САНКТ – ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Разработка и реализация алгоритмов обработки изображений с анатомическими особенностями на основе HTML 5.0

Дипломная работа студента 545 группы

Добролеж Анны Борисовны

Научный руководитель	…… /подпись/	старший преподаватель И. Г. Антипов
Рецензент	…… /подпись/	аспирант А. Г. Петров
“Допустить к защите”, заведующий кафедрой	…… /подпись/	д.ф.-м.н., проф. А.Н. Терехов

Санкт-Петербург

2012

St. Petersburg State University
Mathematics and Mechanics Faculty

Software Engineering Department

Development and implementation of image processing algorithms with anatomical feature based on HTML 5.0

Graduate paper by
Dobrolezh Anna
545 group

Scientific advisor	..... /signature/	Senior Lect. I. G. Antipov
Reviewer	..... /signature/	A.G. Petrov
“Approved by” Head of the Department	..... /signature/	Professor A.N.Terekhov

Saint Petersburg
2012

Введение

В 21 веке все медицинские процессы ускоряются. К примеру, ЭКГ можно снять дома, просто приложив к себе мобильное устройство с датчиком и передав данные врачу за считанные секунды. Устройства, с помощью которых производится диагностирование и мониторинг, уменьшаются в размере, цены на диагностику и прогнозирование падают.

Многое из вышесказанного верно и про пластическую хирургию. Компьютерное имитирование операций в этой области позволяет наглядно показать клиенту эффективность принятых мер, а так же дать ему почувствовать эффект от хирургического вмешательства непосредственно до самого вмешательства.

Создание редакторов 2D изображений для различных косметических процедур и пластической хирургии являются интересной областью разработки. Особенностью таких редакторов является возможность добавления информации об анатомии лица человека, такой как различные фильтры, маски, выявленные алгоритмы старения, ограничения на цвет и форму различных областей изображения.

В качестве одного из примеров таких редакторов можно рассмотреть приложение AlterImage, позволяющее прогнозировать результаты пластических операций. Несмотря на солидный набор функций, приложение обладает очевидным недостатком: оно написано только под платформу Microsoft Windows, и для запуска его на других операционных системах приходится идти на ухищрения, сложно выполнимые для рядового пользователя: например, пользователям Mac OS X предлагается установить виртуальную машину VMWare и пользоваться приложением из-под нее. Кроме того, постоянное обновление версии программы требует дополнительных усилий системных администраторов.

Постановка задачи

В данной работе основными задачами являются косметические операции по удалению морщин и подтяжке овала лица (изменение контура), а так же подготовка framework’а для легкого расширения на новые хирургические операции и части тела.

В работе планировалось обобщение некоторых достижений в прогнозировании операций и портирование их на современную площадку. Несмотря на то, что на данных момент уже существуют средства и инструменты, позволяющие имитировать и демонстрировать клиентам результаты таких пластических операций, как "подтяжка лица" и морщин (например, упомянутый выше AlterImage), все эти инструменты реализованы под специфические платформы и зачастую требуют вмешательства квалифицированных специалистов для их изначальной установки и настройки. Эти средства привередливы к программному обеспечению и рабочим станциям, на которых они исполняются, что зачастую приводит к необходимости покупки новых компьютеров в угоду требованиям программ. Разработка новых версий этих приложений под другие платформы трудоемка, не говоря уже об адаптации этих средств под мобильные платформы и устройства.

Целью дипломной работы было создать графический редактор 2D-изображений, который бы был лишен этих недостатков, был бы легко расширяемым и который был бы разработан с акцентом на развитие современных технологий обработки данных. Создаваемый проект должен работать на практически любой операционной системе, будь то популярная Microsoft Windows, редкий Mac OS X или свободно поставляющийся Linux. Такая кроссплатформенность сделает его удобным в использовании и практически не накладывает ограничений на рабочие станции клиентов. В связи с бурным развитием мобильных технологий и широкого в них проникновения HTML5 можно ожидать, что в скором будущем приложение,

разработанное в рамках дипломной работы, сможет быть успешно запущено на планшетных компьютерах и других мобильных устройствах, что, безусловно, сделает его исключительно удобным в использовании практикующими врачами.

Обзор существующих решений

Мотивацией для выполнения данной работы служило большое количество типовых операций, выполняемых пластическим хирургом. Несмотря на то, что данные операции являются однотипными, полная их автоматизация на данном этапе развития технологи представляется очень трудоемкой. Примером таких операций является имитация и демонстрация результатов пластической хирургии. С внедрением высоких технологий пластические операции могут быть спрогнозированы с помощью упрощенных, полнофункциональных или специализированных редакторов.
Рассмотрим возможность и трудоемкость использования каждого из них для имитации пластической операции.

Упрощенный редактор.

В качестве примера рассмотрим редактор MSPaint, доступный всем пользователям операционной системы Windows.
При освоении редактор не требует особых знаний и умений, все доступные в нем инструменты предельно просты и доступны для освоения. Однако большим минусом данного варианта является то, что основные необходимые хирургу операции либо требуют применения нескольких инструментов совместно, либо вообще не могут быть выполнены посредством данного редактора. Примером невыполнимой операции может служить имитация подтяжки овала лица.

Полнофункциональный редактор.

В качестве примера рассмотрим редактор Adobe Photoshop. Данный редактор имеет очень богатую функциональность и может выполнять качественно практически все доступные на данном этапа развития компьютерных технологий действия с изображениями. Однако минусом этого решения является тяжеловесность редактора, высокие требования к операционным

ресурсам и медлительность в случае использования маломощного компьютера.

Специализированные редакторы.

Специализированные редакторы позволяют оптимизировать и упрощать операции по имитированию результатов пластических операций и являются оптимальным выбором в данном случае. Однако существующие решения имеют проприетарный характер, дороги для лицензирования и в большинстве своем не обладают кроссплатформенностью.

Примером подобного продукта может быть AlterImage 2.0, выпускающийся компанией Seattle Software Design на платной основе. На данный момент цена на продукт превышает 1000 долларов США. AlterImage – это специализированный графический редактор, приспособленный для имитации косметических изменений с лице и теле человека. Программа реализует три основных инструмента – «Warp» (деформирование), «Stretch» (растяжение), and «Smooth» (сглаживание). По сравнению с полнофункциональными редакторами изображений, такими как Photoshop, AlterImage проще в освоении и использовании, имеет более легковесный интерфейс. Программа позволяет производить манипуляции с изображением, изменение формы лица и тела, необходимые для имитации основных пластических операций. Однако данное программное обеспечение реализовано исключительно под операционную систему Windows. В связи с растущей доле на рынке мобильных устройств (в данном случае наиболее актуальны планшетные компьютеры), а так же растущей долей операционной системы Macintosh[12], этот недостаток становится серьезной помехой в использовании программы.

