Федеральное государственное бюджетное образовательное учреждение

высшего образования

“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Мохаммед Заки Хассан М.Н.

Разработка алгоритмов распознавания рукописных символов на основе топологических свойств графов символов

Диссертация

05.13.17 Теоретические основы информатики

Научный руководитель:

к.ф.-м.н., доцент Н.А. Тюкачев

Воронеж 2018

Оглавление

Введение

Для распознавания объектов по изображению возникает достаточно часто, например, для идентификации личности по отпечаткам пальцев или фотографии, распознавания символов и пр.

Хорошей моделью объекта является описание в виде графа скелетной линии, представляющей срединные оси объекта. Существуют множество методов построения такого графа, и на основе результатов скелетизации объектов можно выделять различные признаки для решения задачи распознавания объектов.

Не существует общего метода для сравнения форм растровых изображений. Почти все меры сравнения строятся на основе информации о объектах, исходя из особенностей конкретной прикладной задачи. Поэтому для каждого приложения определение меры сходства решается заново. Часто за метрику в пространстве графов принимается расстояние Левенштейна (редакционное расстояние) между двумя строками — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Определение сходства двух скелетных объектов. Обработка изображений является актуальным разделом исследований в области теоретической информатики, как в области практических приложений, так и в области фундаментальных наук.

Задачи обработки изображений в области фундаментальной науки появляются в физике элементарных частиц, астрономии, фотографии, радиолокации, медицины, географии, в задачах учёта, обработки и систематизации данных. Таким образом, задача обработки изображений состоит из большого класса подзадач. Задача распознавания и рукописных символов является одной из таких подзадач. В последнее время достигнут большой прогресс в распознавании машинописных символов. Разработаны методы, алгоритмы и на их основе программные комплексы, с высокой степенью точности распознающие машинописные символы. Сложнее с алгоритмами и программными продуктами для распознавания рукописных символов.

Рукописный текст является наиболее традиционным способом хранения и использования информации. С появлением современных средств вычислительной техники, сетей и средств обмена информации роль рукописного текста в коммуникации значительно возрастает. Примерами этому являются школьные задания, почтовые адреса на конвертах, медицинские заключения и справки, заполненные руками анкеты и формы, подписи на банковских чеках и документах, общение письмами. Проблема распознавания рукописного текста имеет большое значение и в архивном деле. Существует огромное количество рукописных архивных документов, требующих перевода в цифровой формат и распознавания. Большая часть документов является письменными источниками.

Поэтому необходимо создание систем распознавания рукописных текстов, не требующих больших человеческих трудозатрат.

Все перечисленные проблемы и новые технологические задачи определяют актуальность разработки методов и алгоритмов обработки рукописных текстов. Это и является предметом исследования предлагаемой работы.

Актуальность исследования

Существует большое количество отсканированных рукописных текстов, которые необходимо распознать для последующего лингвистического анализа и поиска. Но нет универсальных программ, способных это сделать.

Цели и задачи исследования

Целью исследования является разработка и теоретическое обоснование алгоритмов, предназначенных для распознавания рукописных символов и на основе развития методов распознавания текстов и развитие новых алгоритмов решения этой задачи. Для достижения цели в диссертации решаются следующие задачи.

Постановка задачи

1. Разработать модифицированный алгоритм выделения контура методом жука. Разработать методы численной оценки качества выделения границ на изображениях, сравнить их быстродействие и качество.
2. Разработать модифицированный алгоритм скелетизации изображения символов.
3. Разработать модифицированный волновой алгоритм.
4. Разработать методы построения и упрощения графа изображения символов.
5. Разработать структуру, метод построения дерева классификации символов. Разработать метод поиска символов в этом дереве.

Методы исследования

Теоретические исследования базируются на использовании методов дискретной математики, теории множеств, методов аналитической геометрии, теории кривых. Экспериментальные исследования основаны на разработанных программах, написанных на языке C# и на сравнении результатов с данными, имеющимися в литературе.

Научная новизна и значимость работы

Заключается в разработке методов, алгоритмов и создании программного обеспечения для распознавания рукописных текстов.

В работе впервые разработаны:

1. Разработан модифицированный алгоритм выделения скелета методом Андреева-Бобкова, отличающийся тем, что отмечаются пройденные пиксели, например, красным цветом и в стек следующих точек цвета $C_{новый}$ заносятся те соседи, у которых есть хотя бы одна соседняя точка цвета $C_{пред}$. Это позволяет не пропускать все точки ветвления контура и не допускать зацикливания.

2. Разработаны методы численной оценки качества выделения границ на изображениях, отличающиеся от метрики пикового отношения сигнала к шуму (Peak Signal to Noise Ratio, PSNR)

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{\max_f^2}{\mathrm{MSE}}\right)$$

тем, что вместо среднеквадратичной ошибки MSE предлагается использовать относительную ошибку,

$$\mathrm{RE} \equiv (b * P_{co} + \alpha)^{-1} + c * P_{fa} + d * P_{nd}$$

определённую через примитивные метрики, где α – константа, ограничивающая *RE* сверху, *b,c,d* – весовые коэффициенты, являющиеся мерами значимости примитивных метрик. Проведено сравнение быстродействия и качества методов выделения границ.

3. Разработан модифицированный волновой алгоритм для построения дерева скелета символа, отличающийся от алгоритма предложенного Денисовым И. и Кузьминым Е. тем, что волна идет по линии толщиной в 1 пиксель и появление в волне двух пикселей служит сигналом для создания узла. Впервые разработаны методы построения и упрощения графа из этого дерева скелета изображения символов.
4. Предложен набор топологических признаков, отличающийся от набора Афонасенко А.В. и Елизарова А.И., и на основе этого набора разработана структура и метод построения дерева принятия решений для классификации символов. Доказана теорема: Каждый лист дерева принятия решения соответствует одному символу.

Практическая ценность работы

Заключается в возможности применении разработанного программного продукта для распознавания рукописных символов. Разработанные методы и алгоритмы могут быть применены в программах распознавания символов на сканированных изображения и для распознавания архивных документов.

Соответствие диссертации паспорту научной специальности

Область исследования соответствует паспорту специальности 05.13.17 «Теоретические основы информатики»:

5. Разработка и исследование моделей и алгоритмов анализа данных, обнаружения закономерностей в данных и их извлечениях разработка и исследование методов и алгоритмов анализа текста, устной речи и изображений.

7. Разработка методов распознавания образов, фильтрации, распознавания и синтеза изображений, решающих правил. Моделирование формирования эмпирического знания.

Математической моделью является дерево принятия решения, предназначенное для классификации графов скелетов печатных или рукописных символов. Узлами этого дерева являются топологические признаки графов скелетов символов, а листьями - печатные или рукописные символы.

Численные методы использовались для вычисления кривизны ребер графа.

Разработано программное обеспечение, предназначенное для обработки сканированного изображения рукописного текста, построения дерева принятия решения и классификации символов. На это программное обеспечение получен патент [43].

1 Анализ методов сегментации

Одной из основных проблем анализа и обработки изображения является сегментация, то есть разделение изображения на области, у которых, например, примерно одинаковая яркость.

Один из часто используемых и простых способов — это построение сегментации с помощью порогов. Пороги — это признаки, которые позволяют разделить изображение на группы. Пороговое разделение сводится в сопоставлении значения яркости каждого пикселя изображения с заданными значениями порогов.

В самом простом случае – выделение областей, яркость которых выше или ниже некоторого порога.

Рисунок 1.1 - Преобразование изображения к черно-белому

1.1 Определение порога для симметричного пика гистограммы

Этот метод применяется в том случае, когда фон изображения дает отчетливый пик гистограммы, симметричный относительно свой середины.

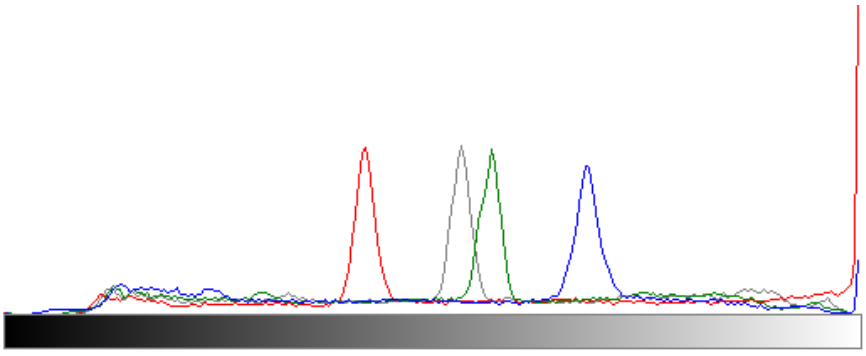


Рисунок 1.2 – Гистограмма изображения

В этом случае необходимо:

1. одним из методов сгладить гистограмму;
2. найти точку гистограммы h_{max} с максимальным значением;
3. на левой или правой от пика стороне гистограммы не относящейся к объекту найти яркость h_p, количество пикселей с яркостью >= h_p равняется p%, например 5%, от пикселей яркости которых >= h_{max};
4. пересчитать порог $T = h_{max} - (h_p - h_{max})$;

1.2 Адаптивная бинаризация

Адаптивная бинаризация используется, если есть неравномерная яркость фона или объекта (рис. 1.3).

Для каждого пикселя изображения I(x, y) выполняются следующие действия:

1. Вычитывается индивидуальный для данного пикселя порог T в окрестности пикселя радиуса r;

2. Если значение I(x, y) > T + C , то результат 1, иначе 0;

Существуют следующие варианты выбора для T:

- T = mean
- T = median
- T = (min + max) / 2

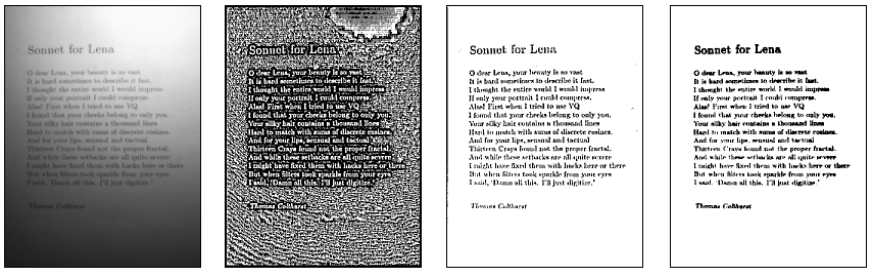


Рисунок 1.3 - Неравномерная яркость фона

1.3 Бинаризация с нижним порогом

Наиболее простой операцией является бинаризация с нижним порогом. В этом случае используется только одно значение порога:

$$f'(m,n) = \begin{cases} 1, f(m,n) \geq t \\ 0, f(m,n) < t \end{cases}$$

Все значения цвета пикселя меньше порога устанавливаются в 255 (белый), а все цвета пикселей, большие порога — в 0 (черный).

1.4 Бинаризации с верхним порогом

Иногда используется вариант бинаризации с нижним порогом, который дает негативное изображение. Операция бинаризации с верхним порогом подчиняется следующим условиям:

$$f'(m,n) = \begin{cases} 0, f(m,n) \leq t \\ 1, f(m,n) > t \end{cases}$$

1.5 Бинаризация с двойным ограничением

Если значения яркости пикселей могут меняться в некотором диапазоне, то для выделения областей вводится бинаризация с двойным ограничением ($t_1<t_2$):

$$f'(m,n) = \begin{cases} 0, f(m,n) \geq t_1 \\ 1, f(m,n) \leq t_2 \\ 0, f(m,n) > t_2 \end{cases}$$

1.6 Неполная пороговая обработка

Это преобразование дает изображение, лишенное фона со всеми деталями, присутствующими на изображении.

$$f'(m,n) = \begin{cases} f(m,n), f(m,n) \geq t \\ 0, f(m,n) < t \end{cases}$$

1.7 Многоуровневое пороговое преобразование

Это преобразование формирует изображение состоящее из сегментов с различной яркостью.

$$f'(m,n) = \begin{cases} 1, f(m,n) \in D_{1;} \\ 2, f(m,n) \in D_{2;} \\ \dots. \\ n, f(m,n) \in D_n \\ 0, \beta\_\text{стальных}\_\text{случаях} \end{cases}$$

1.8 Метод Отсу

Этот метод использует гистограмму, построенную по значениям $p_i=n_i/N$, где N – общее количество пикселей, n_i – количество пикселей с уровнем яркости i. Диапазон яркостей делится на две части пороговым значением уровня яркости k. Каждому поддиапазону соответствуют относительные частоты $\omega_0\omega_1$:

$$\omega_0(k) = \sum_{i=1}^{k} p_i$$

$$\omega_1(k) = \sum_{i=k+1}^{L} p_i = 1 - \omega_0(k)$$

$$\mu_0(k) = \sum_{i=1}^{k} \frac{ip_i}{\omega_0}$$

$$\mu_1(k) = \sum_{i=k+1}^{L} \frac{ip_i}{\omega_1}$$

Затем вычисляется максимальное значение оценки качества:

$$\vartheta(k) = \max_{k=1..L-1}\left(\frac{\sigma^2_{\text{кл}}}{\sigma^2_{\text{общ}}}\right)$$

где $(\sigma_{кл})^2=\omega_0\omega_1(\mu_1-\mu_0)^2$, – дисперсия, а $(\sigma_{общ})^2$ – дисперсия для изображения.

1.9 Определение порога на основе градиента яркости изображения

Применяется, если изображение можно разделить на два части – объекты и фон. Алгоритм вычисления порога состоит из двух шагов:

На первом шаге определим модуль градиента яркости для каждого пикселя изображения:

$G(m,n) = max\ \{|G_m(m,n)|, |G_m(m,n)|\},$

где

$G_m(m,n) = f(m+1,n)-f(m-1,n),$

$G_n(m,n) = f(m,n+1)-f(m,n-1).$

По формуле вычислим значение порога:

$$t = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)G(m,n)}{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} G(m,n)}$$

1.10 Метод Бернсена

1. Квадратным шаблоном с нечетным числом пикселей сканирует все пиксели исходного изображения. Для каждого пикселя определяется минимальное (Min) и максимальное (Max) значение.

2. Определяется среднее значение Avg= (Min + Max) /2.

3. Если цвет пикселя больше Avg<E — он становится белым, иначе — чёрным. E — константа заданная пользователем.

4. Если среднее значение меньше порога — то для пикселя устанавливается цвет «сомнительного пикселя».

Недостатки: после обработки монотонных областей яркости появляются сильные паразитные шумы, иногда приводящих к появлению ложных черных пятен

1.11 Метод Эйквила

Это один из самых эффективных. Часто используется для обработки четких и контрастных изображений. Изображение сканируется с помощью двух квадратных окон: маленького – S, и большого L. Оба окна сканируют изображение с шагом равным S. Для большого окна L вычисляется порог B так, чтобы поделить пиксели на два группы. Если матожидания уровня яркости в двух группах имеют разницу, превышающую некоторый уровень $/\mu_1-\mu_2/\geq l$, то все пиксели внутри окна S бинаризуются по порогу T. Иначе, яркость пикселей окна S заменяется некоторым близким значением.

1.12 Метод Ниблэка

Данный метод состоит в варьировании порога B(x, y) в каждой точке на основании значения стандартного отклонения:

$$B(x,y) = \mu(x,y) + k \cdot s(x,y)$$

где μ(x, y) – среднее и s(x, y) — среднее квадратичное отклонение выборки для окрестности точки.

Размер окрестности должен быть достаточно большим для того, чтобы снизить влияние шума. С другой стороны размер окрестности должен быть минимальным, но таким, чтобы сохранить локальные детали изображения.

Значение параметра k определяет, какая часть границы объекта должна быть взята в качестве самого объекта. При значении k = -0.2 получается достаточно хорошее разделение черных объектов, а при значение k = +0.2, – для объектов белого цвета.

1.13 Метод Яновица и Брукштейна

На основе локальной максимизации градиента яркости строится поверхность потенциалов, которая используется в качестве пороговой поверхности бинаризации.

С помощью контурного оператора Собеля или Кэнни вычисляется значение градиента яркости. Затем изображение фильтруется с целью получения контурных линий толщины в один пиксель усредняющим фильтром 3×3. По итерационной интерполирующей схеме строится потенциальная поверхность. Расчет поверхности начинается с контурных пикселей. Рассчитывается интерполяционный остаток R(x, y) для каждого не контурного пикселя. Новое значение пикселя P(x, y) на n+1-ом шаге вычисляется по формулам:

$$P_n + 1(x, y) = P_n(x, y) + \beta . R_n(x, y)/4 ,$$

$$R_n(x, y) = P_n(x - I, y) + P_n(x + I, y) + P_n(x, y - I) + P_n(x, y + 1) - 4 . P_n(x, y)$$

Значение параметра β для быстрой сходимости принимается в пределах 1≤β≤2.

1.14 Метод Янни

Метод Янни заключается в определении максимального значения амплитуды серого цвета g_{max} и минимального значения амплитуды серого цвета g_{min}. Затем

вычисляется среднее количество пикселей, попавших в диапазон от минимального значения до середины. Это позволяет вычислять оптимальный порог Янни:

$$T_{opt} = (g_{max} - g_{min}) \sum_{g=g_{min}}^{g^*_{mid}} p(g)$$

2 Выделение контура изображения

Контур это протяженный разрыв, скачкообразное изменение или перепад значений яркости. Возникает несколько проблем при выделении контуров:

- если яркость меняется не слишком быстро, то в этих местах появляются разрывы контура;
- вследствие шума на изображении появляются ложные контуры;
- появляются широкие контурные линии из-за размытости или шума.

Можно дать несколько определений контура области:

1. Пиксель принадлежит контуру, если он принадлежит области и существует сосед, не принадлежащий области (внутренняя граница).

2. Пиксель принадлежит контуру, если он не принадлежит области и существует сосед, принадлежащий области (внешняя граница).

Контур зависит от того 4-х или 8-ми связность используется для определения соседей.

