САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Попов Кирилл Владимирович

Детектирование неба на фотографиях

Курсовая работа

Научный руководитель:
доцент Александр Вахитов

Санкт-Петербург
2014

Оглавление

Введение

Сегментация является одной из основных задач в области обработки изображений. С помощью алгоритмов сегментации можно разбить изображение на различные регионы, объединенные по какому-либо признаку. В своей работе я рассмотрел результат работы некоторых наиболее распространенных алгоритмов [2] применительно к задаче распознавания и удаления неба с фотографий. При этом была проанализирована скорость работы этих алгоритмов на мобильных устройствах на платформе Android.

По результатам проведенных исследований было выявлено, что ни один из существующих алгоритмов не в состоянии корректно решать поставленную задачу. В связи с чем, на базе алгоритма GrabCut[5], был разработан собственный оптимизированный вариант, который позволяет решать задачу распознавания неба с меньшими затратами процессорного времени и с более высоким качеством.

Так же в процессе работы были опробованы некоторые подходы, которые не привели к желаемому результату, однако, они позволили выбрать нужный вектор исследований.

Для проверки алгоритмов, замера их производительности, а так же для разработки нового алгоритма было написано тестовое приложение для платформы Android.

1. Постановка задачи

Разработать алгоритм и приложение под OS Android для выделения неба на фотографиях.

Основные требования к алгоритму:

- Быстрота работы на мобильных устройствах
- Автоматическое распознавание неба
- Возможность ручной коррекции
- Качественная сегментация

2. Практическое применение

Данная работа проведена в интересах кампании YoWindow[1], занимающейся разработкой скринсейверов[14] для настольных и мобильных вычислительных устройств. Благодаря проведенным исследованиям, пользователь сможет загрузить в разработанную программу фотографию его любимого пейзажа и удалить из него небо. Причем, для этого ему совершенно не нужно обладать какими-либо знаниями в области обработки изображений, либо графического дизайна. После этапа обработки изображения, вместо вырезанного неба, будет показываться графически нарисованное небо, чье состояние зависит от погоды на ближайшие несколько часов, в той местности, где находится пользователь.

Таким образом, стояла задача разработать максимально быстрый и качественный алгоритм сегментации, чье использование было бы максимально интуитивным и не требовало бы никаких специальных знаний.

Разработанный алгоритм получился достаточно универсальным, поэтому он может применяться не только по прямому назначению, но и для автоматического распознавания и удаления любого однородного фона на изображениях. И при этом обладает возможностью интерактивной коррекции результатов.

3. Тестовое приложение и его функциональность

Программа разрабатывалась для проверки скорости работы рассматриваемых алгоритмов на устройствах под управлением OS Android 2.4 и выше. Так же с помощью него были выявлены правила, следуя которым можно существенно повысить скорость работы алгоритмов компьютерного зрения, не прибегая к использованию JNI[10]