Альтернативные javascript-фреймворки

Ввиду направленности разрабатываемого web-приложения на обработку изображений, было проведено исследование с целью выяснить, какой функционал предлагают фреймворки для обработки изображений на языке JavaScript. Для этого были рассмотрены следующие наиболее популярные библиотеки для обработки изображений

- pixastic http://www.pixastic.com/
- processingjs http://processingjs.org/
- camanjs http://camanjs.com/

1)Pixastic

Библиотека pixastic предоставляет серию высокоуровнеых фильтров изображений. Каждый фильтр может быть наложен как на все изображение сразу, так и на некоторую его прямоугольную область. Однако фильтры, предлагаемые библиотекой, носят самый общий характер, и не обладают достаточной гибкостью настроек для использования в целях имитации пластических операций. Дополнительным препятствием было возможность использования фильтров только в рамках прямоугольных областей.
Тем не менее, некоторые идеи, замеченные при анализе исходного кода фильтров фреймворка, были использованы в дальнейшем в ходе работы над приложением.

2) ProcessingJS

Библиотека processingJS изначально была разработана как порт языка Processing для использование в сети Интернет. Библиотека обладает следующими возможностями:

- Преобразование файлов на языке Processing в серию JavaScript методов, с целью их дальнейшего отображения на заданном элементе canvas.

- Использование API библиотеки для создания изображений средствами чистого JavaScript. Это API является высокоуровневой абстракцией для рисования на элементе canvas

Разработчики библиотеки отдельно подчеркивают [16], что высокоуровневое API не является основным направлением развития библиотеки, и оно не было разработано с целью предоставления высокоуровневых методов для рисования общего назначения. В свою очередь, основной сценарий использования движка предполагает рендеринг кода языка Processing для создания изображений и логотипов, и не предназначен для описания преобразований изображений.

3)CamanJS

Библиотека CamanJS написана на языке CoffeeScript и предоставляет широкие возможности по фильтрации изображений. К сожалению, эта библиотека не предоставляет возможности обрабатывать только части изображения. Итерирование по всем точкам изображения при достаточно больших размерах изображения занимает достаточно много времени, а потому ее использование в данной работе не целесообразно.

Архитектура

Альтернативные решения

1. Desktop-приложение

Desktop-приложения исторически первый тип программного обеспечения, который появился на персональных компьютерах. Эти приложения дают разработчикам определенные достоинства: они позволяют добиваться очень большой производительности при выполнении трудоемких задач за счет использования низкоуровневых команд и функций. В то же время с ними сопряжено большое количество недостатков, о которых, ввиду длинной истории этого типа программного обеспечения, хорошо известно.
Перечислим некоторые из этих недостатков:

- Проблема поддержки разных версий программного обеспечения
- Проблема кроссплатформенности программного обеспечения

Решение этих недостатков отнимает большое количество времени и ресурсов: необходимо разрабатывать специальный код для автоматического обновления программ, а в случае разработки кроссплатформенного приложения архитектура должна быть разработана таким образом, чтобы минимизировать время для портирования кода с одной платформы на другую.
Так как одним из основных сценариев применения приложения было использование его как на стационарных компьютерах в больницах, так и на планшетных компьютерах, то от разработки Desktop-приложения было решено отказаться.

2. Flash

Технология Flash является очень популярной, распространение которой, тем не менее, падает в последние годы в связи с переходом на стандарт HTML5. Flash позволяет писать код на языке программирования ActionScript, который затем будет интерпретирован специальной программой “Flash Player”.

Интерпретаторов кода существует огромное множество под платформы Windows, Linux и Mac OS X. Исполнение flash-скриптов доступно браузерам за счет использования специальных flash-плагинов. Таким образом, данная технология не страдает от проблемы переносимости на разнообразные платформы и вполне успешно следует лозунгу Write-Once-Run-Anywhere.
Дополнительным преимуществом технологии Flash можно назвать большое количество учебных материалов, пособий и вспомогательных библиотек, доступных в сети Интернет.
Однако у этой технологии есть несколько недостатков, которые заставили отказаться от ее использовании при выполнении данной дипломной работы.
Во-первых, для исполнения flash-скриптов пользователю необходимо установить в браузер специальный плагин Flash, что несколько усложнит использование программы пользователями.
Во-вторых, несмотря на изначальные обещания по поддержке технологии Flash на мобильных платформах, компания Adobe официально закрыла разработку проекта Mobile Flash [10]. А значит, эта технология не появится на таких популярных мобильных платформах, как Google Android и Apple iOS, и приложения, написанные с использованием Flash, исполняться на них не будут.
Совокупность этих двух недостатков заставила отказаться от использования этой технологии.

3. Клиент-серверное приложение с тонким клиентом

Тонкий клиент – архитектура клиент-серверных приложений, при которой все или большая часть операций по обработки информации переносится на сервер.

Использование тонких клиентов позволяет решить некоторое количество проблем, связанных с обновлением программного обеспечения, т.к. эволюция бизнес-логики приложения будет проходить на стороне сервера, а потому каждый пользователь будет всегда использовать последнюю версию

программы. Однако проблема поднимается вновь, если изменения стали настолько глобальны, что потребовали изменения программного кода тонкого клиента.

Огромным достоинством этой технологии является возможность проводить неограниченно-трудоемкие вычисления за счет наращивания серверных мощностей, которые никак не затрагивают пользователей системы. Кроме того, пользователь может обладать достаточно слабой конфигурацией персонального компьютера, что никак не помешает ему в использовании системы.

Несмотря на вышеперечисленные достоинства, на данный момент у такой архитектуры есть необходимость в постоянном интернет соединении, которое на данном этапе развития компьютерных технологий присутствует не во всех лечебных учреждениях. Поэтому от нее было решено отказаться.

Архитектура реализуемого приложения

После анализа всевозможных подходов для разработки приложения было решено разработать web-приложение на языках HTML 5.0 и JavaScript. Этот выбор не лишен недостатков. Во-первых, язык JavaScript является достаточно трудным для освоения и написания программ: для него отсутствуют мощные среды программирования, он является объектно-ориентированным, но построен на использовании редкой модели прототипирования.

Во-вторых, язык JavaScript не позволяет создавать потоков (thread), что для программ, проводящих массивные вычисления, чревато “подвисаниями” интерфейса.

Несмотря на перечисленные недостатки, приложение, разработанное на HTML5.0 и Javascript, будет иметь следующие несомненные достоинства:

- кроссплатформенность
- отсутствие проблем с версионированием программы
- возможность использования на мобильных платформах

Таким образом, оно будет удовлетворять основным поставленным задачам, что и обусловило этот выбор.

Описание алгоритмов предлагаемого решения

Основой разрабатываемого приложения является набор алгоритмов работы с изображениями. Данные алгоритмы принимают на вход изображение, а так же некоторую область, выделенную пользователем (исключением является алгоритм выделения, который принимает вместо области массив вершин многоугольника). В качестве результата алгоритмы возвращают обработанное изображение.