Можно получать контуры, также, с помощью операций математической морфологии:

- внутреннее оконтуривание

$$CI = A – (A (-) B)$$

- внешнее оконтуривание

$$CO = (A (+) B) – A$$

Для построения дифференциальных масок необходимого размера используется подход, называемый ЛОГ-фильтрами. JJ его основе лежит функция $f^{(2)}(x)$ являющаяся второй производной функции Гаусса *f(x)* и имеющая вид:

$$f(x) - ce^{-\frac{x^2}{2\sigma^2}}$$

$$f^{(1)}(x) = c_1 x e^{-\frac{x^2}{2\sigma^2}}$$

$$f^{(2)}(x) = c_2 x^2 e^{-\frac{x^2}{2\sigma^2}} - c_3 e^{-\frac{x^2}{2\sigma^2}} \qquad (2.1)$$

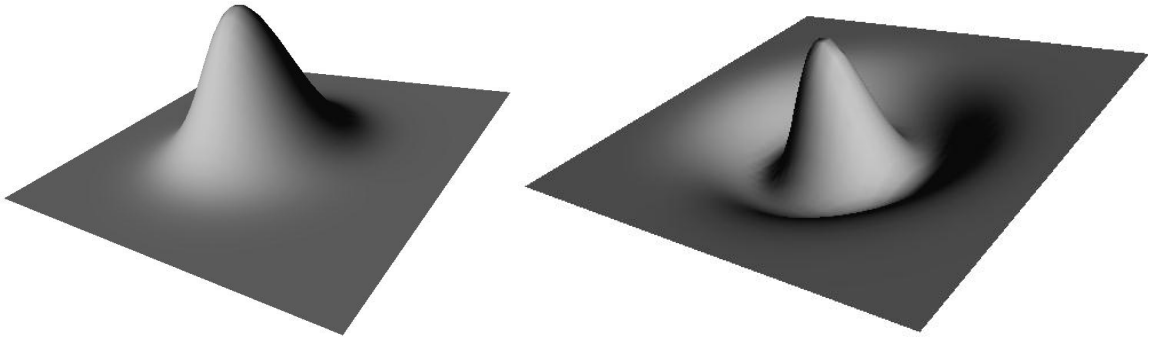


Рисунок 2.1 - Поверхности Гаусса

2.1 Обзор методов выделения контура

Выделению границ посвящено большое количество работ [13,16,20,37,39-41,44,45].

2.1.1 Метод Робертса

Оператор выделения границ Робертса введен Лоуренсом Робертсом в 1964 году. Он выполняет простые и быстрые вычисления двумерного пространственного измерения на изображении. Этот метод подчеркивает области высокой пространственной частоты, которые зачастую соответствуют краям. На вход оператора подается полутоновое изображение. Значение пикселей выходного изображения в каждой точке предполагает некую величину пространственного градиента входного изображения в этой же точке.

Оператор Робертса использует четыре значения яркости на изображении [37] и имеет следующий вид

$$G_{i,j} = \sqrt{(E_{(i+1).(j+1)} - E_{i,j})^2 + (E_{(i+1).j} - E_{i.(j+1)})^2}$$

или

$$G_{i,j} = \left|E_{(i+1),(j+1)} - E_{i,j}\right| + \left|E_{(i+1),j} - E_{i,(j+1)}\right| \qquad (2.2)$$

где $[E_{i,j}]$ - элемент матрицы исходного изображения.

Листинг 2.1. Метод Робертса

```
public static void МетодРобертса()
{
    C.threeImage = new Bitmap(C.twoImage.Width,
```

```
        C.twoImage.Height, PixelFormat.Format16bppArgb1555);
    for (int i = 0; i < C.twoImage.Width - 1; i++)
        for (int j = 0; j < C.twoImage.Height - 1; j++)
        {
            Color a = C.twoImage.GetPixel(i, j);
            Color b = C.twoImage.GetPixel(i, j + 1);
            Color c = C.twoImage.GetPixel(i + 1, j);
            Color d = C.twoImage.GetPixel(i + 1, j + 1);
            double x = Math.Abs(a.B - d.B);
            double y = Math.Abs(b.B - c.B);
            double w = Math.Sqrt(x * x + y * y);
            if (w > 255)
                w = 255;
            byte p = Convert.ToByte(w);
            c = Color.FromArgb(p, p, p);
            C.threeImage.SetPixel(i, j, c);
         }
}
```

Робертс для выделения перепадов и контрастирования предложил следующую нелинейную операцию двумерного дискретного дифференцирования:

$$G_R(j, k)=([F(j,k)-F(j+1, k+1)]^2 + [F(j,k+l)-F(j+1, k]^2)^{1/2} \quad (2.3)$$

Меньше вычислений требует другая операция пространственного дифференцирования:

$$G_A(j,k) = |F(j, k)-F(j+1, k+l)| + |F(j,k+1)-F(j+1,k)| \quad (2.4)$$

Можно легко показать, что

$G_R(j, k) \le G_A(j, k) \le \sqrt{2} G_R(j, к)$.

Информацию о приблизительной ориентации перепада дает тот из четырех элементов изображения, расположенных около обнаруженной точки, который имеет наибольшее значение яркости.

2.1.2 Метод Лапласа (LoG)

Используя свертку пикселей изображения с оператором Лапласа, представленным в виде шаблона, можно получить повышение контраста перепадов без учета их ориентации. Ниже перечислено несколько видов масок оператора Лапласа:

Первая маска

$$H = \begin{matrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{matrix}$$

Вторая маска

$$H = \begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$

Третья маска

$$H = \begin{matrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{matrix}$$

В методе Лапласа осуществляется умножение каждого элемента двумерной апертуры 3 • 3 на соответствующий элемент матрицы Лапласа (рис. 2.2):

1	1	1		0	-1	0		-1	1	1		1	1	1
1	-2	1		-1	4	-1		-1	-2	1		-1	-2	1
-1	-1	-1		0	-1	0		-1	1	1		-1	-1	1

Рисунок 2.2 - Матрицы Лапласа

Листинг 2.2. Метод Лапласа

```
public static void МетодЛапласа()
{
     int[,] a = {{-1,-2,-1}, { -2, 12, -2 }, { -1, -2, -1 } };
     C.threeImage = new Bitmap(C.twoImage.Width,
```

```
            C.twoImage.Height, PixelFormat.Format16bppArgb1555);
        for (int i = 1; i < C.twoImage.Width - 1; i++)
            for (int j = 1; j < C.twoImage.Height - 1; j++)
            {
                int sum = 0;
                Color c;
                for (int x = -1; x <= 1; x++)
                    for (int y = -1; y <= 1; y++)
                    {
                        c = C.twoImage.GetPixel(i + x, j + y);
                        sum += c.B * a[x + 1, y + 1];
                    }
                byte p = (byte)sum;
                c = Color.FromArgb(p, p, p);
                C.threeImage.SetPixel(i, j, c);
            }
    }
```

2.1.3 Метод Уоллеса

Уоллис в работе [37] для обнаружения перепадов предложид нелинейный метод, опирающийся на гомоморфную обработку изображения. Идея этого метода состоит в следующем: точка принадлежит перепаду, если значение логарифма яркости в этой точке больше среднего значения логарифмов яркостей четырех ближайших соседних пикселей на некоторое значение. Контрастированность изображения определяется по формуле:

$$G(j,k) = \log|F(j,k) - \frac{1}{4}\log(A_1) - \frac{1}{4}\log(A_3) - \frac{1}{4}\log(A_5) - \frac{1}{4}\log(A_7)$$

Или , что эквивалентно

$$G(j,k) = \frac{1}{4} log\left\{\frac{[F(j,k)]^4}{A_1 A_3 A_5 A_7}\right\} \quad (2.5)$$

Проверка принадлежности G (j, k) диапазану между верхним и нижним порогами эквивалентна сравнению дроби в скобках выражения (2.5) с видоизмененным порогом. Поэтому не нужно точно вычислять значения логарифмов. Основное преимущество логарифмического детектора перепадов кроме простоты вычислений состоит в том, что он не чувствителен к мультипликативным изменениям уровня яркости.

2.1.4 Метод Собела

Оператор Собеля введен Собелем в 1970 году. Данный метод обнаружения границ использует приближение к производной. Это позволяет обнаруживать край в тех местах, где градиент самый высокий. Данный способ обнаруживает количество градиентов на изображении, тем самым выделяя области с высокой пространственной частотой, которые соответствуют границам. В целом это привело к нахождению предполагаемой абсолютной величине градиента в каждой точке входного изображения. Данный оператор состоит из двух матриц, размером 3×3. Вторая матрица отличается от первой только тем, что повернута на 90 градусов. Это очень похоже на оператор Робертса.

Идея этого метода основана в наложении на каждую точку изображения двух масок вращения. Эти маски представляют собой две ортогональные матрицы размерностью 3×3 (рис. 2.3).

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

Рисунок 2.3 - Маски Собеля

Эти маски выявляют границы, расположенные вертикально и горизонтально на изображении. При раздельном наложении этих масок на изображение можно получить оценку градиента по каждому из направлений G_x, G_v. Конечное значение градиента определяется по формуле

$$G = \sqrt{G_x^2 + G_y^2}$$
$$G(j,k) = \sqrt{X^2 + Y^2} \ , \qquad (2.7)$$

где

$$X = (A_2 + A_3 + A_4) - (A_0 + 2A_7 + A_6)$$
$$Y = (A_0 + 2A_1 + A_2) - (A_6 + 2A_5 + A_4) \qquad (2.8)$$

Листинг 2.3. Метод Собела

```
public static void МетодСобела()
{
    C.threeImage = new Bitmap(C.twoImage.Width,
        C.twoImage.Height, PixelFormat.Format16bppArgb1555);
    for (int i = 1; i < C.twoImage.Width - 1; i++)
        for (int j = 1; j < C.twoImage.Height - 1; j++)
        {
            byte a1 = C.twoImage.GetPixel(i - 1, j - 1).B;
            byte a2 = C.twoImage.GetPixel(i, j - 1).B;
            byte a3 = C.twoImage.GetPixel(i + 1, j - 1).B;
            byte a4 = C.twoImage.GetPixel(i + 1, j).B;
            byte a5 = C.twoImage.GetPixel(i + 1, j + 1).B;
            byte a6 = C.twoImage.GetPixel(i, j + 1).B;
            byte a7 = C.twoImage.GetPixel(i - 1, j + 1).B;
            byte a8 = C.twoImage.GetPixel(i - 1, j).B;
            int x = (a3 + 2 * a4 + a5) - (a1 + 2 * a8 + a7);
            int y = (a1 + 2 * a2 + a3) - (a7 + 2 * a6 + a5);
            double w = Math.Sqrt(x * x + y * y);
            if (w > 255)
                 w = 255;
            byte p = (byte)w;
            Color c = Color.FromArgb(p, p, p);
```

```
            C.threeImage.SetPixel(i, j, c);
        }
 }
```

2.1.5 Оператор Превитта

Обнаружение границ данным оператором предложено Превиттом в 1970 году. Правильным направлением в данном алгоритме была оценка величины и ориентация границы. Даже при том, что выделение границ является весьма трудоемкой задачей, такой подход дает весьма неплохие результаты. Данный алгоритм базируется на использовании масок размером 3 на 3, которые учитывают 8 возможных направлений, но прямые направления дают наилучшие результаты. Все маски свертки рассчитаны.

2.1.6 Метод Кирша

В 1971 году Кирш для обнаружения границ предложил алгоритм, основанной на использовании одной маски, которую поворачивают по восьми главным направлениям соседей.

Значение границы определяется как максимальное значение, найденное с помощью этой маски. Отметим, что последние четыре маски являются зеркальным отражением первых четырех относительно центральной оси матрицы.

Для контрастирования перепадов Киршем [37] предложен еще один нелинейный оператор с окном из 3X3 элементов:

A_0	A_1	A_2
A_7	$F(j,k)$	A_3
A_0	A_5	A_4

Рисунок 2.4 - Обозначение элементов операторов

$$G(j,k) = \max\left\{1, \max_{i=0\div7}[|5S_i - 3T_i|]\right\}, \quad (2.9)$$

где

$$S_i = A_i + A_{i+1} + A_{i+2}$$

$$T_i = A_{i+3} + A_{i+4} + A_{i+5} + A_{i+6} + A_{i+7}$$

Оператор Кирша (2.9) дает максимальное значение градиента в точке изображения без учета значения *F* (j, *k).*

2.1.7 Оператор Робинсона

Метод Робинсона, введенный в 1977, подобен методу Кирша, но является более простым в реализации в силу использования коэффициентов 0, 1 и 2. Маски данного оператора симметричны относительно центральной оси, заполненной нулями. Достаточно получить результат от обработки первых четырех масок, остальные же можно получить, инвертируя первые.

Максимальное значение, полученное после применения всех четырех масок к пикселю и его окружению считается величиной градиента, а угол градиента можно аппроксимировать как угол линий нулей в маске, которые дают максимальный отклик.

2.1.8 Детектор краев Кенни

Дж. Кэнни предположил, что детектор должен реагировать на границы, игнорируя ложные границы, точно определять линию границы и реагировать на каждую границу только один раз.

Алгоритм состоит из следующих шагов:

1. размытие изображения для удаления шума (сглаживание);
2. границы находятся там, где градиент яркости изображения имеет максимальное значение;
3. удаление немаксимумов — только локальные максимумы фиксируются как границы;
4. применение двойной пороговой фильтрации;
5. окончательные границы устанавливаются путем подавления всех краев, не связанных с сильными границами.

Применяется первая производная от гауссианы для уменьшения чувствительности алгоритма к шуму (рис. 2.5).

Рисунок 2.5 - Иллюстрация работы контурного детектора Кэнни

Каждый из рассмотренных выше методов обнаружения границ обрабатывает пикселы независимо друг от друга. Полученные границы представляют собой по линии без разрывов, а просто области изображения, которые могут иметь произвольную толщину и форму. Детектор краёв Кении является более развитым методом обнаружения границ гарантирует, что все найденные границы являются неразрывными линиями толщиной в один пиксел.

На вход метода поступает черно-белое изображение, размах сглаживания, нижний и верхний пороги. Назначение каждого параметра будет указано далее. Алгоритм состоит из нескольких шагов: сглаживание по Гауссу, вычисление градиентов, немаксималыюе подавление и прослеживание контуров.

На первом шаге происходит усреднение шумов исходного изображения с помощью фильтра Гаусса с указанным размахом сглаживания. Это приводит к тому, что обнаруживаемые границы становятся более монотонны и имеют примерно одинаковый локальный контраст. Если величина размаха сглаживания слишком мала, то границы остаются немонотонными и в результате работы одна отчётливо заметная тонкая линия может быть обнаружена как набор разрозненных сегментов. Если же величина размаха сглаживания слишком велика, то локальный контраст может оказаться потерян и заметные границы не будут обнаружены вовсе.

После выполнения сглаживания происходит вычисление градиента в каждом пикселе изображения, что дает возможность для определения границ и их направления.

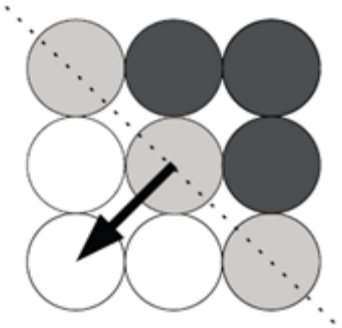


Рисунок 2.6 -. Немаксимальное подавление

Далее выполняется операция, которая называется немаксимальным подавлением. Для каждого пиксела рассматривается два его соседа, расположенных вдоль направления градиента. Такими пикселами могут быть верхний / нижний, левый / правый, левый верхний / правый нижний, или левый нижний / правый верхний. Сравниваются величины градиентов текущего пиксела и его соседей. Если величина градиента текущего пиксела оказывается не максиммалыюй из трчх, то она обнуляется сразу же после окончания работы данного шага алгоритма. Это приводит к тому, что вместо толстых областей, в которых пикселы имеют большое значение градиента, остачтся тонкая линия. В иеч попадют только те пикселы, которые имеют максимальное значение градиента среди своих соседей.

На заключительном шаге алгоритма происходит прослеживание контуров п отсеивание пикселов, которые относятся к второстепенным границам. Сначала происходит определение пикселов, где величина градиента больше указанного на входе верхнего порога. Начиная от каждого такого пиксела происходит поиск связного фрагмента изображении. Сначала этот фрагмент состоит из одного начального пиксела, но потом туда добавляются все соседние пиксели, у которых величина градиента больше указанного нижнего порога. Затем аналогичным

образом рассматриваются соседи соседей и так далее. Каждый обнаруженный таким образом фрагмент является набором тонких, связанных между собой линий. Они состоят из пикселей, которые либо вошли 15 один из фрагментов, либо имеют величину градиента, большую верхнего порога. Если пиксель не вошел ни в один из фрагментов и величина его градиента меньше верхнего порога, то он не считается граничным.

Изображение, переведенное в оттенки серого, сглаживается за счет использования специального фильтра. Этот фильтр обнаруживает и устраняет найденные разрывы, применяя перемещаемую по изображению маску. Обычно ее называют ядром, которое представляет собой квадратную матрицу. Элементами этой матрицы являются цвета пикселей изображения, над которыми в данный момент находится шаблон. Согласно значениям яркости этих пикселей изменяется яр- кость пикселя под центром маски. Схема пространственной фильтрации изображена на рис. 2, где *f*(*x*,*y*) – двумерный массив пикселей изображения.

Оператор Канни [1,4] для сглаживания использует фильтр Гаусса. Уравнение распределения Гаусса в *N* измерениях имеет вид:

$$\text{Gauss(x,y,}\sigma) = \frac{e^{\frac{-(x^2+y^2)}{2\sigma^2}}}{2\pi.\sigma^2}$$

где $(x^2 + y^2)$ – радиус размытия; σ – отклонение распределения Гаусса.

Пиксели, в которых распределение отлично от нуля, используются для построения матрицы свертки, которая применяется к изображению. Значение яркости каждого пикселя вычисляется как средневзвешенное для окрестности. Значение центрального пикселя принимает наибольший вес, а соседние пиксели принимают меньшие веса.

В применяемом методе используются специализированные ядра фильтра, являющиеся разновидностью оператора Собеля.

Фильтры Собеля:

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

Именно такой оператор использует маски, с помощью которых сворачивает исходное изображение и вычисляет приближенное значение производной по вертикали, горизонтали или диагоналям [4].

Градиент определяет, как быстро изменяется яркость изображения в каждой точке, что дает возможность определить границу и ее ориентацию. При применении оператора Собеля в области постоянной яркости, т. е. области, где нет резких перепадов или границ, будет получен вектор очень маленький или близкий к нулю. В точке, относящейся к границе областей, имеющих различную яркость, величина вектора будет значительно больше, с направлением в сторону увеличения яркости.

К пикселям границ относят пиксели, которые дают локальный максимум градиента в направлении его вектора. Угол направления должен быть кратен 45°.

Большая часть пикселей в примере имеют ориентацию по направлению вверх. Это говорит о том, что значение градиентов в этих точках будет сопоставлено с пикселями, где ориентация иная. Пиксели, помеченные белым, будут рассматриваться как потенциальная граница, все прочие будут исключены.

