum_{i=1}^{N} A_i}{N} - \max_{i=1,N} A_i \right|,$$

где A_i – число стрелок, соединяющихся с i-ым блоком. Необходимо стремиться чтобы он был минимален [21]. В таблице 3.6 представлен расчет коэффициента сбалансированности для всех уровней. В таблицах 3.7 и 3.8 даны описания элементов контекстной диаграммы.

Таблица 3.6

Уровень	K_d	K_b
Первый	3	$\left\|\frac{3+4+3}{3} - 4\right\| = \frac{2}{3}$
Второй (А2)	1,5	$\left\|\frac{3+2+3}{3} - 3\right\| = \frac{1}{3}$
Второй (А3)	1	$\left\|\frac{3+2}{2} - 3\right\| = \frac{1}{3}$

Таблица 3.7 – Описание функциональных блоков

Блок	**Описание**
A0	Нахождение восстанавливающего фильтра на основе входных изображений и восстановление переданных размытых изображений.
A1	Считывание входных изображений по типу использования.

Продолжение таблицы 3.7

A2	С помощью алгоритма идентифицирует параметры восстанавливающего фильтра.
A3	С помощью алгоритма восстанавливает тестовые размытые изображения.
A21	Для заданного в системе параметрического уравнения, на основе входных изображение строится система нелинейных уравнений.
A22	Производится решение системы нелинейных уравнений.
A23	Нормирование найденных коэффициентов восстанавливающего фильтра (сумма коэффициентов должна быть равна единице).
A31	Параллельно восстанавливаются все переданные изображения.
A32	Производится подсчет статистики по качеству восстановления изображений.

Таблица 3.8 – Описание интерфейсов и управлений

Интерфейсная дуга	**Тип**	**Описание**
Параметрическое уравнение	Управление	Параметрическое равнение на основе которого происходит построение системы нелинейных уравнений.
Параметры чтения изображений	Управление	Параметры, определяющие тип входного изображения.
Данные	Выход Вход	Внутренне представление изображения в системе.
Восстанавливающий фильтр	Выход Управление	Параметрическое уравнение и его идентифицированные параметры.
Уведомление	Выход	Информация о статистике и этапе работы программы.

Продолжение таблицы 3.8

Восстановленные изображения	Выход Управление	Результат работы программы.
Система уравнений	Выход Управление	Система нелинейных уравнений, построенная на основе параметрического уравнения.
Решение системы	Выход Управление	Численные коэффициенты параметрического уравнения, являющиеся решения системы нелинейных уравнений.

В таблице 3.9 показана связность всех блоков контекстной диаграммы по типу связи.

Таблица 3.9

Блоки	**A0**	**A1**	**A2**	**A3**	**A21**	**A22**	**A23**	**A31**	**A32**
A0	-	Р	Р	Р	Р	Р	Р	Р	Р
A1	Д	-	К	П	К	П	П	П	П
A2	Д	Ф	-	К	Р	Р	Р	П	П
A3	Д	Ф	Ф	-	Ф	Ф	Ф	Р	Р
A21	Д	Ф	Д	П	-	К	П	П	П
A22	Д	Ф	Д	П	Ф	-	К	П	П
A23	Д	Ф	Д	П	Ф	Ф	-	К	П
A31	Д	Ф	Ф	Д	Ф	Ф	Ф	-	К
A32	Д	Ф	Ф	Д	Ф	Ф	Ф	Ф	-

Обозначения связностей: С – случайная, Л – логическая, В – временная, Пр – процедурная, К – коммуникационная, П – последовательная, Ф – функциональная, Д – дочерняя, Р – родительская.

3.4 Диаграммы потоков

На рисунках 3.8-3.13 представлены диаграммы потоков для каждого блока иерархии контекстной диаграммы.