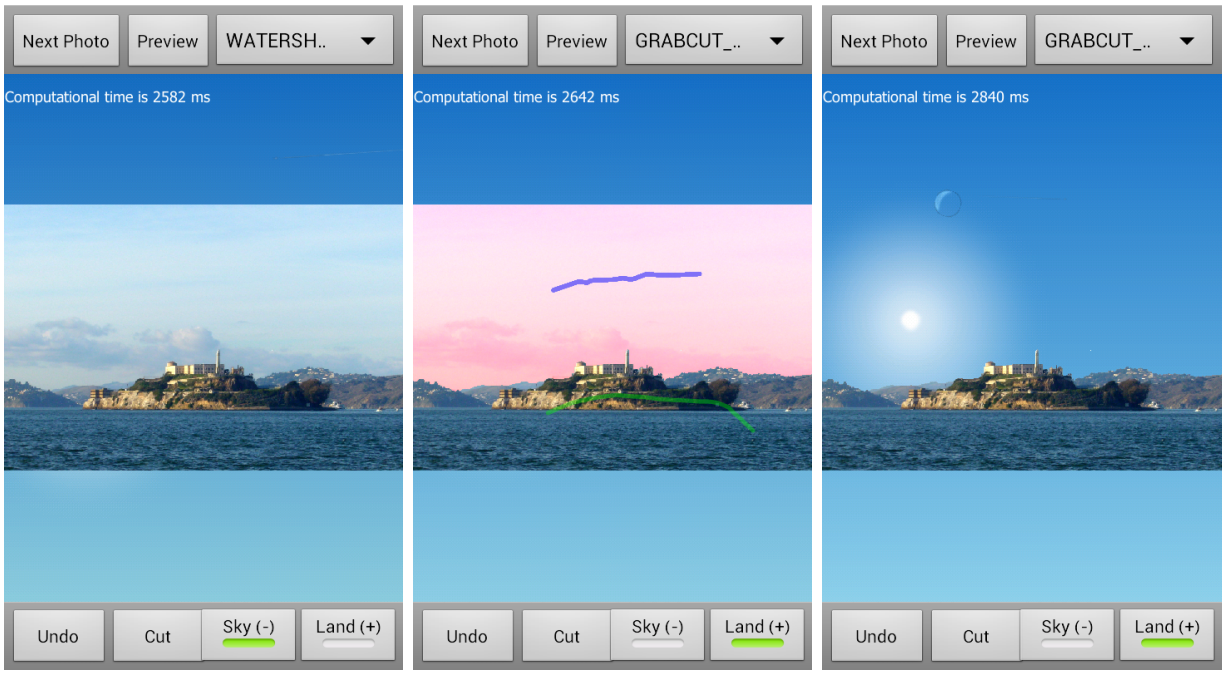


Рис. 1: Пользовательский интерфейс приложения

Тестовое приложение имеет следующую функциональность:

- Интерактивный выбор алгоритма вырезания
- Зум и перетаскивание пальцами
- Выделение областей, определенных алгоритмом как небо
- Отображение времени, затраченного на работу алгоритма
- Кнопка Undo. Позволяет отменить до 20 последних действий
- Кнопка Preview - удаление областей отмеченных как небо
- Кнопка выбора новой фотографии
- Кнопка Cut, предназначенная для ручного запуска медленных алгоритмов
- Кнопки Sky и Land, предназначенные для пользовательского ввода примерных регионов принадлежащих земле и небу

4. Анализ существующих алгоритмов сегментации

4.1. K-Means

K-Means[7] - алгоритм кластеризации, стремящийся минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров. Для задачи распознавания неба, алгоритм применялся следующим образом. Пользователь отмечал участки, по его мнению, принадлежащие классам "земля" или "небо". После этого для каждого класса с помощью алгоритма K-Means вычислялось оптимальное положение для N центроидов, исходя из интенсивности пикселей, находящихся в данном классе.

После проведения данной процедуры у нас имеется N центроидов для пикселей принадлежащих классу "земля" и N центроидов для пикселей принадлежащих классу "небо".

Следующим шагом алгоритма было вычисление для каждого пикселя изображения наиболее близкий к нему центроид. В качестве метрики использовалось как обычное Евклидово[11], так и Манхэттенское расстояние[13] в $\mathbb{R}^3$. Исходя из того, к какому классу относится наиболее близкий центроид, определялась принадлежность к земле или небу самого пикселя.

Существенным недостатком данного подхода является то, что он никак не учитывает взаимное расположение пикселей относительно друг друга. Следовательно, если среди пикселей, принадлежащих земле, найдется пиксель, с интенсивностью близкой к пикселям неба(к примеру блики на машинах и стеклах домов), то он будет причислен к классу, не соответствующему ему.

Так же, еще одним недостатком алгоритма является вопрос о том, каким должно быть число N. В случае маленьких значений, возрастает число ошибок сегментации, а в случае больших, увеличивается сложность вычислений, и, следовательно, время работы алгоритма.

4.2. GrabCut

GrabCut[5] - алгоритм сегментации изображения, оценивающий с помощью GMM[4] распределение цветов в регионах, отнесенных пользователем к двум различным классам(В нашем случае ”земля” и ”небо”). После этого, по полученным данным строится Markov Random Field[8] с использованием энергетической функции[3], которая предпочитает соединенные пиксели, имеющие один и тот же класс. После чего запускается оптимизационный метод, основанный на минимальном разрезе графа[6]

На выходе получается маска, чьи значения определяют к какому классу относится соответствующий пиксель. Данный алгоритм, в отличие от K-Means, использует информацию о взаиморасположении пикселей. Благодаря этому удается минимизировать ложные срабатывания на объектах поверхности.

Из всех рассмотренных алгоритмов GrabCut показал наилучший результат по качеству работы. К сожалению, он же оказался и самым медленным среди них. Существенное влияние на скорость работы оказывает количество циклов обучения GMM. В случае, если он мало, алгоритм теряет способность к сегментации больших однородных регионов. В противном случае, время работы увеличивается пропорционально количеству итераций.(см. Рис. 2)

Так как данный алгоритм показал наилучшее качество работы, было выбрано решено взять его за основу.