После вычисления оператора Собеля определяется угол направления вектора границы. Направления вектора округляется до ближайшего угла, кратного 45° (0, 45, 90 и 135°). Затем проверяется достижение локального максимума величиной градиента в найденном направлении вектора [2].

В результате подавления локальных неопределенностей толщина линии границы становится равномерной и тонкой, что увеличивает точность ее расположения.

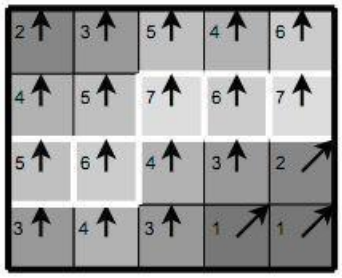


Рисунок 2.7 - Принцип подавления «ложных» максимумов

Метод порогов заключается в определении областей на изображении, где могут располагаться границы, за счет значений порогов. Чем выше пороги, тем меньше границ будет найдено, но и шум на изображении меньше. Поэтому высокий порог может ослабить обнаружение границ вплоть до ее фрагментации, а слишком низкий пропустит много лишней информации и шума.

2.1.9 Методы жука

Необходимо отметить одно важное достоинство методов жука: в результате получается упорядоченная последовательность точек контура, что значительно упрощает следующую задачу – векторизацию контура.

Метод жука (Прэтт)

Метод заключается в последовательном вычерчивании границы между объектом и фоном. Прослеживающая точка в виде «жука» ползает по изображению до тех пор, пока не доходит до темной области (объект). Тогда «жук» поворачивается налево и движется по кривой, пока не достигнет границ объекта, после этого поворачивается направо и повторяет процесс, пока не достигнет окрестности начальной точки (рис. 2.8) [37].

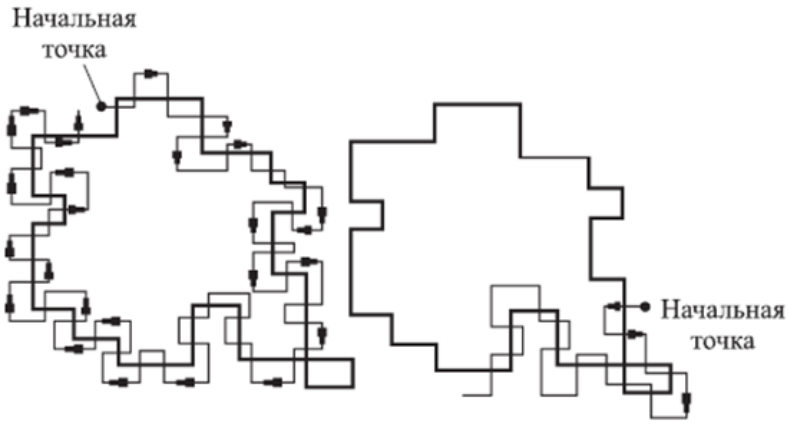


Рисунок 2.8 - Метод прослеживания контуров

Метод напоминает движение жука, обходящего препятствие. В случае черно-белого изображения, жук начинает сканирование на белом фоне. После пересечения черного элемента, он поворачивается налево и переходит к следующему элементу. Если элемент черный, то жук поворачивает налево, если же элемент - белым, то жук поворачивает направо. Процедура повторяется до тех пор, пока жук не вернется в исходную точку [37]. Координаты точек перехода с черного пикселя на белый или с белого пикселя на черный определяют местоположение границы.

Метод жука (Андреев - Бобков)

У метода жука Прэтта есть недостатки:

1) жук может не заметить ветвления контура;

2) жук может не заметить областей белых пикселей внутри объекта.

3) метод не работает в случае, если контур заканчивается на границе изображения;

4) в контур попадают не только черные, но и белые пиксели;

5) многие пиксели попадают в контур дважды.

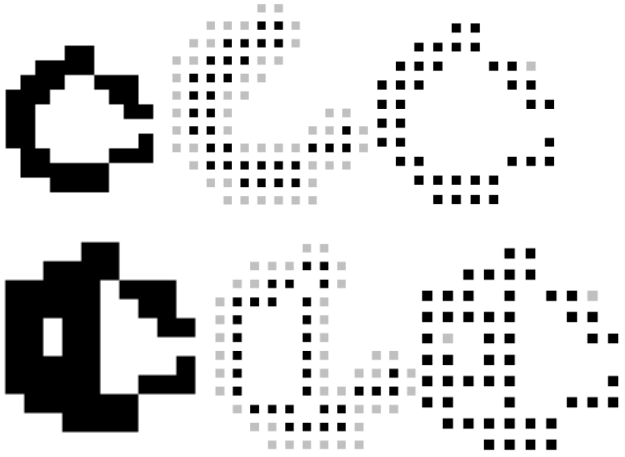
Для решения этих проблем для задачи сегментации изображения Андреев А.Ю., Бобков С.П. [14] предложили модифицированный метод жука. Основная идея их модификации заключается в следующем: при переходе с белого пикселя

через угловой черный по диагонали остается непроверенный пиксель, который может быть черным. В этом случае предлагается занести его в стек и после завершения обхода контура начать обход с этого пикселя. Необходимо отметить, что отслеживание только угловых пикселей не решает проблемы ветвления граничных линий.

2.2 Модифицированный метод жука

Тестирование этого алгоритма Андреева-Бобкова показало, что в стек попадают не только непроверенные пиксели, но и те пиксели, которые жук уже прошел как граничные. Так для изображения, представленного на рис. 2.9, вместо одного пикселя в стек попадает 9. А это приводит к зацикливанию алгоритма.

На рис. 2.9 представлены результаты работы методов жука Прэтта (б) и модифицированного метода (в). Первое изображение содержит два узловых соединения, второе – контур внутри контура.



а) б) в)

Рисунок 2.9 - Сравнение работы методов жука Прэтта (б) и модифицированного метода (в)

Из рис. 2.9 видно, что метод жука Прэтта потерял часть границы и не нашел внутренний контур.

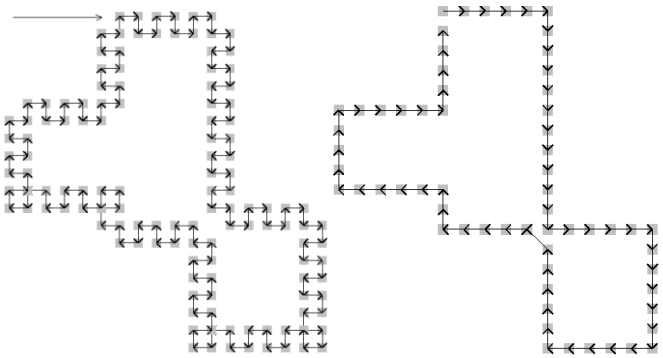


Рис. 2.10. Траектория движения жука по методу Прэтта и по модифицированному методу

Все эти проблемы алгоритма порождаются тем, что точки контура не отмечаются как пройденные. Модифицированный алгоритм [28] предназначен для обработки изображения нескольких оттенков серого и использует три цвета: $C_{пред}$ – цвет пикселей за жуком; $C_{новый}$ – цвет пикселей перед жуком; $C_{контур}$ – красный цвет пикселей контуров, пройденных жуком. Если, например, $C_{пред}$ – белый, то $C_{новый}$ – черный, если $C_{пред}$ – черный, то $C_{новый}$ – белый. Упорядоченные точки контуров накапливаются в двумерном списке Контуры[,]. Конвертация цветов $C_{пред}$ и $C_{новый}$ происходит после завершения построения очередного контура.

Основные идеи алгоритма заключаются в следующем:

1) отмечать пройденные пиксели, например, красным цветом;

2) в стек следующих точек цвета $C_{новый}$ заносить тех соседей, у которых есть хотя бы одна соседняя точка цвета $C_{пред}$.

Алгоритм построения контуров состоит из следующих шагов:

1. Назначить цвета $C_{пред}$ и $C_{новый}$ по цвету пикселя (0, 0).
2. Сканировать изображение, пропуская пиксели $C_{пред}$ и $C_{контур}$ до тех пор, пока не встретится пиксель цвета $C_{новый}$.
3. Построить очередной контур, перекрашивая пройденные точки.
4. Если сканирование не завершено, то конвертировать цвета $C_{пред}$, $C_{новый}$ и продолжить сканирование.

Сложность построения одного контура заключается: во-первых, в том, контур может иметь точки ветвления; во-вторых, начинаться и заканчиваться на границе изображения. Алгоритм построения одного контура заключается в следующем:

1. Положить начальную точку в стек и добавить в массив контура.
2. Найти следующую точку контура.
    1. Если точка найдена, то перекрасить в красный цвет.
        1. Если стек не пуст, то взять точку из стека и добавить в массив контура.
        2. Иначе вставить точку в начало массива контура.
    2. Точка не найдена. Если стек не пуст, то взять точку из стека и организовать новый контур. Цикл завершить.
3. Если найденная точка не совпадает с начальной, то перейти на пункт

Алгоритм определения следующей точки контура получает предыдущую точку, два цвета $C_{пред}$, $C_{новый}$ и заключается в следующем:

1. Опрашиваются соседние 8 точек.
2. Если цвет соседней точки $C_{новый}$ и среди ее 8 соседей есть соседние точки цвета $C_{пред}$, то она попадает в стек соседних точек кандидатов на следующую точку.

Необходимо отметить так же, что алгоритм жука не уступает по скорости самому простому и некачественному алгоритму Робертса и вдвое быстрее популярного алгоритма Собела [37].

На рис. 2.11. Приведен пример выделения контуров на изображении с 5 оттенками серого цвета.

Рисунок 2.11 - Выделение контуров на изображении с 5 оттенками серого цвета

2.3. Алгоритмы утоньшения линий

Основным предназначением любого алгоритма скелетизации является максимальное утоньшение всех линий на бинарном изображении, что приводит к получению скелета, в котором все линии имеют толщину в один пиксель.

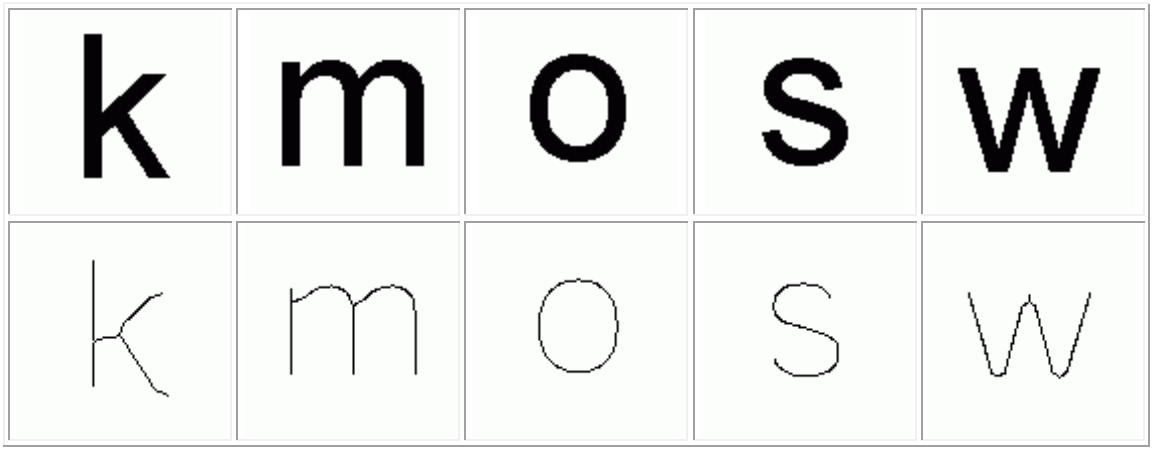


Рисунок 2.12 Результат скелетизации изображений

Такой растровый скелет является весьма удобным для векторизации.

Скелетизация изображения, как правило, реализуется за несколько проходов. За один проход снимаются все граничные пиксели. Данный проход повторяется пока на изображении не останутся "не удаляемые" пиксели.

Рисунок 2.13 - Этапы скелетизации

В этом пункте рассмотрены следующие алгоритмы утоньшения линий:

- шаблонная скелетизация;
- волновой метод;
- алгоритм Зонга-Суня;
- алгоритм.

2.3.1 Шаблонная скелетизация линий

Шаблонный метод использует два набора шаблонов. Каждый шаблон соответствуют матрице 3*3, в которой центральный элемент является текущим пикселем при сканировании изображения.

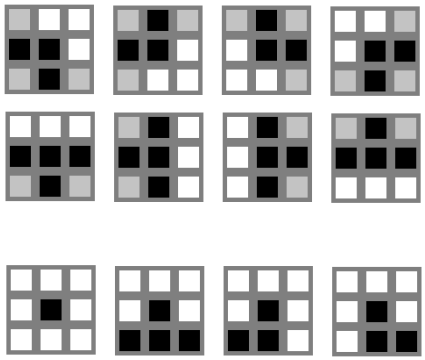


Рисунок 2.14 - Наборы шаблонов. Серым цветом обозначены пиксели, цвет которых не имеет значения

В основном, используются первые первых шаблонов. Нижние четыре предназначены для устранения «шума». Эти четыре так же могут быть повернуты на 90, 180 и 270 градусов, и используются при повторном сканировании изображения.

Если во время сканирования изображения алгоритм натыкается на шаблон, то текущий пиксель окрашивается в белый цвет. Сканирование продолжается, пока продолжаются перекрашивания.

Текст методов шаблонной скелетизации приведен в Приложении 1.

2.3.2 Алгоритм скелетизации Зонга-Суня (Zhang-Suen)

Алгоритм использует следующую матрицу:

P9	P2	P3
P8	P1	P4
P7	P6	P5

Рисунок 2.15 - Матрица алгоритма Зонга-Суня

Матрица попиксельно накладывается на изображение таким образом, чтобы P1 поочередно совмещался со всеми пикселями изображения (там где это возможно). Белый пиксель на изображении имеет значение 0, а черный 1.

Каждый проход по изображению состоит из двух итераций.

Итерация 1: Пиксель P1 можно удалить, если выполняются следующие условия:

1) 2 <= P2+P3+...+P8+P9 <=6

2) S(P1) = 1

3) P2*P4*P6 = 0

4) P4*P6*P8 = 0

где S(P1) - количество найденных последовательностей 01 в последовательности P2, P3, P4, P5, P6, P7, P8, P9, P2. То есть для удаления пикселя, вокруг него должен существовать только один переход от нуля к единице.

В данном случае удаляются все пиксели на юго-восточной границе и северо-западные угловые пиксели.

Итерация 2: Чтобы удалить пиксели на северо-западной границе и юго-восточные угловые пиксели, необходимо выполнить еще одну выше итерацию заменив пункты 3 и 4 на следующие:

3) P2*P4*P8 = 0

4) P2*P6*P8 = 0

Если после полного прохода по изображению были удалены какие-либо пиксели, то необходимо повторить проход. Такое повторение должно быть до тех пор, пока во время прохода не будет удален ни один пиксель.

Скелетизация Зонга-Суня относиться к классу параллельных алгоритмов, поскольку для корректной работы алгоритма, обработанные данные (информация об удаленных пикселях) должны заноситься в отдельный массив, который предохранят от возможности рекурсивного удаления пикселей.

2.3.3 Алгоритм Андреева-Бобкова

У метода жука есть недостатки. Основные недостатки заключаются в следующем:

1) в ситуации, когда в объекте будет последовательность черных пикселей по диагонали друг от друга, то может возникнуть такая ситуация, при которой жук упрется в одну финальную - стартовую точку и закончит выполнение, так и не пройдя по всему контуру символа;

2) наличием «дыр» (областей белых пикселей) внутри объекта, которые жук может пропустить.

3) метод не работает, если контур заканчивается на границе изображения;

4) в контур попадают не только черные, но и белые пиксели;

5) многие пиксели попадают в контур дважды.

Для решения проблем первого рода для задачи сегментации изображения Андреев А.Ю., Бобков С.П. [14] предложили модифицированный метод жука. Основная идея их модификации заключается в следующем: при переходе с белого пикселя через угловой черный по диагонали остается непроверенный пиксель, который может быть черным. В этом случае предлагается занести его в стек и после завершения обхода контура начать обход с этого пикселя. Необходимо отметить, что отслеживание только угловых пикселей не решает проблемы ветвления граничных линий.

2.3.4 Модифицированный метод Андреева-Бобкова

У метода Андреева-Бобкова есть недостатки. Тестирование этого алгоритма показало, что в стек попадают не только непроверенные пиксели, но и те пиксели, которые жук уже прошел как граничные. Так для изображения, представленного на рис. 2, вместо одного пикселя в стек попадает 9. А это приводит к зацикливанию алгоритма.

Все эти проблемы алгоритма порождаются тем, что точки контура не отмечаются как пройденные. Предлагается модифицированный алгоритм [31, 34], предназначенный для обработки изображения нескольких оттенков серого и использует три цвета: $C_{пред}$ – цвет пикселей за жуком; $C_{новый}$ – цвет пикселей перед жуком; $C_{контур}$ – красный цвет пикселей контуров, пройденных жуком. Если, например, $C_{пред}$ – белый, то $C_{новый}$ – черный, если $C_{пред}$ – черный, то $C_{новый}$ – белый. Упорядоченные точки контуров накапливаются в двумерном списке Контуры[,]. Конвертация цветов $C_{пред}$ и $C_{новый}$ происходит после завершения построения очередного контура.

Основные идеи алгоритма заключаются в следующем: 1) отмечать пройденные пиксели, например, красным цветом; 2) в стек следующих точек цвета $C_{новый}$ заносить тех соседей, у которых есть хотя бы одна соседняя точка цвета $C_{пред}$.

Алгоритм построения контуров состоит из следующих шагов:

1. Назначить цвета $C_{пред}$ и $C_{новый}$ по цвету пикселя (0, 0).
2. Сканировать изображение, пропуская пиксели $C_{пред}$ и $C_{контур}$ до тех пор, пока не встретится пиксель цвета $C_{новый}$.
3. Построить очередной контур, перекрашивая пройденные точки.
4. Если сканирование не завершено, то конвертировать цвета $C_{пред}$, $C_{новый}$ и продолжить сканирование.