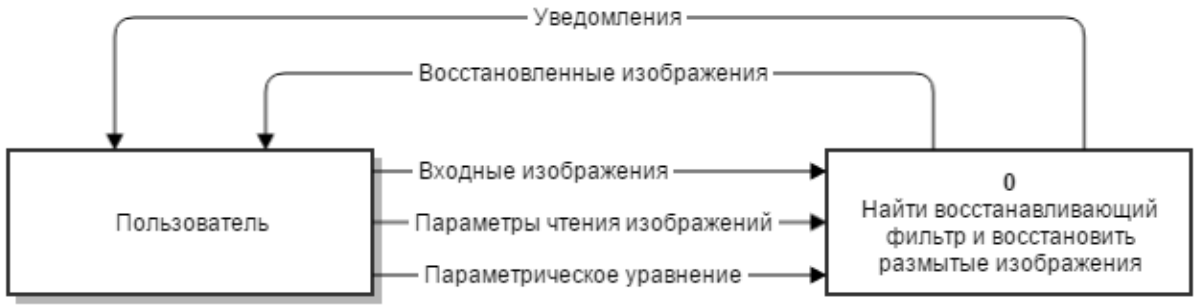

Рисунок 3.8 – Контекстная диаграмма

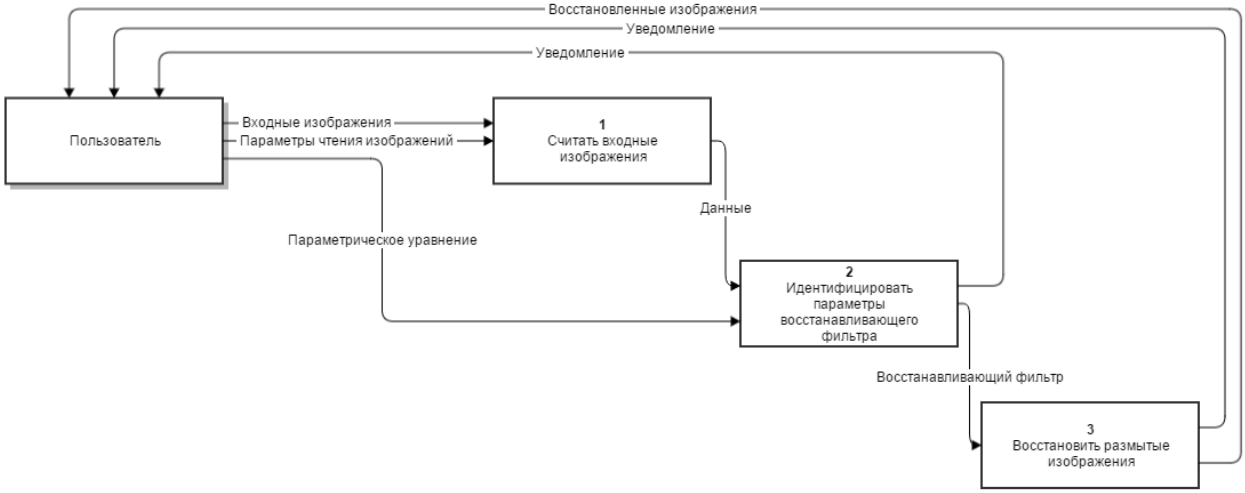

Рисунок 3.9 – Диаграмма нулевого уровня

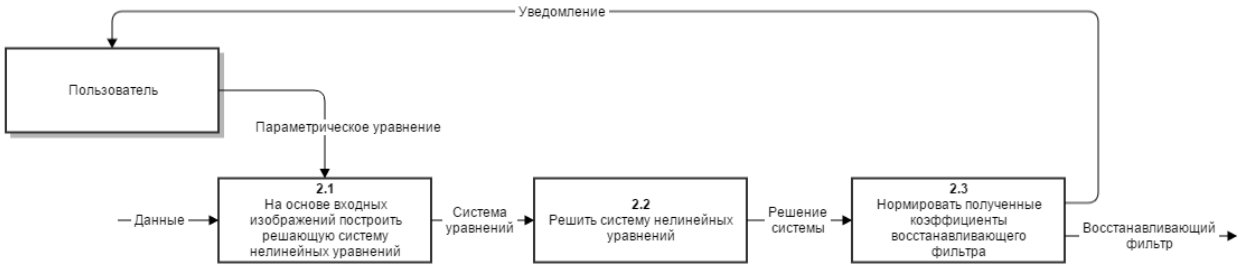

Рисунок 3.10 – Диаграмма первого уровня для подсистемы 2

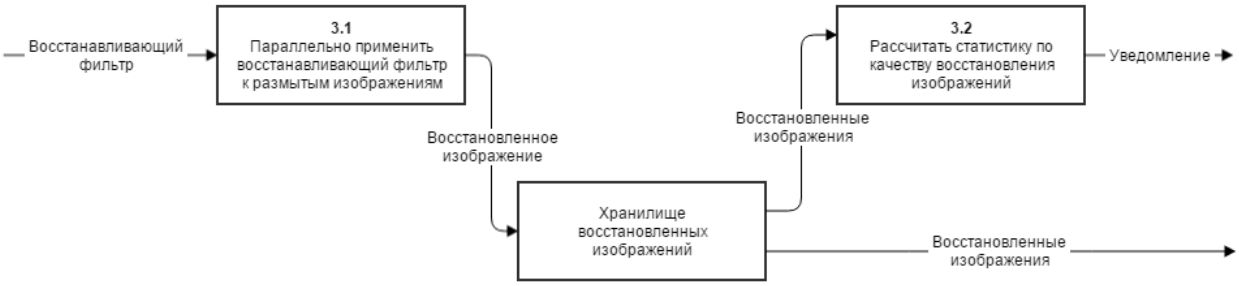

Рисунок 3.11 – Диаграмма первого уровня для подсистемы 3

Рисунок 3.12 – Диаграмма второго уровня для подсистемы 2.1

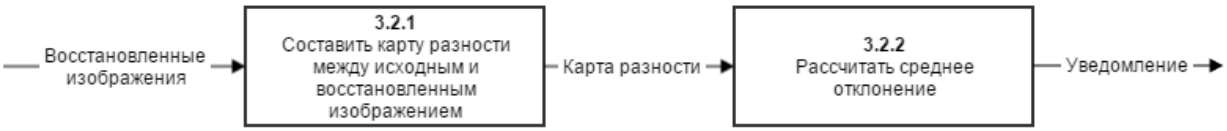

Рисунок 3.13 – Диаграмма второго уровня для подсистемы 3.2

В таблицах 3.10-3.13 приведены описания для всех элементов диаграмм потоков.

Таблица 3.10 – Таблица внешних сущностей

Внешняя сущность	**Описание**
Пользователь	Человек, который подает системе на вход необходимые данные и ожидает получить результат работы системы

Таблица 3.11 – Таблица подсистем и процессов

Подсистема или процесс	**Описание**
Найти восстанавливающий фильтр и восстановить размытые изображения	Нахождение восстанавливающего фильтра на основе входных изображений и восстановление переданных размытых изображений.
Считать входные изображения	Считывание входных изображений по типу использования.
Идентифицировать параметры восстанавливающего фильтра	С помощью алгоритма идентифицирует параметры восстанавливающего фильтра.
Восстановить размытые изображения	С помощью алгоритма восстанавливает тестовые размытые изображения.
Параллельно применить восстанавливающий фильтр к размытым изображениям	Параллельно восстанавливаются все переданные изображения.
Рассчитать статистику по качеству восстановления изображений	Производится подсчет статистики по качеству восстановления изображений.
На основе входных изображений построить решающую систему нелинейных уравнений	Для заданного в системе параметрического уравнения, на основе входных изображение строится система нелинейных уравнений.
Решить систему нелинейных уравнений	Производится решение системы нелинейных уравнений.
Нормировать полученные коэффициенты восстанавливающего фильтра	Нормирование найденных коэффициентов восстанавливающего фильтра (сумма коэффициентов должна быть равна единице).

Продолжение таблицы 3.11

Сформировать вектор решения системы на основе тестового не размытого изображения	Из тестового не размытого изображения производится выбор случайных точек, в соответствии с размерностью системы.
Для установленной маски найти значения аргументов параметрического уравнения на основе тестового размытого изображения	По выбранным точкам из тестового не размытого изображения – выбираются соответствующие точки из соответствующего размытого изображения и прилежащие к ним точки, на основе установленной в системе маски.
Проверить ранг системы	Производится проверка условия на соответствие ранга системы и размерности системы.
Составить карту разности между исходным и восстановленным изображением	Составление матрицы разностей значений точек между исходным и восстановленным изображением.
Рассчитать среднее отклонение	На основе карты разностей производится подсчет среднего значения отклонения.

Таблица 3.12 – Таблица накопителей данных

Накопитель данных	**Описание**
Хранилище восстановленных изображений	Хранилище, которое сохраняет все восстановленные изображения в ходе параллельного процесса.

Таблица 3.13 – Таблица потоков данных

Поток данных	**Описание**
Входные изображения	Электронные файлы, которые можно считать как RGB матрицу.
Параметры чтения изображений	Параметры, определяющие тип входного изображения.
Параметрическое уравнение	Параметрическое равнение на основе которого происходит построение системы нелинейных уравнений.
Уведомление	Информация о статистике и этапе работы программы.
Восстановленные изображения	Результат работы программы.
Данные	Внутренне представление изображения в системе.
Восстанавливающий фильтр	Параметрическое уравнение и его идентифицированные параметры.
Система уравнений	Система нелинейных уравнений, построенная на основе параметрического уравнения.
Решение системы	Численные коэффициенты параметрического уравнения, являющиеся решения системы нелинейных уравнений.
Координаты выбранных точек	Двумерные координаты точек, которые были выбраны на тестовом не размытом изображении.
Матрица решающей системы	Матрица значений аргументов параметрического уравнения для выбранных точек.
Карта разности	Матрица разностей значений точек между исходным и восстановленным изображением.

3.5 Программная реализация

В текущей работе для идентификации параметров фильтра используется математический пакет Matlab. Так как идентифицировать параметры фильтра можно на малом фрагменте изображения, то вычисления занимают очень малое время и не требуют большого количества памяти для хранения изображения и промежуточных данных. Код программы приведен в приложении А1.

Непосредственное применение найденного восстанавливающего фильтра осуществляется с помощью технологии CUDA. Данная технология позволяет осуществлять вычисления на графическом процессоре, который имеет много большее количество вычислительных ядер и работает с числами с плавающей точкой [9]. Это позволит быстро в большом изображении (например, космического снимка) к каждому пикселю – применить построенный восстанавливающий фильтр. Код программы приведен в приложении А2.

На рисунках 3.14-3.15 приведены графики, показывающие преимущество использования параллельной реализации с помощью технологии CUDA.