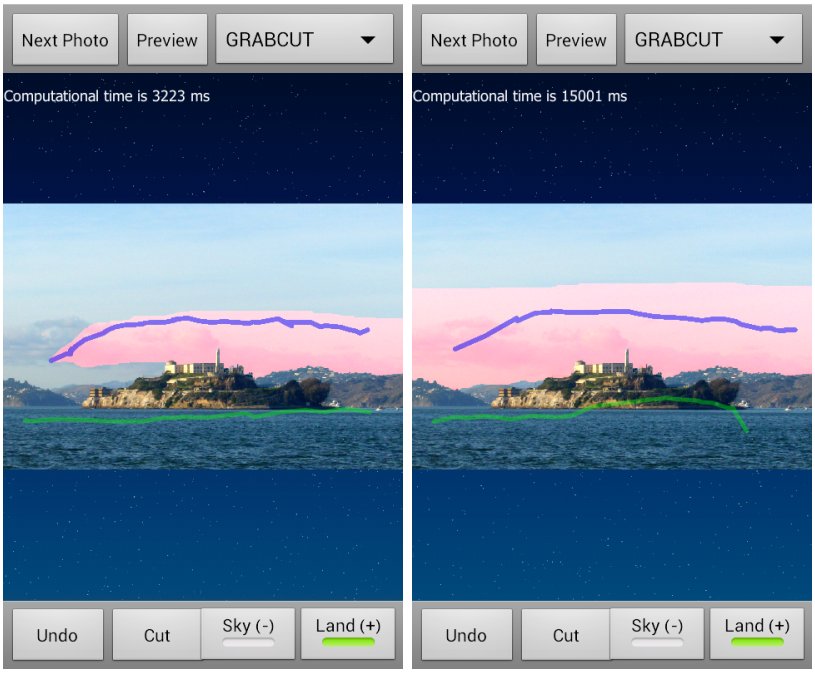


Рис. 2: GrabCut при 1 и 5 итерация обучения GMM

4.3. Watershed

Watershed[9] - еще один популярный алгоритм сегментации. Используется во многих мобильных приложениях для вырезания объектов с фотографий. Работает следующим образом. В начале, пользователь определяет, какие регионы относятся к ”земле”, а какие к ”небу”. Далее к изображению применяется оператор Лапласа, благодаря чему на выходе получается карта высот, которую, образно говоря, начинают ”заливать водой” до тех пор, пока вода не скроет самый высокий пик. Озера, слившиеся с регионами, отмеченными пользователем, попадают в один и тот же сегмент. Если встречаются два ”озера” из разных кластеров, по месту их встречи проходит граница между разными сегментами.

Одним из плюсов данного алгоритма является то, что он является линейным. Благодаря чему может работать в режиме реально времени даже на мобильных устройствах.

Минусом является то, что в случае, если граница между небом и землей не достаточно четкая, либо же на ландшафте присутствуют мелкие детали, алгоритм дает серьезный сбой, что видно на Рис. 3. Так же обратите внимание на время работы алгоритма, отмеченное в левом верхнем углу. Он существенно меньше, чем у GrabCut, даже с 1 итерацией обучения.