Сложность построения одного контура заключается: во-первых, в том, контур может иметь точки ветвления; во-вторых, начинаться и заканчиваться на

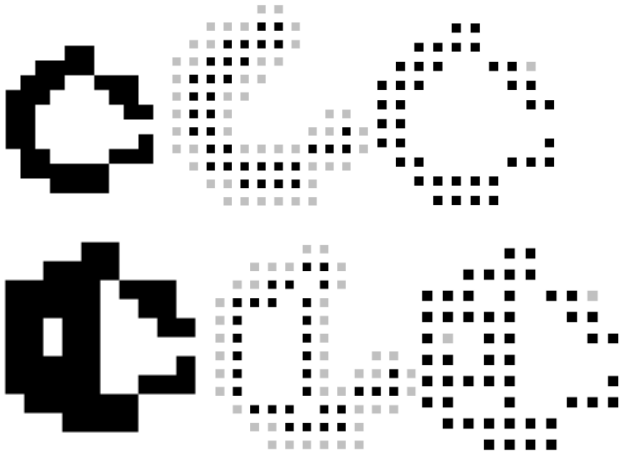
границе изображения. Алгоритм построения одного контура заключается в следующем:

1. Положить начальную точку в стек и добавить в массив контура.
2. Найти следующую точку контура.
3. Если точка найдена, то перекрасить в красный цвет.
   - Если стек не пуст, то взять точку из стека и добавить в массив контура.
   - Иначе вставить точку в начало массива контура.
4. Точка не найдена. Если стек не пуст, то взять точку из стека и организовать новый контур. Цикл завершить.
5. Если найденная точка не совпадает с начальной, то перейти на пункт 2.

Алгоритм определения следующей точки контура получает предыдущую точку, два цвета $C_{пред}$, $C_{новый}$ и заключается в следующем:

1. Опрашиваются соседние 8 точек.
2. Если цвет соседней точки $C_{новый}$ и среди ее 8 соседей есть соседние точки цвета $C_{пред}$, то она попадает в стек соседних точек кандидатов на следующую точку.

На рис. 2.16 представлены результаты работы методов жука Прэтта (б) и модифицированного метода (в). Первое изображение содержит два узловых соединения, второе – контур внутри контура.



а) б) в)

Рисунок 2.16 - Сравнение работы методов жука Прэтта (б) и модифицированного метода (в)

Из рис. 2.16 видно, что метод жука Прэтта потерял часть границы и не нашел внутренний контур. Необходимо отметить так же, что алгоритм жука не уступает по скорости самому простому и некачественному алгоритму Робертса и вдвое быстрее популярного алгоритма Собела [37].

На рис. 2.17 Приведен пример выделения контуров на изображении с 5 оттенками серого цвета.

Рис. 2.17 - Выделение контуров на изображении с 5 оттенками серого цвета

2.4 Сравнение быстродействия алгоритмов выделения контуров

В этом пункте [26, 34] сравнивается быстродействие алгоритмов выделения контуров.

Ни один из алгоритмов выделения контуров, как правило, не применяется ко всем изображениям, поэтому оценка эффективности этих алгоритмов является необходимой. В то время как разработка новых алгоритмов выделения контуров привлекает значительное внимание, относительно меньше усилий потрачено на их оценку. Алгоритмы выделения контуров могут быть аналитически или эмпирическими, так что методы оценки могут быть разделены на три категории: аналитические методы, эмпирические методы и оценка быстродействия. Аналитические методы непосредственно изучают и оценивают сами алгоритмы выделения контуров, анализируя их принципы и свойства. Эмпирические методы косвенно сравнивают алгоритмы выделения контуров, применяя для их проверки качество результатов выделения контуров. Были предложены различные эмпирические методы. Большинство из них подразделяются на два типа:

качественные методы и методы несоответствия. В первой категории некоторые желательные свойства контуров устанавливаются в соответствии с человеческой интуиции и измеряются параметрами "качества".

Проведено сравнение быстродействия модернизированного метода жука Андреева-Бобкова с другими алгоритмами выделения контуров: алгоритм жука, алгоритм Собела, алгоритм Робертса, алгоритм Лапласа, алгоритм Кирша, статистический алгоритм.

Алгоритмы применялись к 8 сегментированным изображениям двух типов: черно-белым и к изображениям 6 оттенков серого (рис. 2.18, 2.19).

Рисунок 2.18 - Примеры исходных изображений

Рисунок 2.18 - Примеры выделения контуров из черно-белого изображения с помощью алгоритмов жука, Собела, Робертса, Лапласа, Кирша, статистического, модифицированного Андреева-Бобкова

Рис. 2.19. Примеры выделения контуров из изображения 6 оттенков серого с помощью алгоритмов жука, Собела, Робертса, Лапласа, Кирша, статистического, модифицированного Андреева-Бобкова

Сравнение быстродействия алгоритмов представлено на рис 2.20 и 2.21. По горизонтали откладывается размер файлов в байтах, по вертикали – сотые доли секунды.

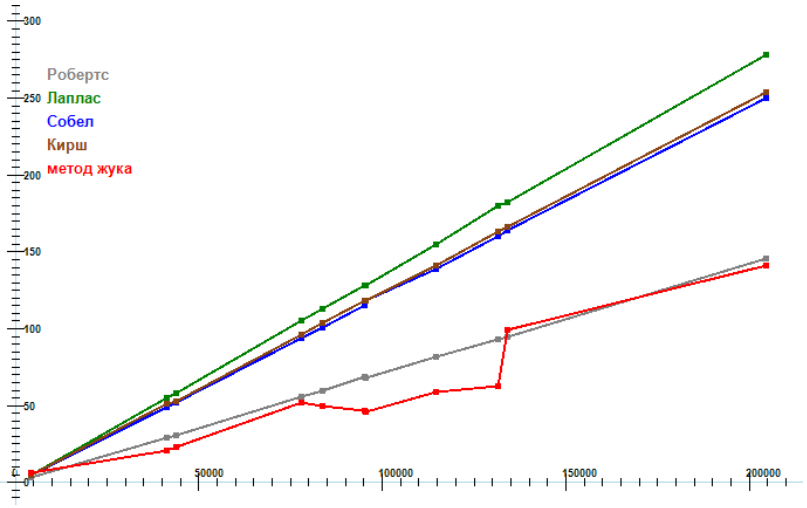


Рисунок 2.20 - Быстродействие алгоритмов жука, Собела, Робертса, Лапласа, Кирша, статистического, модифицированного Андреева-Бобкова для черно-белых изображений

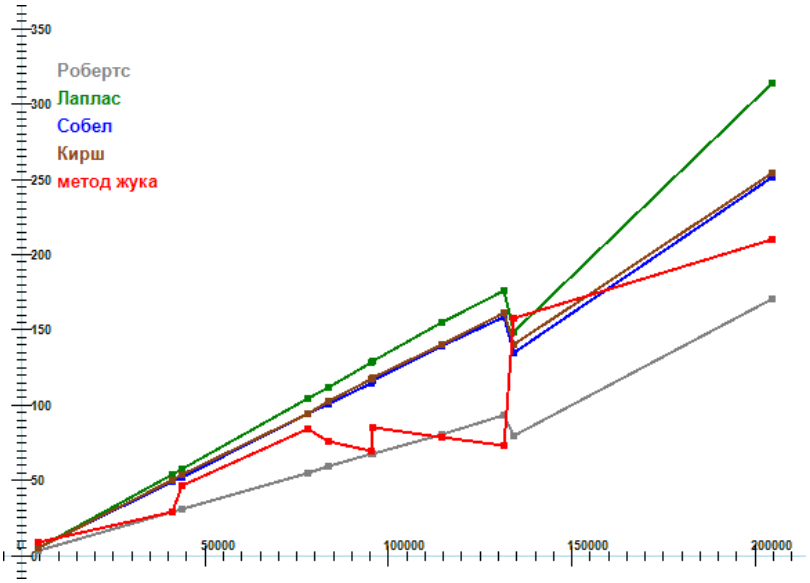


Рисунок 2.21 - Быстродействие алгоритмов жука, Собела, Робертса, Лапласа, Кирша, статистического, модифицированного Андреева-Бобкова для черно-белых изображений.

Необходимо отметить, что алгоритмы Собела, Робертса, Лапласа, Кирша, статистического просто перекрашивают точки изображения, и дальнейшая их векторизация требует дополнительного времени. Алгоритм жука сразу дает упорядоченный наборы точек контуров в отличие от остальных алгоритмов.

Из графиков, представленных на рис. 2.20 и 2.21 видно, что алгоритм жука не уступает по скорости самому простому и некачественному алгоритму Робертса и вдвое быстрее популярного алгоритма Собела.

2.5 Методы численной оценки качества выделения границ

Важное место в процессе машинного зрения занимает этап сегментации изображений. Этап сегментации находится в самом начале цепочки преобразований системы машинного зрения и является нетривиальной задачей, от качества решения которой зависит качество решения задачи машинного зрения в целом. Очень часто сегментация сводится к выделению границ или контуров объектов на изображении [16]. Границы объектов, выделяемые человеком, и границы объектов, выделяемые компьютером, очевидно, будут отличаться. Для достижения оптимального результата сегментации необходимо использовать численные методы оценки качества выделения границ объектов.

Результаты этого раздела опубликованы в работе [23]. В разделе 1 представлен краткий обзор существующих метрик качества выделения границ объектов на изображении. В разделе 2 рассмотрены существующие методы численной оценки качества выделения границ объектов на изображениях. В разделе 3 предложен альтернативный метод оценки качества выделения границ объектов на изображении базирующийся на метриках, рассмотренных в разделе 2. В разделе 4 представлены результаты оптимальной сегментации, проведённой по критерию максимума представленной метрики.

2.5.1. Методы выделения границ объектов на изображениях

На практике широкое применение получили методы выделения границ объектов на основе операторов дифференцирования первого и второго рода [40, 41]. Принцип работы таких методов представлен на рисунке 2.22.

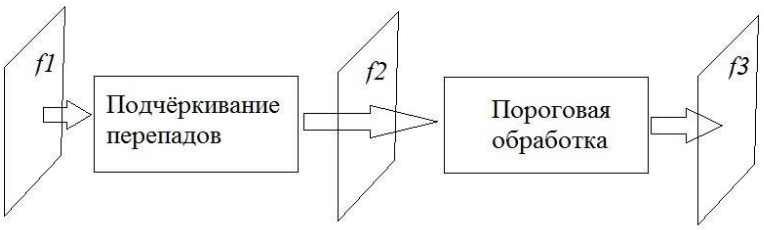


Рисунок 2.22 - Выделение контуров при помощи операторов дифференцирования

Кроме того, существуют методы основанные на выделении контуров по регионам, также известные как методы жука. В этом случае операция пороговой обработки изображения осуществляется первой, после чего изображение подвергается обработке соответствующим алгоритмом. Подробные описания данных методов можно найти в [10, 11].

2.5.2. Существующие методы численной оценки качества выделения границ объектов на изображениях

На практике широкое применение находят следующие метрики численной оценки качества выделения границ объектов на изображениях [37]:

$$P_{CO} = \frac{TP}{\max(N_I, N_B)} \qquad (2.10)$$

– вероятность верного детектирования, где N_I – количество граничных точек в идеальном контуре объекта, N_B – количество граничных точек в полученном в процессе детектирования контуре объекта, TP (true positive) – количество верно определённых граничных точек объекта.

$$P_{nd} = \frac{FN}{\max(N_I, N_B)} \qquad (2.11)$$

– вероятность ошибки первого рода, где FN (false negative) – количество точек, которые являются граничными точками объекта, но которые не были обнаружены детектором границ.

$$P_{fa} = \frac{FP}{\max(N_I, N_B)} \qquad (2.12)$$

– вероятность ошибки второго рода, где FP (false positive) – количество точек, которые не являются граничными точками объекта, но которые были определены детектором границ.

Для оценки качества выделения границ, которые могут быть смещены, применяется метрика Прэтта имеющая вид [37]:

$$IMP = \frac{1}{\max(N_I, N_B)} \sum_{i=1}^{N_B} \frac{1}{1 + \alpha * d_i} \qquad (2.13)$$

где α – коэффициент, регулирующий величину штрафа за смещение граничной точки, d_i – расстояние от граничной точки идеального контура до граничной точки полученного в результате детектирования границ контура.

На основе всех вышеперечисленных метрик можно ввести комплексную метрику вида:

$$\mathrm{D4} = \sqrt{(\mathrm{P_{co}} - 1)^2 + (\mathrm{IMP} - 1)^2 + \mathrm{P_{nd}} + \mathrm{P_{fa}}} \qquad (2.14)$$

Данная метрика может принимать значения из диапазона [0..2]. Детектор, для которого данная метрика имеет значение 0 является идеальным. Напротив, детектор, значение метрики для которого равно 2 считается наихудшим.

Метрика (2.13), как и базирующаяся на ней метрика (2.14), имеют алгоритмическую сложность $O(kN)$, где k – размер окна, в котором происходит поиск смещённых относительно идеального положения точек. Такая сложность приводит к значительным временным затратам. В то же время метрики (2.10), (2.11), (2.12) имеют алгоритмическую сложность $O(N)$, но не могут быть по отдельности эффективно использованы для выбора оптимальных порогов сегментации.

2.5.3. Предложенный метод оценки качества выделения границ объектов на изображениях

На основе метрик (2.10), (2.11), (2.12) была предложена альтернативная метрика качества выделения границ объектов на изображениях.

Пиковое отношение сигнала к шуму (Peak Signal to Noise Ratio, PSNR) — широко используемая метрика для оценки качества выделения сигнала, позаимствованная из теории сигналов.

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{\max_{\mathrm{f}}^{2}}{\mathrm{MSE}}\right) \qquad (2.15)$$

Общепринятое определение среднеквадратичной ошибки имеет следующий вид:

$$\mathrm{MSE} = \frac{1}{\mathrm{mn}}\sum_{\mathrm{i}=0}^{\mathrm{m}-1}\sum_{\mathrm{j}=0}^{\mathrm{n}-1}\| \mathrm{f}(\mathrm{i},\mathrm{j}) - \mathrm{g}(\mathrm{i},\mathrm{j}) \|^{2} \qquad (2.16)$$

В данной работе предлагается определить *MSE* как:

$$\mathrm{MSE} \equiv \mathrm{RE}$$

$$\mathrm{RE} \equiv (\mathrm{b} * \mathrm{P}_{\mathrm{co}} + \alpha)^{-1} + \mathrm{c} * \mathrm{P}_{\mathrm{fa}} + \mathrm{d} * \mathrm{P}_{\mathrm{nd}} \qquad (2.17)$$

– относительная ошибка, определённая через рассмотренные ранее примитивные метрики, где α – константа, ограничивающая RE сверху. b, c, d – весовые коэффициенты, являющиеся мерами значимости примитивных метрик.

Критерий оптимальной сегментации предлагается рассматривать в виде:

$$\mathrm{PSNR} \to \max \qquad (2.18)$$

Предложенная метрика имеет алгоритмическую сложность $O(N)$.

2.5.4. Результаты оптимальной сегментации по критерию максимума предложенной метрики

На рисунке 2.23 представлены результаты эксперимента по применению определённой ранее метрики, проведённого в среде Matlab.

Для каждого из методов обнаружения границ выделены локальные максимумы значений вычисленной метрики. Соответствующие данным

максимумам пороги сегментации принимаются за оптимальные. Полученные пороги сегментации могут быть использованы для последующей автоматической обработки с целью решения конкретной прикладной задачи.

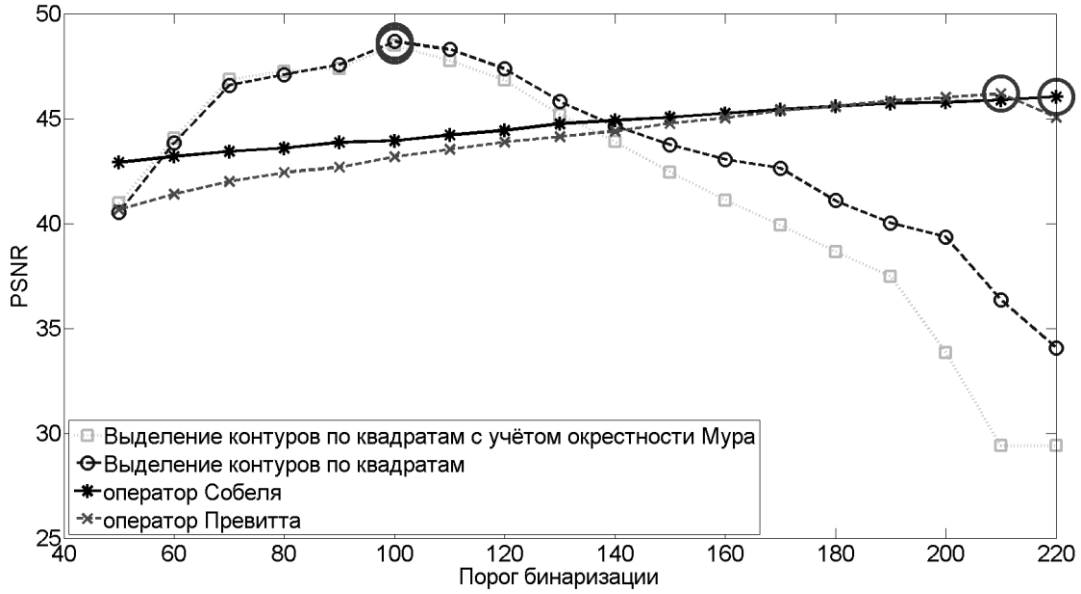


Рисунок 2.23 - Выбор оптимальных порогов сегментации согласно критерию максимума пикового соотношения С/Ш

Изображение было обработано четырьмя методами выделения границ объектов с различными порогами сегментации. Шаг дискретизации порогов сегментации составил 10 единиц. В результате были получены четыре набора изображений с выделенными границами объектов. Каждый из наборов содержит 17 изображений. Из каждого набора было выбрано одно изображение, для которого значение метрики PSNR является наибольшим. Данные четыре изображения содержат оптимально выделенные границы объектов в смысле максимума определённой выше метрики.

Для проведения эксперимента были использованы изображения из набора BSDS (Berkley Segmentation Data Set). На рисунке 2.24 представлено изображение под номером 106025, входящее в состав набора BSDS. Сегментация представленного изображения является тяжёлой задачей в силу незначительного отличия значений интенсивности части точек объекта от значений интенсивности точек фона.

Для вычисления значений метрик были использованы изображения из того же набора, содержащие контуры объектов, сегментированных человеком [8]. На рисунке 2.25 изображены результаты проведённой оптимальной сегментации. На представленном рисунке можно видеть выделенный контур объекта, который в данном случае представляет собой животное.