Далее остановимся подробнее на каждом алгоритме.

Алгоритм сглаживания морщин с учетом цвета кожи

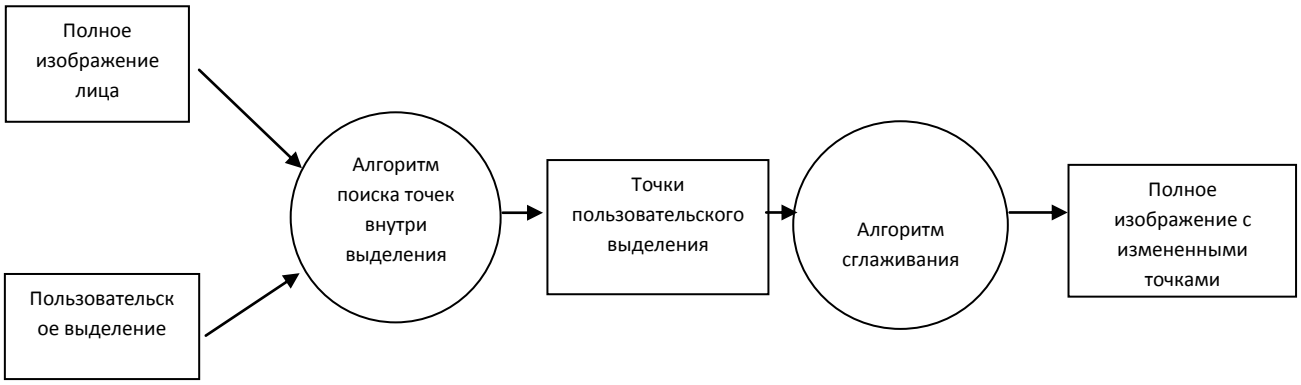

Рисунок 1

Одной из важных задач было разглаживание морщин. Задача заключается в следующем: необходимо реализовать алгоритм, который будет работать с определенной областью и заменять отдельные части области на пиксели другого цвета, не теряя при этом текстуры рисунка. Дополнительным требованием к работе алгоритма было условие, чтобы человеческий глаз минимально отличал края обработанной области от необработанной.

Входными данными для алгоритма была область и множество точек, принадлежащие этой области. Для получения множества точек по одному лишь выделению был разработан специальный алгоритм, о котором подробнее будет рассказано в соответствующем разделе работы. Для того, чтобы удовлетворить поставленным требованиям, было предложено несколько вариантов реализаций.

За основу для первого варианта реализации был взят алгоритм нахождения среднего цвета по области, чтобы в дальнейшем работать с ним. Однако проведенные эксперименты показали, что в ряде случаев (см. Рисунок 2) средний цвет может сильно отличаться от желаемого за счет отдельных пикселей, сильно отличающихся по одному из показателей RGB

Рисунок 2

Алгоритм был переработан таким образом, что за основу был взят не средний, а наиболее часто встречающийся цвет, т.е. тот цвет, количество пикселей которого в выделении наибольшее.
Проведенные эксперименты работы алгоритма показали лучшие результаты, чем первоначальный вариант, (см. рисунок 3) однако во многих случаях воздействие алгоритма смотрелось неестественно в связи с тем, что все точки закрашивались одним цветом. Таким образом изначальная текстура кожи не сохранялась.

Рисунок 3

Для того, чтобы избежать нежелательно эффекта и сохранить текстуру кожи, нужно было сделать зависимость между тем, насколько отличается цвет пикселя от посчитанного среднего цвета и цветом, которым будет закрашен данный пиксель. В качестве метрики отличия была взято отношение разности компонент RGB изначального цвета точки и среднего цвета по выделению к 255 (число от 0 до 1).

$$diffDegree = |averageColor- pixColor|/255$$

Для подсчета нового цвета исследуемой точки в первую очередь был реализован вариант выпуклой комбинации:

$$newColor = [(diffDegree*averageColor+ (1-diffDegree)*pixColor)]$$

Здесь newColor - новый вычисляемый цвет;
diffDegree - метрика отличия изначального цвета от среднего по выделению (от 0 до 1);
averageColor - средний по выделению цвет;
pixColor - изначальный цвет обрабатываемой точки.
Для улучшения работы на сильно отличающихся и сильно похожих по цвету точках была разработана функция зависимости, которая принимала diffDegree, и возвращала так же число от 0 до 1.

От функции зависимости требовались следующие свойства

1)отображать отрезок (0,1) в отрезок (0,1)

2)в значениях близких к единице быть практически равной 1.

Были рассмотрены несколько вариантов функций и в качестве подходящей была взята следующая:

y = ((sin((x -0.5)*PI)+1)/2)^0.3 from 0 to 1

Такие преобразования оказали положительное влияние на такой параметр, как натуральность сглаживания, но на некоторых тестовых изображениях они не выполняли основную функцию алгоритма сглаживания - морщины оставались практически в неизменном виде. (см. рисунок 4).

Для того, чтобы решить эту проблему, функция была поднята на некоторую константу. Таким образом, график становился следующим:

Были проведены эксперименты с различными значениями константы, наилучшие результаты на тестовой выборке изображений дало значение const = 0.3. (см. рисунок 5)

Рисунок 4 Рисунок 5

Этот вариант дал гораздо более приближенные к ожидаемым результаты, однако на тех тестовых изображениях, в которых имелись дефекты кожи (например, темные родинки) или нестандартное освещение, которое давало голубой или зеленый оттенок на морщинах, алгоритм давал сбой.

Рисунок 6

Решением этого могла стать следующая поправка: при подсчёте среднего цвета (в нашем случае наиболее часто встречающегося) надо учитывать только те пиксели изображения, которые принадлежат коже. Для этого необходимо ввести некоторую метрику для подсчета схожести цветов. Метрика должна быть как можно более подобна тому, как человеческий глаз воспринимает и отличает цвета. Для этого определим понятия “согласованности” метрики и цветового пространства.

Определение. Будем называть метрику и цветовое пространство “согласованными”, если для любых двух пар цветов, расположенных на одинаковым расстоянии друг от друга по этой метрике, человеческое восприятие различности этих цветов тоже будет одинаковым.

Определение. Будем называть цветовое пространство “линейным”, если эвклидова метрика является согласованной в этом пространстве.

Изначально все вычисления в алгоритмах проходили в цветовом пространстве RGB.

RGB (аббревиатура английских слов Red, Green, Blue — красный, зелёный, синий) — аддитивная цветовая модель, как правило, описывающая способ синтеза цвета для цветовоспроизведения. Ее минус состоит в том, что она изначально разрабатывалась для отображения цвета на экране и не обладает свойством линейности.

На основании статей [3] и [4] был сделан вывод, что переход от цветной модели RGB к цветной модели CIELab должен сделать метрику определения схожести цветов для человеческого восприятия тривиальной.