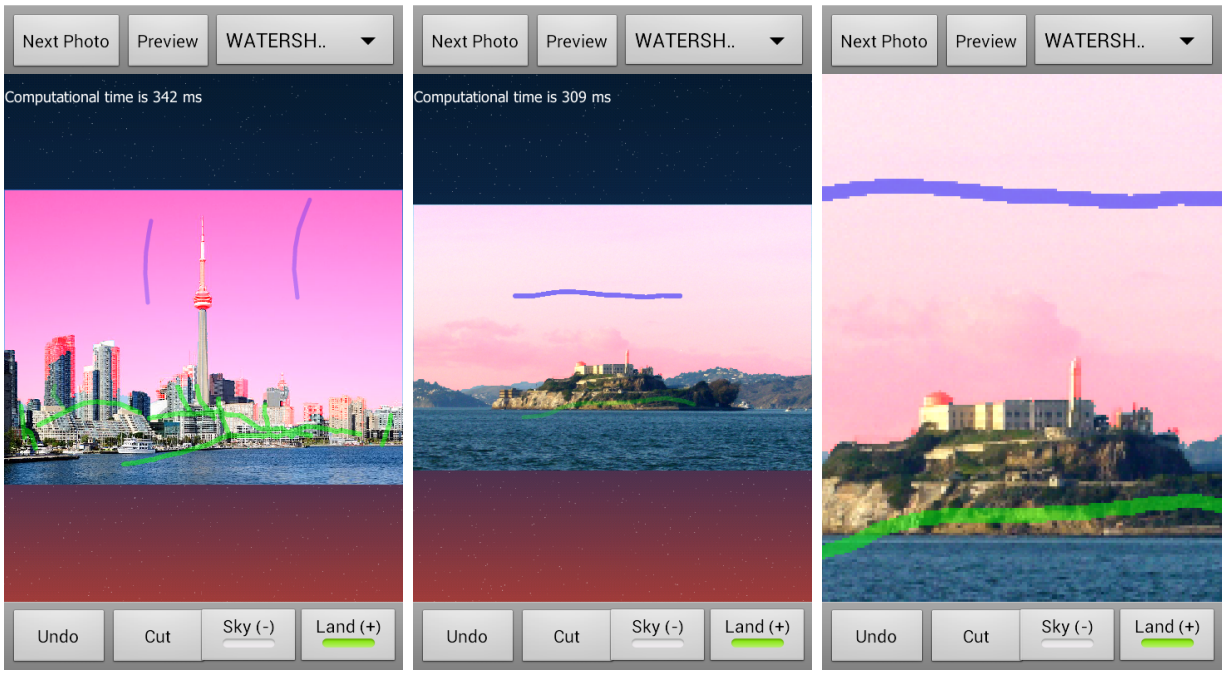


Рис. 3: Результат работы Watershed’a при плохо различимых границах и мелких выступающих объектах ландшафта