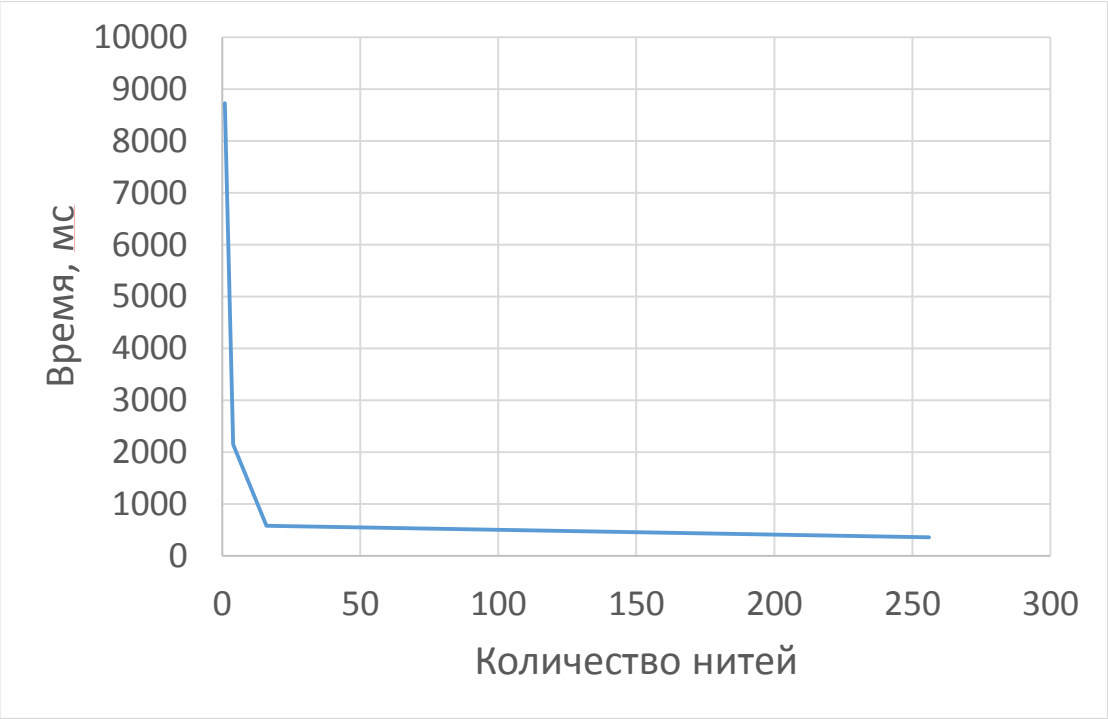

Рисунок 3.14 – График зависимости времени работы программы от числа нитей

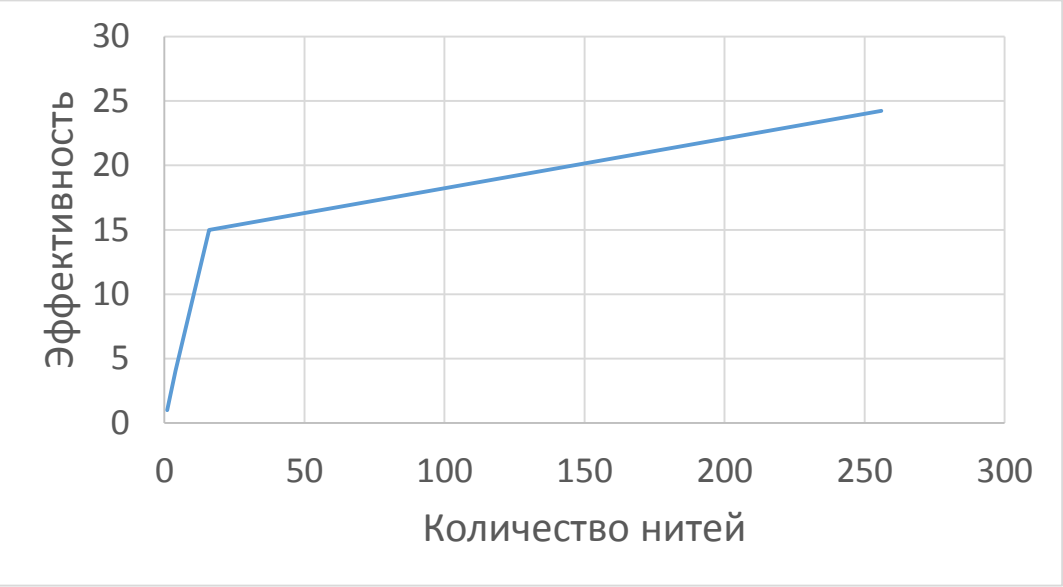

Рисунок 3.15 – График показателя эффективности работы программы в зависимости от числа нитей

Эксперимент проводился на изображении размером 12191×17198 на графическом процессоре ASUS 90YV07Q1-M0NA00 NVIDIA GEFORCE GTX 960 4GB. Последовательный алгоритм выполнялся на том же персональном компьютере с процессором Intel Core i5-4690 CPU @ 3.50Ghz.

4 ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ

В данном разделе приведены результаты по восстановлению изображений различными фильтрами, описанными в разделе 2.2. В качестве исходных (неискаженных) изображений взяты изображения, представленные на рисунках 4.1 и 4.2.

Рисунок 4.1 – Исходное изображение «Город»

Рисунок 4.2 – Исходное изображение «Лена»

Так же эксперименты рассматриваются для различной степени искажений изображений. Изображения искажались искусственно с помощью фильтра Гаусса нижних частот, который в качестве характеристики искажения принимает число – σ-отклонение. На рисунках 4.3 и 4.4 приведены искаженные изображения рисунка 4.1 с параметрами $\sigma=3$ и $\sigma=5$

соответственно. И аналогично на рисунках 4.5 и 4.6 приведены искаженные изображения для рисунка 4.2.

Рисунок 4.3

Рисунок 4.4

Рисунок 4.5

Рисунок 4.6

4.1 Полная модель

На рисунках 4.7-4.10 приведены результаты восстановления изображений с помощью полной модели:

$$y(k_1,k_2)=c_0+\sum_{i=1}^{6}c_i x(r_i)+\sum_{i=1}^{6}\sum_{j=1}^{6}c_{i,j}x(r_i)x(r_j).$$

Рисунок 4.7 – Восстановленное изображение для рисунка 4.3

Рисунок 4.8 – Восстановленное изображение для рисунка 4.4

Рисунок 4.9 – Восстановленное изображение для рисунка 4.5

Рисунок 4.10 – Восстановленное изображение для рисунка 4.6

По представленным изображениям видно, что построенная модель даёт хороший результат. Отсутствует резкое появление артефактов или импульсного шума на восстановленных изображениях.

4.2 Упрощенные модели

На рисунках 4.11-4.14 приведены результаты восстановления изображений с помощью упрощённой модели:

$$y(k_1,k_2)=c_0+\sum_{i=1}^{6}c_{1,i}x(r_i)+\sum_{i=1}^{6}c_{2,i}x^2(r_i)+\sum_{i=1}^{6}c_{2,i}x^3(r_i).$$

Рисунок 4.11 – Восстановленное изображение для рисунка 4.3

Рисунок 4.12 – Восстановленное изображение для рисунка 4.4

Рисунок 4.13 – Восстановленное изображение для рисунка 4.5

Рисунок 4.14 – Восстановленное изображение для рисунка 4.6

Как видно из приведенных рисунков упрощение модели визуально не вносит существенных ухудшений в качество восстановления.

4.3 Сравнительные результаты

На рисунках 4.15-4.18 представлены результаты восстановления изображений с помощью Винеровского фильтра из открытой библиотеки OpenCV. Параметры Винеровского фильтра подбирались таким образом, чтобы достигалось наименьшее значение средне квадратичного отклонения восстановленного изображения от исходного (не учитывая границы).

Рисунок 4.15 – Восстановленное изображение для рисунка 4.3

Рисунок 4.16 – Восстановленное изображение для рисунка 4.4

Рисунок 4.17 – Восстановленное изображение для рисунка 4.5

Рисунок 4.18 – Восстановленное изображение для рисунка 4.6

По представленным изображениям уже можно судить о существенном преимуществе использования построенной модели нелинейного фильтра перед стандартным линейным Винеровским фильтром. При восстановлении сильно размытых изображений Винеровским фильтром возникают множественные артефакты вокруг границы объектов на изображении.