В цветовом же пространстве Lab (в работе использован более известный и распространенный стандарт CIELAB) значение светлоты отделено от значения хроматической составляющей цвета (тон, насыщенность). Светлота задана координатой L (изменяется от 0 до 100, то есть от самого темного до самого светлого), хроматическая составляющая — двумя декартовыми координатами a и b. Первая обозначает положение цвета в диапазоне от зеленого до пурпурного, вторая — от синего до желтого.
Отличительной особенностью этого цветового пространства является свойство линейности. Таким образом, если мы хотим узнать, насколько один цвет отличается от другого для человека, нам достаточно взять евклидово расстояния между векторами (L,a,b) для каждого из двух цветов.

В качестве тестовой выборки были взяты фотографии с сайтов [13] и [14]. Эти изображения представляют собой лицо человека до и после определённой операции пластической хирургии. С помощью данного набора изображений был выбран цвет, который является средним (в данном случае использовалось именно среднее арифметическое значение) по пикселям кожи на тестовой выборки. Так же было посчитано максимальное отклонение от среднего цвета на различных частях кожи лица. Используя эти данные, возможно почитать сферу в цветовом пространстве (L,a,b), которая с большой вероятностью содержит часто встречающиеся цвета кожи. Для достижения достаточной вероятности попадания в сферу в качестве радиуса было взято максимальное отклонение, посчитанное на тестовой выборке .

В результате, на указанных ранее фотографиях с дефектами кожи или нестандартным освещением, или при случайном захвате выделением точек, не принадлежащим коже пациента, алгоритм работал правильно, и разглаживание текстур выглядело реалистично.

Рисунок 7

Также важным моментом в разработке алгоритма являлась разработка той его части, которая отвечает за незаметность перехода от обработанной части к необработанной. За основу было взято следующее утверждение: чем дальше от ближайшей границы выделения располагается обрабатываемая точка, тем “сильнее” он должна быть обработана. В качестве выражения силы обработки был взят четвертый параметр данных от точки в отображении на html canvas - alpha. Этот параметр отвечает за прозрачность отрисовки пикселя на изображении.

Вопрос стоял в том, каким образом должна выражаться зависимость прозрачности от удаленности от границы. Были проведены эксперименты с квадратичной, кубической, линейной и логарифмической зависимостями. Наиболее близкие к ожидаемым результаты показала наиболее простая линейная зависимость, так что в конечной реализации алгоритма было решено использовать именно ее.

В результате был реализован алгоритм, который работает на заданном выделении по следующей схеме

1. Обнаруживает наиболее часто встречающийся тон кожи среди выделения
2. С помощью него исправлять сильно отличающиеся по цвету участки, сохраняя при этом общую текстуру и неровности кожи, но удаляя глубокие морщины, которые на изображении сильно отличаются цветом

Так же алгоритм отлично осуществляет переход к границе обработанной области, т. о. применение сглаживания выглядит максимально натурально.

Алгоритм выделения

При работе со сглаживанием морщин важной частью было написание алгоритма, который бы возвращал все точки, лежащие внутри выделение. Формально задачу можно описать следующим образом.

На двумерной плоскости задан многоугольник, полностью лежащий внутри некоторого прямоугольника. Многоугольник может быть невыпуклым и с самопересечениями. Необходимо найти все точки с целочисленными координатами, которые принадлежат многоугольнику.

В этой формализации прямоугольник является абстракцией изображения, а многоугольник – выделения.

Схема алгоритма выглядит следующим образом:

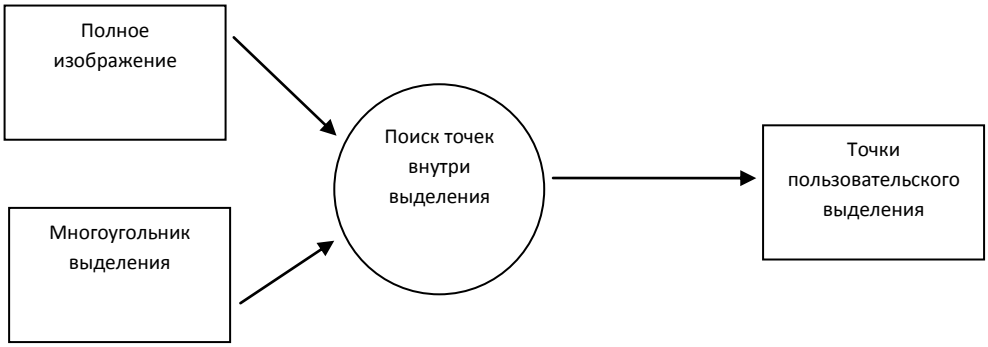

Рисунок 8

Самым простым решением был бы следующий алгоритм, использующий метод трассировки лучей [5].

Суть метода заключается в применении следующей последовательности действий для каждой из точек прямоугольника

1. Проводим луч строго горизонтально налево с началом в этой точке
2. Считаем количество пересечений с ребрами многоугольника

3. Если количество ребер нечетно, то мы добавляем точку в множество точек, лежащих внутри многоугольника.

Асимптотика данного алгоритмы O(n*m*k),где n - ширина прямоугольника, m- высота прямоугольника, k - количество вершин многоугольника.
На практике такой алгоритм не работает быстро. Самым простым ускорением его работы является обход не всех точек изображения, а только минимального описывающего прямоугольника для заданного многоугольника.

Однако общая асимптотика в таком случае так же остается кубической и не приносит большого выигрыша на больших выделениях.

Для того, чтобы добиться приемлемого быстродействия, была применена оптимизация, основанная на методе заметающей прямой. Этот метод использует вертикальную прямую, передвигающуюся слева на право, и останавливающуюся в точках событий. В предложенном алгоритме этот метод применяется к каждому ряду независимо, а точками событий являются точки многоугольника, принадлежащие этому ряду.

В методе заметающей прямой возникает необходимость управления статусом. Статус — это объект, который хранит некоторую вспомогательную информацию о процессе заметания и который обновляется в точках событий. В нашей оптимизации статус является переменной логического типа, которая показывает, находится ли заметающая прямая внутри многоугольника или снаружи.

При использовании такой асимптотики алгоритм работает за O(L * K), где L – количество точек внутри многоугольника.

Такая оптимизация позволила добиться достаточно быстрой работы алгоритма в браузерах.

Результаты работы алгоритма, а так же их сравнение с классическим алгоритмом выделения представлены в таблицах.

При выделении многоугольником с количеством вершин 30.

	Количество точек внутри выделения		
	2*10^3	4*10^4	1.5*10^5
Оптимизированный алгоритм (мс)	9	15	77
Классический алгоритм (мс)	10	94	510

Таб. 1

При выделении многоугольником с количеством вершин 250.