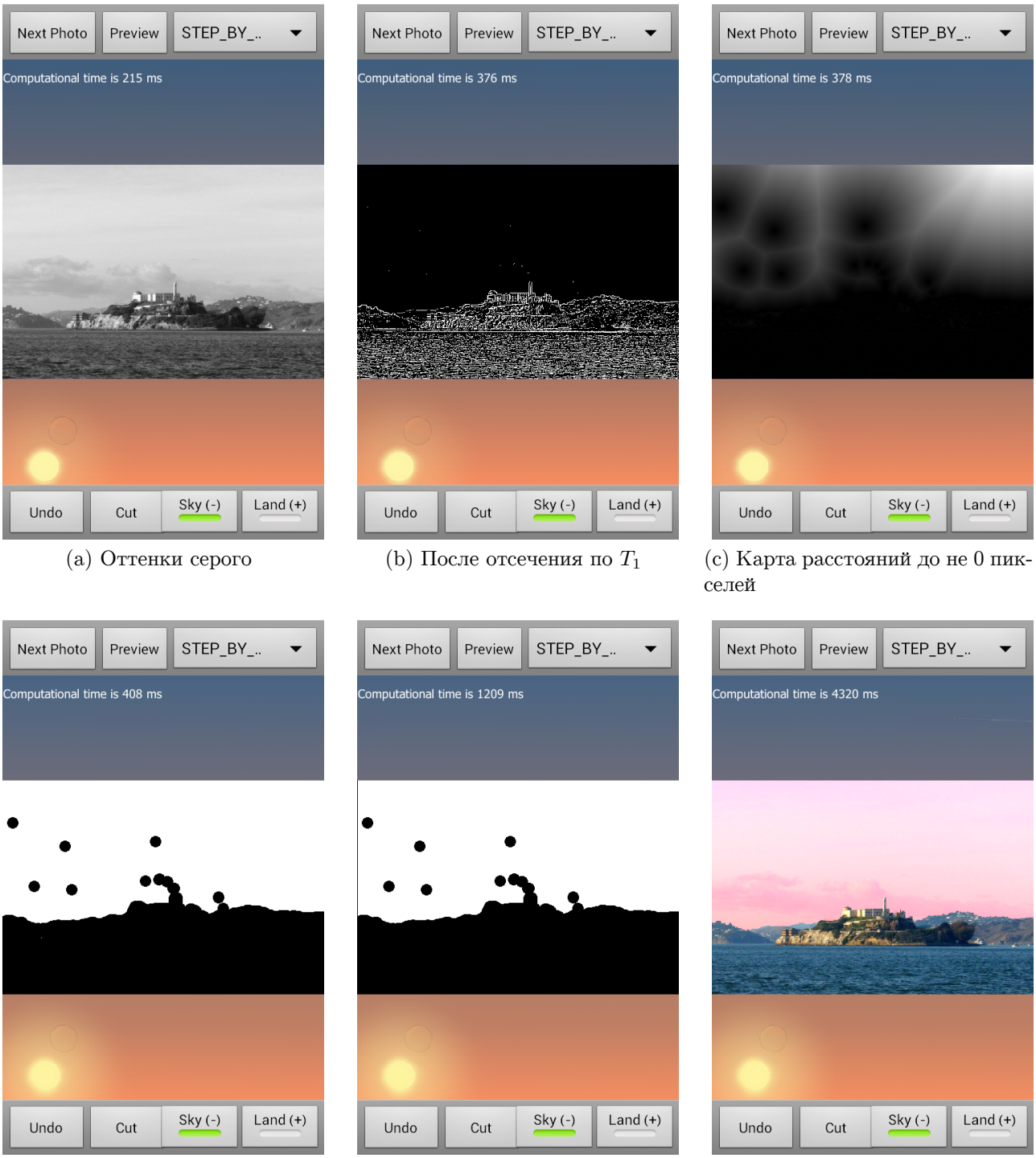
5. Описание работы алгоритма

Разработанный алгоритм вычисляет специальную маску для изображения следующим образом (см. Рис. 4):

1. Конвертируем исходное изображение в оттенки серого 4a
2. Применяем к полученному изображению фильтр высоких частот
3. Обнуляем пиксели с интенсивностью ниже, чем пороговое значение T_1. (см. Рис. 4b)
4. Если пользователь произвел какой-либо ввод, то на данном шаге мы добавляем к изображению из предыдущего пункта пиксели, помеченные им как "земля"
5. Считаем расстояние от каждого пикселя до пикселя с не 0 значением (см. Рис. 4c)
6. Полученный результат линейно отображаем на отрезок [0, 255]
7. Обнуляем пиксели, чье расстояние до границы меньше чем пороговое значение T_2
8. В полученном изображении ищем пиксели с интенсивностью большей, чем пороговое значение T_3. Считаем, что они принадлежат классу "небо". Таким образом, даже, если пользователь не производил ввода, у нас всегда найдутся пиксели, интерпретируемые как небо. Т.к. данные пиксели являются геометрическим центром больших однородных областей.
9. Ищем компоненты связности, которым принадлежат пиксели, точно известные как небо. Мы получили такие пиксели из предыдущего пункта и пользовательского ввода, если он был.(см. Рис. 4d)
10. Обнуляем пиксели, которые не попали в компоненты связности, найденные в предыдущем пункте (см. Рис. 4e)

Не 0 пиксели в полученной маске помечаются как "вероятно небо", кроме тех, которые отметил сам пользователь. Они всегда лежат в классах "небо" и "земля" соответственно. Полученная маска передается на вход алгоритму GrabCut. Т.к. большие участки уже размечены алгоритмом, GrabCut остается только уточнить границы неба.(см. Рис. 4f) За счет этого мы во-первых, экономим время на обучении GMM. Потому что, выполнив грубый предподсчет, снижаем количество пикселей, которые должны быть размечены. А во-вторых, мы получили автоматическое распознавание неба.

Дальнейшая оптимизации алгоритма будет связана с тем, что на последней стадии GrabCut будет считаться не от всего изображения, а только от квадратного окна шириной $3 * T_2$ вдоль границ маски.



(a) Оттенки серого

(b) После отсечения по T_1

(c) Карта расстояний до не 0 пикселей

(d) Поиск связных компонент. В левом нижнем углу находится пиксель, который будет удален на следующем этапе

(e) Получившаяся маска

(f) Окончательный результат

Рис. 4: Основные шаги разработанного алгоритма

6. Порядок выделения неба, предлагаемый пользователю

В качестве подготовки, после того, как пользователь загружает фотографию, из которой он хочет вырезать небо, происходит сжатие изображения. Этот шаг необходим для того, чтобы работать только с тем количеством пикселей, которое нам необходимо. Поскольку обработанное изображение будет выступать в качестве части скринсевера[14], необходимая ширина изображения равна ширине экрана.

После этого алгоритм, предложенный в этой работе, пытается определить небо на фотографии. В случае, если ему это удается, и пользователя устраивает качество вырезания, которое он может оценить с помощью функции Preview, работа программы завершается.

В противном случае, пользователь с помощью кнопки Sky выбирает кисть "небо", и, проводя её по экрану, помечает части изображения, относящиеся к небу, но не помеченные алгоритмом как таковые. То же самое пользователь может сделать с кнопкой Land. В случае ее нажатия будет выбрана кисть "земля", которой может пометить землю, ошибочно выделенную алгоритмом как небо.

