МОСКОВСКИЙ КОМИТЕТ ОБРАЗОВАНИЯ

ЛИЦЕЙ №1533 (ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ)

ВЫПУСКНАЯ РАБОТА

(специальность "Прикладное программирование")

учащейся группы 11.3 Николаевой Натальи Сергеевны

Название работы:

"Система поддержки экспериментов с клеточными культурами"

Руководитель: Каневский Даниил Юрьевич

Консультант: Макарова Екатерина Владимировна

Заказчик: Гроздова Ирина Дмитриевна

Оглавление

Введение

В наши дни активно ведутся исследования в области цитологии. Особенно остро стоит проблема поиска лекарств против рака, лекарств, способных бороться против любых видов раковых клеток. В лабораториях, специализирующихся на этом, проводится множество экспериментов.

В частности, на кафедре Высокомолекулярных соединений Химфака МГУ ведутся подобные исследования.

В каждом эксперименте есть несколько этапов, на каждом из которых необходим контроль за численностью популяции. Лаборатории, не имеющие достаточного финансирования, не могут позволить себе приобрести достаточно дорогостоящее оборудование для оптимизации и автоматизации работы и контроля за клеточной культурой, поэтому все подсчеты ведутся вручную или крайне неточными и неэффективными методами, что сильно снижает скорость и точность подсчетов. Разработанная программа сможет помочь ученым несколько оптимизировать процесс работы и включить в него компьютерные возможности.

. Данная работа сделана по просьбе сотрудников кафедры Высокомолекулярных соединений Химфака МГУ. (И.Д. Гроздовой и д.х.н. Н.С. Мелик-Нубарова).

Актуальность темы

На кафедре Высокомолекулярных соединений МГУ исследуют возможность создания фармакологических препаратов, которые способствуют уничтожению злокачественных клеток.

Стандартное исследование препарата проводится в несколько этапов.

На первом этапе такие препараты тестируют на клетках в культуре, при успешном результате тестируют на животных, и, уже в финале, на людях.

Эта лаборатория специализируется именно на первом этапе тестирования, то есть работе с клетками.

Обычная процедура работы с препаратом такова:

Клетки засеивают в 96-луночные планшеты, в какие-то лунки добавляют тестируемое вещество, в другие нет.

После проведения эксперимента сравнивают кол-во клеток в опытных лунках (с препаратом) и в контроле (ячейки, куда препарат добавлен не был). Причем сравнение происходит не прямым подсчетом, а только лишь определяется по активности одного из ферментов митохондрий.

Этот метод имеет 2 серьезных недостатка.

1) В разных лунках исходно разное кол-во клеток, а добиться одинакового невозможно, так как невозможно отмерить точное количество клеток - слишком мелкий материал. Как следствие этого, происходит ошибка в количественной оценке эффективности препарата. Чтобы увеличить точность исследований, ученые проделывают одну и ту же работу несколько раз, что резко понижает производительность труда и увеличивает его трудоемкость.

2) Кол-во клеток определяют по активности одного из ферментов митохондрий. Так, строго говоря, тестируется влияние препарата на активность этого фермента, а цель работы - уничтожение клеток, поэтому возникает задача создать программу, которая бы считала *клетки*, а не демонстрировала активность фермента.

Данная программа способна решить эту проблему, причем заказчикам достаточно, чтобы обрабатывалось по одной фотографии с каждой ячейки, так как клетки распределены равномерно.

Также этот продукт может помочь в решении ряда других проблем.

Когда ученые выращивают клеточную культуру, над которой впоследствии будет проводиться эксперимент, каждые три дня, без всяких выходных и праздников, они

вынуждены проводить по 4 часа в боксе в лаборатории, занимаясь крайне нудной и долгой процедурой пересеивания клеток, так как клетки размножаются и их становится слишком много в той емкости, которую они занимают, поэтому, если их не пересеивать, им будет не хватать места и питания, и популяция погибнет.

Процедура занимает 4 часа, причем три из них - механическая работа по контролю за численностью популяции. Два часа тратится на приготовление препарата в камере Горяева, после чего ученый около двух часов проводит перед микроскопом, считая клетки, которые он видит. При этом достаточно велика погрешность, вероятность сбиться или ошибиться в подсчете. Так, если эти клетки возможно считать, просто фотографируя их через микроскоп, то скорость и продуктивность работы ученых была бы в разы выше.

Далее - эти же ученые работают с раковыми клетками. Чтобы определить эффективность воздействие препарата, клетки убивающего, доксорубицина, используют флуорисцентную микроскопию - клетки, пораженные препаратом, начинают светиться в УФ-излучении. Так, программа может подсчитывать пораженные клетки и из этого будут делаться выводы об эффективности действия того или иного метода введения доксорубицина, эти исследования призваны помочь в разработке новых лекарств против рака, применимых даже при образовании устойчивых к химиотерапии клеток.

Также возможно применение в вирусологии, нанотехнологиях и везде, где требуется подсчет объектов на достаточно контрастном фоне.

Обзор аналогов

Аналоги данной программе существуют и алгоритмы, выполняющие аналогичные функции, применяются в лабораториях, оборудованных в соответствии с западными сертификатами. Однако в России таких лабораторий крайне мало, так как это оборудование стоит баснословных денег.

Однако, в Серпухове появилась недавно лаборатория, в которой присутствует аппарат, в котором культура растет и развивается, и при этом отслеживается динамика развития культуры: аппарат делает фотографию раз в n часов и анализирует снимок.

Такое оборудование не по карману лабораториям, находящимся на государственном финансировании.

Одна из западных программ, использующаяся для подобных целей – Imaris, производства компании Bitplane. Она, безусловно, имеет куда большие возможности, но для её использования необходимо соотвествующее оборудование. Большая часть программ этой компании платная и не находится в открытом доступе.

Функции системы.

Данная программа является обработчиком фотографий, полученных с помощью светового микроскопа, который распознает в культуре клетки и подсчитывает их количества.

Существует несколько режимов работы программы - одна фотография, один эксперимент или серия экспериментов.

Для каждой фотографии:

а) В начале для обработки загружается полученное с помощью микроскопа изображение.

б) После этого с помощью алгоритма распознавания контура в изображении выделяются клетки.с помощью алгоритма поиска и прослеживания границ.

в) Далее, считается количество клеток на изображении, выводятся данные о количестве клеток на фотографии и их концентрации на единицу поверхности.

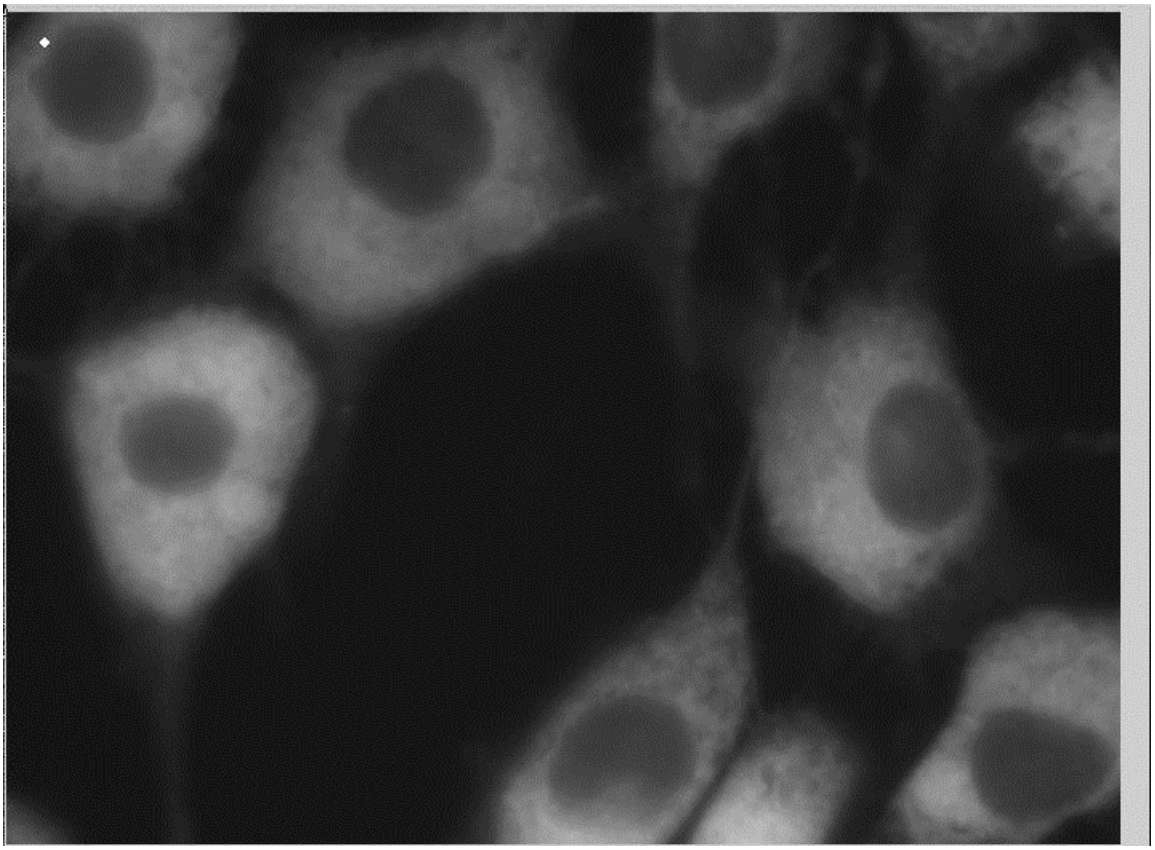
г) Полученные результаты могут быть сохранены и в дальнейшем показана динамика изменения состояния культуры.