В таблицах 4.1 и 4.2 приведены значения среднеквадратичных отклонений представленных изображений от оригинала для рассмотренных моделей, а также для различных упрощений или дополнений в полной модели.

Таблица 4.1 – Значения СКО для различных фильтров при восстановлении изображений с показателем размытия σ =3

Фильтр	**СКО «Лена»**	**СКО «Город»**
$y(k_1,k_2)=c_0+\sum_{i=1}^{6}c_{1,i}x(r_i)+\sum_{i=1}^{6}c_{2,i}x^2(r_i)+\sum_{i=1}^{6}c_{2,i}x^3(r_i)$	8,7016	15,1467
$y(k_1,k_2)=c_0+\sum_{i=1}^{6}c_{1,i}x(r_i)+\sum_{i=1}^{6}c_{2,i}x^2(r_i)$	8,7043	15,1555
$y(k_1,k_2)=c_0+\sum_{i=1}^{6}c_{1,i}x(r_i)+\sum_{i=1}^{6}c_{2,i}x^2(r_i)+\sum_{i=1}^{6}c_{2,i}x^3(r_i)+\sum_{i=1}^{6}c_{2,i}x^4(r_i)$	8,863	15,2931

Продолжение таблицы 4.1

Полная модель с усредненными значениями отсчетов на третьей-четвертой окружностях и на пятой-шестой окружностях	8,9613	15,6698
$y(k_1,k_2)=c_0+\sum_{i=1}^{6}c_i x(r_i)+\sum_{i=1}^{6}\sum_{j=1}^{6}c_{i,j}x(r_i)x(r_j)$	8,9870	15,2948
$y(k_1,k_2)=c_0+\sum_{i=1}^{3}c_{1,i}x(r_i)+\sum_{i=1}^{3}c_{2,i}x^2(r_i)+\sum_{i=1}^{3}c_{2,i}x^3(r_i)$	8,8932	16,8469
OpenCV	9,8238	19,3741

Таблица 4.2 – Значения СКО для различных фильтров при восстановлении изображений с показателем размытия σ =5

Фильтр	**СКО «Лена»**	**СКО «Город»**
$y(k_1,k_2)=c_0+\sum_{i=1}^{6}c_{1,i}x(r_i)+\sum_{i=1}^{6}c_{2,i}x^2(r_i)+\sum_{i=1}^{6}c_{2,i}x^3(r_i)$	11,6262	23,5168
$y(k_1,k_2)=c_0+\sum_{i=1}^{6}c_{1,i}x(r_i)+\sum_{i=1}^{6}c_{2,i}x^2(r_i)$	11,7534	23,5207
$y(k_1,k_2)=c_0+\sum_{i=1}^{6}c_{1,i}x(r_i)+\sum_{i=1}^{6}c_{2,i}x^2(r_i)+\sum_{i=1}^{6}c_{2,i}x^3(r_i)+\sum_{i=1}^{6}c_{2,i}x^4(r_i)$	11,7193	23,5356
Полная модель с усредненными значениями отсчетов на третьей-четвертой окружностях и на пятой-шестой окружностях	11,8041	23,6472
$y(k_1,k_2)=c_0+\sum_{i=1}^{6}c_i x(r_i)+\sum_{i=1}^{6}\sum_{j=1}^{6}c_{i,j}x(r_i)x(r_j)$	11,9505	23,7339
$y(k_1,k_2)=c_0+\sum_{i=1}^{3}c_{1,i}x(r_i)+\sum_{i=1}^{3}c_{2,i}x^2(r_i)+\sum_{i=1}^{3}c_{2,i}x^3(r_i)$	12,1636	23,6343
OpenCV	13,2166	23,8003

Приведенные численные результаты также показывают существенное преимущество применения построенного нелинейного фильтра. А упрощение

выбранной модели не несёт существенного ухудшения показателей качества восстановления.

ЗАКЛЮЧЕНИЕ

В данной работе была выполнена поставленная цель, а именно был разработан алгоритм нелинейной фильтрации на основе идентификации линейной по параметрам модели. Было также реализовано программное средство для реализации данного алгоритма и для визуального представления и оценивания результатов работы алгоритма.

Было теоретически описано преимущество использования алгоритма идентификации линейных по параметрам моделей, а также преимущество использования нелинейных фильтров. Был полностью описан процесс разработки алгоритма нелинейной фильтрации на основе идентификации линейной по параметрам модели, применимый для реставрации изображений. И в конце были представлены практические результаты использования данного алгоритма.

Был разработан массивно-многопоточный алгоритм восстановления изображений с помощью нелинейного фильтра в CUDA-среде.

Практически показано что использование линейной по параметрам модели нелинейного фильтра позволяет получить более высокое качество восстановления. Использование моделей различной размерности показывает также, что возможно получение хорошего качества при существенных упрощениях.

Работа прошла апробацию на конференции:

- Елкин Д.А., Фурсов В.А. Восстановление изображений нелинейными фильтрами, полученными идентификацией линейной по параметрам модели. // II Международная Конференция и молодёжная школа Информационные технологии и нанотехнологии (ИТНТ-2016), 17-19 мая 2016, Самара, Россия, издательство СГАУ, 2016 г.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Сойфер В. А. Методы компьютерной обработки изображений / Под ред. Сойфера В. А. 2-е изд., испр. М. Физматлит, 2003. 784 с.

2 Фурсов В. А. Восстановление изображений КИХ-фильтрами, построенными путем непосредственной идентификации инверсного тракта // Компьютерная оптика. Вып. 16. 1996. С. 103-108.

3 Фурсов В. А. Адаптивная идентификация по малому числу наблюдений [Текст] / Фурсов В.А. // Приложение к журналу «Информационные технологии» №9. 2013. 32 с.

4 Fursov V. Construction of adaptive identification algorithms, using the estimates conformity principle [Text] / V. Fursov // 11th International Conference on Pattern Recognition and Image Analysis: New Information Technologies (PRIA-11-2013). Samara, September 23-28, 2013. Conference Proceedings (Vol. I-II). 2013. V.1. P. 22-25.

5 Фурсов В. А. Два подхода к оценке точности и достоверности согласованной идентификации. / Труды X Международной конференции «Идентификация систем и задачи управления», Москва, 26-30 января 2015 г. М.: Институт проблем управления им. В.А. Трапезникова РАН, 2015. С. 907-918.

6 Льюнг Л. Идентификация систем. Теория для пользователя: Пер. с англ./ Под ред. Я.З. Цыпкина. М.: Наука. Гл. ред.физ.-мат. Лит., 1991. 432 с.

7 Щербаков М. А. Итерационный метод оптимальной нелинейной фильтрации изображений // Известия ВУЗов. Поволжский регион. Технические науки. 2011. №4. С. 43-56.

8 Прэтт У. Цифровая обработка изображений. М.: Мир, 1982. Т. 1. 312 с.

9 Jia Tse. Image processing with CUDA. University of Nevada, Las Vegas. August 2012. P. 56.

10 Bouman C. A. Digital Image Processing. January 12, 2015. P. 13.

11 Цифровая обработка изображений в информационных системах: Учебное пособие для студентов V курса РЭФ (специальности

«Радиотехника» и «Средства связи с подвижными объектами») / И. С. Грузман, В. С. Киричук, В. П. Косых, Г.И. Перетягин, А. А. Спектор. Новосибирск, 2000. 168 с.

12 Michael Johansen. Nonlinear filtering in digital image processing. Norwegian University of Science and Technology. August, 2011. P. 33.

13 Теоретические основы цифровой обработки изображений / В. А. Сойфер, В. В. Сергеев, С. Б. Попов, В.В. Мясников. Самара, 2000. 256 с.