	Количество точек внутри выделения		
	3*10^3	2*10^4	1.5*10^5
Оптимизированный алгоритм (мс)	9	52	249
Классический алгоритм (мс)	31	332	3070

Таб. 2

Замеры производились на следующем оборудовании:

Браузер: Google Chrome v.19.0

Процессор: Intel Core i5, 1.7ГГц, кэш-память третьего уровня в 3 Мб

Память: 4 ГБ DDR3 1333МГц

Графический процессор: Intel HD Graphics 3000 с памятью 384

Алгоритм сдвига

Второй важной задачей приложения была возможность показать или имитировать результат коррекции овала лица. С математической точки зрения, требовался алгоритм трансформации области, который бы в зависимости от определенных параметров искажал изображения, сдвигая некоторые пиксели вдоль определённого вектора. Вектор планировалось задавать с помощью управления мышью. Т. о. разрабатывалась программа, имеющая следующее поведение:

1. при наведении на изображение, пользователю показывается, какая область подлежит трансформации
2. по нажатию мышки и сдвигу ее в какую либо сторону происходит искажение изображения вдоль вектора движения

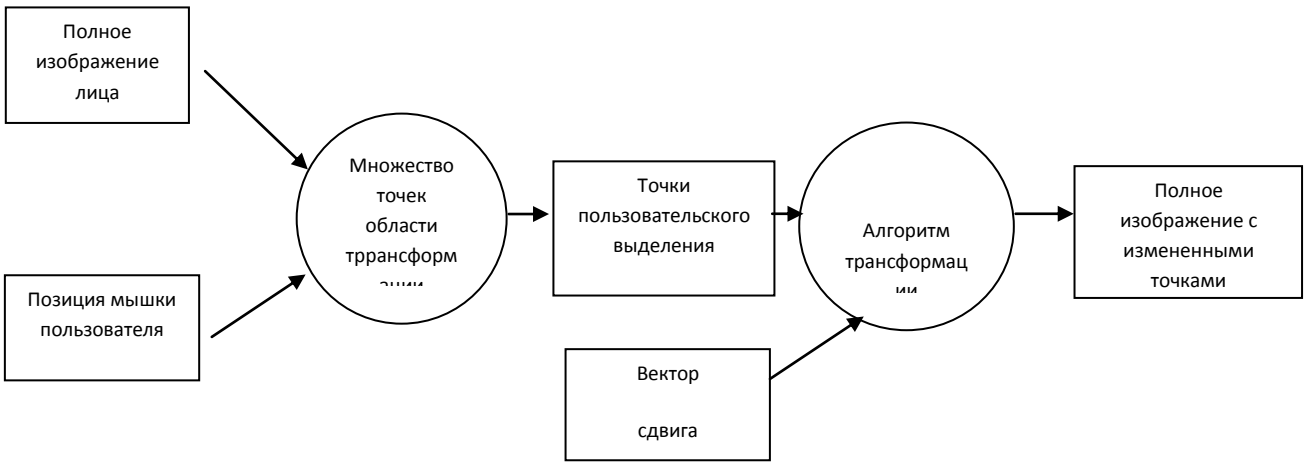

Рисунок 9

Основными требованиями было, чтобы после трансформации изображение осталась максимально натуралистичным и похожим на реальный результат пластической коррекции овала лица. В результате за основу взяли параметризуемый алгоритм сдвига. В качестве параметров выступают следующие величины:

1. Форма трансформируемой области.

Работа с точками изображениями ведется в области вокруг пикселя, который был выбран пользователем с помощью мыши. Форма этой области может быть различной, наиболее естественными вариантам являются круг либо эллипс. Именно выбор одной из этих форм, а так же параметрическая настройка выбранной формы - для круга это изменение радиуса, для эллипса - коэффициента сжатия и радиуса является первым параметром описываемого алгоритма

2. Коэффициент затухания

Для того чтобы искажение сохраняло естественность изображения, необходим плавный переход от трансформируемой области к остальному изображению. Для этого необходимо чтобы у границ области трансформирования искажение было практически незаметным, в то время как если расстояние от границы области велико, трансформация может быть гораздо более сильной. Функция изменения трансформации от расстояния до ближайшей границы области трансформирования, т.е. коэффициент затухания, является вторым настраиваемым параметром.

3. Коэффициент сдвига

В зависимости от разности между координатами двух последовательных положений мышки возможно различное значение вектора трансформации. Так же радиус трансформированной области может влиять на длину этого вектора.

Алгоритм действует по следующей схеме:

1. Получаем точки трансформирования - те точки, которые находятся в заданном радиусе относительно координат мыши
2. Разбиваем полученный круг на радиальные области

Для каждой радиальной области подсчитывается вектор, на который точки из этой области будут сдвинуты:

$$\frac{d}{R} * shift_x$$

3. На изначальном отображение отрисовываются пиксели из области трансформирования, сдвинутые на вектор, соответствующей радиальному кольцу, в котором они лежат.

$$\begin{cases} x_{new} = x_{old} + \frac{d}{R} * shift_x \\ y_{new} = y_{old} + \frac{d}{R} * shift_y \end{cases}$$

При небольших сдвигах данный алгоритм показывал хорошие результаты, однако при увеличении сдвига была выявлена следующая проблема: появление выделяющихся точек на области трансформации (см. рисунок 7).

Рисунок 10

Исследование показало, что данный эффект вызван тем, что заданное преобразование отображало изначальную область (круг) в область большего размера (некоторые точки при сдвиге на вектор выходили за пределы изначального круга). Конечное множество точек было больше чем

изначальное, и не все из них заполнялись образами изначальных точек. Если рассматривать область, в которой происходили изменение в процессе выполнения алгоритма трансформации, то внутри области окажется некоторое количество точек, оставшихся неизменными. Именно они порождали вышеуказанную проблему.

Первым вариантом решения было переносить на вектор сдвига квадратную область 2 на 2 пикселя вместо переноса отдельных пикселей. Эксперименты показали, что это помогло избавиться от вышеупомянутой проблемы, однако появился другой недостаток: неровность получающейся картинки. Особенно хорошо это было видно на границах сильно отличающихся по цвету областей.

В связи с тем, что трансформация границы лица с фоном была основной задачей, было принято решения отказаться от такого метода решения проблемы.

Благодаря проведенному исследованию, установившему причины появления нежелательных радиальных эффектов, удалось устранить первопричину благодаря следующему приему: вместо вычисления образов точек, которые получались не целыми и округления которых не заполняли требуемую область, было решено вычислять прообразы нужных точек области трансформации. Иначе говоря,для каждой точки из области, в которой происходит изменение, искать пиксель, координаты которого были бы ближайшими к действительному (нецелому вещественному) прообразу. Прообраз вычислялся следующим образом:

$$\begin{cases} x_{old} = x_{new} - \dfrac{d}{R} * shift_x \\ y_{old} = y_{new} - \dfrac{d}{R} * shift_y \end{cases}$$

Однако в указанный выше методе важным вопросом было то, как определить область, в которой происходит изменение. Первым определением была выпуклая оболочка всех точек, которые отрисовываются в течение работы первого варианта алгоритма.