Для эксперимента:

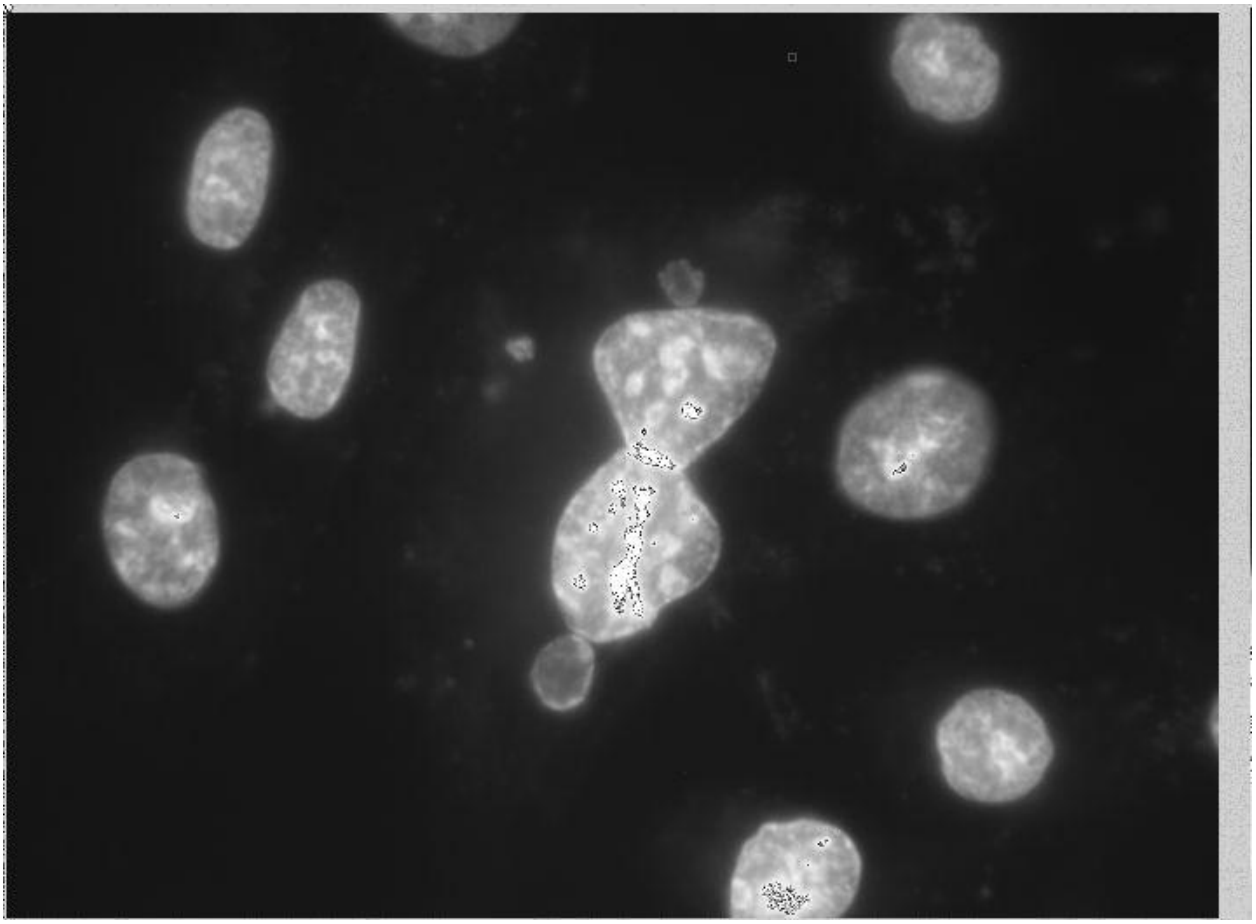
а) Изображения должны быть объединены в папки по экспериментам. Загрузка изображений для одного эксперимента происходит через путь, указанный к этой папке.

б) В связи с этим возможно получение статистических данных по разовому эксперименту, а в случае проведения серии экспериментов, прослеживать динамику развития культуры, а также построение графиков по данным, полученным за некоторый промежуток времени.

Примеры обрабатываемых фотографий:



(флуоресценция цитоплазмы)



(флуоресценция ядер)

Применяемые термины и алгоритмы.

Клетка

Кле́тка — элементарная единица строения и жизнедеятельности всех живых организмов (кроме вирусов, о которых нередко говорят как о неклеточных формах жизни), обладающая собственным обменом веществ, способная к самостоятельному существованию, самовоспроизведению и развитию.

Клетка снаружи ограничена мембраной.

В клетке присутствуют различные функциональные постоянные структуры клеток – органоиды.

Ядро – органиод, в котором содержится генетическая информация клетки.

Цитоплазма – вязкая жидкость, которой наполнена клетка, в которой находятся все органоиды.

Доксорубицин

Доксорубицин – лекарство, применяемое в борьбе против раковых опухолей в процессе химиотерапии. При слишком интенсивной терапии вырабатываются устойчивые к данному препарату клетки, которые практически невозможно уничтожить. Это одна из проблем, из-за которой не все больные раком люди могут быть излечены.

Камера Горяева

Камера Горяева – это прибор, которым пользуются биологи для посчета числа клеток в данном объеме жидкости уже много десятилетий.

Внешне представляет собой прозрачный параллелепипед (предметное стекло), с бороздами и нанесённой микроскопической сеткой. Размеры малых делений клетки сетки составляют 0,05 мм, а больших — 0,2 мм. При этом сетка нанесена на площадку (участок стекла), расположенный на 0,1 мм ниже, чем две соседние площадки. Эти площадки служат для притирания покровного стекла. В результате объем жидкости над квадратом, образованным большими делениями сетки Гаряева, составляет 0,004 микролитра.

Bitmap изображения

BMP (от англ. *Bitmap Picture*) — формат хранения растровых изображений. При этом изображение представляет собой матрицу, каждый элемент которой (пиксель) имеет множество параметров, такие как цвет (задается тремя составляющими RGB), яркость и прочие.

Все алгоритмы для обработки изображений проще реализованы для монохромных изображений (имеющих значением одного параметра цвета яркость, например R=Brightness).

Алгоритм Кэнни выделения контуров

Алгоритм выделения контуров Кэнни состоит из 5 отдельных друг от друга шагов. Он работает с монохромным изображением и основан на поиске наиболее сильного градиента (перехода от меньшего значения яркости к большему).

Шаги алгоритма Кэнни:

1) Размытие изображения

Данный пункт реализован для того, чтобы убрать шумы (дефектные пиксели с яркостью, значительно отличающейся от яркости рядом находящихся пикселей) из изображения. Для этого необходим фильтр Гаусса (Gayssian smoothing filter), который "накладывается" на изображение в каждом из пикселей и тем самым размывает его.

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}$$

Где A – фрагмент изображения 9*9, B – сумма поэлементного произведения матрицы фильтров на матрицу фрагмента изображения.

2) Нахождение градиента.

Далее, по тому же принципу изображение обрабатывается оператором Собеля. Это - матрица 2*2, состоящая из определенных чисел, так, что попиксельное наложение на изображение дает возможность выделить места усиления градиента наиболее ярко. Однако, в процессе использования оператор Собеля не дал желаемых результатов и был заменен на аналогичный оператор Щарра, который отличается тем, что имеет матрицу 3*3 и другое численное наполнение, что дает ему возможность выделять места градиента ярче. На этом этапе вычисляется модуль и направление градиента (theta).

$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$$

3)Отсеивание "ложных" краев

В качестве краев возможно выделить только границы с сильным градиентом, а с более слабым необходимо оставить в стороне. Также это зависит от направления градиента.

4) Двойной порог

Некоторые края в алгориме должны быть помечены как "сильные" и "слабые". Соответственно, для них разный порог.

5) Замыкание контуров

Большинство контуров должно получиться замкнутыми. Для этого анализируется расположение "сильных" краев и "слабых" относительно их, после чего контуры замыкаются.

Для данной задачи и данного типа фотографий этот алгоритм оказался не применим, поскольку разница яркостей подчас может быть очень маленькой, но должна быть отслежена.

Алгоритм симплексного прослеживания граничных коридоров при гексагональной смежности.

Симплексное прослеживание - это метод трассировки граничного корридора в случае, когда структура соседства определяется гексагональной смежностью. Существует 2 варианта смежности, определяющие разные треугольные решетки. Диагональная - решетка, образованная диагоналями с разницей в 45 градусов, антидиагональная - в -45.

Шаги алгоритма:

1) Построение начального треугольника.

Пусть нам нужно проследить границы фигуры, состоящей из черных точек на белом фоне. Сначала для поиска двух граничных точек необходимо провести построчное сканирование изображения и нахождение левой верхней точки, от которой пойдет отсчет. Прослеживание идет так, чтобы черные точки были слева по отношению к движению треугольника, а белые - справа. Как только 2 точки разного цвета найдены, можно обозначить их L - левая, черная точка, R - правая белая.

Положение третьей вершины треугольника - T - определяется в соответствии с правилами:

T.x=R.x; T.y=R.y+(R.x-L.x); для диагональной решетки

T.x=L.x; T.y=L.y+(R.x-L.x); для антидиагональной решетки.

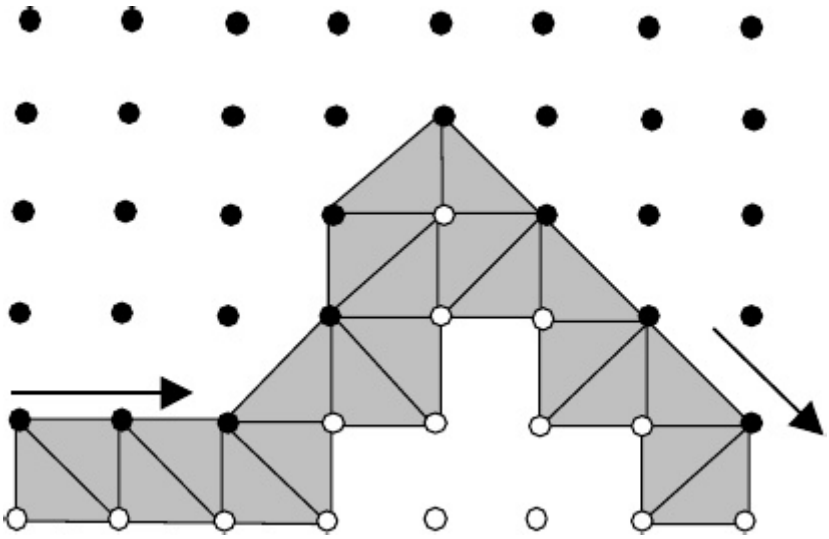
Это - начальное положение катящегося симплекса.

2) Переворот треугольника.

Треугольник переворачивается в соответствии с правилами:

а) Переворот выполняется через сторону треугольника RT или LT, через ту из них, у которой точки имеют разные цвета.

б) Новый треугольник является центрально симметричным старому относительно центра стороны, через которую выполняется переворот.



3) Завершение прослеживания.

Если новый треугольник полностью совпадает с изначальным, прослеживание завершается.

Так как алгоритм симплексного прослеживания годен только для бинаризованных изображений, то необходимо исходные фотографии привести в бинарный вид. Для этого использован алгоритм k-средних, адаптированный для бинаризации.

Алгоритм k-средних для бинаризации.

Вообще алгоритм k-средних применяется не только для того, чтобы бинаризовать изображения. Метод k-средних – метод **кластеризации** данных.Целью задачи кластеризации является разбиение множества объектов на классы (кластеры) на основе некоторой меры сходства объектов.