14 Давыдов А. В. Цифровая обработка сигналов: обработка изображений / Электрон. текстовые данные – УГГУ – Режим доступа: http://prodav.exponenta.ru/dsp/index.html, свободный.

15 Даджион Д., Мерсеро Р. Цифровая обработка многомерных сигналов. М.: Мир, 1988. 488 с.

16 Гонсалес Р., Вудс Р., Эддинс С. Цифровая обработка изображений в среде MATLAB. М.: Техносфера, 2006. 616 с.

17 Гонсалес Р., Вудс Р. Цифровая обработка изображений. М.: Техносфера, 2005. 1072 с.

18 Куприянов А.В. Внешнее описание программных средств: методическое указание к лабораторной / Самар. гос. аэрокосм. ун-т. Самара, 2010. 24 с.

19 Куприянов А. В. Моделирование функциональной спецификации программного средства: методическое указание к лабораторной работе / Самар. гос. аэрокосм. ун-т. Самара, 2010. 20 с.

20 Куприянов А. В. Документирование функциональной спецификации программного средства: методическое указание к лабораторной работе / Самар. гос. аэрокосм. ун-т. Самара, 2010. 18 с.

21 Куприянов А. В. Функциональное моделирование архитектуры программного средства: методическое указание к лабораторной работе / Самар. гос. аэрокосм. ун-т. Самара, 2010. 20 с.

22 Куприянов А. В. Диаграммы потоков данных: методическое указание к лабораторной работе / Самар. гос. аэрокосм. ун-т. Самара, 2010. 20 с.

23 Куприянов А. В. Объектно-ориентированное проектирование: методическое указание к лабораторной работе / Самар. гос. аэрокосм. ун-т. Самара, 2010. 30 с.

24 Gonzalo R. Arce, Jan Baca, José L. Paredes. Nonlinear Filtering for Image Analysis and Enhancement. University of Delaware, Universidad de Los Andes. 2009. P. 292.

25 Елкин Д. А., Фурсов В. А. Восстановление изображений нелинейными фильтрами, полученными идентификацией линейной по параметрам модели / Материалы международной конференции и молодежной школы «Информационные технологии и нанотехнологии», Самара, 17-19 мая 2016 года. Самарский государственный аэрокосмический университет имени академика С.П. Королева, 2016. С. 549-555.

ПРИЛОЖЕНИЕ А

ТЕКСТ ПРОГРАММЫ

А.1 Код программы по построению фильтра

```
function [result] = getRandom(min, max, cnt)
    result = randi([min, max], 1, cnt);
end


function [result] = final_solver(I, blur_I, samples_count)
    length_a = 20;
    new_I = double(I);
    new_blur_I = double(blur_I);
    X = zeros(samples_count, length_a);
    iter = 0;
    while rank(X) ~= length_a
        y = zeros(samples_count, 1);
        min_image_size = min(length(I(1,:)), length(I(:,1)));
        image_index_i = getRandom(4, min_image_size - 3, samples_count);
        image_index_j = getRandom(4, min_image_size - 3, samples_count);
        for i = 1 : samples_count
            y(i, 1) = new_I(image_index_i(i), image_index_j(i));
            src_i = image_index_i(i);
            src_j = image_index_j(i);
            b1 = new_blur_I(src_i, src_j);
            b2 = (new_blur_I(src_i + 1, src_j) + new_blur_I(src_i - 1, src_j)
+ new_blur_I(src_i, src_j + 1) + new_blur_I(src_i, src_j - 1)) / 4;
            b3 = (new_blur_I(src_i + 1, src_j - 1) + new_blur_I(src_i - 1,
src_j + 1) + new_blur_I(src_i + 1, src_j + 1) + new_blur_I(src_i - 1, src_j -
1)) / 4;
            b4 = (new_blur_I(src_i + 2, src_j) + new_blur_I(src_i - 2, src_j)
+ new_blur_I(src_i, src_j - 2) + new_blur_I(src_i, src_j + 2)) / 4;
            b5 = (new_blur_I(src_i + 1, src_j + 2) + new_blur_I(src_i + 1,
src_j - 2) + new_blur_I(src_i - 1, src_j + 2) + new_blur_I(src_i - 1, src_j -
2) + new_blur_I(src_i + 2, src_j - 1) + new_blur_I(src_i + 2, src_j + 1) +
new_blur_I(src_i - 2, src_j + 1) + new_blur_I(src_i - 2, src_j - 1)) / 8;
            b6 = (new_blur_I(src_i + 2, src_j - 2) + new_blur_I(src_i - 2,
src_j + 2) + new_blur_I(src_i + 2, src_j + 2) + new_blur_I(src_i - 2, src_j -
2)) / 4;
            b7 = (new_blur_I(src_i + 3, src_j + 3) + new_blur_I(src_i + 3,
src_j - 3) + new_blur_I(src_i - 3, src_j + 3) + new_blur_I(src_i - 3, src_j -
3)) / 4;
            X(i, :) = [1 b1 b2 b3 b4 b5 b6 b7 ...
                b2^2 b2^3 ...
                b3^2 b3^3 ...
                b4^2 b4^3 ...
                b5^2 b5^3 ...
                b6^2 b6^3 ...
                b7^2 b7^3];
        end;
        iter = iter + 1;
        disp(iter);
        disp(rank(X));
    end;
    result = X \ y;
%     result = result ./ sum(result);
end


function [result] = final_unblur(blur_I, a)
```

```
    result = blur_I;
    temp = double(result);
    for i = 4 : length(blur_I(:,1)) - 3
        for j = 4 : length(blur_I(1,:)) - 3
            b1 = temp(i, j);
            b2 = (temp(i + 1, j) + temp(i - 1, j) + temp(i, j + 1) + temp(i,
j - 1)) / 4;
            b3 = (temp(i + 1, j - 1) + temp(i - 1, j + 1) + temp(i + 1, j +
1) + temp(i - 1, j - 1)) / 4;
            b4 = (temp(i + 2, j) + temp(i - 2, j) + temp(i, j - 2) + temp(i,
j + 2)) / 4;
            b5 = (temp(i + 1, j + 2) + temp(i + 1, j - 2) + temp(i - 1, j +
2) + temp(i - 1, j - 2) + temp(i + 2, j - 1) + temp(i + 2, j + 1) + temp(i -
2, j + 1) + temp(i - 2, j - 1)) / 8;
            b6 = (temp(i + 2, j - 2) + temp(i - 2, j + 2) + temp(i + 2, j +
2) + temp(i - 2, j - 2)) / 4;
            b7 = (temp(i + 3, j + 3) + temp(i + 3, j - 3) + temp(i - 3, j +
3) + temp(i - 3, j - 3)) / 4;
            func = [1 b1 b2 b3 b4 b5 b6 b7 ...
                b2^2 b2^3 ...
                b3^2 b3^3 ...
                b4^2 b4^3 ...
                b5^2 b5^3 ...
                b6^2 b6^3 ...
                b7^2 b7^3];
            result(i, j) = sum(func .* a');
        end
    end
end


clear;
dispersion = 5;
I = imread('lena.jpg');
if length(I(1,1,:)) > 1
    temp_I = I(:,:,1);
    I=temp_I;
end
figure;
imshow(I);
disp('now img monochrome');
H = fspecial('gaussian', length(I(:, 1)), dispersion);
blur_I = imfilter(I, H, 'replicate');
disp('img blurred');
figure;
imshow(blur_I);
a = final_solver(I, blur_I, 500);
disp('solver solved');
unblur_I = final_unblur(blur_I, a);
disp('img unblurred');
figure;
imshow(unblur_I)

shift = 4;
sum = 0;
count = 0;
I = double(I);
unblur_I = double(unblur_I);
for i = shift : 1 : (length(I(:, 1)) - shift)
    for j = shift : 1 : (length(I(1, :)) - shift)
        sum = sum + (I(i, j) - unblur_I(i, j))^2;
        count = count + 1;
    end;
end;
```

```
deviation = sqrt(1 / count * sum);
```