Поиск выпуклой оболочки был реализован с помощью алгоритма Грэхема [1] Однако, несмотря на то, что этот алгоритм имеет оптимальную сложность (O(n log n)), его использование сильно замедлило работу программы. Было предложено решение в качестве области трансформирования брать объединение двух кругов, один из которых был изначальной областью трансформации, а второй имел тот же радиус, но центр был сдвинут на вектор, перемещения мыши.

Определенная таким образом область практически полностью совпадала с выпуклой оболочкой, а для ее вычисления не требовалось дополнительного времени. (Сдвиг центра мог быть посчитан во время исполнения предыдущих шагов алгоритма)

При тестировании различных ситуаций использования трансформирования в связи с радиальным разбиением области трансформирования на кольца, для которых применяются различные сдвиги, при последовательном применении несколько раз сдвига в одном направлении проявлялась следующая проблема, проиллюстрированная на рисунке 11.

Рисунок 11

Поэтому был предложен второй вариант реализации алгоритма трансформации. Отличительной особенностью этого алгоритма было то, что все точки из области трансформации должны были остаться внутри той же области.

Перед рассмотрением алгоритма рассмотрим небольшое утверждение, на которое будем в дальнейшем опираться при рассмотрении алгоритма.

Утверждение. Пусть дана окружность и ее дуга APB. Утверждение: если прямая пересекает хорду AB, то она пересекает дугу APB.

Т.к это утверждение представляется очевидным, перейдем сразу к описанию алгоритма.

Для иллюстрации работы алгоритма рассмотрим рисунок.

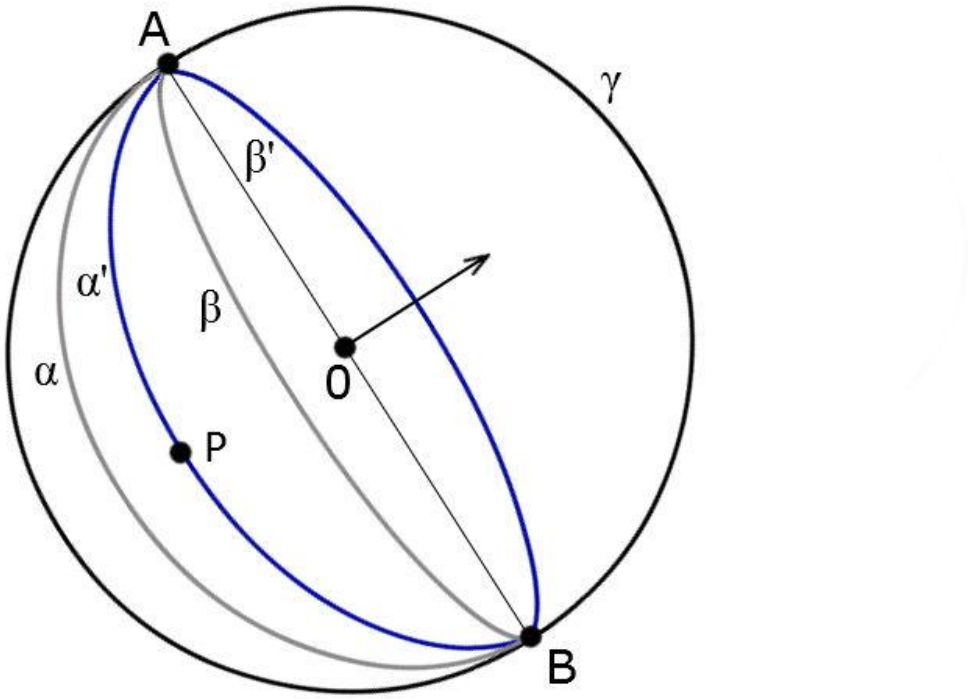

Рисунок 12

Окружность с центром в точке O задает область трансформации. Трансформация определяется вектором направления v. AB - диаметр окружности, задающей область трансформации, перпендикулярный вектору трансформации v. Точки A и B будем для удобства называть “седловыми”. Для того чтобы трансформация была естественной, необходимо, чтобы точки A и B, а также все точки на границах окружности оставались в результате отображения неподвижными.

Рассмотрим некоторую точку P внутри области трансформации и не лежащую на диаметре AB. Новый алгоритм дает ответ на вопрос о том, какая точка перейдет в эту после преобразования.

Рассмотрим сам алгоритм. Через точки P, A, B можно однозначно построить окружность. Дугу APB назовем Альфа-штрих. Наш алгоритм считает прообразом каждой такой дуги некоторую другую дугу какой-то окружности, чуть отдаленную от центра O в противоположном вектору сдвигу направлении: на рисунке прообразом дуги Альфа-штрих является дуга Альфа, а прообразом дуги Бета-Штрих является дуга Бета.

Эти дуги отображают структуру преобразования: все точки дуги Альфа-Штрих биективно перешли в нее в результате преобразования из дуги Альфа. После построения дуги прообраза необходимо придумать биекцию точек из дуги-образа в дугу-прообраз, чтобы найти точку, которая перешла в точку P.

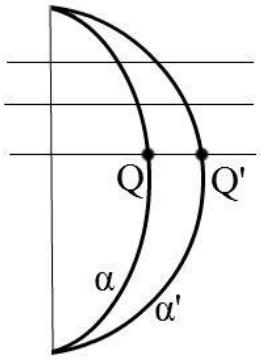

Рисунок 13

На рисунке 12 точки A и B - седловые точки.

Наиболее логичным отображением оказалось следующее биективное отображение. Рассмотрим семейство параллельных прямых, заданных вектором сдвига v. Для любой точки Q’ дуги Альфа-Штрих найдется прямая из этого семейства, проходящая через эту точку (ввиду того, что семейство прямых покрывает всю плоскость). Ввиду того, что эта прямая перпендикулярна хорде AB дуг Альфа и Альфа-Штрих, она (по Лемме) пересекает дугу Альфа в некоторой точке Q, которая и будет прообразом для точки Q’.

На данном этапе рассмотрения алгоритма неразрешенным остается вопрос об определении дуг-прообразов для заданной дуги-образа. Был предложен

следующий вариант ответа для данной проблемы.