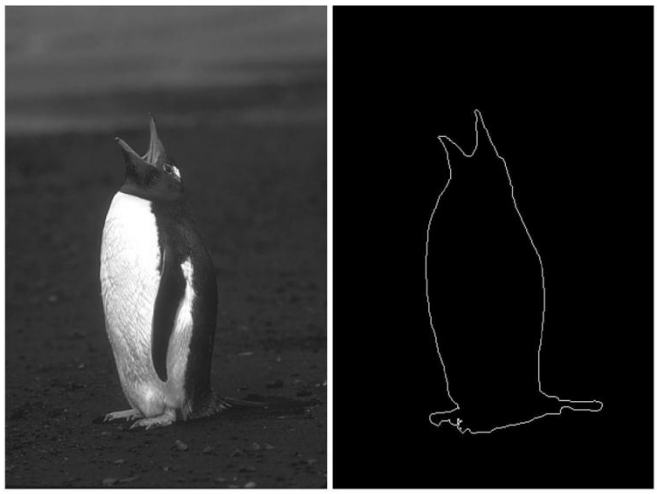


Рисунок 2.24 - Оригинальное изображение и контур

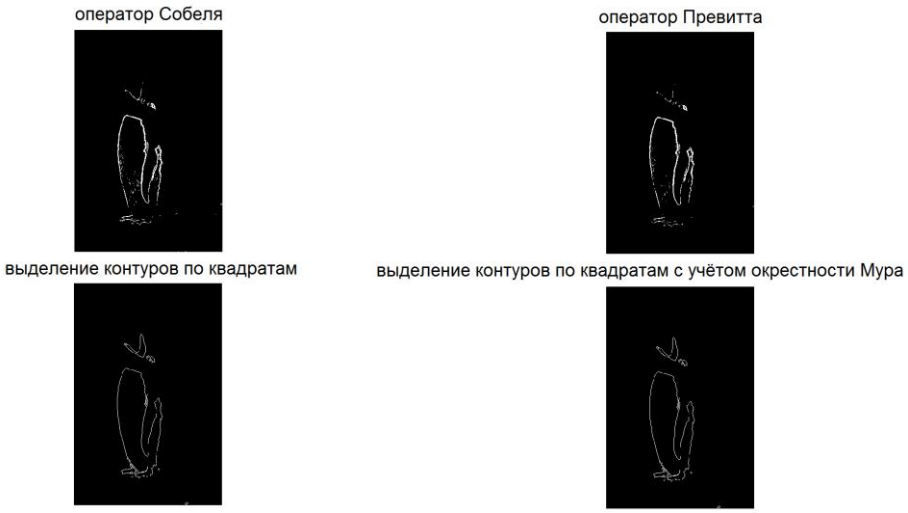


Рисунок 2.25 - Результаты оптимальной сегментации по критерию максимума пикового отношения С/Ш

Полученные контуры объекта, сегментированного автоматически, визуально согласуются с ожидаемым результатом, что позволяет сделать вывод о состоятельности проведённого эксперимента.

3 Векторизация

Для распознавания изображений необходимо преобразовать растровые данные в векторные [9,12,16,37].

При распознавании изображений выделение контуров или скелетов, с последующей векторизацией, является одним из наиболее сложных этапов. Векторизация необходима для уменьшения количества параметров, описывающих изображение.

Перечислим основные методы векторизации:

1. Методы, использующие преобразование Хафа. Они основываются на преобразовании изображения по заданному правилу и получению пространства параметров, из которого затем извлекаются объекты, параметры которых находятся в локальных максимумах этого пространства.
2. Методы, основанные на процедурах определения серединных линий исходных линий на растровом изображении:
    a. алгоритмы утончения;
    b. алгоритмы скелетизации линий,
    c. алгоритмы отслеживания контуров.

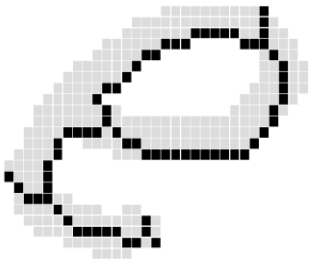
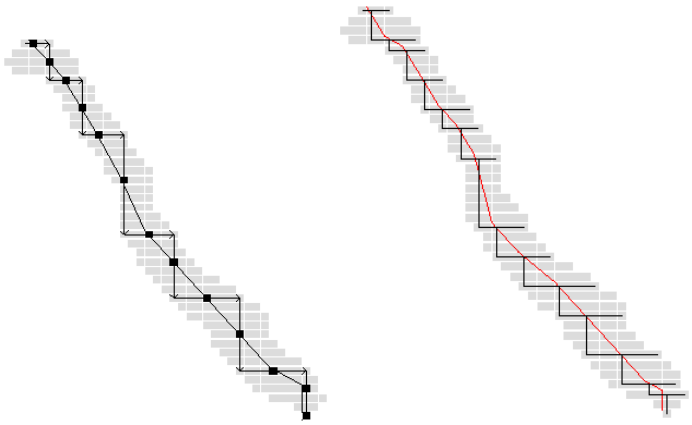


Рисунок 3.1 - Скелетизация



б в

Рисунок 3.2 - Методы векторизации:
а — скелетизации; б — OZZ; в — SPV

3. Методы предварительной обработки растрового изображения, которые заключаются в его представлении в виде количественных характеристик соседних пикселей с последующим анализом этих характеристик.
4. Разреженно-пиксельные методы, использующие для векторизации часть пикселей изображения.
    d. Метод OZZ (Orthogonal Zig-Zag) (рис. 1, *б*) заключается в отслеживании линий на изображении вертикальными и горизонтальными ходами, заканчивающихся при достижении края линии. Недостатком этого метода является малая точность построения векторных линий, в частности, для кривых линий.
    e. Метод SPV (Sparse Pixel Vectorization) (рис. 1, в) позволяет получить более точное построение траектории по сравнению с OZZ. Отличается более сложной, но точной процедурой поиска первой точки линии на изображении и использованием одинаковых циклов ходов для любых линий [4,37].
    f. Метод LNG (Line Net Global), отличается от методов OZZ и SPV тем, что обеспечивают отслеживание линий на изображении с

произвольными направлениями, не ограничиваясь вертикальными и горизонтальными ходами [37].

g. Волновой метод [9,12,16,37]отличается от OZZ и SPV тем, что обеспечивают отслеживание линий на изображении с произвольными направлениями, не ограничиваясь вертикальными и горизонтальными ходами

5. сопоставлении контуров;
6. графах бегущих штрихов;
7. разбиении изображения регулярной сеткой.

3.1 Обзор методов векторизации

3.1.1 Преобразование Хафа

Любой примитив может быть задан аналитически с помощью уравнения:

$$f(x, y, \boldsymbol{a}) = 0$$

В этом уравнении x и y являются координатами пикселя, а вектор $\boldsymbol{a}$ задает форму примитива. В зависимости от типа примитива вектор имеет различное число и назначение компонент. Например, для прямой линии он состоит из трех чисел, а функция задается формулой $f(x, y, \boldsymbol{a}) = a_1x+a_2y+a_3$. Для окружностей вектор $\boldsymbol{a}$ по-прежнему состоит из компонент, которые задают координаты центра и радиус, а функция f задается формулой $f(x, y,\boldsymbol{a}) =(x-a_1)^2+(y-a_2)^2-a_3$.

Поиск примитивов заключается в определении таких значений a, при которых уравнение выполняется для наибольшего количества граничных пикселей. В некоторых случаях учитывают не только факт прохождения границы, но и величину градиента, оценивая не только суммарное количество пикселей, удовлетворяющих уравнению, а сумму величин их градиентов. Для этого заводится массив C[a], индексы которого являются значениями параметра a. Элементы этого массива содержат количества пикселей и дополнительно могут содержать список координат этих пикселей. Алгоритм сканирует все пиксели изображения и если пиксель является границей, то соответствующий элемент

массива C[*a*] увеличивается на единицу. Затем определяются максимальные элемента массива C[*a*]. Соответствующие им индексы, то есть значения a, задают форму наиболее заметных графических примитивов в виде уравнения $f(x, y, \boldsymbol{a}) = 0$.

Сложность реализации алгоритма сводится к эффективной организации массива C[**a**], который должен быть небольшим по размеру, позволять указывать для всех возможных $\boldsymbol{a}$, и обращение к его элементам должно осуществляться максимально быстро. Для этого используют следующие приемы:

- выполняется дискретизация значений компонент вектора a;
- функция $f(x, y, f)$ модифицируется таким образом, чтобы множество возможных значений a было как можно меньше;
- для организации массива используются специальные структуры данных.
- Перечисленные приемы хорошо видны на примере преобразования Хафа для прямых линий и окружностей.

3.1.2 Векторизация прямых линий

Принято задавать прямые линии уравнением вида $ax+by+c = 0$, но оно непригодно для алгоритма векторизации. Проблема заключается в том, что индексами массива C[**a**] будут являться тройки чисел (a, b, c). Множество возможных значений этих троек велико после дискредитации значений. Если использовать более простой вариант задания линии в виде уравнения $y = bx + c$, то множество возможных значений пар (b,c) все равно велико.

Решение этой проблемы заключается в рассмотрении уравнения прямой в виде:

$$x \cos\vartheta + y \sin\vartheta - d = 0$$

Числа x и y являются координатами пикселя, ϑ задает направление нормали к прямой, а d показывает расстояние от начала координат до прямой.

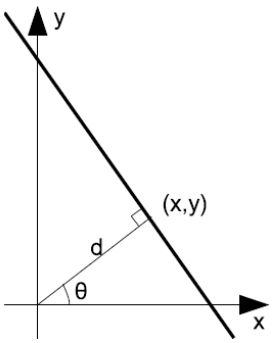


Рисунок 3.3 - Параметрическое задание прямой

Значение ϑ изменяется в интервале от 0 до $\pi/2$ и может быть дискретизировано с шагом $\pi/180$. Величина может быть любой, но тоже может быть дискретизирована. Это позволяет достаточно эффективно организовать массив C[***a***], где ***a*** представляет собой два числа (ϑ, d).

Алгоритм обработки каждого граничного пикселя достаточно прост. Направление градиента всегда перпендикулярно границе и, фактически, задает направление перпендикуляра к прямой, а зная координаты пикселя (x, y) и угол ϑ несложно вычислить значение d.

3.1.3 Определитель прямых линий Барнса

Определитель прямых Барнса предназначен для векторизации отрезков и основан на несколько иных принципах, отличных от преобразования Хафа. Алгоритм включает в себя несколько шагов:

- вычисление градиентов;
- дискредитация градиентов;
- поиск связных компонент;
- голосование.

Первый шаг (вычисление градиентов) основан на использовании дифференциальных масок и его результатом является двумерный массив, элементы которого соответствуют пикселям изображения и содержат значения градиентов. По результатам первого шага определяются граничные пиксели.

На втором шаге происходит дискретизация направлений градиента граничных пикселей по двум шкалам. Первая шкала содержит восемь значений, соответствующих направлениям градиента от 0° до 45°, от 45° до 90°, от 90° до 135° и так далее. Вторая шкала так же содержит восемь значений со сдвинутыми диапазонами значений на 22,5°, то есть они соответствуют направлениям от 22,5° до 67,5°, от 67,5° до 112,5° и так далее. Таким образом, каждый пиксель получает две метки.

Поиск связных компонент заключается в группировке пикселей, которые имеют одинаковое значение метки. Способ группировки достаточно прост: два соседних пикселя относятся к одной группе, если имеют одинаковое значение метки (то есть их градиенты направлены примерно в одну сторону). Далее эта схема рекурсивно применяется к соседним пикселям и, в результате, все изображение разбивается на набор непересекающихся связных групп пикселей.

Обе метки обрабатываются отдельно, поэтому строится два разбиения изображения, а каждый пиксель входит в две компоненты, Отдельно определяется длина всех обнаруженных компонент.

На заключительном шаге, называемом голосованием, для компонент заводятся счетчики проголосовавших за них пикселей. Каждый пиксель входит сразу в две компоненты, но увеличивает счетчик более длинной из них. По окончании работы анализируются итоговые значения счетчиков и выбираются набравшие наибольшее количество голосов компоненты. У входящих в них пикселей градиенты направлены примерно одинаково, поэтому соответствующие фрагменты изображения являются прямыми отрезками, расположенными перпендикулярно данным градиентам.

3.2 Модифицированный волновой метод

Волновой метод [9,12,16,37], в отличие от OZZ и SPV, обеспечивают отслеживание линий на древовидном растровом изображении. Волновые методы могут быть сплошными [9] – они отслеживают все пиксели изображения и

разреженными [12], отслеживающие только часть точек изображения. В этой работе предлагается алгоритм, использующий сплошную волну.

Предлагаемый алгоритм [27, 30, 35] отличается от алгоритма предложенного Денисовым И. и Кузьминым Е. тем, что волна идет по линии толщиной в 1 пиксель и появление в волне двух пикселей служит сигналом для создания узла. Вот для этого и нужен модифицированный алгоритм Щепина.

Любой волновой алгоритм строит дерево линий, даже если изображение представляет собой граф (рис. 3.4).

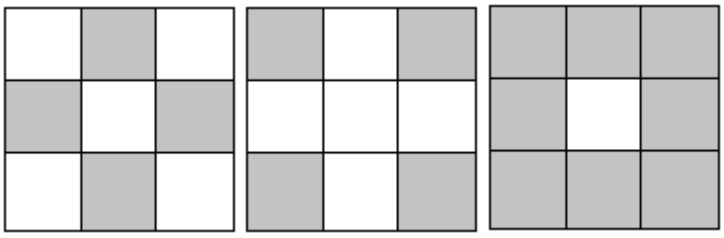
Рисунок 3.4 - Символ «а» представляет собой граф с 2 узлами

Для построения графа требуются дополнительные усилия. В отличии от алгоритма, описанного в работе [22], предлагаемый алгоритм строит граф с минимальным количеством узлов.

Волновой метод заключается в анализе пути прохождения волны по изображению. На каждом шаге анализируется смещение центра волны, который образует скелет изображения.

Алгоритм устойчив к белому шуму, но желательно от него избавиться.

Законы распространения волн отличаются для 4-х связных шаблонов растра (рис. 3.5а, 3.5б), 8-ми связного шаблона (рис. 3.5в).

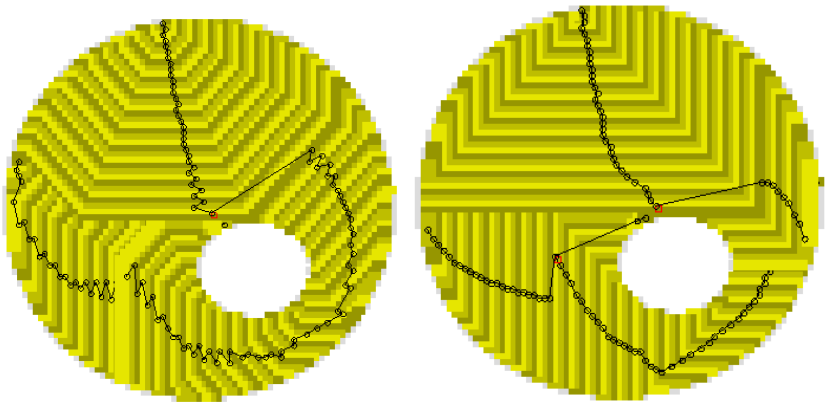


а б в

Рисунок 3.5 - Шаблоны волны

При использовании четырех связного шаблона распространение волны идет в форме ромба или квадрата, при 8-связном – в виде квадрата.

Для генерации восьмиугольной волны необходимо скомбинировать четырех и восьми связное распространение волны (рис. 3.6а). Это достигается попеременным применением 4-х и 8-и связного распространения.



а б

Рисунок 3.6 - Комбинация 8-ми и 4-х связных шаблонов

Применение только 4-х связных шаблонов дает худший результат (рис. 3.7).

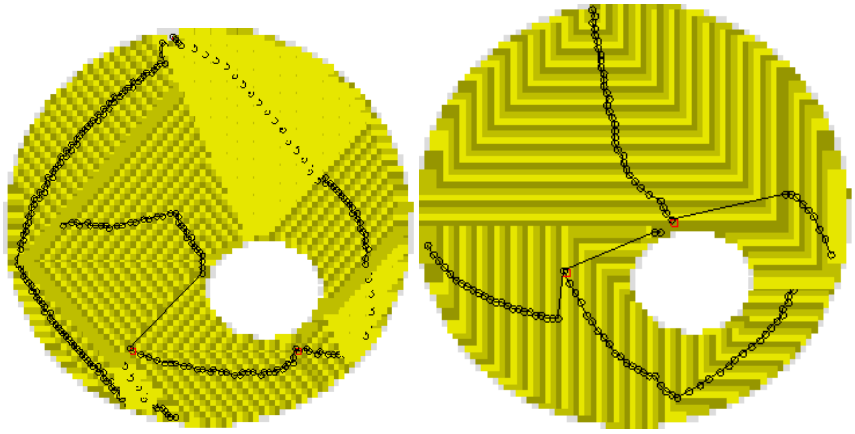


Рисунок 3.7 - Волны, образованные 4-х связными шаблонами

Алгоритм предполагает построение скелетов нескольких несвязанных объектов, каждый из которых представляет собой граф. Поэтому в классе «Волна» объявлен список «Вершины», каждый элемент которого показывает на начальный узел графа (рис. 3.8).