A.2 Код программы по восстановлению изображения

```
#ifndef _BOXFILTER_KERNEL_CH_
#define _BOXFILTER_KERNEL_CH_

#include <helper_math.h>
#include "device_launch_parameters.h"
// CUDA utilities and system includes
#include <cuda_runtime.h>

// Helper functions
#include <helper_functions.h>  // CUDA SDK Helper functions
#include <helper_cuda.h>       // CUDA device initialization helper functions

#include <stdio.h>
#include <stdlib.h>

#include <iostream>

#if defined(WIN32) || defined(_WIN32) || defined(WIN64) || defined(_WIN64)
#   pragma warning( disable : 4996 ) // disable deprecated warning
#endif

#pragma pack(1)

typedef struct
{
	short type;
	int size;
	short reserved1;
	short reserved2;
	int offset;
} BMPHeader;

typedef struct
{
	int size;
	int width;
	int height;
	short planes;
	short bitsPerPixel;
	unsigned compression;
	unsigned imageSize;
	int xPelsPerMeter;
	int yPelsPerMeter;
	int clrUsed;
	int clrImportant;
} BMPInfoHeader;

const char *image_filename = "lena_24-bit.bmp";
unsigned int width, height;
unsigned int *h_img = NULL;

struct cudaGraphicsResource *cuda_pbo_resource; // handles OpenGL-CUDA exchange

StopWatchInterface *kernel_timer = NULL;

// Auto-Verification Code
unsigned int g_TotalErrors = 0;
```

```
int   *pArgc = NULL;
char **pArgv = NULL;

extern "C" int  runSingleTest(char *ref_file, char *exec_path);
extern "C" void loadImageData(int argc, char **argv);

// These are CUDA functions to handle allocation and launching the kernels
extern "C" void   initTexture(int width, int height, void *pImage);

extern "C" double unblur(unsigned int *d_dest, int width, int height, StopWatchInterface
*timer, float *a);

extern "C" void LoadBMPFile(uchar4 **dst, unsigned int *width, unsigned int *height,
const char *name);

texture<uchar4, 2, cudaReadModeNormalizedFloat> tex;
cudaArray *d_array;

////////////////////////////////////////////////////////////////////////////////
// These are CUDA Helper functions

// This will output the proper CUDA error strings in the event that a CUDA host call
returns an error
#define checkCudaErrors(err)           __checkCudaErrors (err, __FILE__, __LINE__)

inline void __checkCudaErrors(cudaError err, const char *file, const int line)
{
       if (cudaSuccess != err)
       {
              fprintf(stderr, "%s(%i) : CUDA Runtime API error %d: %s.\n", file, line,
(int)err, cudaGetErrorString(err));
              exit(EXIT_FAILURE);
       }
}

// RGBA version
// reads from 32-bit unsigned int array holding 8-bit RGBA

// convert floating point rgba color to 32-bit integer
__device__ unsigned int rgbaFloatToInt(float4 rgba)
{
       rgba.x = __saturatef(rgba.x);   // clamp to [0.0, 1.0]
       rgba.y = __saturatef(rgba.y);
       rgba.z = __saturatef(rgba.z);
       rgba.w = __saturatef(rgba.w);
       return ((unsigned int)(rgba.w * 255.0f) << 24) |
              ((unsigned int)(rgba.z * 255.0f) << 16) |
              ((unsigned int)(rgba.y * 255.0f) << 8) |
              ((unsigned int)(rgba.x * 255.0f));
}

__device__ float4 rgbaIntToFloat(unsigned int c)
{
       float4 rgba;
       rgba.x = (c & 0xff) * 0.003921568627f;       //  /255.0f;
       rgba.y = ((c >> 8) & 0xff) * 0.003921568627f;  //  /255.0f;
       rgba.z = ((c >> 16) & 0xff) * 0.003921568627f; //  /255.0f;
       rgba.w = ((c >> 24) & 0xff) * 0.003921568627f; //  /255.0f;
       return rgba;
}

__global__ void
apply_linear(unsigned int* result, int width, int height, float* a)
{
       int i = blockIdx.x * blockDim.x + threadIdx.x;
```

```
	int j = blockIdx.y * blockDim.y + threadIdx.y;

	if (j < 3 || i < 3 || j >= height - 4 || i >= width - 4)
	{
		return;
	}

	float4 sum = make_float4(a[0]);
	float4 b1 = tex2D(tex, i, j);
	float4 b2 = (tex2D(tex, i + 1, j) + tex2D(tex, i - 1, j) + tex2D(tex, i, j + 1) +
tex2D(tex, i, j - 1)) / 4;
	float4 b3 = (tex2D(tex, i + 1, j - 1) + tex2D(tex, i - 1, j + 1) + tex2D(tex, i +
1, j + 1) + tex2D(tex, i - 1, j - 1)) / 4;
	float4 b4 = (tex2D(tex, i + 2, j) + tex2D(tex, i - 2, j) + tex2D(tex, i, j - 2) +
tex2D(tex, i, j + 2)) / 4;
	float4 b5 = (tex2D(tex, i + 1, j + 2) + tex2D(tex, i + 1, j - 2) + tex2D(tex, i -
1, j + 2) + tex2D(tex, i - 1, j - 2) + tex2D(tex, i + 2, j - 1) + tex2D(tex, i + 2, j +
1) + tex2D(tex, i - 2, j + 1) + tex2D(tex, i - 2, j - 1)) / 8;
	float4 b6 = (tex2D(tex, i + 2, j - 2) + tex2D(tex, i - 2, j + 2) + tex2D(tex, i +
2, j + 2) + tex2D(tex, i - 2, j - 2)) / 4;
	float4 b7 = (tex2D(tex, i + 3, j + 3) + tex2D(tex, i + 3, j - 3) + tex2D(tex, i -
3, j + 3) + tex2D(tex, i - 3, j - 3)) / 4;
	sum += a[1] * b1;
	sum += a[2] * b2;
	sum += a[3] * b3;
	sum += a[4] * b4;
	sum += a[5] * b5;
	sum += a[6] * b6;
	sum += a[7] * b7;

	result[j * width + i] = rgbaFloatToInt(sum);
}

__global__ void
apply_square(unsigned int* result, int width, int height, float* a)
{
	int i = blockIdx.x * blockDim.x + threadIdx.x;
	int j = blockIdx.y * blockDim.y + threadIdx.y;

	if (j < 3 || i < 3 || j >= height - 4 || i >= width - 4)
	{
		return;
	}

	int offset = 7;

	float4 sum = make_float4(a[0]);
	float4 b2 = (tex2D(tex, i + 1, j) + tex2D(tex, i - 1, j) + tex2D(tex, i, j + 1) +
tex2D(tex, i, j - 1)) / 4;
	float4 b3 = (tex2D(tex, i + 1, j - 1) + tex2D(tex, i - 1, j + 1) + tex2D(tex, i +
1, j + 1) + tex2D(tex, i - 1, j - 1)) / 4;
	float4 b4 = (tex2D(tex, i + 2, j) + tex2D(tex, i - 2, j) + tex2D(tex, i, j - 2) +
tex2D(tex, i, j + 2)) / 4;
	float4 b5 = (tex2D(tex, i + 1, j + 2) + tex2D(tex, i + 1, j - 2) + tex2D(tex, i -
1, j + 2) + tex2D(tex, i - 1, j - 2) + tex2D(tex, i + 2, j - 1) + tex2D(tex, i + 2, j +
1) + tex2D(tex, i - 2, j + 1) + tex2D(tex, i - 2, j - 1)) / 8;
	float4 b6 = (tex2D(tex, i + 2, j - 2) + tex2D(tex, i - 2, j + 2) + tex2D(tex, i +
2, j + 2) + tex2D(tex, i - 2, j - 2)) / 4;
	float4 b7 = (tex2D(tex, i + 3, j + 3) + tex2D(tex, i + 3, j - 3) + tex2D(tex, i -
3, j + 3) + tex2D(tex, i - 3, j - 3)) / 4;
	sum += a[offset + 1] * b2 * b2;
	sum += a[offset + 2] * b3 * b3;