Следует отметить, что на предыдущем шаге пользователь не обязан точно выделять контуры соответствующих областей. Достаточно лишь сделать это декларативно. (см. Рис. 1)

В случае, если и этого оказалось не достаточно, благодоря функциям "приближение" и "перетаскивание", у пользователя остается возможность в ручную провести разделение сложных участков изображения с точностью "до пикселя". Для этого он может использовать все те же кнопки Sky и Land

7. Моделирование неба

В процессе работы была опробована идея, подойти к проблеме не с точки зрения алгоритмов сегментации, а с точки зрения математического моделирования. Мной было сделано предположение, что небо, если на нем нет облаков, можно промоделировать с помощью кривой второго порядка. Принцип работы алгоритма должен был состоять в следующем. Пользователь отмечает участки неба, свободные от облаков. По этим участкам, с помощью МНК, строится модель неба. После чего, все пиксели, которые лежат близко к моделируемым значениям, считаются небом и подлежат удалению.

Для того, чтобы проверить правильность рассуждений, предлагаю посмотреть на то, как выглядит профиль интенсивности изображения конвертированного в оттенки серого(см. Рис. 5)

На графике (5b) видна модель, построенная с помощью МНК[12] для первых 320 пикселей.

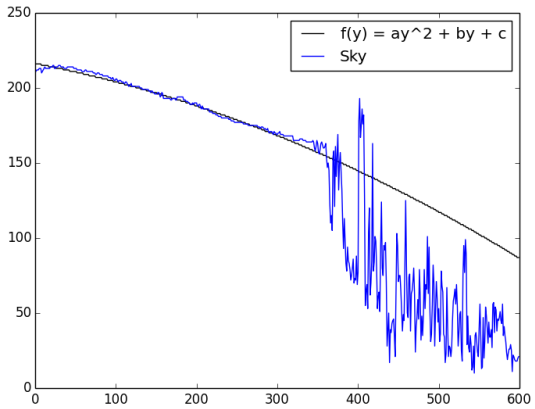
Небо моделировалось с помощью уравнения $f(y) = ay^2 + by + c$, где:

- y - координата пикселя по оси ОУ
- f(y) - интенсивность пикселя
- a, b, c - неизвестные коэффициенты модели. Для их нахождения используется МНК[12]

Как видно из рисунков (5c) и (5d) моделирование работает, в целом, достаточно хорошо и справляется с большими регионами. Однако, т.к. модель не учитывает взаиморасположение пикселей, некоторая часть пикселей вырезается из земли. Кроме того, в силу особенностей работы цифровых камер пиксели лежащие на границе неба и земли, не подпадают под данную модель и следовательно не могут быть убраны.

Так же, в процессе работы были опробованы модели для цветных фотографий(3 цветовых канала, в каждом по 3 коэффициента. Всего: 9) и модели, учитывающие 2е координаты пикселей $f(x, y) = ax^2 + bxy + cy^2 + dx + ey + k$ (3 цветовых канала, 6 коэффициентов. Всего: 18) К сожалению, ни одна из них не смогла существенно улучшить результат сегментации.

(a) Исходное изображение, конвертированное в оттенки серого



(b) Профиль интенсивности столбца Y=750

(c) Восстановленное небо

(d) Результат отсечения пикселей, совпадающих с моделью

Рис. 5: Проверка возможности моделирования неба

8. Характеристики тестового набора

В процессе разработки алгоритм тестировался на фотографиях, обладающих различными характеристиками. Т.к. для некоторых фотографий, алгоритм чувствителен к пользовательскому вводу, то приходится ограничиться ручным тестированием. В связи с этим стояла задача разбить потенциальные фотографии, с которыми придется работать алгоритму, на некоторые классы, и составить тестовый набор из представителей каждого из них. Ниже приведены некоторые особенности, которыми обладают некоторые фотографии из тестового набора.

- Имеют разное разрешение
- Получены с применением различных цифровых фильтров
- Сжаты jpg с большим коэффициентом потерь
- Полученны при разных видах облачности
- Сделаны при разном освещении
- Хорошо различим смог на городом
- Имеют обилие мелких деталей
- Имеют трудно различимую границу между землей и небом
- Присутствуют большие однотонные области не являющиеся небом(В основном вода)
- Небо разделено окружающим ландшафтом на несколько больших частей

9. Способы ускорения работы с изображениями в OS Android

В процессе работы над реализацией алгоритма выяснилось, что Bitmap - класс в Android отвечающий за хранение и доступ к пикселям изображения, хранит их в Native memory. Как следствие, каждый запрос на получение значения конкретного пикселя, с помощью метода

```
public int getPixel(int x, int y)
```

приводит к вызову метода

```
native int nativeGetPixel(int nativeBitmap, int x, int y);
```