В общем случае он выглядит таким образом:

Дано:

Набор векторов x_i i=1,…,p;

k – число кластеров, на которые нужно разбить набор x_i;

Найти:

k средних векторов m_j j=1,…,k (центров кластеров);

отнести каждый из векторов x_i к одному из k кластеров;

Шаги алгоритма:

1. Случайным образом выбрать k средних m_j j=1,…,k;
2. Для каждого x_i i=1,…,p подсчитать расстояние до каждого из m_j j=1,…,k, Отнести (приписать) x_i к кластеру j’, расстояние до центра которого $m_{j'}$ минимально;
3. Пересчитать средние m_j j=1,…,k по всем кластерам;

4. Повторять шаги 2, 3 пока кластеры не перестанут изменяться;

Для того, чтобы адаптировать этот алгоритм уже для бинаризованного изображения, необходимо составить гистограмму яркостей - массив в 256 элементов, в каждом элементе которого будет указано количество пикселей, значение яркости которых равно порядковому номеру элемента массива.

Вариант k-средних для бинаризации

1. Выбрать порог бинаризации T равным середине диапазона яркостей (порог бинаризации - это значение яркости, выше которого всем пикселям присваивается белый цвет, а ниже - черный);
2. Вычислить среднюю яркость всех пикселей с яркостью < T m_1, аналогично m_2 для пикселей с яркостью > T;
3. Пересчитать порог $T = (m_1 + m_2) / 2$;
4. Повторять шаги 2, 3 порог не перестанет изменяться;

Ход работы

Изначально тема моей работы звучала как "Моделирование процесса мембранного транспорта". Это достаточно сложная область, в ней необходимы глубокие знания органической химии и молекулярной биологии.

Программа должна была быть призвана проанализировать сформулированную гипотезу о механизмах этого явления, так как при разных условиях экспериментальные данные получаются разные и тяжело говорить о чем-то определенно.

Однако, в процессе анализа предметной области выяснилось, что данных для построения математической модели данного процесса либо не существует в принципе, либо они находятся в закрытом доступе. После того, как стало понятно, что данную задачу при моих возможностях выполнить невозможно, было решено попробовать взяться за другую задачу, не менее актуальную.

В лабораториях, работающих с разного рода клеточными культурами, начиная с 50-60х годов прошлого столетия достаточно остро стоит проблема заражения клеточных культур мельчайшими прокариотическими одноклеточными организмами - микоплазмой. Эти микроорганизмы, паразитируют на клетке, и, когда их много, мешают нормальной её жизнедеятельности, делая невозможным движение и деление, то есть, в случае сильного заражения микоплазмой, клеточная культура через несколько дней погибает.

В связи с этим проводится достаточно много исследований на предмет взаимодействия клеток и микоплазмы, поскольку точные механизмы этого еще не установлены.

Изначально мне предлагалось попробовать поучаствовать в определении механизмов этого взаимодействия, однако опять встал вопрос нехватки точных математических данных.

Однако, эксперименты тоже нуждаются в технической поддержке. И в данном случае эта идея оказалась небесполезной.

Для лаборатории, в которой работает заказчик данного проекта, необходима данная программа для определения наличия заражения в культуре и, возможно, ориентируясь на полученные данные, сделать выводы о механизмах взаимодействия.

Далее, встала проблема о том, что на фотографиях микоплазму сложно отличить от клетки даже "на глаз", так как фото крайне нечеткие, черно-белые и маленького размера, не говоря уже о "компьютерном зрении". Кроме того, необходимо было разобраться с четырьмя абсолютно разными видами фотографий.

Поэтому задача свелась лишь к подсчету количества этих самых клеток, поскольку, как описано в пункте "актуальность темы", это тоже немаловажная для этой лаборатории задача.

Далее начался поиск алгоритма.

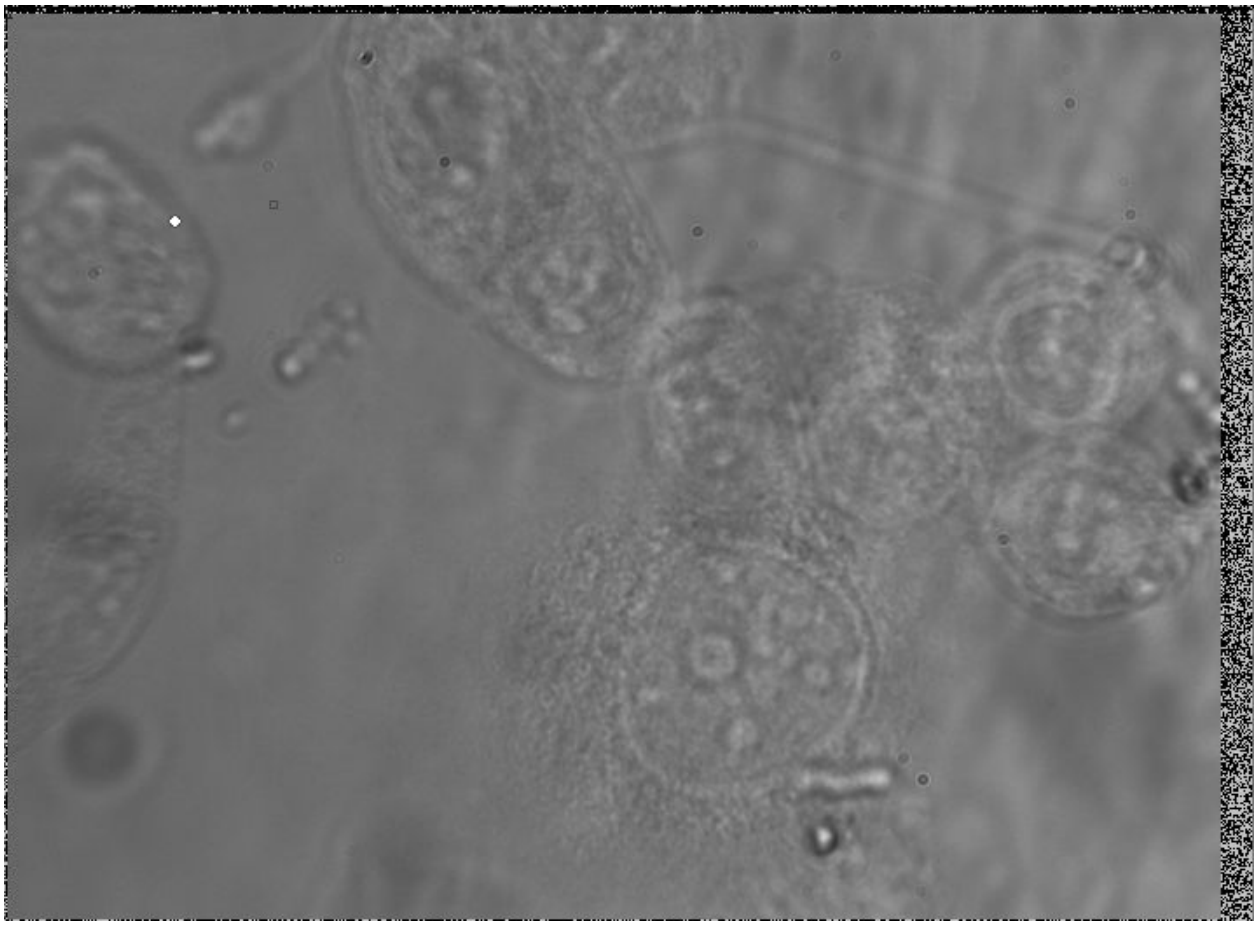
Первоначально испробованный алгоритм Кэнни для выделения контуров желаемых результатов не дал - из-за нечеткости и плохого качества изображений.

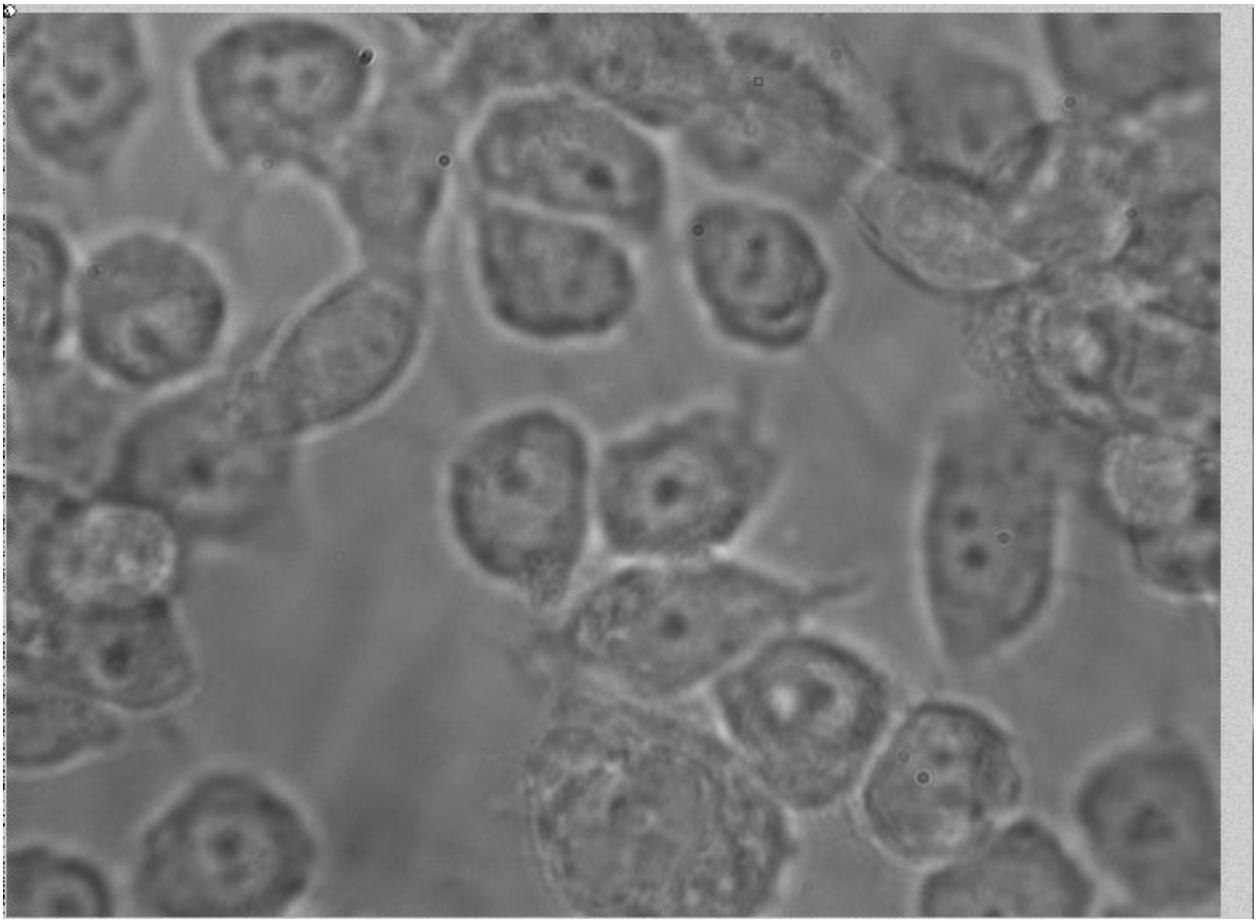
После этого были применены в комплексе алгоритм бинаризации k-средних и, после этого, алгоритм симплексного прослеживания гексагональной смежности (см. пункт "Использованные алгоритмы"). Они дали желаемые результаты и, при определенной доработке, трудностей с подсчетом клеток не возникло.