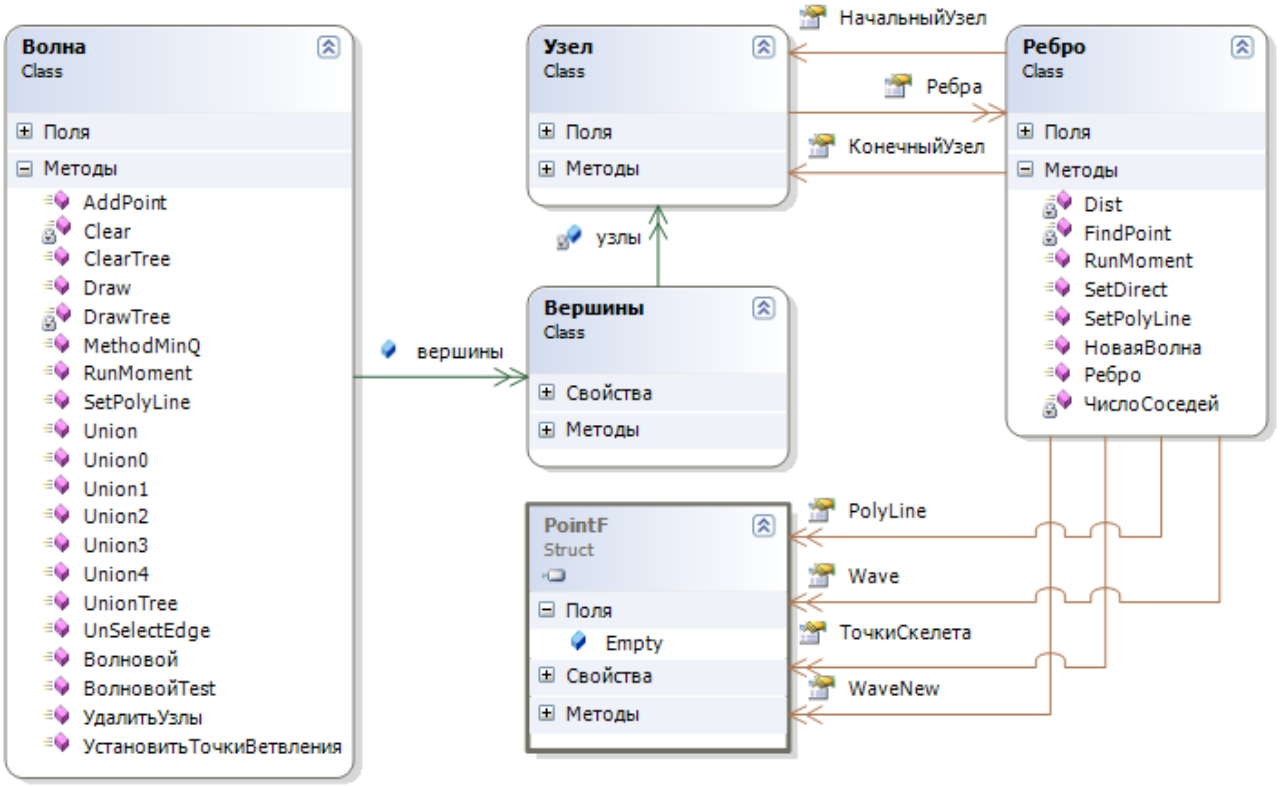


Рисунок 3.8 - Структура данных

Графы состоят из узлов. Узлы имеют список ребер. У ребра есть указатели на начальный и конечный узлы. Если конечного узла нет, то «КонечныйУзел» = null. У каждого узла есть список точек старой волны «WaveOld» и новой волны «WaveNew». Точки старой волны перекрашиваются в белый цвет. В процессе прохождения волны строится список точек скелета. По этим точкам далее строится сглаженная линия «PolyLine».

Новый узел появляется в случае, когда волна разбивается на две или более частей. Каждая часть волны порождает новое ребро и запускает на нем метод построения волны. Метод построения волны является рекурсивным, который заканчивает свою работу в случае, если длина волны равна 0 или выходит на границы изображения.

Алгоритм состоит из следующих шагов:

1. Сканируется изображение до первой не белой точки. Эта точка назначается узлом – вершиной графа и старой волной ребра.

2. Как правило, первая вершина идет поперек растровой линии и имеет единственное короткое ребро. Поэтому ее необходимо удалить.
3. У ребра запускается метод построения волны.
4. Упорядочиваются точки новой волны.
5. Если новая волна не имеет разрывов, то середина волны добавляется к ребру дерева.
6. Иначе создается новый узел дерева и для каждого фрагмента новой волны рекурсивно запускается метод построения волны. Переходим к пункту 3.
7. Метод построения волны заканчивается, если длина волны равна нулю.

На рис. 3.9 показаны этапы работы алгоритма.

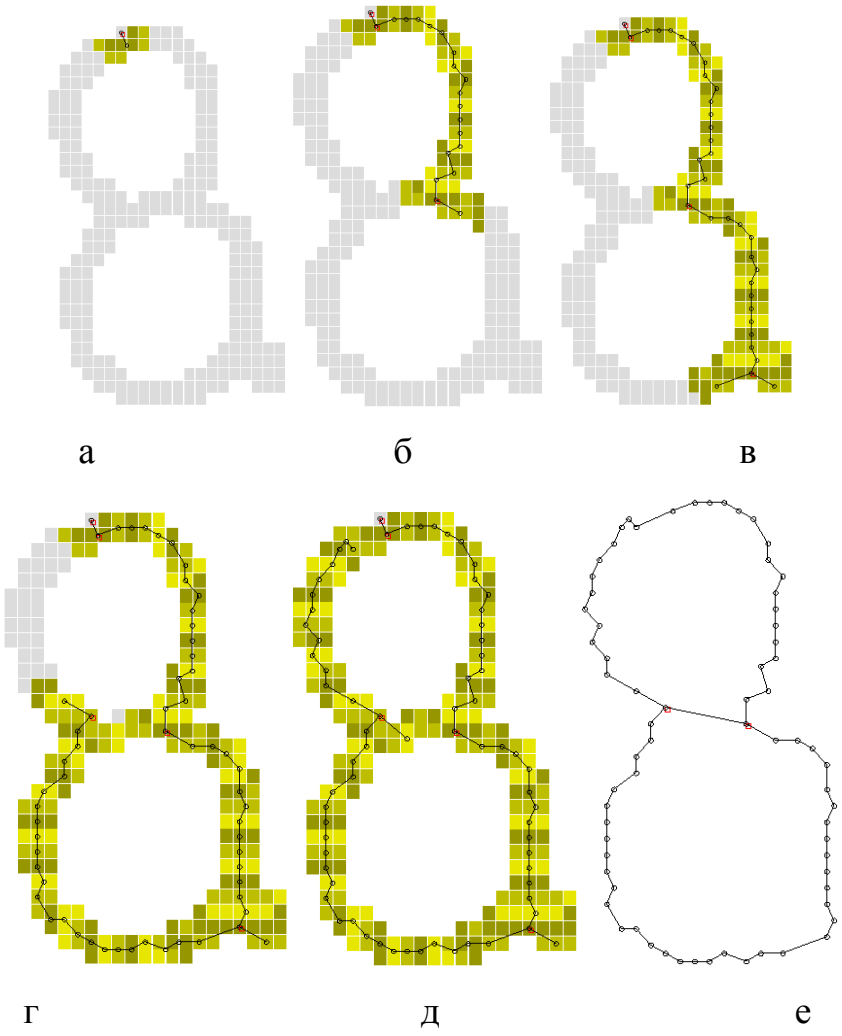


Рисунок 3.9 - Этапы работы алгоритма

Полученный скелет изображения не является графом и не является оптимальным.

Далее необходимо:

1. Найти узлы близкие к концам ребер и добавить в них найденные ребра.
2. Удалить узлы с короткими ребрами.
3. Удалить узлы с одним ребром.
4. Соединить ребром близкие узлы.
5. Удалить узлы с двумя ребрами.

Окончательный результат работы алгоритма представлен на рис. 6е.

Представленный алгоритм может использоваться при создании программ распознавания текста, конструкторской документации, в ГИС и САПР.

Текст методов волнового алгоритма представлен в Приложении 2.

3.3 Упрощение графа скелета изображения

В результате применения волнового метода к скелетам изображений символов получаются деревья, состоящее из узлов и ребер (рис. 3.10).

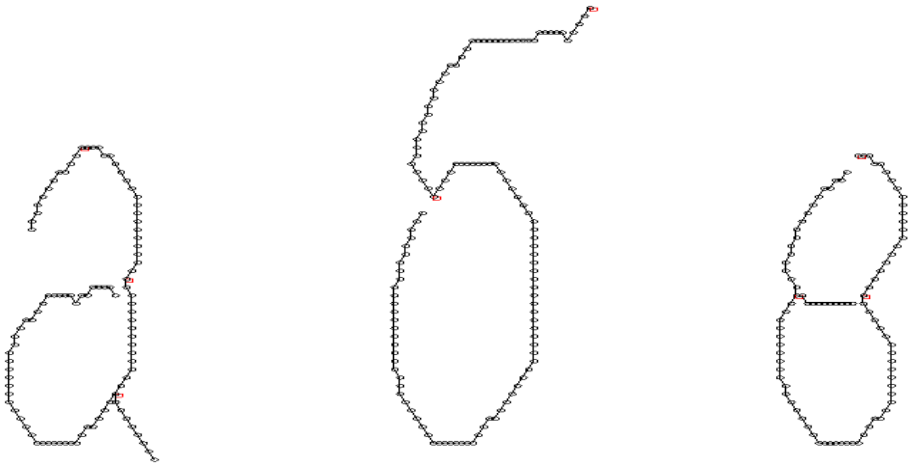


Рисунок 3.10 - Деревья символов

Эти деревья необходимо превратить в графы, а графы максимально упростить. Это означает, что не может быть узлов с индексом 2, то есть узлов разрезающих ребро. Не может быть коротких ребер.

Для каждого графа вычисляются следующие топологические признаки:

- Число диакритических знаков;

- Число узлов;
- Число ребер.

Для каждого ребра определяются следующие признаки:

- Количество узлов на концах (1 или 2);
- Ребро прямая, дуга или кольцо;
- Для дуги определяются секторы круга, которые она накрывает.
- Характер изменения кривизны.

Для построения графа и его упрощения предлагается следующая последовательность действий:

1. Удалить узлы с короткими ребрами. Если у узла несколько ребер и среди них есть короткое, которое не показывает на другой узел.
2. Найти узлы близкие к концам ребер.
3. Удалить узлы с одним ребром.
4. Удалить узлы с двумя ребрами.
5. Найти точки близкие к узлам.
6. Удалить узлы с одним ребром.
7. Найти точки близкие к концам.
8. Удалить одинаковые ребра.

4 Анализ изображения. Распознавание

Распознаванию изображений посвящено множество работ [42, 44,45,47].

4.1 Анализ по геометрическим признакам

Для каждой области можно подсчитать некий набор простейших числовых характеристик:

- площадь – количество пикселей в области;

$$A = \sum_{x=0}^{m} \sum_{y=0}^{n} I(x, y)$$

- центр масс;

$$\bar{x} = \frac{\sum_{x=0}^{m} \sum_{y=0}^{n} xI(x, y)}{A};\ \bar{y} = \frac{\sum_{x=0}^{m} \sum_{y=0}^{n} yI(x, y)}{A}$$

- количество пикселей границы области;
- отношение квадрата периметра к площади. Круг – наиболее компактная фигура;

$$C = \frac{P^2}{A}$$

- ориентация главной оси инерции;
- эксцентриситет.

4.2 Статистические моменты области

Дискретные центральные моменты m_{ij} области определяется таким образом [Josef Bigun. Vision with Direction. A Systematic Introduction to Image Processing and Computer Vision. O Springer-Verlag Berlin Heidelberg 2006]:

$$m_{ij} = \sum_{x,y \in S}^{n} (x - \bar{x})^i (y - \bar{y})^j I(x, y)$$

Для распознавания изображений используются характеристики инвариантные по отношению к масштабированию, переносу, повороту:

Удлиненность, эксцентриситет

$$elongation = \frac{m_{20} + m_{02} + \sqrt{(m_{20} - m_{02})^2 + 4m_{11}^2}}{m_{20} + m_{02} - \sqrt{(m_{20} - m_{02})^2 + 4m_{11}^2}}$$

Компактность $C = \frac{P^2}{A}$

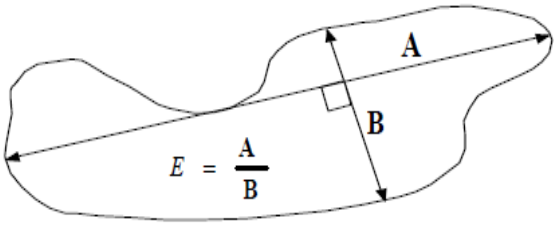


Рисунок 4.1 - Отношение ширины к высоте

Ориентация главной оси инерции не инвариантна к повороту, но иногда предоставляет полезную информацию об ориентации объекта:

$$\theta = \frac{1}{2}\arctan(\frac{2m_{11}}{m_{20} - m_{02}})$$

Рисунок 4.2 - Ориентация главной оси

Другие инвариантные характеристики области:

$$M_1 = m_{20} + m_{02}$$

$$M_2 = (m_{20} - m_{02})^2 + 4m_{11}^2$$

$$M_3 = (m_{30} - 8m_{12})^2 + (8m_{21} - m_{03})^2$$

$$M_4 = (m_{30} + m_{12})^2 + (m_{21} - m_{03})^2$$

$$M_5 = (m_{30} + 8m_{12})(m_{30} + m_{12})[(m_{30} + m_{12})^2 - 8(m_{21} + m_{03})^2]$$

$$+(6m_{21} - m_{12})(m_{21} + m_{03})[8(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2]$$

$$M_6 = (m_{20} + m_{02})[(m_{30} + m_{12})^2 - 8(m_{21} + m_{03})^2]$$

$$+(4m_{11}(m_{30} - m_{12})(m_{03} + m_{21})$$

$$M_7 = (8m_{21} - m_{03})(m_{12} + m_{30})[(m_{30} + m_{12}^2 - 8(m_{21} + m_{03})^2]$$
$$-(m_{30} - 8m_{12})(m_{12} + m_{03})[8(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2]$$

Где m_{pq} представляют центральные моменты, было предложено Hu [M.K. Hu. Visual pattern recognition by moment invariants. *IRE Trans, on Information Theory*, pages 179-187. 1962], инвариантные относительно вращений.

4.3 Фурье-дескрипторы

Способ, основанный на использовании Фурье-дескрипторов [2] который позволяет получить признаки, обладающие инвариантностью к сдвигу, повороту и масштабу. Фурье-дескрипторы определяются для непрерывных или дискретизированных кривых с равноудалёнными точками.

Несмотря на ограничения, связанные с представлением контуров в виде двумерного отсчётов, инвариантность признаков, основанных на Фурье-дескрипторах, делает их весьма привлекательными.

Значения дескрипторов Фурье вычисляются с помощью преобразования Фурье.

Пусть границы объекта заданы на плоскости (*xy*) (на рис. 4.3). Начиная с произвольной точки (x_0,y_0), обойдем границу и обозначим координаты точек границы (x_0,y_0); (x_1,y_1); (x_2,y_2);…(x_{N-1},y_{N-1}). Запишем эти координаты в форме $x(k)=x_k$ и $y(k) = y_k$. Границу объекта представить в виде последовательности координатных пар

$$f(k) = [x(k), y(k)] \qquad (4.1)$$

где $k = 0,1,2,\ldots<N-1$.

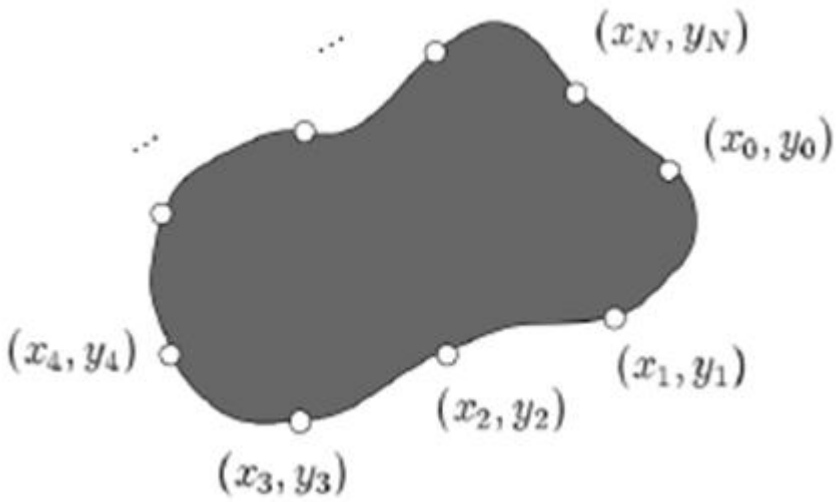


Рисунок 4.3 - Расстановка точек на границе

Каждую пару координат точки рассмотрим как комплексное число

$$f(k) = x(k) + i \cdot y(k). \quad (4.2)$$

Дискретное преобразование Фурье для последовательности задается уравнением:

$$\tilde{F}(u) = \frac{1}{N} \sum_{k=0}^{N-1} f(k) exp\left(-i \frac{2\pi uk}{N}\right) \quad (4.3)$$

для $u = 0,1,2,...,N-1$. Комплексные коэффициенты *F(u)* называются фурье-дескрипторами границы. Обратное преобразование Фурье, примененное к этим коэффициентам, позволяет восстановить границу *f(k)*:

$$f(k) = \frac{1}{N} \sum_{u=0}^{N-1} \tilde{F}(u) exp\left(i \frac{2\pi uk}{N}\right), k = 0,1,2, \dots N-1 \quad (4.4)$$

Фурье-дескрипторы однозначно описывают границы объекта.

Дескрипторы нижних частот описывают общие сведения о форме контуров, а дескрипторы высоких частот – мелкие детали.

На рис. 4.4 представлены результаты описания контура объекта в виде шурупа (рис. 4.5) при использовании, соответственно, 100, 90, 80, 20, 10 и 5 дескрипторов, что составляет примерно 9,92%, 9,1%, 8%, 2%, 1% и 0,5% от 992 имеющихся дескрипторов.

Рисунок 4.4 - Изображение шурупа

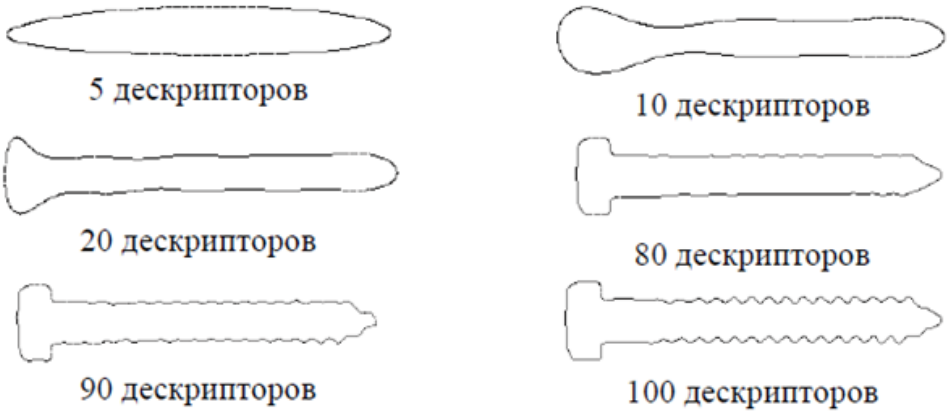


Рисунок 4.5 - Изображение шурупа, восстановленные по дескрипторам

При использовании 5 дескрипторов получаем изображение эллипса, искажения уменьшаются при увеличении числа дескрипторов и при использовании 100 дескрипторов получается описание области, близкое к исходному.

Результаты показывают, что для описания формы объектов требуются некоторое число дескрипторов.