соответственно, в случае, если необходимо обработать все пиксели изображения, генерируется очень много вызовов использующих JNI[10]. Однако, есть способ свести количество вызовов до 2. Для этого, следует получать все пиксели из Native memory сразу.

```
Bitmap image = /*Ваше изображение*/;
int h = image.getHeight(), w = image.getWidth();
int pixels[] = new int[h * w]; (1)
image.getPixels(pixels,0,w,0,0,w,h); (2)
for (int i = 0; i < h; ++i) {
    for (int j = 0; j < w; ++j) {
        int index = (i * w + j) ;(3)
        int pixel = pixel[index];

        //...
    }
}

image.setPixels(pixels, 0, w, 0, 0, w, h); (4)
```

где цифрами отмечены:

1. Выделяем память для загружаемых пикселей
2. 1-ый вызов JNI
3. По координатам пикселя на изображении (i,j) получаем его индекс в массиве
4. 2-ой вызов JNI

Для моего проекта данный метод позволил ускорить обход пикселей изображения в 30 раз, благодаря чему я смог отказаться от написания собственного Native кода под OS Android.

10. Сравнение алгоритмов

Для того, чтобы провести сравнение алгоритмов, был выбран следующий способ. Некий испытуемый с помощью тестового приложения должен был удалить небо с тестовых фотографий, используя один из следующих алгоритмов:

- GrabCut
- Watershed
- Алгоритм моей разработки(Grabcut+).

В процессе работы испытуемого, для каждого тестового изображения и каждого исследуемого алгоритма записывались следующие параметры, выбранные как критерии оценки качества:

- Количество штрихов, которое совершил пользователь(см. Рис. 6a)
- Количество пикселей, которое пользователю пришлось закрасить вручную, чтобы получить нужное качество(см. Рис. 6b)
- Общее время работы алгоритма.(см. Рис. 6c)
- Среднее время обработки пользовательского ввода(см. Рис. 6d)

На графиках (6a) и (6b) Grabcut+ в некоторых точках принимает значение 0. Это означает, что на данных тестовых изображениях, алгоритм распознал и удалил небо, с требуемой точностью.

Так же, на графике (6d) виден скачек на изображении с ID равным 13. Оно представляет из себя изображение ландшафта, на 45% состоящее из воды, на 45% из неба и на 10% из суши. Мы предполагаем, что в процессе эксплуатации приложения, такие картинки будут составлять не более 5%, поэтому данный эффект является приемлемым.