Направления дальнейших разработок

1) В дальнейшем планируется добавить обработку других видов фотографий, получаемых с микроскопа. На данный момент обработка возможна только для фотографий, в которых присутствует флуоресценция ядра или цитоплазмы клетки. Однако, существуют еще фотографии, где клетки сфотографированы в фазово-контрастном режиме, для них необходим принципиально другой способ обработки, так как фона, на котором расположены клетки, не видно.

Пример фазово-контрастных фотографий:





Для этих фотографий однозначно необходим другой способ бинаризации, а, возможно, и другой способ оконтуривания.

2) Применение уже реализованных алгоритмов к другим задачам.

В лаборатория, сотрудничающая с той, в которой работает мой заказчик, специализируется на изучении насекомых. Им необходимо решить на программном уровне задачу, которая так же решается вручную, хотя технологической сложности из себя не представляет.

Существует фотография жука. У изображенного на ней насекомого есть голова и туловище. Необходимо выяснить отношение размеров головы к размерам туловища, при этом считая длину туловища и головы по прямой, не учитывая объем. В случае сегментированного туловища необходимо считать отношение каждого из сегментов к голове.

Данная программа могла бы быть крайне полезной и экономящей время работы ученых.

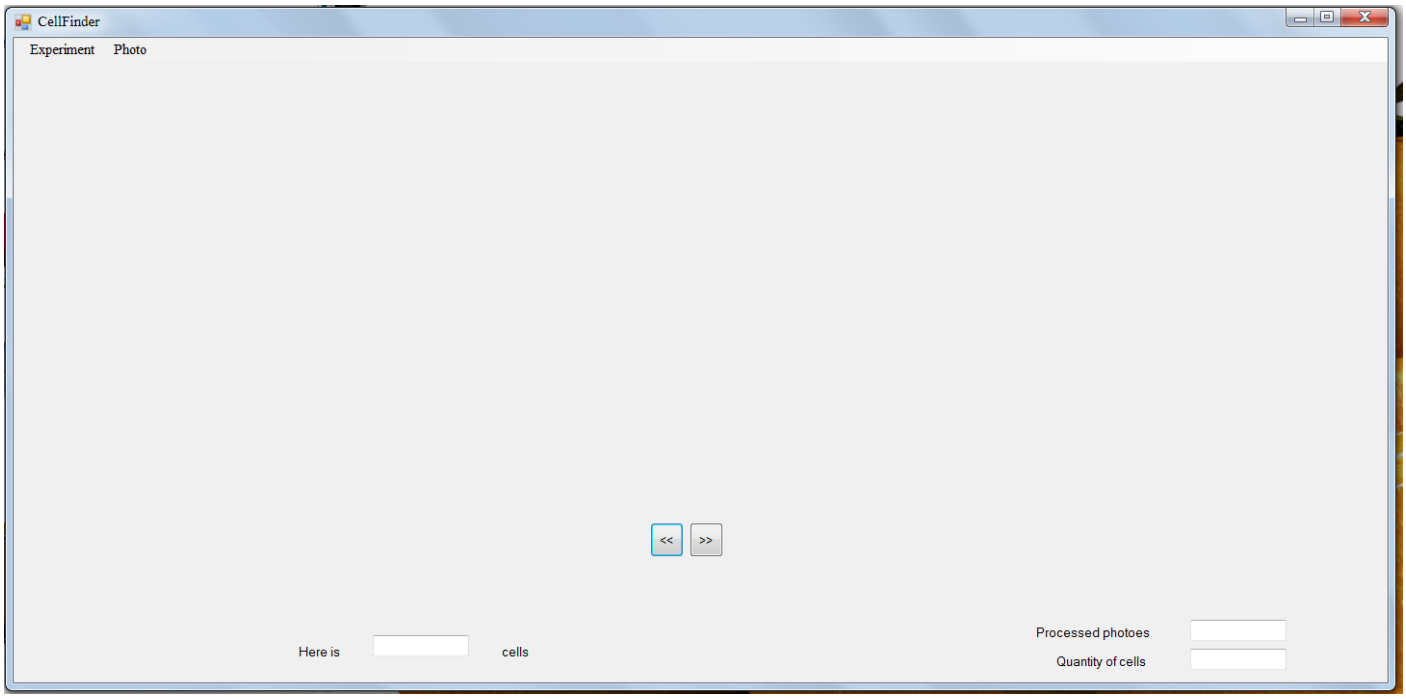
Примеры фотографий жуков:

Parena sp.

Пример использования

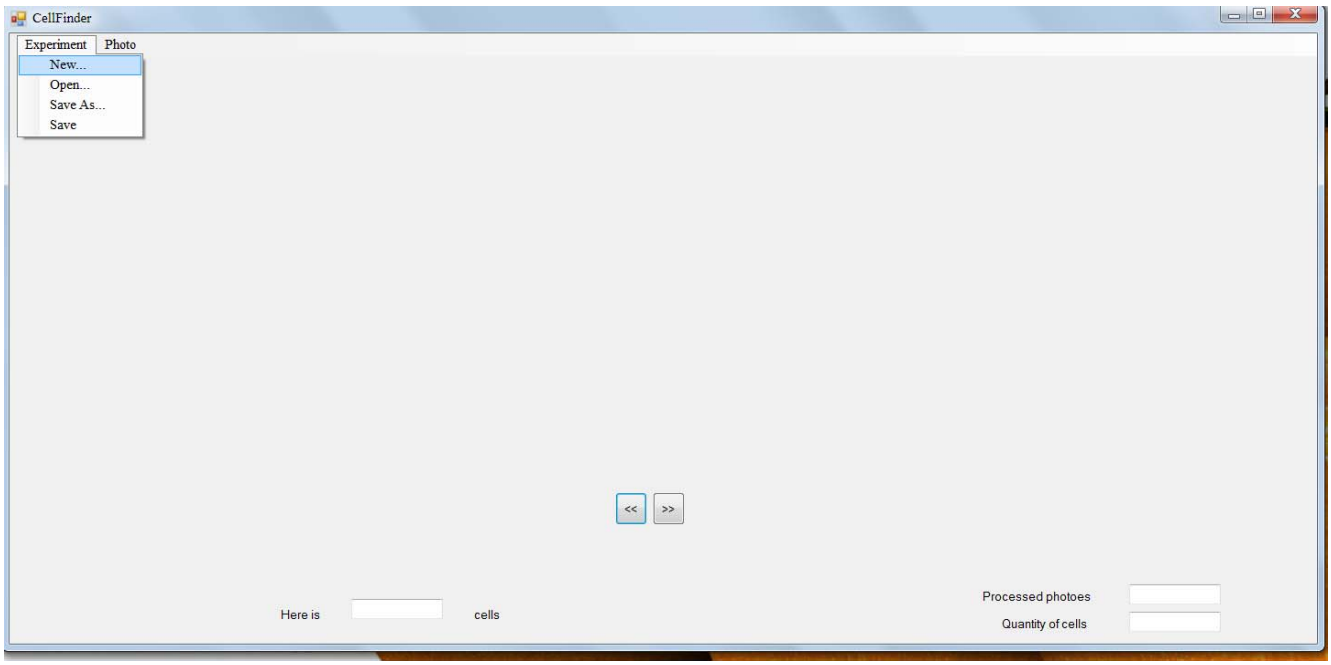
1)

Перед вами стартовое окно программы.

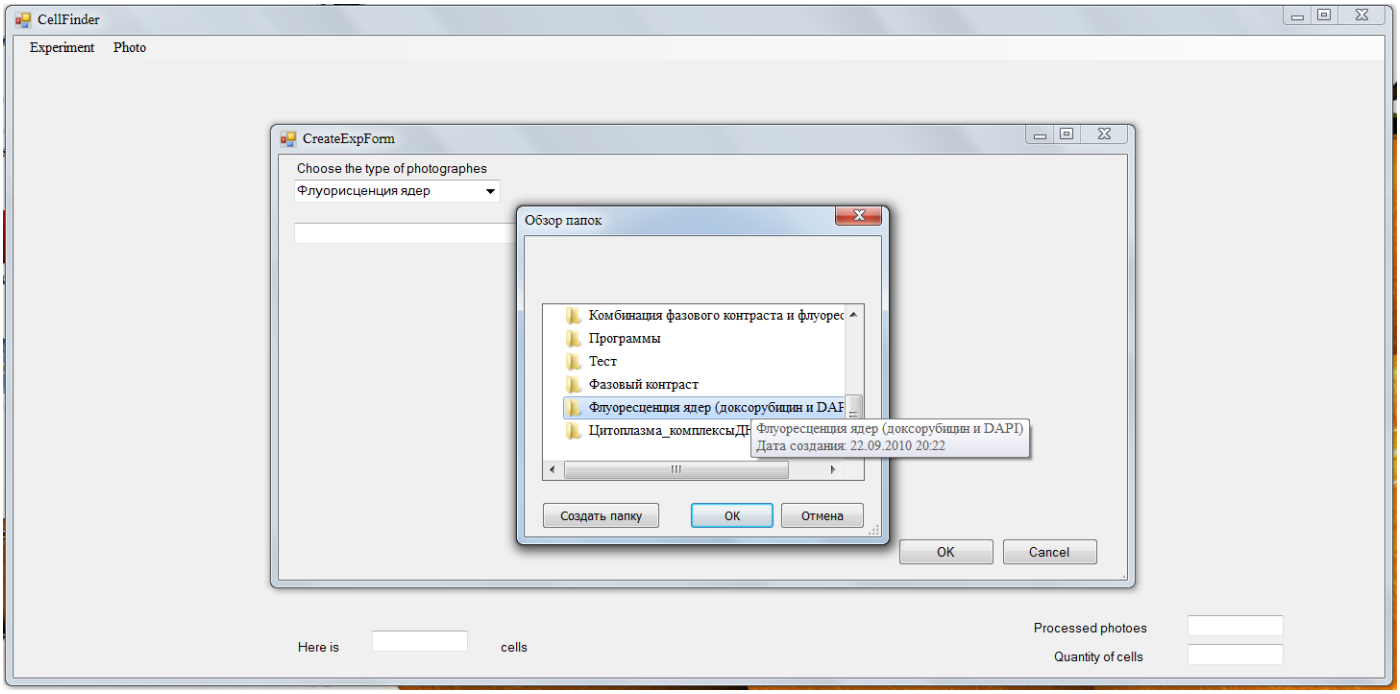


2) Основной режим.