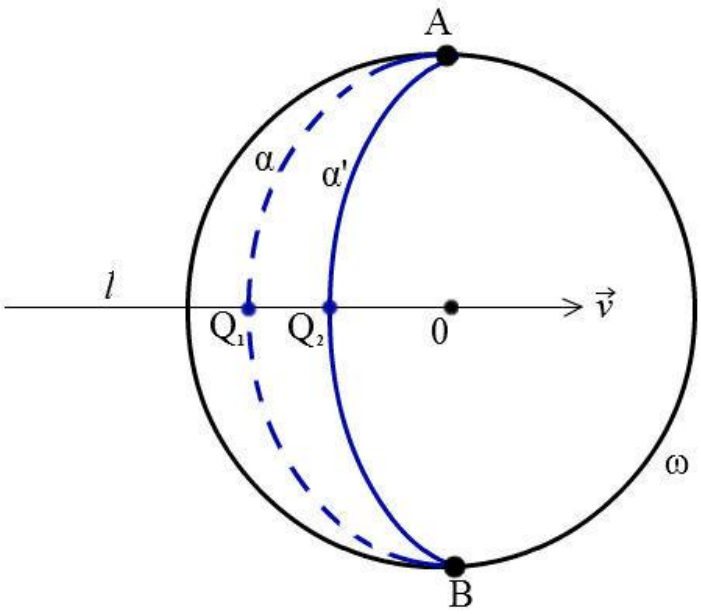

Рисунок 14

Пусть Омега - окружность, задающая область трансформации, вектор v - вектор сдвига, точки A и B - седловые точки, O - центр окружности, задающей область деформации, а Альфа-Штрих - дуга. Рассмотрим алгоритм построения дуги-прообраза Альфа для Альфа-Штрих.
Прежде всего, проведем прямую L, заданную вектором сдвига v и точкой O. По лемме, эта прямая пересечет дугу Альфа-Штрих в некоторой точке Q1, а дугу Альфа в точке Q2, причем Q2 (как и дуга Альфа) нам не известны. Обратим внимание, что для того, чтобы определить дугу Альфа, достаточно определить точку Q2, а дуга Альфа будет строиться по трем точкам A, Q2, B. Рассмотрим расстояние от точки Q1 до окружности Омега, и возьмем его отношение к радиусу окружности Омега.

$$k=\frac{\text{dist}(Q_1,w)}{r}=\frac{\text{r-dist}\ (Q_1,0)}{r}$$

Полученный коэффициент - коэффициент приращения, принимающий значения от нуля (в случае, если дуга Альфа-Штрих совпадает с границей) до единицы (в случае, если дуга выродилась в диаметр). Используя этот коэффициент, вычислим Q2 как точку на прямой L, удаленную от точки Q1 на расстояние k * RAPIDNESS_COEFF в направлении, противоположном вектору сдвига. В приведенной формуле коэффициент RAPIDNESS_COEFF является параметром с интервалом значений от 0 до r, задающим “агрессивность” преобразования.

Прежде чем переходить к самой реализации алгоритма, формализуем математически еще один момент, описанный до этого только словесно. Речь пойдет о проекции точки на дугу-прообраз.

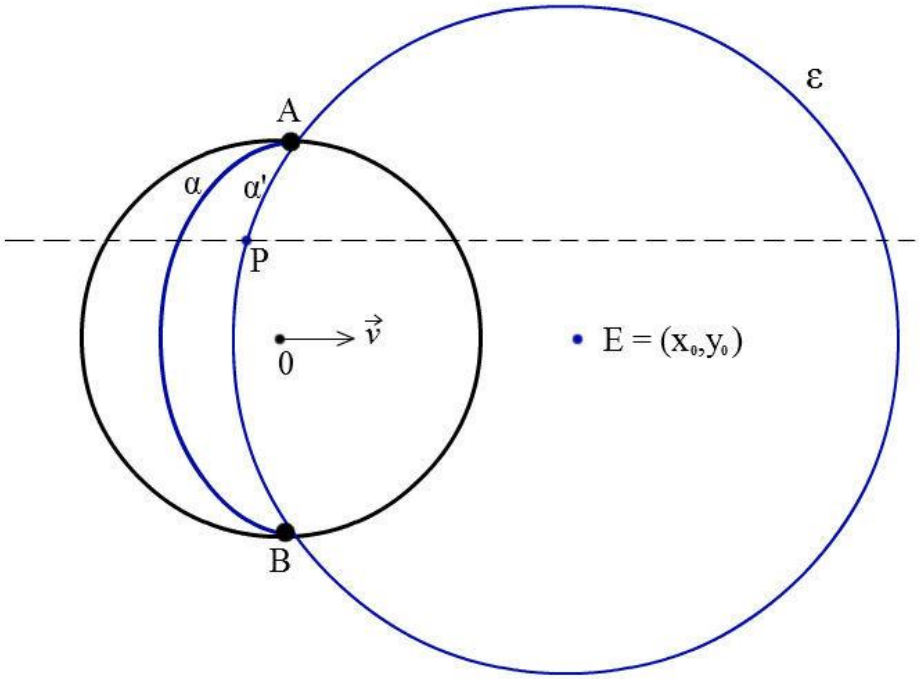

Рисунок 15

Пусть v - вектор сдвига, и пусть у нас есть дуга Альфа-Штрих и точка P на ней. Дуга-прообраз Альфа вместе с окружностью Эпсилон нам тоже известны. Центр окружности Эпсилон - точка E с координатами (x0, y0), а ее радиус обозначим за R. Необходимо найти проекцию точки P на дугу Альфа. Для этого воспользуемся следующим геометрическим приемом.
Известно, что геометрическое место точек окружности описывается формулой с двумя неизвестными, которая в случае окружности Эпсилон будет выглядеть так

$$(x - x_0)^2 + (y - y_0)^2 = R^2$$

Кроме того, геометрическое место точек прямой, заданной вектором v и точкой P, описывается следующей системой уравнений

$$\begin{cases} x = P_x + k * V_x \\ y = P_y + k * V_y \end{cases}$$

Пересечение этих двух ГМТ будет иметь не более двух точек, и выражается формулой

$$(P_x + k * V_x - x_0)^2 + (P_y + k * V_y - y_0)^2 = R^2$$

Легко показать, опираясь на геометрический смысл и лемму, что у этого уравнения всегда два различных корня. Нас интересует минимальный из них по модулю. После получения k точка-прообраз может быть получена по следующей формуле

$$(P_x + k * V_x, P_y + k * V_y)$$

Рассмотрим небольшой отрывок кода реализации алгоритма

```
function invertPoint(point, toolCircle, settlePoints,
shift) {
   var pc = getCircleCenter(point, settlePoints.a,
settlePoints.b);
```

```
    // this means that point lays on the same line as
settlePoints
    if (!pc) {
        pc = scaleVectorToLength(shift, 10000);
    }
    pc.r = distPoints(pc, settlePoints.a);
    var ic = getInvertedCircleCenter(toolCircle, pc,
settlePoints, shift);
    ic.r = distPoints(ic, settlePoints.a);
    var alpha = point.x - ic.x;
    var betta = point.y - ic.y;
    var A = shift.x * shift.x + shift.y * shift.y;
    var B = 2 * (shift.x * alpha + shift.y * betta);
    var C = (alpha * alpha + betta * betta) - ic.r *
ic.r;
    var res = solveQuadraticEquation(A, B, C);

    var k = res.x2;
    if (Math.abs(res.x1) < Math.abs(res.x2)) {
       k = res.x1;
    }
    return {
        x: point.x + k * shift.x,
        y: point.y + k * shift.y
    };
}
```

Эта функция на вход принимает точку, для которой нужно посчитать обратную, окружность, описывающую область трансформации, набор седловых точек и сдвиг. Функция возвращает точку-прообраз данной по версии вышеописанного алгоритма.