Фурье-дескрипторы сами по себе не инвариантны к параллельному переносу, повороту и изменению масштаба объектов, но на основе Фурье-дескрипторов можно получить инвариантные признаки:

$$X = \left(\frac{|F_{-L/2}|}{|F_1|} \ldots . \frac{|F_{-2}|}{|F_1|}, \frac{|F_{-1}|}{|F_1|} \frac{|F_2|}{|F_1|} \ldots \frac{|F_{(L/2)-1}|}{|F_1|} \right)^T \qquad (4.9)$$

4.4 Сравнение графов изображения символов

В работе [38] было предложено использовать для распознавания топологическое описание изображений, если последние можно считать плоскими графами. При этом в отдельных задачах (автоматическое чтение текста и др.) все возможные изображения, составляющие тот или другой класс, можно представить при отсутствии помех как результат гомеоморфных преобразований эталонного изображения, соответствующего этому классу. Проблема распознавания в этом случае сводится к установлению эквивалентности изучаемого изображения с одним из эталонных. Эту проблему можно решить с помощью топологических инвариантов - таких свойств изображения, не изменяющихся при его гомеоморфных преобразованиях. Инвариантами, позволяющим описать изображение, являются, например, число узлов, индексы узлов графа, число диакритических знаков (у «й» один диакритический знак, у «ё» - 2).

У арабских символов система диакритических знаков развита значительно сильнее (рис. 4.6).

ء ا ب ة ت ث ج ح خ

د ذ ر ز س ش ص ض ط

ظ ع غ ف ق ك ل م ن

ه و ي

Рисунок 4.6 - Арабские символы

В работе [32] проведен анализ и сравнение методов распознавания рукописных символов.

В частности, в работе [22] для каждого узла скелетного представления предлагается использовать следующие топологические признаки:

- нормированные координаты узлов графа (нечеткий);
- длина ребра до следующего узла в процентах от длины ребер всего графа (нечеткий);
- нормированное направление из узла на следующий узел (нечеткий);
- нормированное направление входа в узел, выхода из узла: для узлов с индексом больше 2 эти значения различаются, для узлов индекса 1 совпадают с точностью до знака (нечеткий);
- "левая" и "правая" кривизна дуги, соединяющей узел со следующим узлом. Кривизна - это отношение максимального расстояния от узлов дуги до прямой, соединяющей узлы, к длине отрезка, соединяющего те же узлы (нечеткий).

Распознавания сводится к сравнению топологических признаков изображения и эталонных изображений классов. Важное достоинство топологического описания - его нечувствительность к деформациям изображения, включающим преобразования подобия и поворота. Обучение заключается в ведении набора эталонных описаний.

В этой работе предлагается [29] все символы любого алфавита классифицировать по следующим топологическим признакам:

1. число диакритических знаков;
2. положение диакритических знаков (сверху, снизу, в центре, справа);
3. число узлов графа скелета;
4. число ребер;
5. число узлов у ребра;
6. ребро дуга или кольцо;
7. число линейных уравнений для лучшей интерполяции производной дуги (1, 2 или 3);
8. для 2-х или 3-х линейных уравнений смена знака кривизны (да или нет);

9. разрыв кривизны для 2-х или 3-х линейных уравнений, если нет смены знака кривизны;
10. секторы дуги.

В таблице 4.1 приведена классификация символов кириллицы по первым трем топологическим признакам.

Таблица 4.1.

Классификация символов кириллицы по топологическим признакам.

Топология	Диакри-тические	Узлов	ребер	прямых	дуг	колец	*символы*
	0	1	1	1			*и, г, п, л, с, м*
	0	1	3	3			*ц, у, е, ш, з, э, ч, т*
	0	1	4	4			*к, х*
	0	1	1			1	*о*
	0	1	2	1		1	*ъ, ы, р, ь, б, д*
	0	2	3	1	2		*в, в*
	0	2	4	2	2		*а, д, я, е*
	0	2	4	3		1	*ю*
	0	2	5	4	1		*н, щ*
	0	2	5	3	2		*ф*
	0	3	8	6	2		*ж*
	1	1	1	1	0	0	*й*
	2	2	4	2	2	0	*ё*

В приложении 3 приведена классификация арабских символов по первым трем топологическим признакам.

Эти топологические признаки собираются в базу данных и используются для распознавания символов.

Многие символы, например «*ъ, ы, р, ь, б, д*», по топологическим признакам собираются в группы. Из рис. 4.7 видно, что символы «*б*» и «*д*» имеют одинаковую топологическую структуру: 1 узел; ребро-кольцо; ребро-дуга.

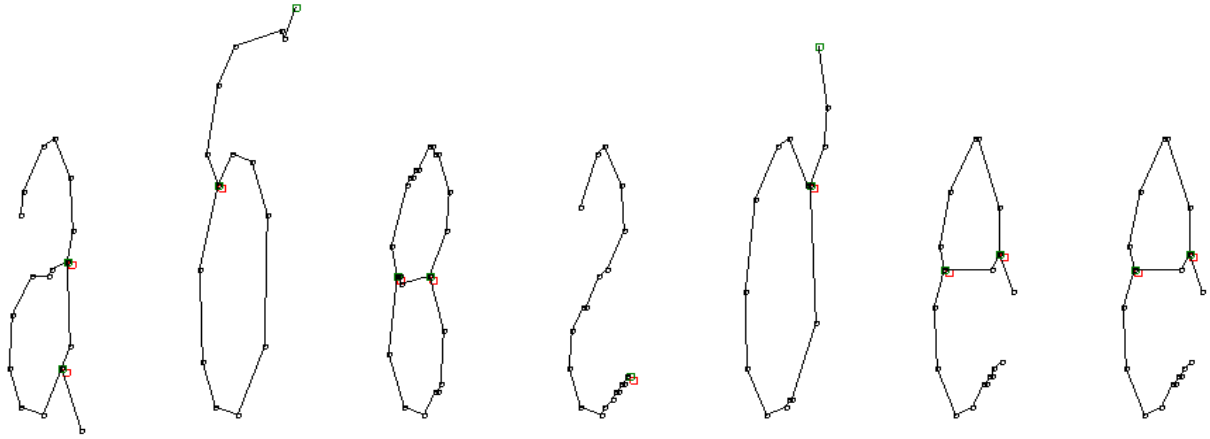


Рисунок 4.7 - Графы для нескольких первых символов кириллицы

Поэтому необходим дополнительный анализ ребер.

Вторым дополнительным признаком являются секторы круга, которые накрывает ребро. По трем точкам дуги (первой, последней и средней) вычисляются координаты и радиус окружности. Круг разбивается на 8 секторов. Попадание ребра в сектор отмечается как 1 в 8-ми битовом представлении ребра. Так, например, дуге символа *д* на рис. 4 соответствует число 11000001_2 (рис. 4.8)

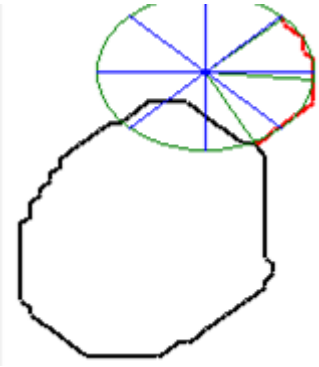


Рисунок 4.8 – Секторы для дуги символа *д*

Эта окружность разбивается на 8 секторов. Вводится секторальный признак типа byte размером 8 бит. Секторы пронумерованы от оси 0X против часовой стрелки. Каждый бит отвечает за один сектор и принимает значение 1, если сектор заполнен точками дуги не менее, чем на 75%. Так для дуги, изображенной на рис. 4.8, соответствующий признак равен 10000001_2.

Для символа «б» (рис. 4.9)

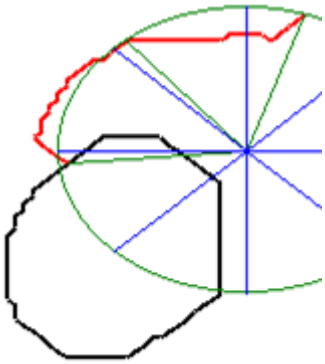


Рисунок 4.9 – Секторы для дуги символа *б*

секторальный признак равен 00110000_2. Не совпадение секторальных признаков позволяет разрешать некоторые коллизии.

Первым дополнительным признаком является изменение кривизны дуги (рис. 4.10). Для этого построим график изменения первой производной ребра. Аппроксимируем эти точки 1, 2 или 3 уравнениями y = k*x+b и методом наименьших квадратов определим коэффициенты k и b. Выбирается число уравнений с наименьшим средне квадратичным отклонением. Коэффициент k отвечает за кривизну, а коэффициент b – за разрыв первого рода первой производной ребра.

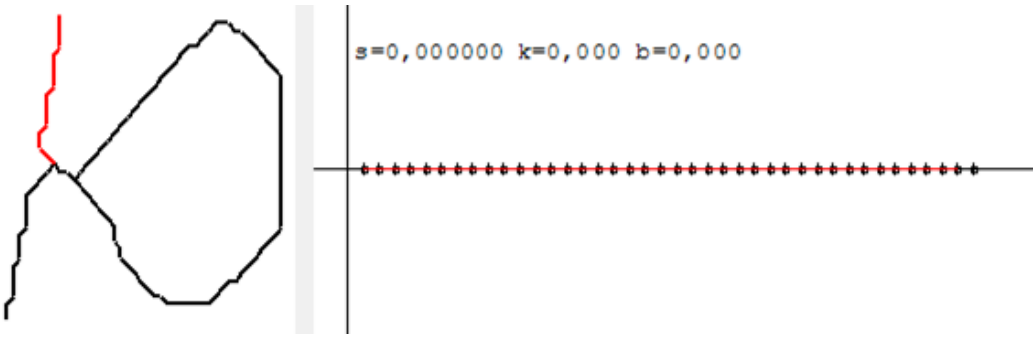


Рисунок 4.10 - График изменения первой производной прямого ребра

Аппроксимируем эти точки уравнением y = k*x+b и методом наименьших квадратов определим коэффициенты k и b. Для прямых линий коэффициент k близок к нулю, для дуг – существенно отличается от 0 (рис. 4.15).

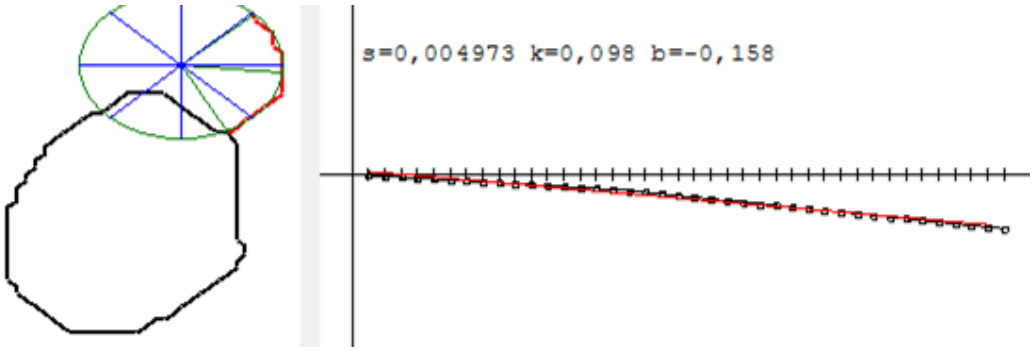


Рисунок 4.11 - График изменения первой производной дуги

На рис. 4.10 и 4.11 выводится также значение критерия – суммы квадратов отклонения S.

Некоторые дуги, например символов «П» и «Г», состоят из нескольких звеньев (рис. 4.12).

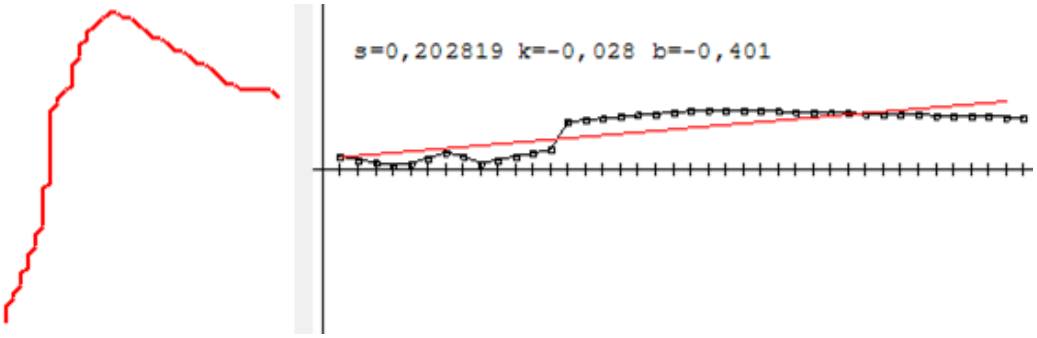


Рисунок 4.12 - График изменения первой производной из двух звеньев

Поэтому необходимо производить аппроксимацию двумя или тремя уравнениями вида y = k*x+b. Так, если производить аппроксимацию двумя уравнениями, то критерий S будет существенно меньше (рис. 4.13).

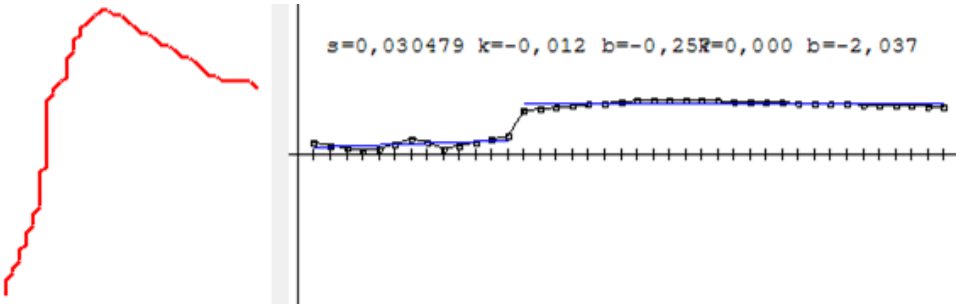


Рисунок 4.13 - График изменения первой производной из двух звеньев

Коэффициенты k_1 и k_2 – малы, а это означает, что дуга состоит из 2 прямых звеньев.

Для символа «*г*» аппроксимация двумя уравнениями дает лучший результат, но кривизны на разных участках имеют разные знаки. Разрыва первой производной при этом не происходит (рис. 4.14).

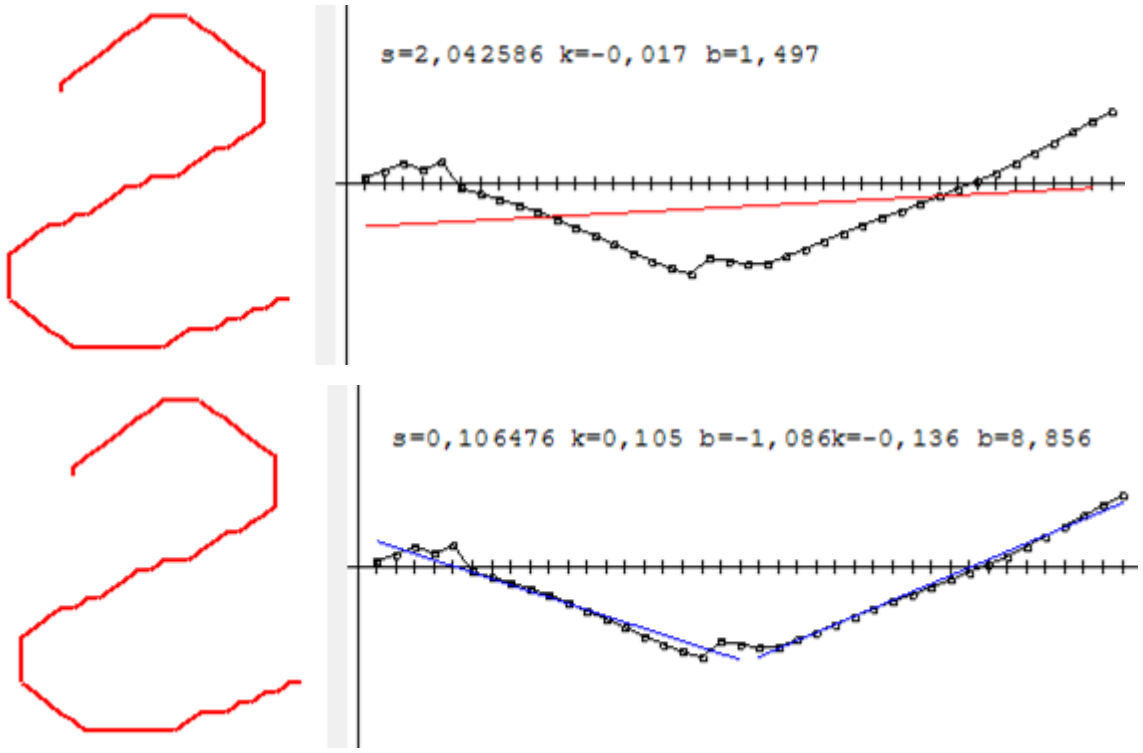
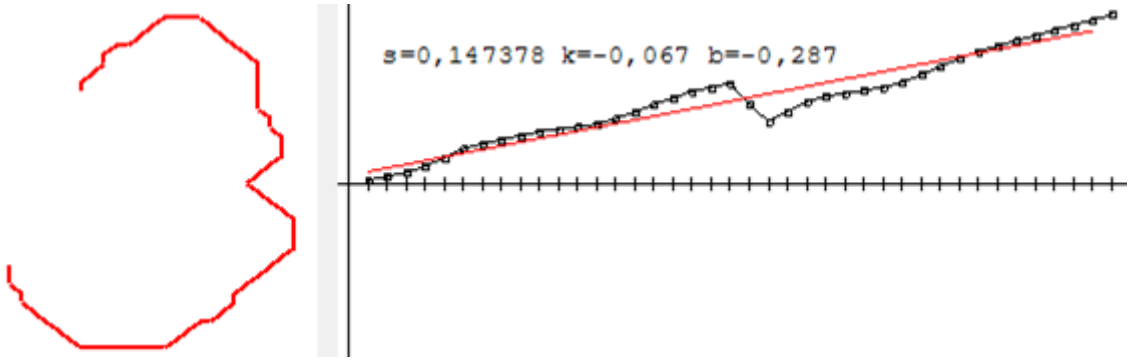


Рисунок 4.14 - График изменения первой производной из двух звеньев с кривизнами разных знаков

Для символа «*з*» аппроксимация двумя уравнениями также дает лучший результат, кривизны на разных участках совпадают. Первая производная терпит разрыв (рис. 4.15).