Чтобы загрузить фотографии с нового эксперимента, выберите Experiment>New...

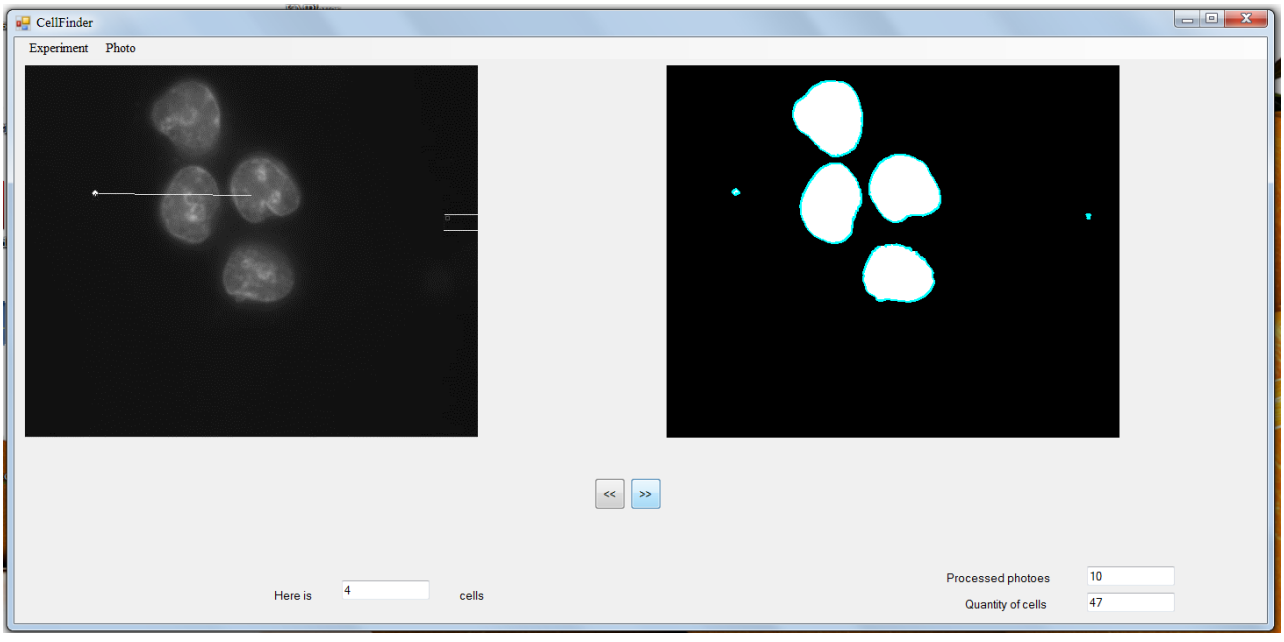


3) Далее в новом окне выберите вид изображений, с которыми бы вы хотели работать и папку, в которой они хранятся. Можно выбрать изображения вида "Флуоресценция ядер" или "Флуоресценция цитоплазмы".

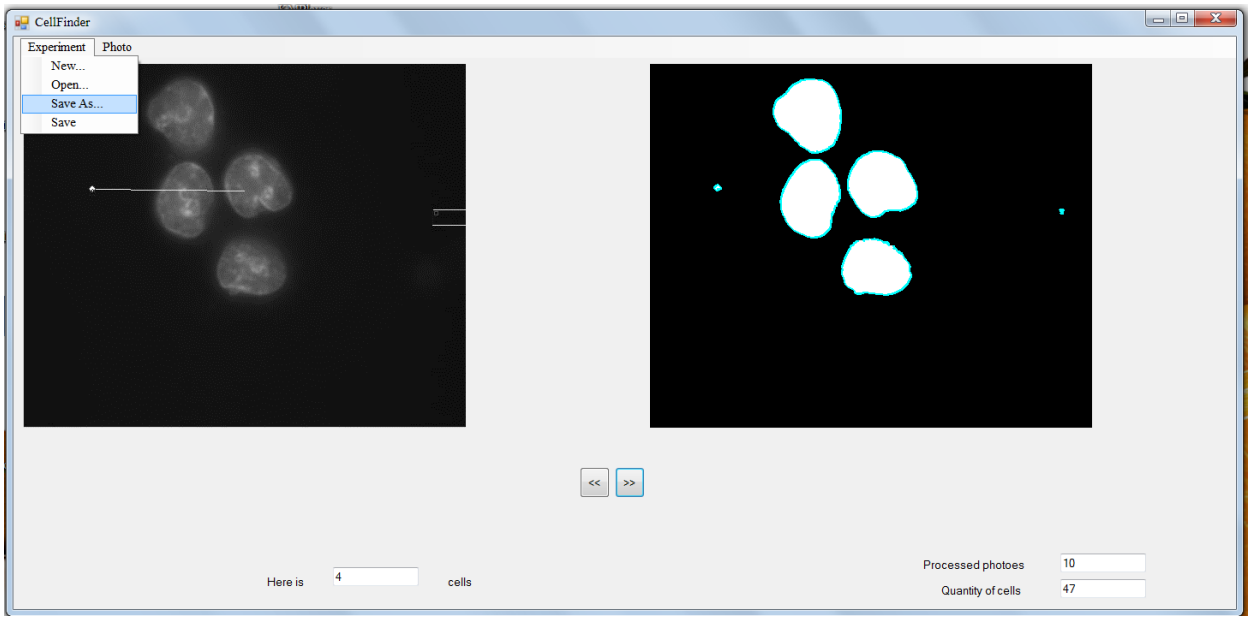


4) На экране появятся изображения: слева - исходное, слева - обработанное. Можно просматривать все изображения в папке стрелками "вперед" и "назад".

В соответствующих полях отображается количество клеток на этой фотографии (слева) и на всех фотографиях, участвующих в эксперименте, а также количество обработанных фотографий (в правом нижнем углу).

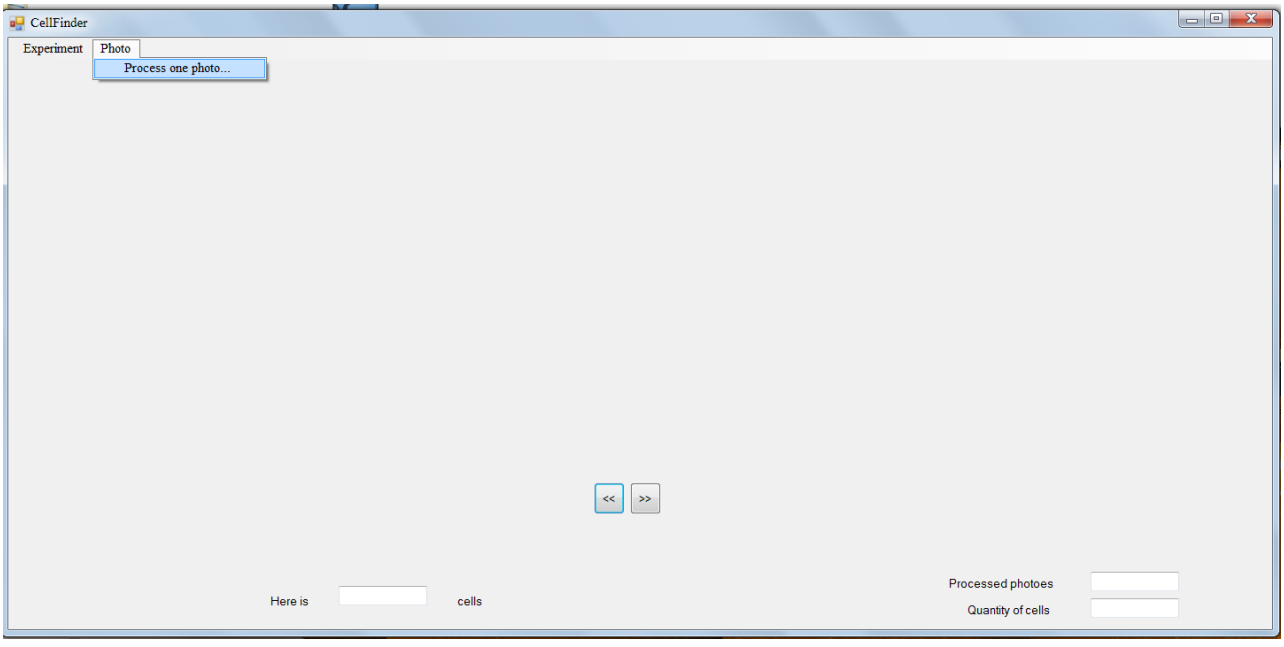


5) Проведенный эксперимент есть возможность сохранить в xml - таблицу, чтобы потом загрузить все данные этого эксперимента еще раз.

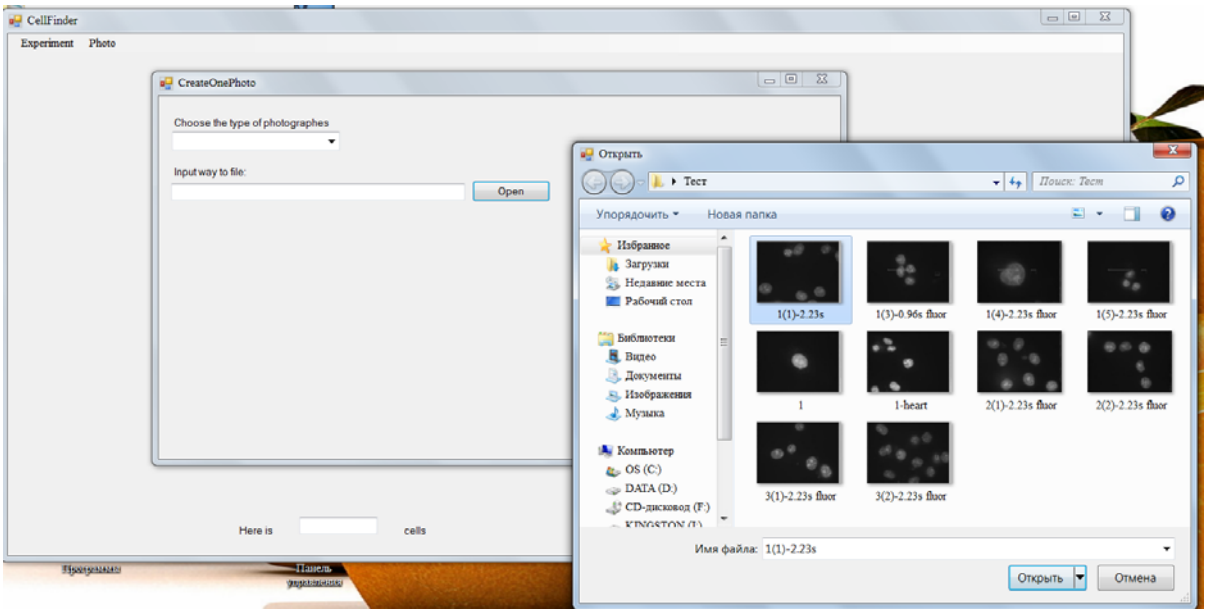


6) Другой режим работы программы - обработка одной фотографии.