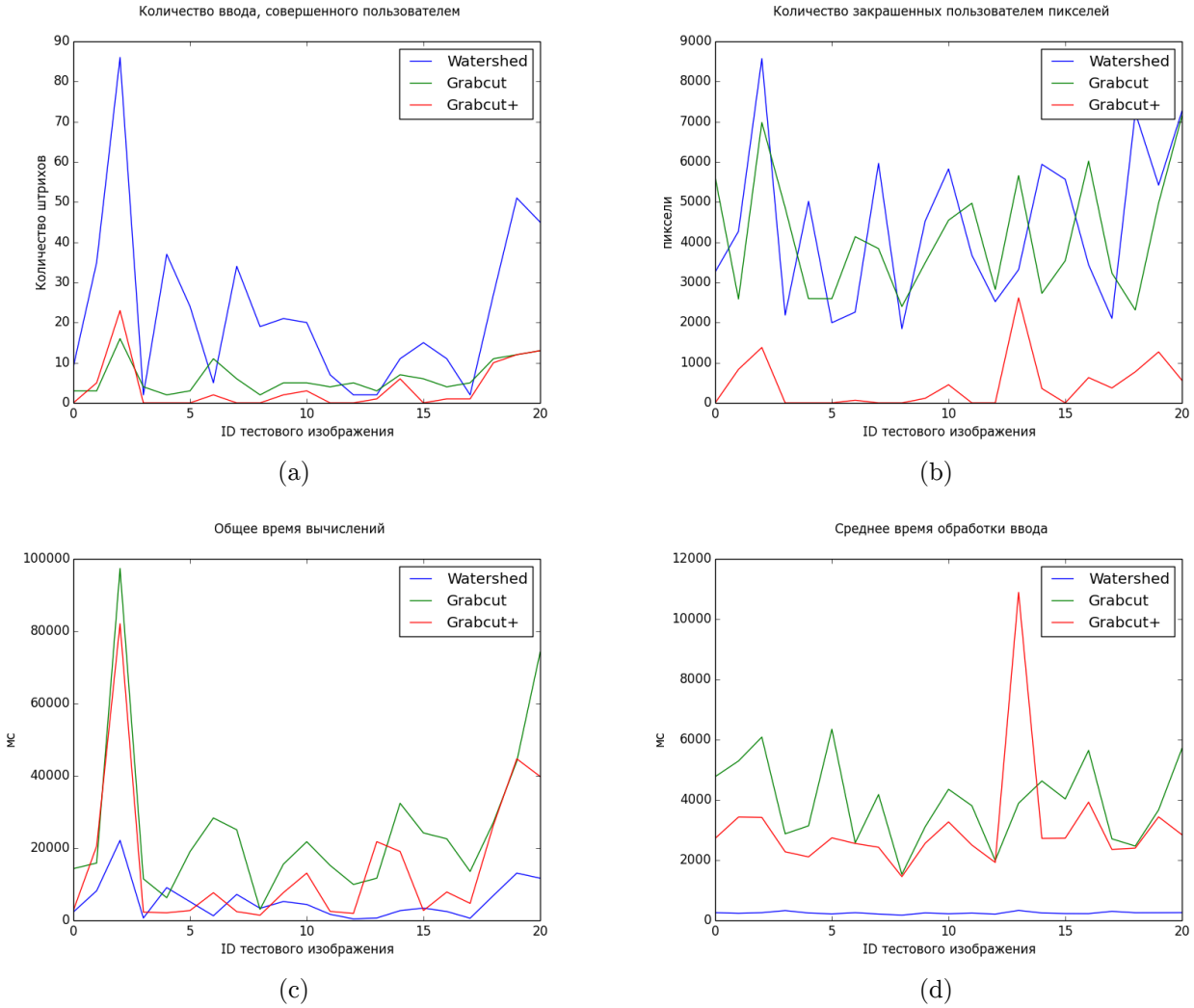


Рис. 6: Графики сравнения алгоритмов по различным показателям

11. Результаты

В ходе выполнения курсовой работы, были достигнуты следующие результаты:

- Разработано приложение под OS Android для вырезания неба
- Разработан алгоритм, обладающий возможностью автоматического распознавания неба на фотографиях
- Произведено сравнение разработанного алгоритма, с существующими алгоритмами сегментации
- Проверена возможность моделировать небо по фотографиям
- Определены способы, с помощью которых можно существенно ускорить разработанный алгоритм.

Список литературы

[1] RepkaSoft. YoWindow // Официальный сайт. — 2014. — http://yowindow.com.

[2] Szeliski Richard. Computer Vision: Algorithms and Applications. Microsoft Research. — PDF : http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf.

[3] Wikipedia. Energy function // Википедия, свободная энциклопедия. — 2014. — http://en.wikipedia.org/wiki/Energy_function.

[4] Wikipedia. Gaussian mixture model // Википедия, свободная энциклопедия. — 2014. — http://en.wikipedia.org/wiki/Gaussian_mixture_model.

[5] Wikipedia. GrabCut // Википедия, свободная энциклопедия. — 2014. — http://en.wikipedia.org/wiki/GrabCut.

[6] Wikipedia. Graph cuts in computer vision // Википедия, свободная энциклопедия. — 2014. — http://en.wikipedia.org/wiki/Graph_cuts_in_computer_vision.

[7] Wikipedia. K-Means // Википедия, свободная энциклопедия. — 2014. — http://ru.wikipedia.org/wiki/K-means.

[8] Wikipedia. Markov random field // Википедия, свободная энциклопедия. — 2014. — http://en.wikipedia.org/wiki/Markov_random_field.

[9] Wikipedia. Watershed // Википедия, свободная энциклопедия. — 2014. — http://en.wikipedia.org/wiki/Watershed_(image_processing).

[10] Wikipedia. jni // Википедия, свободная энциклопедия. — 2014. — http://en.wikipedia.org/wiki/JNI.

[11] Wikipedia. Евклидово расстояние // Википедия, свободная энциклопедия. — 2014. — http://ru.wikipedia.org/wiki/Евклидово_расстояние.

[12] Wikipedia. МНК // Википедия, свободная энциклопедия. — 2014. — http://ru.wikipedia.org/wiki/МНК.

[13] Wikipedia. Манхэттенское расстояние // Википедия, свободная энциклопедия. — 2014. — http://ru.wikipedia.org/wiki/Расстояние_городских_кварталов.

[14] Wikipedia. Скринсейвер // Википедия, свободная энциклопедия. — 2014. — http://ru.wikipedia.org/wiki/Скринсейвер.