Вначале считается окружность по трем точкам, содержащая дугу Альфа-Штрих (см. рис. 11). Обратите внимание, что если построить окружность не удалось (например, точки лежат на одной прямой), то следующим условным оператором мы избавляемся от этого крайнего случая, присваивая в качестве ответа окружность, находящуюся очень далеко.
После этого вызывается метод getInvertedCircleCenter, который делает в точности то, что описано в алгоритме, и возвращает центр окружности с дугой Альфа (см. рис. 13). Далее для проекции точки на дугу мы решаем квадратное уравнение, проверяем на правильность его ответы и возвращаем искомую точку.

В результате, этот алгоритм показал лучшие результаты на тестовой выборке фотографий, и избавлял от проблемы с последовательным применением нескольких сдвигов в одном направлении. Поэтому было принято решение использовать именно его.

Результат

Результатом дипломной работы является графический редактор двухмерных изображений, написанный на языках HTML5.0 и JavaScript. Редактор предоставляет возможность имитировать результаты пластической хирургии на основе первоначальной фотографии. В качестве средств, доступных для имитации, доступны следующие инструменты, специально разработанные для работы с лицами.

1. Инструмент “сглаживание выделения” (Рисунок 16, 1)
   Инструмент позволяет исправлять дефекты кожи сложной формы, такие как складки и глубокие морщины. Этот инструмент применяет алгоритм сглаживания текстур к выделенной пользователем области. Для достижения наилучшего результата дефект кожи должен по возможности лежать как можно ближе к центру области выделения.
2. Инструмент “размытие”(Рисунок 16, 2)
   Инструмент позволяет исправлять родинки и небольшие пигментные пятна на коже, а так же подходит для ретуширования мелких деталей. Инструмент применяет алгоритм сглаживания к области в виде окружности с центром, указанным пользователем. Радиус окружности настраивается с помощью специального ползунка.
3. Инструмент “подтяжка”(Рисунок 16, 3)
   Инструмент позволяет исправлять форму овала лица, изгиб бровей, форму разреза глаз. Инструмент применяет алгоритм к области, обозначенной в виде круга. Радиус области действия настраивается с

помощью специального ползунка. (Рисунок 16, 4)

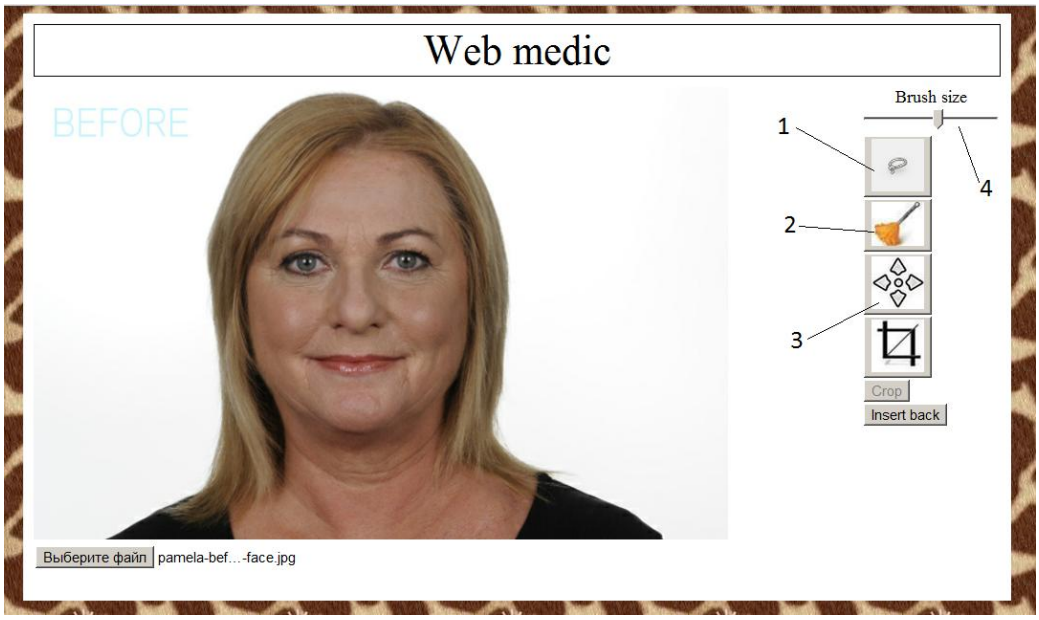

Рисунок 16

Работа приложения была протестирована на следующих браузерах:

1. Google Chrome версии 19.0
2. Apple Safari версии 5.1.3
3. Safari Mobile версии 5.1.1
4. Internet Explorer версии 9.0.6
5. Opera версии 11.64

По результатам испытаний приложение показало свою абсолютную работоспособность в браузерах Google Chrome, Opera и Apple Safari. Испытания на мобильном браузере Safari Mobile показали, что приложению требуется оптимизация интерфейса для мобильных устройств. К сожалению, ввиду отсутствия должной поддержки HTML5 в браузере Internet Explorer, приложение функционально не полностью, однако позволяет выполнять базовые операции без настройки размера кисти.

Список литературы

1. Graham, R.L. (1972). An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set
2. Linda G. Shapiro and George C. Stockman (2001). Computer Vision.
3. Weatherall IL, Coombs BD : Skin color measurements in terms of CIELAB color space values. J Invest Dermatol 99 : 468-473, 1992
4. A Top Down Description of S-CIELAB and CIEDE2000. Garrett M. Johnson,* Mark D. Fairchild
5. Notes on efficient ray tracing. Solomon Boulos, University of Utah
6. 2D Image Canvas - http://www.w3.org/TR/2dcontext/
7. JCrop - http://deepliquid.com/content/Jcrop.html
8. Сайт по ECMAScript - http://www.ecmascript.org
9. CIELAB - http://www.fho-emden.de/~hoffmann/cielab03022003.pdf
10. Pixastic - http://pixastic.com/
11. Adobe - http://blogs.adobe.com/conversations/2011/11/flash-focus.html)
12. http://www.w3schools.com/browsers/browsers_os.asp
13. http://www.facialsurgery.com
14. http://www.smartplasticsurgery.com
15. http://processingjs.org/articles/jsQuickStart.html#waystouseprocessingjs