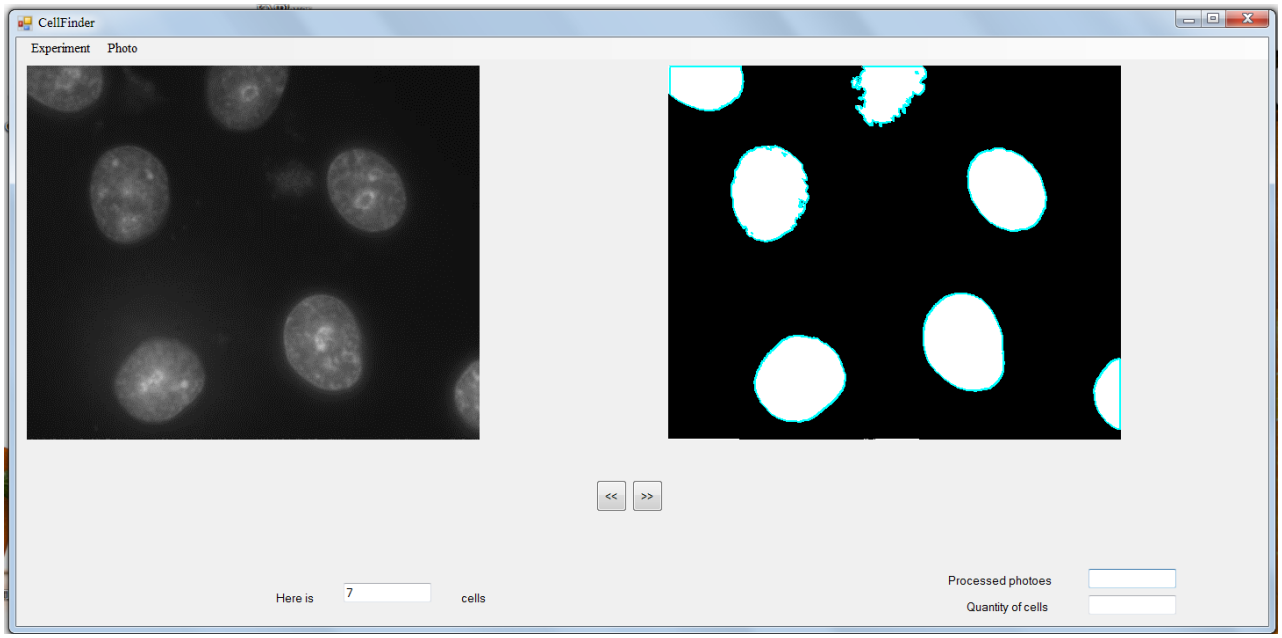
Для этого выберите в меню Photo>Process one photo.



7) Далее выберите вид фотографии и определите в диалоговом окне её местоположение.



8) На экране после обработки появятся в том же виде обработанная и необработанная фотографии. В левом окне выведется количество клеток на фотографии.



Заключение

Итак, была разработана программа, которая может помочь ученым автоматизировать процесс разработки и сделать его точнее.

Применение она может находить во всех областях, где необходимо считать объекты на контрастном им фоне, а таких задач может быть немало.

Данная разработка вызвала большой интерес у сотрудников кафедры Высокомолекулярных соединений, а также у некоторых людей, также связанных с наукой.

При дальнейшей разработке расширятся возможности программы, и она будет применима в куда более широком спектре задач и к более широкому классу фотографий.

Программа включает в себя пользовательские возможности для удобства работы с ней, такие как сохранение эксперимента и повторная загрузка. Она несет в себе только те функции, которые необходимы заказчику, при этом все данные, которые могут быть полезны, отображаются.

В ходе работы была достаточно глубоко изучена предметная область по нескольким темам, при этом не был найден математический аппарат, по которому возможно было бы моделировать процессы.

В итоговом проекте реализованы возможности поддержки эксперимента, которые могут помочь, в частности, в проведении исследований для более подробного выяснения математических закономерностей клеточных процессов.

Использованная литература

- Л. Шапиро, Д. Стокман "Компьютерное зрение" (Москва, БИНОМ. Лаборатория знаний, 2006).
- Л.М. Местецкий "Непрерывная морфология бинарных изображений. Фигуры, скелеты, цикруляры". (Москва, Физматлит, 2009).
- http://msdn.microsoft.com/
- http://en.wikipedia.org/
  http://ru.wikipedia.org/ - Википедия, статьи об изображениях и их хранении, а также об алгоритмах их обработки.

Приложение: Фрагменты исходного кода

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
using System.Xml;
using System.Xml.Serialization;


namespace Diplom
{
   public partial class CellFinder : Form
   {
      PhotoSerie TempSerie;
      CellImage TempImage;
      int k;

      public CellFinder()
      {
         InitializeComponent();

      }

      private void CellFinder_Paint(object sender, PaintEventArgs e)
      {

         Graphics g = e.Graphics;
         Bitmap PH1;
         Bitmap TempPhoto;

         if (TempSerie!=null)
         {
            string[] filePaths = Directory.GetFiles(TempSerie.folderAdress);
            string ext = Path.GetExtension(filePaths [k]);
            if (ext == ".jpg")
            {
               TempPhoto = new Bitmap(TempSerie[k].adress);
               Rectangle rect = new Rectangle(Img.widthCut, Img.heightCut, TempPhoto.Width - 2 * Img.widthCut,
TempPhoto.Height - Img.heightCut);
               PH1 = TempPhoto.Clone(rect, System.Drawing.Imaging.PixelFormat.Format32bppArgb);
               TempPhoto = PH1;

               pictureBox1.Image = TempPhoto;
               pictureBox1.Show();
               pictureBox2.Image = TempSerie[k].Photo;
               pictureBox2.Show();
               textBox4.Clear();
               textBox4.SelectedText = TempSerie[k].count.ToString();
            }
            else
               k++;

            textBox1.Clear();
            textBox1.SelectedText = TempSerie.count.ToString();
            textBox2.Clear();
            textBox2.SelectedText = TempSerie.cellCount.ToString();
         }

         else
            if (TempImage != null)
            {
               TempPhoto = new Bitmap(TempImage.adress);
```

```
            Rectangle rect = new Rectangle(Img.widthCut, Img.heightCut, TempPhoto.Width - 2 * Img.widthCut,
TempPhoto.Height - Img.heightCut);
            PH1 = TempPhoto.Clone(rect, System.Drawing.Imaging.PixelFormat.Format32bppArgb);
            TempPhoto = PH1;

            pictureBox1.Image = TempPhoto;
            pictureBox1.Show();
            pictureBox2.Image = TempImage.Photo;
            pictureBox2.Show();

            textBox1.Clear();
            textBox2.Clear();
        }
    }



    private void button1_Click(object sender, EventArgs e)
    {
       if (k > 0)
          k--;
       Refresh();
    }

    private void button2_Click(object sender, EventArgs e)
    {
       if (k < TempSerie.count - 1)
          k++;
       Refresh();
    }

    private void experimentToolStripMenuItem_Click(object sender, EventArgs e)
    {
       TempSerie = null;
       k = 0;
       int val = 0;
       DialogResult result = new DialogResult();
       CreateExpForm fm = new CreateExpForm();
       fm.buttonOK.DialogResult = DialogResult.OK;
       fm.buttonCancel.DialogResult = DialogResult.Cancel;
       result = fm.ShowDialog();

       if (result == DialogResult.OK)
       {

          while (val == 0)
          {

             string text = fm.comboBox1.Text;

             if (text == "Фазовый контраст")
                val = 1;
             if (text == "Фазовый контраст+флуоресценция")
                val = 2;
             if (text == "Флуоресценция ядер")
                val = 3;
             if (text == "Флуоресценция цитоплазмы")
                val = 4;
             if (val == 0)
             {
                MessageBox.Show("Error: No image type");
                fm.Visible = false;
                result = fm.ShowDialog();
             }
          }
          try
          {
             if (result==DialogResult.OK)
             {
                string folderAdress = fm.folderBrowserDialog1.SelectedPath;
```

```
            if (val != 0)
            {
               TempSerie = new PhotoSerie();
               TempSerie.type = val;
               TempSerie.folderAdress = folderAdress;
               TempSerie.SerieProcessing();

               Refresh();
            }
         }
      }
      catch
      {
         MessageBox.Show("Error: No way to folder");
         fm.Close();
      }

   }
   if (result == DialogResult.Cancel)
   {
      fm.Close();
   }
   Refresh();
}

private void processOnePhotoToolStripMenuItem_Click(object sender, EventArgs e)
{
   TempSerie = null;
   int val = 0;
   DialogResult result = new DialogResult();
   CreateOnePhoto fm = new CreateOnePhoto();
   fm.buttonOK.DialogResult = DialogResult.OK;
   fm.buttonCancel.DialogResult = DialogResult.Cancel;
   result = fm.ShowDialog();

   if (result == DialogResult.OK)
   {
      while (val==0)
      {


         string text = fm.comboBox1.Text;

         if (text == "Фазовый контраст")
            val = 1;
         if (text == "Фазовый контраст+флуоресценция")
            val = 2;
         if (text == "Флуоресценция ядер")
            val = 3;
         if (text == "Флуоресценция цитоплазмы")
            val = 4;
         if (text == "")
         {
            fm.Show();
            fm.Visible = false;
            result = fm.ShowDialog();
         }
      }

      try
      {
         string n = fm.openFileDialog1.FileName;
         if (val != 0)
         {
            TempImage = new CellImage();
            TempImage.adress = n;
            TempImage.Resave();
            ImgSequence ImageSeq = new ImgSequence();
            Contrasting C = new Contrasting();
```