	sum += a[offset + 3] * b4 * b4;
	sum += a[offset + 4] * b5 * b5;
	sum += a[offset + 5] * b6 * b6;
```

```
	sum += a[offset + 6] * b7 * b7;

	result[j * width + i] = rgbaFloatToInt(sum);
}

__global__ void
apply_cube(unsigned int* result, int width, int height, float* a)
{
	int i = blockIdx.x * blockDim.x + threadIdx.x;
	int j = blockIdx.y * blockDim.y + threadIdx.y;

	if (j < 3 || i < 3 || j >= height - 4 || i >= width - 4)
	{
		return;
	}

	int offset = 13;

	float4 sum = make_float4(a[0]);
	float4 b2 = (tex2D(tex, i + 1, j) + tex2D(tex, i - 1, j) + tex2D(tex, i, j + 1) +
tex2D(tex, i, j - 1)) / 4;
	float4 b3 = (tex2D(tex, i + 1, j - 1) + tex2D(tex, i - 1, j + 1) + tex2D(tex, i +
1, j + 1) + tex2D(tex, i - 1, j - 1)) / 4;
	float4 b4 = (tex2D(tex, i + 2, j) + tex2D(tex, i - 2, j) + tex2D(tex, i, j - 2) +
tex2D(tex, i, j + 2)) / 4;
	float4 b5 = (tex2D(tex, i + 1, j + 2) + tex2D(tex, i + 1, j - 2) + tex2D(tex, i -
1, j + 2) + tex2D(tex, i - 1, j - 2) + tex2D(tex, i + 2, j - 1) + tex2D(tex, i + 2, j +
1) + tex2D(tex, i - 2, j + 1) + tex2D(tex, i - 2, j - 1)) / 8;
	float4 b6 = (tex2D(tex, i + 2, j - 2) + tex2D(tex, i - 2, j + 2) + tex2D(tex, i +
2, j + 2) + tex2D(tex, i - 2, j - 2)) / 4;
	float4 b7 = (tex2D(tex, i + 3, j + 3) + tex2D(tex, i + 3, j - 3) + tex2D(tex, i -
3, j + 3) + tex2D(tex, i - 3, j - 3)) / 4;
	sum += a[offset + 1] * b2 * b2 * b2;
	sum += a[offset + 2] * b3 * b3 * b3;
	sum += a[offset + 3] * b4 * b4 * b4;
	sum += a[offset + 4] * b5 * b5 * b5;
	sum += a[offset + 5] * b6 * b6 * b6;
	sum += a[offset + 6] * b7 * b7 * b7;

	result[j * width + i] = rgbaFloatToInt(sum);
}

__global__ void
apply_simple(unsigned int* result, int width, int height, float* a)
{
	int i = blockIdx.x * blockDim.x + threadIdx.x;
	int j = blockIdx.y * blockDim.y + threadIdx.y;

	if (j < 3 || i < 3 || j >= height - 4 || i >= width - 4)
	{
		return;
	}

	float4 sum = make_float4(a[0]);
	float4 b1 = tex2D(tex, i, j);
	float4 b2 = (tex2D(tex, i + 1, j) + tex2D(tex, i - 1, j) + tex2D(tex, i, j + 1) +
tex2D(tex, i, j - 1)) / 4;
	float4 b3 = (tex2D(tex, i + 1, j - 1) + tex2D(tex, i - 1, j + 1) + tex2D(tex, i +
1, j + 1) + tex2D(tex, i - 1, j - 1)) / 4;
	float4 b4 = (tex2D(tex, i + 2, j) + tex2D(tex, i - 2, j) + tex2D(tex, i, j - 2) +
tex2D(tex, i, j + 2)) / 4;
	float4 b5 = (tex2D(tex, i + 1, j + 2) + tex2D(tex, i + 1, j - 2) + tex2D(tex, i -
1, j + 2) + tex2D(tex, i - 1, j - 2) + tex2D(tex, i + 2, j - 1) + tex2D(tex, i + 2, j +
1) + tex2D(tex, i - 2, j + 1) + tex2D(tex, i - 2, j - 1)) / 8;
```

```
	float4 b6 = (tex2D(tex, i + 2, j - 2) + tex2D(tex, i - 2, j + 2) + tex2D(tex, i +
2, j + 2) + tex2D(tex, i - 2, j - 2)) / 4;
	float4 b7 = (tex2D(tex, i + 3, j + 3) + tex2D(tex, i + 3, j - 3) + tex2D(tex, i -
3, j + 3) + tex2D(tex, i - 3, j - 3)) / 4;
	//linear
	sum += a[1] * b1;
	sum += a[2] * b2;
	sum += a[3] * b3;
	sum += a[4] * b4;
	sum += a[5] * b5;
	sum += a[6] * b6;
	sum += a[7] * b7;

	//square
	int offset = 7;
	sum += a[offset + 1] * b2 * b2;
	sum += a[offset + 2] * b3 * b3;
	sum += a[offset + 3] * b4 * b4;
	sum += a[offset + 4] * b5 * b5;
	sum += a[offset + 5] * b6 * b6;
	sum += a[offset + 6] * b7 * b7;

	//cube
	offset = 13;
	sum += a[offset + 1] * b2 * b2 * b2;
	sum += a[offset + 2] * b3 * b3 * b3;
	sum += a[offset + 3] * b4 * b4 * b4;
	sum += a[offset + 4] * b5 * b5 * b5;
	sum += a[offset + 5] * b6 * b6 * b6;
	sum += a[offset + 6] * b7 * b7 * b7;

	result[j * width + i] = rgbaFloatToInt(sum / 37);
}

__global__ void
d_blur(unsigned int* result, int width, int height) {
	int col = blockIdx.x * blockDim.x + threadIdx.x;
	int row = blockIdx.y * blockDim.y + threadIdx.y;

	if (row < 6 || col < 6 || row >= height - 7 || col >= width - 7)
		return;

	int mask[7][7] = { 1, 2, 3, 4, 3, 2, 1, 2, 5, 6, 7, 6, 5, 2, 3, 6, 8, 9, 8, 6, 3,
4, 7, 9, 10, 9, 7, 4, 3, 6, 8, 9, 8, 6, 3, 2, 5, 6, 7, 6, 5, 2, 1, 2, 3, 4, 3, 2, 1 };

	float4 sum = make_float4(0.0f);
	for (int j = -3; j <= 3; j++) {
		for (int i = -3; i <= 3; i++) {
			//int color = arr[(row + j) * width + (col + i)];
			sum += tex2D(tex, col + i, row + j) * mask[i + 3][j + 3];
		}
	}

	result[row * width + col] = rgbaFloatToInt(sum / 234);

}

extern "C"
void initTexture(int width, int height, void *pImage)
{
	int size = width * height * sizeof(uchar4);

	// copy image data to array
	cudaChannelFormatDesc channelDesc;
```

```
        channelDesc = cudaCreateChannelDesc(8, 8, 8, 8, cudaChannelFormatKindUnsigned);

        checkCudaErrors(cudaMallocArray(&d_array, &channelDesc, width, height));
        checkCudaErrors(cudaMemcpyToArray(d_array, 0, 0, pImage, size,
cudaMemcpyHostToDevice));

        // Bind the array to the texture
        checkCudaErrors(cudaBindTextureToArray(tex, d_array, channelDesc));
}

/*
Unblur image using CUDA

Parameters:
d_src  - pointer to input image in device memory
d_dest - pointer to destination image in device memory
width  - image width
height - image height

*/
// RGBA version
extern "C"
double unblur(unsigned int *d_dest, int width, int height, StopWatchInterface *timer,
float *a)
{
        checkCudaErrors(cudaBindTextureToArray(tex, d_array));

        // var for kernel computation timing
        double dKernelTime;

        // sync host and start kernel computation timer_kernel
        dKernelTime = 0.0;
        checkCudaErrors(cudaDeviceSynchronize());
        sdkResetTimer(&timer);

        dim3 block(16, 16);
        dim3 grid(width / 16, height / 16);
        //d_blur<<< grid, block >>>(d_dest, width, height);
        apply_simple<<< grid, block >>>(d_dest, width, height, a);

        cudaThreadSynchronize();

        checkCudaErrors(cudaThreadSynchronize());

        // sync host and stop computation timer_kernel
        checkCudaErrors(cudaDeviceSynchronize());
        dKernelTime += sdkGetTimerValue(&timer);

        return dKernelTime;
}

void initCuda()
{
        // Refer to boxFilter_kernel.cu for implementation
        initTexture(width, height, h_img);

        sdkCreateTimer(&kernel_timer);