```
                Binary B = new Binary(val);
                Noise N = new Noise();
                Polygones P = new Polygones();
                BlackEdge E = new BlackEdge();
                //ImageSeq.Add(C);
                ImageSeq.Add(B);
                ImageSeq.Add(N);
                ImageSeq.Add(E);
                ImageSeq.Add(P);

                try
                {
                   TempImage = (CellImage) ImageSeq.Launch((Img)TempImage);
                   Refresh();
                   textBox4.Clear();
                   textBox4.SelectedText = TempImage.count.ToString();
                }
                catch
                {
                }
             }
          }
          catch
          {
             MessageBox.Show("Error: No way to file");
          }

       }
       if (result == DialogResult.Cancel)
       {
          fm.Close();
       }
       Refresh();

    }

    private void saveToolStripMenuItem_Click(object sender, EventArgs e)
    {
       if (TempSerie != null)
       {

          saveFileDialog1.ShowDialog();


          try
          {
             TextWriter WriteFileStream = new StreamWriter(saveFileDialog1.FileName + ".xml");
             XmlSerializer SerializerObj = new XmlSerializer(typeof(PhotoSerie));
             SerializerObj.Serialize(WriteFileStream, TempSerie);
             WriteFileStream.Close();
          }

          catch
          {
             MessageBox.Show("Error: no file name");
          }
       }
    }

    private void openToolStripMenuItem_Click(object sender, EventArgs e)
    {
       openFileDialog1.ShowDialog();


       try
       {
          FileStream ReadFileStream = new FileStream(openFileDialog1.FileName, FileMode.Open, FileAccess.Read,
FileShare.Read);
       XmlSerializer SerializerObj = new XmlSerializer(typeof(PhotoSerie));
       TempSerie = (PhotoSerie)SerializerObj.Deserialize(ReadFileStream);
```

```
            ReadFileStream.Close();
            Refresh();
            }
            catch
            {
               MessageBox.Show("Error: cannot open this file");
            }
         }

      }
   }

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Diplom
{
   class BlackEdge: ImgProcess
   {
      public override Img Process(Img A)
      {
         int x1, y1;

         x1 = 0;
         y1 = 0;
         for (x1 = 0; x1 < A.Photo.Width; x1++)
         {
            A.Arraybr[x1 * A.Photo.Height + y1] = 0.0;
            A.Arraybr[x1 * A.Photo.Height + y1 + 1] = 0.0;
         }
         x1--;
         for (y1 = 0; y1 < A.Photo.Height; y1++)
         {
            A.Arraybr[x1 * A.Photo.Height + y1] = 0.0;
            A.Arraybr[(x1 - 1) * A.Photo.Height + y1] = 0.0;
         }
         y1--;
         for (x1 = 0; x1 < A.Photo.Width; x1++)
         {
            A.Arraybr[x1 * A.Photo.Height + y1] = 0.0;
            A.Arraybr[x1 * A.Photo.Height + y1 - 1] = 0.0;
         }
         x1 = 0;
         for (y1 = 0; y1 < A.Photo.Height; y1++)
         {
            A.Arraybr[x1 * A.Photo.Height + y1] = 0.0;
            A.Arraybr[(x1+1) * A.Photo.Height + y1] = 0.0;
         }
         return base.Process(A);
      }
   }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Diplom
{
   class Binary:ImgProcess
   {
      double val;
      int [] DiagBri;
      int type;

    /*   public Binary(int value)
      {
```

```
    val = value;
}
*/
public Binary(int type1)
{
    val = 256;
    DiagBri = new int[256];
    type=type1;
}


public override Img Process(Img A)
{
    A.ToArray();
    for (int i = 0; i < A.Photo.Width * A.Photo.Height; i++)
    {
        DiagBri[(int)A.Arraybr[i]] = DiagBri[(int)A.Arraybr[i]]+1;
    }


    if (val == 256)
        val = Threshold(A);

    for (int i = 0; i < A.Photo.Width*A.Photo.Height; i++)
    {
        if (A.Arraybr[i] < val)
            A.Arraybr[i] = 0;
        else A.Arraybr[i] = 255;
    }

    A.ToImage(A.Arraybr);
    return base.Process(A);
}

public double Threshold(Img A)
{

    int T = 128, T1 = 256;
    int midBriMin = 0, midBriMax = 0;
    int maxCount = 0, minCount = 0;

    while (T!=T1)
    {
        if (T1 != 256)
            T = T1;
        for (int i = 0; i < T; i++)
        {
            midBriMin += DiagBri[i] * i;
            minCount += DiagBri[i];
        }
        if (minCount == 0)
            minCount = 1;
        midBriMin = midBriMin / minCount;

        for (int i = T; i < 256; i++)
        {
            midBriMax += DiagBri[i] * i;
            maxCount += DiagBri[i];
        }

        if (maxCount == 0)
          maxCount = 1;


        midBriMax = midBriMax / maxCount;
        T1 = (midBriMax + midBriMin) / 2;
        midBriMax = 0;
        midBriMin = 0;
        minCount = 0;
        maxCount = 0;
```

```
            }
            return T1;

        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Drawing.Imaging;
using System.Drawing;
using System.Runtime.InteropServices;
using System.ComponentModel;
using System.Data;
using System.Windows.Forms;

namespace Diplom
{
    [Serializable()]
    public class CellImage : Img
    {

        public CellImage()//конструктор
        {
                widthCut = 100;
                heightCut = 100;


            /* Gxmap = new double[Photo.Width * Photo.Height];
             Gymap = new double[Photo.Width * Photo.Height];
             Gmap = new double[Photo.Width * Photo.Height];
             theta = new double[Photo.Width * Photo.Height];
             Temparraybr = new double[Photo.Width * Photo.Height];
            */
            //Cutting();

            // realHeight = Photo.Height - 2 * heightCut;
            //realWidth = Photo.Width - 2 * widthCut;

        }
    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Diplom
{
    public partial class CreateExpForm : Form
    {
        public CreateExpForm()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            folderBrowserDialog1.ShowDialog();
            string n = folderBrowserDialog1.SelectedPath;
            textBox1.Clear();
            textBox1.SelectedText = n;
```

```
        }

        private void buttonCancel_Click(object sender, EventArgs e)
        {

        }

        private void buttonOK_Click(object sender, EventArgs e)
        {
        }

        private void CreateExpForm_Load(object sender, EventArgs e)
        {
            comboBox1.Text = "Флуоресценция ядер";
        }

    }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Diplom
{
    public partial class CreateOnePhoto : Form
    {
        public CreateOnePhoto()
        {
            InitializeComponent();
        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {
            openFileDialog1.ShowDialog();
            string n = openFileDialog1.FileName;
            textBox1.Clear();
            textBox1.SelectedText = n;
        }

        private void CreateOnePhoto_Load(object sender, EventArgs e)
        {
            comboBox1.Text = "Флуоресценция ядер";
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Diplom
{
    class Experiment:List<PhotoSerie>
    {
        int expCount;
        public Experiment()
        {
```

```
        }

        public void ExpStats()
        {
            int k = 0;
            int m = 0;
            foreach (PhotoSerie A in this)
            {
                k += A.count;
                m++;
            }
            expCount = k / m;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Drawing.Imaging;
using System.Drawing;
using System.Runtime.InteropServices;
using System.ComponentModel;
using System.Data;
using System.Windows.Forms;


namespace Diplom
{
    [Serializable()]
    public class Img
    {

        public string adress;//адрес, где расп. фотография

        /*public double[] Gxmap;//массив градиента по Ox
        public double[] Gymap;//массив градиента по Oy
        public double[] Gmap;//масссив итогового градиента
        public double[] theta;//массив направлений градиента
        public double[] Temparraybr;//временный массив яркостей (нужен, чтобы не портить исходный в процессе сглаживания)
       */
        public int count;

       public static int widthCut;
       public static int heightCut;

        [System.Xml.Serialization.XmlIgnore()]
        //[System.NonSerialized()]
        public Bitmap Photo;//фотография

        //[NonSerialized()]
        [System.Xml.Serialization.XmlIgnore()]
        public double[] Arraybr;//массив яркостей

      // public int realHeight;
      // public int realWidth;

        public Img()
        {
        }

        public static Bitmap Indexed2Image(Image img, System.Drawing.Imaging.PixelFormat fmt)
        {
            Bitmap bmp = new Bitmap(img.Width, img.Height, fmt);
            Graphics gr = Graphics.FromImage(bmp);
            gr.DrawImage(img, 0, 0);
            gr.Dispose();
```

```
   return bmp;
}

public void ToArray()//перевод изображения в массив яркостей
{
   for (int i = 0; i < Photo.Width; i++)
      for (int j = 0; j < Photo.Height; j++)
      {
         Color Pix = Photo.GetPixel(i, j);
         Arraybr[i * Photo.Height + j] = 255*Pix.GetBrightness();
      }

}

public void SetColor(byte br, int k, int l)//присвоить цвет указанному пикселю
{
   Color C1 = Color.FromArgb
            (
               (int)(br),
               (int)(br),
               (int)(br));

   Photo.SetPixel(k, l, C1);
}

public void ToImage(double [] Ar)//перевод массива яркостей в монохромное изображение
{
   for (int i = 0; i < Photo.Width; i++)
      for (int j = 0; j < Photo.Height; j++)
      {
         byte br = (byte)(Ar[i*Photo.Height+j]);
         SetColor(br, i, j);

      }

}

public void Edging()
{
   //делать черную грань по краю
      int x1, y1;

      x1 = 0;
      y1 = 0;
      for (x1=0; x1<Photo.Width; x1++)
      Arraybr[x1 * Photo.Height + y1] = 0.0;
      x1--;
      for (y1=0; y1<Photo.Height; y1++)
         Arraybr[x1 * Photo.Height + y1] = 0.0;
      y1--;
      for (x1=0; x1<Photo.Width; x1++)
         Arraybr[x1 * Photo.Height + y1] = 0.0;

      x1=0;
      for (y1=0; y1<Photo.Height; y1++)
         Arraybr[x1 * Photo.Height + y1] = 0.0;


}

public void Resave()
{
    Bitmap PH1 = new Bitmap(adress);
      Bitmap bmp = new Bitmap(PH1.Width, PH1.Height);
      Graphics gr = Graphics.FromImage(bmp);
      gr.DrawImage(PH1, 0, 0);
      Photo = bmp;


  Rectangle rect = new Rectangle(widthCut, heightCut, Photo.Width-2*widthCut,Photo.Height - heightCut);
```

```
            PH1 = Photo.Clone(rect, System.Drawing.Imaging.PixelFormat.Format32bppArgb);

            Photo = PH1;
               gr.Dispose();
            // Photo = Indexed2Image((Image)Photo, PixelFormat.DontCare);
               Arraybr = new double[Photo.Width * Photo.Height];
               Edging();

         }


      }
   }

      using System;
   using System.Collections.Generic;
   using System.Linq;
   using System.Text;

   namespace Diplom
   {
      class ImgProcess
      {
         public virtual Img Process(Img A) { return A; }//сделано коряво для совместимости
      }
   }

   using System;
   using System.Collections.Generic;
   using System.Linq;
   using System.Text;

   namespace Diplom
   {
      class Noise:ImgProcess
      {
         double [] matr;
         int redcolor;
         int blackcolor;

         public Noise()
         {
            matr = new double[9];
         }

         public override Img Process(Img A)
         {
            double [] Temparraybr = new double[A.Photo.Width * A.Photo.Height];
            for (int i = 0; i < A.Photo.Width * (A.Photo.Height-2)-2; i++)
            {
               matr[0] = A.Arraybr[i];
               matr[1] = A.Arraybr[i + 1];
               matr[2] = A.Arraybr[i + 2];
               matr[3] = A.Arraybr[i + A.Photo.Height];
               matr[4] = A.Arraybr[i + A.Photo.Height + 1];
              matr[5] = A.Arraybr[i + A.Photo.Height + 2];
               matr[6] = A.Arraybr[i + 2 * A.Photo.Height];
               matr[7] = A.Arraybr[i + 2 * A.Photo.Height+1];
               matr[8] = A.Arraybr[i + 2 * A.Photo.Height+2];

               for (int j=0; j<9; j++)
               {
                  if (matr[j] == 0)
                     blackcolor++;
                  else
                     redcolor++;
               }

               if (redcolor > blackcolor)
```

```
            {
               for (int k = 0; k < 9; k++)
               {
                  Temparraybr[i] = 255.0;
                  Temparraybr[i + 1] = 255.0;
                  Temparraybr[i + 2] = 255.0;
                  Temparraybr[i + A.Photo.Height] = 255.0;
                  Temparraybr[i + A.Photo.Height + 1] = 255.0;
                  Temparraybr[i + A.Photo.Height + 2] = 255.0;
                 Temparraybr[i + 2 * A.Photo.Height] = 255.0;
                  Temparraybr[i + 2 * A.Photo.Height + 1] = 255.0;
                  Temparraybr[i + 2 * A.Photo.Height + 2] = 255.0;

               }
            }

            else
            {
               for (int k = 0; k < 9; k++)
               {
                  Temparraybr[i] = 0.0;
                  Temparraybr[i + 1] = 0.0;
                  Temparraybr[i + 2] = 0.0;
                  Temparraybr[i + A.Photo.Height] = 0.0;
                  Temparraybr[i + A.Photo.Height + 1] = 0.0;
                  Temparraybr[i + A.Photo.Height + 2] = 0.0;
                  Temparraybr[i + 2 * A.Photo.Height] = 0.0;
                  Temparraybr[i + 2 * A.Photo.Height + 1] = 0.0;
                  Temparraybr[i + 2 * A.Photo.Height + 2] = 0.0;

               }
            }
            redcolor = 0;
            blackcolor = 0;
         }

         for (int i = 0; i < A.Photo.Width * A.Photo.Height; i++)
            A.Arraybr[i] = Temparraybr[i];

         A.ToImage(A.Arraybr);
         return base.Process(A);
      }
   }
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;

namespace Diplom
{
   [Serializable()]
   public class PhotoSerie:List<CellImage>
   {
      public int type;
      public string folderAdress;

      public int cellCount;
      public int count;

      public PhotoSerie()
      {
```

```
        }

        public void LoadImages()
        {

        }

        public void SerieProcessing()
        {
            CellImage K;
            this.cellCount = 0;
            this.count = 0;

            ImgSequence ImageSeq = new ImgSequence();
             Binary B = new Binary(type);
            Noise N = new Noise();
            Polygones P = new Polygones();
            BlackEdge E = new BlackEdge();
            ImageSeq.Add(B);
            ImageSeq.Add(N);
            ImageSeq.Add(E);
            ImageSeq.Add(P);

            string[] filePaths = Directory.GetFiles(folderAdress);
            foreach (string k in filePaths)
            {
                string ext = Path.GetExtension(k);
                if (ext == ".jpg")
                {
                    CellImage A = new CellImage();
                    A.adress = k;
                    A.Resave();
                    K = (CellImage) ImageSeq.Launch(A);
                    this.cellCount += K.count;
                    this.count++;
                    this.Add(K);
                }

            }

        }

        public void ClearList()
        {
            foreach (CellImage K in this)
            {
                K.Photo = null;
                K.Arraybr = null;
                /*K.Gmap = null;
                K.Gxmap = null;
                K.Gymap = null;
                K.theta = null;
                 */
            }
        }
    }
}

using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using System.Drawing;

    namespace Diplom
    {
        class Points
        {
            public int x;
            public int y;
```

```
    public int col;
    public bool IsUsed;
    Img A;


    public Points(int x1, int y1, Img A1)
    {
        x = x1;
        y = y1;
        IsUsed = false;
        A = A1;
        GetPoint();
    }

    public void GetPoint()
    {
        //Color C = A.Photo.GetPixel(x, y);
        if (A.Arraybr[x * A.Photo.Height + y] == 0.0)
        {
            col = 0;
        }
        else
        {
            if (A.Arraybr[x * A.Photo.Height + y] == 255.0)
                col = 1;
            else
            {
                col = 2;
                IsUsed = true;
            }
        }
    }
}

class Polygone : List<Points>
{
    public List<int> xcoord;
    public List<int> ycoord;
   public bool IsConvex;

    public Polygone(int x, int y, Img A)
    {
        xcoord = new List<int>();
        ycoord = new List<int>();

        xcoord.Add(x);
        xcoord.Add(x + 1);
        ycoord.Add(y);
        ycoord.Add(y);
        if ((A.Arraybr[x * A.Photo.Height + y] == 255) && (A.Arraybr[(x + 1) * A.Photo.Height + y] == 0))
        {
            IsConvex = false;
        }
        else
        {
            IsConvex = true;
        }
    }

    public void PolyEdge(Img A)//определять границы многоугольника
    {
        Points L, R, T;
        int m = 1;

        R = new Points(xcoord[0], ycoord[0], A);
        this.Add(R);
        L = new Points(xcoord[1], ycoord[1], A);
        this.Add(L);
        T = new Points(L.x, L.y - (R.x - L.x), A);
```

```
            //переворот треугольниа, прослеживание границ
            Points N;

            while ((this.Count == 2) || ((T.x + (L.x - R.x) < A.Photo.Width) && (T.y + (L.y - R.y) < A.Photo.Height)) && ((T.x + (L.x - R.x) > 0) && (T.y + (L.y - R.y) > 0)))
            {
               if (T.col == 2)
                  break;

               if (this[0] == T)
                  break;
               if ((T.col == 0) && (L.col == 0) && (R.col == 0))
                  break;

               if (((xcoord[0] == xcoord[m]) && (ycoord[0] == ycoord[m])) || ((xcoord[1] == xcoord[m]) && (ycoord[1] == ycoord[m])) && (m != 1))
                  break;



               if (L.col != T.col)
               {
                  try
                  {
                     this.Add(T);
                     xcoord.Add(T.x);
                     ycoord.Add(T.y);
                     m++;
                     N = new Points(T.x + (L.x - R.x), T.y + (L.y - R.y), A);
                     R = T;
                     T = N;
                  }
                  catch
                  {
                     break;
                  }
               }

               if (R.col != T.col)
               {
                  try
                  {
                     this.Add(T);
                     xcoord.Add(T.x);
                     ycoord.Add(T.y);
                     m++;
                     N = new Points(T.x + (R.x - L.x), T.y + (R.y - L.y), A);
                     L = T;
                     T = N;
                  }

                  catch
                  {
                     break;
                  }
               }
               int n = 0;
               foreach (Points P in this)
               {
                  if ((P.x == T.x) && (P.y == T.y))
                     break;
                  n++;
               }
               if (n != this.Count)
                  break;
            }

            Recolor(A);
         }
```

```
    public void Recolor(Img A)
    {
        Form1 f = new Form1();
        foreach (Points K in this)
            K.IsUsed = true;
        for (int i = 0; i < this.Count; i++)
        {
            A.Arraybr[xcoord[i] * A.Photo.Height + ycoord[i]] = 120;
        }
        f.Refresh();
    }
}

class Polygones : ImgProcess
{
    public List<Polygone> MultiPolygone;
    public int realCount;

    public Polygones()
    {
        MultiPolygone = new List<Polygone>();
    }


    public void Detection(Img A)
    {
        //искать пару точек разного цвета
        int x = 0, y = 0;

        Color P1 = new Color();
        P1 = A.Photo.GetPixel(x, y);
        Color P2 = new Color();
        P2 = A.Photo.GetPixel(x + 1, y);

        while ((x + 1 < A.Photo.Width) && (y + 1 < A.Photo.Height))
        {

            while (P1 == P2)//(A.Arraybr[x*A.Photo.Height+y]==A.Arraybr[(x+1)*A.Photo.Height+y])
            {
                if (y + 1 >= A.Photo.Height)
                {
                    break;
                }
                if ((x + 2) == A.Photo.Width)
                {
                    x = 0;
                    y++;
                }

                x++;

                P1 = P2;
                P2 = A.Photo.GetPixel(x + 1, y);


            }

            Points K = new Points(x, y, A);
            Points K1 = new Points(x+1, y, A);


            if ((K.IsUsed == false) && (K1.IsUsed == false)&&(K.col!=K1.col))
            {
                Polygone Poly = new Polygone(x, y, A);
                Poly.PolyEdge(A);
                MultiPolygone.Add(Poly);

            }

            {
```

```
                if ((x + 2) == A.Photo.Width)
                {
                    x = 0;
                    y++;
                }

                x++;
            }

        }

        for (int r = 0; r<A.Photo.Width; r++)
            for (int l=0; l<A.Photo.Height; l++)
            if (A.Arraybr[r*A.Photo.Height+l] == 120)
                A.Photo.SetPixel(r, l, Color.Aqua);
    }

    public void Counting()
    {
        realCount = 0;
        foreach (Polygone P in MultiPolygone)
        {
            if ((P.IsConvex == true)&&(P.Count>100))
                realCount++;
        }
        MultiPolygone.Clear();
    }

    public override Img Process(Img A)
    {
        A.count = 0;
        //запускать остальные функции
        Detection(A);
        //определять количество многоугольников
        Counting();
        A.count = realCount;
        return base.Process(A);
    }
  }
}
```