}

void cleanup()
{
        sdkDeleteTimer(&kernel_timer);

        if (h_img)
        {
```

```
            free(h_img);
            h_img = NULL;
        }

        // Refer to boxFilter_kernel.cu for implementation
        checkCudaErrors(cudaFreeArray(d_array));

        cudaGraphicsUnregisterResource(cuda_pbo_resource);

        // cudaDeviceReset causes the driver to clean up all state. While
        // not mandatory in normal operation, it is good practice.  It is also
        // needed to ensure correct operation when the application is being
        // profiled. Calling cudaDeviceReset causes all profile data to be
        // flushed before the application exits
        cudaDeviceReset();
}

// This test specifies a single test
int runSingleTest(char *ref_file, char *exec_path)
{
        int nTotalErrors = 0;
        char dump_file[256];

        initCuda();
        sdkStartTimer(&kernel_timer);

        unsigned int *d_result;
        unsigned int *h_result = (unsigned int *)malloc(width * height * sizeof(unsigned
int));
        checkCudaErrors(cudaMalloc((void **)&d_result, width*height*sizeof(unsigned
int)));

        float *a = (float *)malloc(20 * sizeof(float));
        float *coefficients;
        checkCudaErrors(cudaMalloc((void **)&coefficients, 20 * sizeof(float)));
        checkCudaErrors(cudaMemcpy((float *)coefficients, (float *)a, 20 * sizeof(float),
cudaMemcpyHostToDevice));

        // run the sample radius
        {
            double spentTime = unblur(d_result, width, height, kernel_timer,
coefficients);
            std::cout << "Time spent: " << spentTime << std::endl;
            // check if kernel execution generated an error
            getLastCudaError("Error: unblur Kernel execution FAILED");
            checkCudaErrors(cudaDeviceSynchronize());

            // readback the results to system memory
            cudaMemcpy((unsigned char *)h_result, (unsigned char *)d_result,
width*height*sizeof(unsigned int), cudaMemcpyDeviceToHost);

            sprintf(dump_file, "lenaRGB_result.ppm");

            sdkSavePPM4ub((const char *)dump_file, (unsigned char *)h_result, width,
height);
        }
        printf("\n");

        free(h_result);
        free(a);
        checkCudaErrors(cudaFree(d_result));
        checkCudaErrors(cudaFree(coefficients));

        sdkStopTimer(&kernel_timer);
        cleanup();
```

```
        return nTotalErrors;
}

void loadImageData(int argc, char **argv)
{
        LoadBMPFile((uchar4 **)&h_img, &width, &height, image_filename);
        //sdkLoadPPM4(image_filename, (unsigned char **)&h_img, &width, &height);

        if (!h_img)
        {
                printf("Error opening file '%s'\n", image_filename);
                exit(EXIT_FAILURE);
        }

        printf("Loaded '%s', %d x %d pixels\n", image_filename, width, height);
}

extern "C" void LoadBMPFile(uchar4 **dst, unsigned int *width, unsigned int *height,
const char *name)
{
        BMPHeader hdr;
        BMPInfoHeader infoHdr;
        int x, y;

        FILE *fd;


        printf("Loading %s...\n", name);

        if (sizeof(uchar4) != 4)
        {
                printf("***Bad uchar4 size***\n");
                exit(EXIT_SUCCESS);
        }

        if (!(fd = fopen(name, "rb")))
        {
                printf("***BMP load error: file access denied***\n");
                exit(EXIT_SUCCESS);
        }

        fread(&hdr, sizeof(hdr), 1, fd);

        if (hdr.type != 0x4D42)
        {
                printf("***BMP load error: bad file format***\n");
                exit(EXIT_SUCCESS);
        }

        fread(&infoHdr, sizeof(infoHdr), 1, fd);

        if (infoHdr.bitsPerPixel != 24)
        {
                printf("***BMP load error: invalid color depth***\n");
                exit(EXIT_SUCCESS);
        }

        if (infoHdr.compression)
        {
                printf("***BMP load error: compressed image***\n");
                exit(EXIT_SUCCESS);
        }

        *width = infoHdr.width;
```

```
        *height = infoHdr.height;
        *dst = (uchar4 *)malloc(*width **height * 4);

        printf("BMP width: %u\n", infoHdr.width);
        printf("BMP height: %u\n", infoHdr.height);

        fseek(fd, hdr.offset - sizeof(hdr) - sizeof(infoHdr), SEEK_CUR);

        for (y = 0; y < infoHdr.height; y++)
        {
                for (x = 0; x < infoHdr.width; x++)
                {
                        (*dst)[(y * infoHdr.width + x)].w = 0;
                        (*dst)[(y * infoHdr.width + x)].z = fgetc(fd);
                        (*dst)[(y * infoHdr.width + x)].y = fgetc(fd);
                        (*dst)[(y * infoHdr.width + x)].x = fgetc(fd);
                }

                for (x = 0; x < (4 - (3 * infoHdr.width) % 4) % 4; x++)
                {
                        fgetc(fd);
                }
        }


        if (ferror(fd))
        {
                printf("***Unknown BMP load error.***\n");
                free(*dst);
                exit(EXIT_SUCCESS);
        }
        else
        {
                printf("BMP file loaded successfully!\n");
        }

        fclose(fd);
}

////////////////////////////////////////////////////////////////////////////////
// Program main
////////////////////////////////////////////////////////////////////////////////
int
main(int argc, char **argv)
{
        int devID = 0;
        char *ref_file = NULL;

        pArgc = &argc;
        pArgv = argv;

        // start logs
        printf("%s Starting...\n\n", argv[0]);

        // load image to process
        loadImageData(argc, argv);

        // This overrides the default mode.  Users can specify the radius used by the
filter kernel
        devID = findCudaDevice(argc, (const char **)argv);
        g_TotalErrors += runSingleTest(ref_file, argv[0]);
        exit(g_TotalErrors == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
}

#endif // #ifndef _BOXFILTER_KERNEL_H_
```