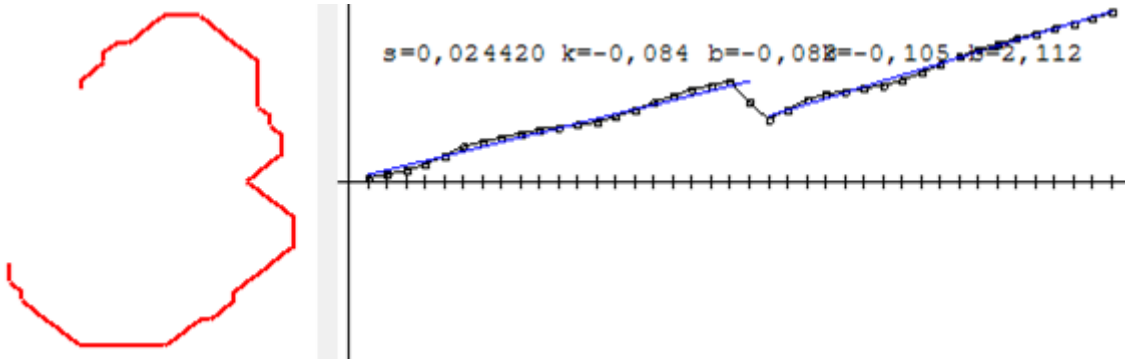


Рисунок 4.15 - График изменения первой производной из двух звеньев с разрывом первого рода

На основе анализа всего алфавита специальным методом (приложение 3) строится дерево 5-ти уровневое дерево, листьями которого являются графы символов (рис. 4.16).

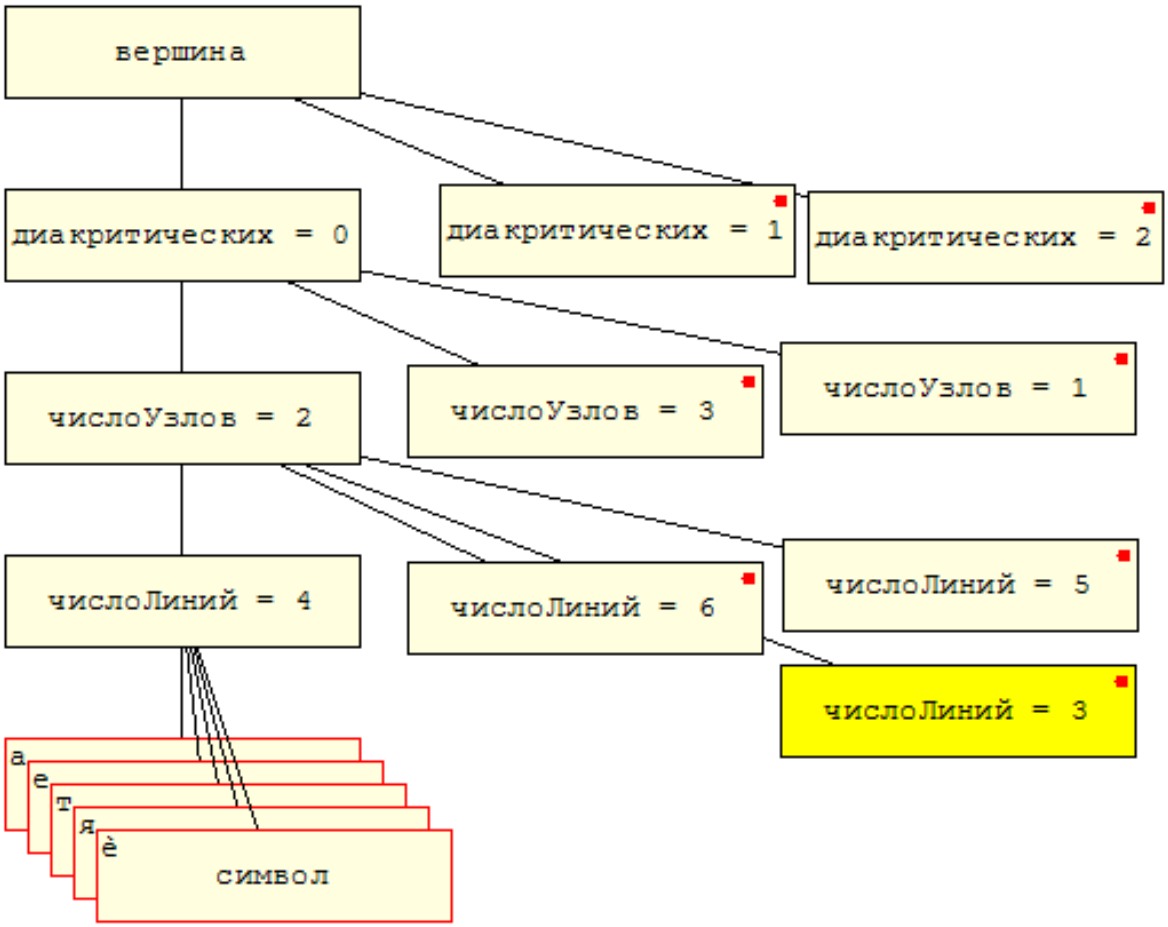


Рисунок 4.16 - Дерево графов символов

Первый уровень – вершина дерева, второй – узлы с различным числом диакритических знаков, третий – узлы с различным числом узлов, четвертый –

узлы с различным числом ребер, пятый уровень содержит листья с графами символов.

Алгоритм построения дерева заключается в следующем. Для каждого графа символа:

1. составляем массив ребер;
2. ищем диакритический узел;
3. если не найден, то создаем диакритический узел, иначе переходим на найденный;
4. если число диакритических знаков <> 0, то найти узел положения знаков (вверху, в центре, внизу) ;
5. если не найден, то создать узел положения знаков (вверху, в центре, внизу), иначе переходим на найденный узел;
6. ищем узел с подходящим числом узлов;
7. если не найден, то создаем узел с соответствующим числом узлов, иначе переходим на найденный узел;
8. ищем узел с подходящим числом ребер;
9. если не найден, то создаем узел с соответствующим числом ребер, иначе переходим на найденный узел;
10. создаем лист для графа символа;
11. для всех ребер графа символа создаем линию и определяем тип ребра, число узлов на концах ребра.

Теорема. Каждый лист дерева принятия решения, построенного по признакам 1-10, соответствует одному символу.

Для доказательства теоремы сформируем таблицу топологических признаков всех символов алфавита. На таблице, представленной ниже, показан фрагмент этой таблицы.

Таблица 4.2. Топологические признаки символов

	1	2	3	4	5	6	7	8	9	10
ё	2	Сверху	1	2	1 1	кольцо дуга	1 1			
й	1	Сверху	1	3	1 1 1	дуга дуга дуга	1 1 1			
ы	1	Справа	1	2	1 1	дуга кольцо	1 1			
о	0		1	1	1	кольцо	1			
с	0		1	1	1	дуга	1			
г	0		1	1	1	дуга	2	да		
л	0		1	1	1	дуга	2	нет	нет	
з	0		1	1	1	дуга	2	нет	есть	11000011
м	0		1	1	1	дуга	2	нет	есть	00001111
п	0		1	1	1	дуга	3	нет		
б	0		1	2	1 1	кольцо дуга	1 1			00001100
д	0		1	2	1 1	кольцо дуга	1 1			00000011
е	0		1	2	1 1	кольцо дуга	1 1			00110000
ъ	0		1	2	1 1	дуга кольцо	2 1	нет		
ь	0		1	2	1 1	дуга кольцо	1 1			00000000
и	0		1	3	1	дуга	1			

					1 1	дуга дуга	1 1			
э	0		1	3	1 1 1	дуга дуга дуга	1 1 1			00000011 00000000 11000000
у	0		1	3	1 1 1	дуга дуга дуга	1 1 1			00000000 00000000 11000000
ц	0		1	3	1 1 1	дуга дуга дуга	1 1 1			00110000 00000000 00000011
ч	0		1	3	1 1 1	дуга дуга дуга	1 1 1			00110000 00000000 00000000
х	0		1	4	1 1 1 1	дуга дуга дуга дуга	1 1 1 1			
в	0		2	3	2 2 2	дуга дуга дуга	1 1 1			
а	0		2	4	2 2 1 1	дуга дуга дуга дуга	1 1 1 1			01111110 00000000 00110000 00000000
р	0		2	4	2 2	дуга дуга	1 1			11100111 00000000

					1	дуга	1			00000000
					1	дуга	1			00000000
ю	0		2	4	2	дуга	1			
					1	кольцо	1			
					1	дуга	1			
					1	дуга	1			
я	0		2	4	2	дуга	1			
					2	дуга	1			
					1	дуга	1			
					1	дуга	1			
к	0		2	5	2	дуга	1			00000000
					1	дуга	1			00000000
					1	дуга	1			00000000
					1	дуга	1			00000011
					1	дуга	1			11000000
н	0		2	5	2	дуга	1			00000000
					1	дуга	1			00000000
					1	дуга	1			00000000
					1	дуга	1			00000000
					1	дуга	1			00000000
т	0		2	5	2	дуга	1			00001111
					1	дуга	1			00000000
					1	дуга	1			00000000
					1	дуга	1			00000000
					1	дуга	1			00001111
ш	0		2	5	2	дуга	1			11110000
					1	дуга	1			11110000
					1	дуга	1			00000000

					1	дуга	1			00000000
					1	дуга	1			00000000
щ	0		2	5	2	дуга	1			11110000
					1	дуга	1			11110000
					1	дуга	1			00000000
					1	дуга	1			00000000
					1	дуга	1			00000011
ф	0		2	5	2	дуга	1			
					2	дуга	1			
					2	дуга	1			
					1	дуга	1			
					1	дуга	1			
ж	0		3	8	2	дуга	1			
					2	дуга	1			
					1	дуга	1			
					1	дуга	1			
					1	дуга	1			
					1	дуга	1			
					1	дуга	1			
					1	дуга	1			

Эти же 10 признаков разрешают коллизии в остальных группах символов.

Обратное утверждение не верно. Одному символу может соответствовать несколько листьев в дереве принятия решения.

Например, арабские символы  и  имеют различное число ребер.

Для любого листа дерева (графа символа) может быть вызвано окно топологических свойств (рис. 4.17).

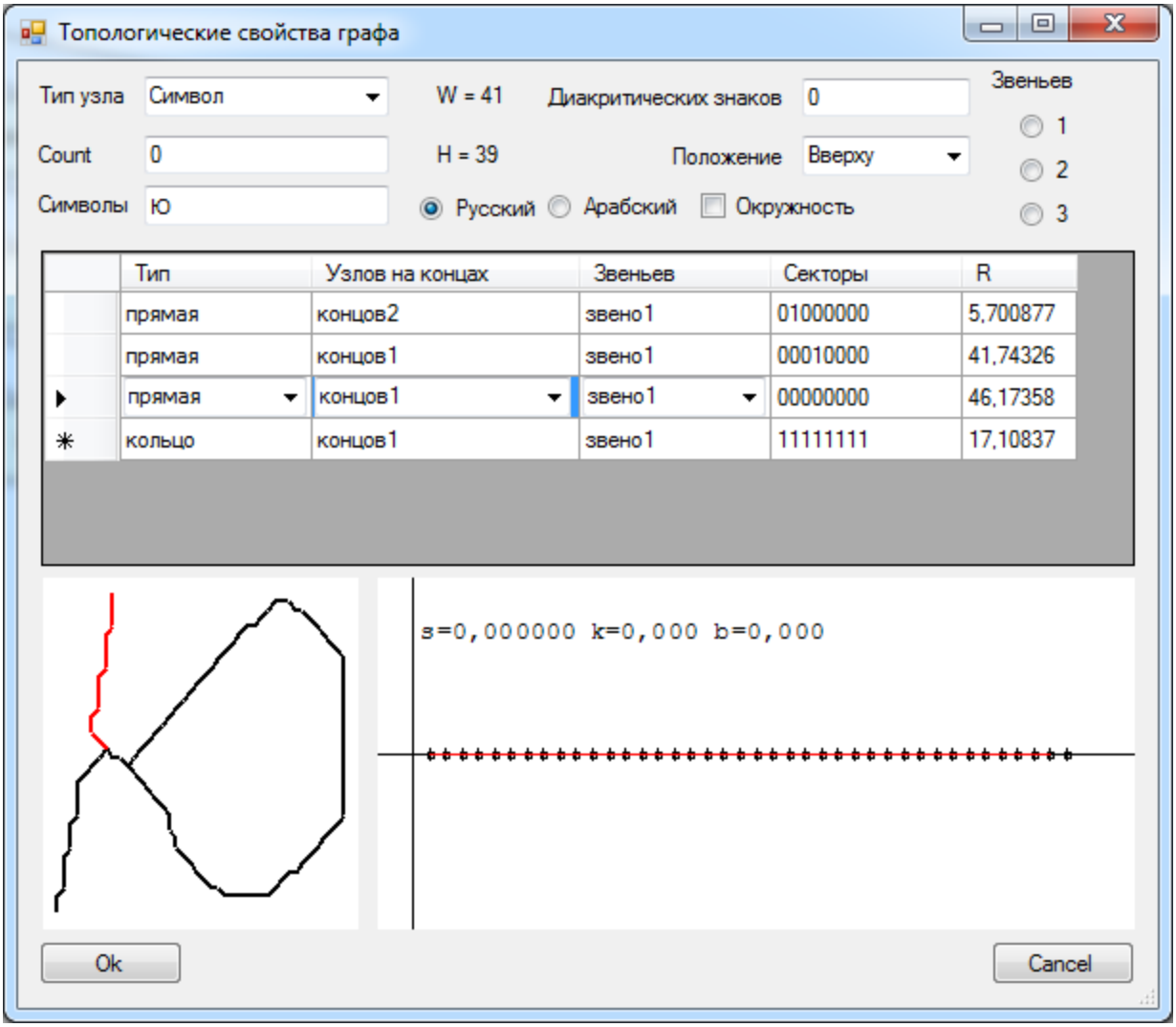


Рисунок 4.17 -. Окно топологических свойств графа

В этом окне отображаются следующие свойства для каждого ребра:

- характер ребра: прямая, дуга или кольцо;
- количество узлов на концах ребер: 1 или 2;
- количество звеньев. У символа «Г» два прямых звена, у символа «П» три прямых звена;
- для дуги определяются секторы круга, которые она накрывает;
- радиус окружности дуги;
- характер изменения кривизны.

В качестве теста программе было предложено распознать строку, представленную на рис. 4.18.

Рисунок 4.18 – Распознаваемая строка

Результат распознавания представлен на рис. 4.19.

Рисунок 4.19 – Распознанная строка

Заключение

1. Разработан модифицированный алгоритм выделения контура методом жука.
2. Разработаны методы численной оценки качества выделения границ на изображениях. Проведено сравнение быстродействия и качество методов выделения границ.
3. Разработан модифицированный алгоритм скелетизации изображения символов.
4. Разработан модифицированный волновой алгоритм.
5. Разработаны методы построения и упрощения графа изображения символов.
6. Разработана структура дерева принятия решений для классификации символов, метод построения дерева принятия решений для классификации символов и метод поиска символов в дереве принятия решений для классификации символов.
7. Доказана теорема: Каждый лист дерева принятия решения, построенного по 10 топологическим признакам, соответствует одному символу.

Литература

1. Canny J.F. Finding edges and lines in images. Master's thesis. / J.F. Canny // MIT, Cambridge, USA, 1983, pp.50–67.
2. Dengsheng Zhang. A Comparative Study on Shape Retrieval Using Fourier Descriptors with Different Shape Signatures / Dengsheng Zhang , Lu Guojun // Journal of Visual Communication and Image Representation. Издатель: Academic Press, 2003. No. 14 (1). P. 41-60
3. Dori D. Sparse Pixel Vectorization: An Algorithm and Its Performance Evaluation / D. Dori, W. Liu // IEEE Pattern Analysis and Machine Intelligence. — 1999. — 21, N 3. — P. 202–215.
4. Green B. Canny Edge Detection Tutorial. URL: http: / B. Green // dasl.mem.drexel.edu/ alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html. (дата обращения 29.05.2016).
5. Gonzaga, A. Method to Evaluate the Performance of Edge Detector / A. Gonzaga // The XXIInd Brazilian Symposium on Computer Graphics and Image Processing – 2009 – P. 87–91.
6. Heikkila M. A texture-based method for modeling the background and detecting moving objects. / , M. Pietikainen // IEEE Transactions on Pattern Analysis and Machine Intelligence, 2006, vol. 28, no. 4, pp. 657–662. doi: 10.1109/TPAMI.2006.68.
7. Janssen R.D.T. Adaptative vectorization of line drawing images / R.D.T. Janssen, A.M. Vossepoel // Computer vision and image understanding. — 1997. — 65. — P. 38–56.
8. Martin, D. A database of human segmented natural images and its application to evaluating algorithms and measuring ecological statistics / D. Martin, C. Fowlkes, D. Tal, J. Malik // 8th International Conference Computer Vision – 2001 – Vol. 2 – P. 416–423.

9. Song J., Su F., Chen J., Tai C., Cai S. Line Net Global Vectorization: an Algorithm and Its Performance Evaluation / J. Song, F. Su, J. Chen, C. Tai, S. Cai // Computer Vision and Pattern Recognition 2000. — 2000. — P. 1383–1388.
10. Toussaint G. Introduction to pattern recognition / G. Toussaint – Режим доступа: http://cgm.cs.mcgill.ca/~godfried/teaching/mir-reading-assignments/Chapter-1-Introduction-Pattern-Recognition.pdf
11. Toussaint G. Grids, connectivity and contour tracing / G. Toussaint – Режим доступа: http://cgm.cs.mcgill.ca/~godfried/teaching/mir-reading-assignments/Chapter-2-Grids-Connectivity-Contour-Tracing.pdf
12. Wenyin L., Dori D. From Raster to Vectors: Extracting Visual Information from Line Drawings / L. Wenyin, D. Dori // Pattern Analysis and Applications. — 1999. — 2, N 1. — P. 10–21.
13. Абламейко С. В. Обработка изображений: технология, методы, применение. Учеб. пособие/ С. В. Абламейко, Д. М. Лагуновский – Москва: Амалфей, 2000. – 304 с.
14. Андреев А.Ю. Сегментация символов в изображении модифицированным методом жука / А.Ю. Андреев, С.П. Бобков // «Современные наукоемкие технологии. Региональное приложение» №1 (37) 2014.
15. Анисимов Б.В. Распознавание и цифровая обработка изображений: Учеб. Пособие / Б.В. Анисимов, В.Д. Курганов, В. К. Злобин – Москва: Высш. школа, 1983. – 295 с.
16. Гонсалес, Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс – Москва: Техносфера, 2005 – 1070 с.
17. Денисов И. Эффективный алгоритм построения остова растрового изображения / И. Денисов, Е. Кузьмин // Доклад на конференции Графикон-99
18. Дуда Р. Распознавание образов и анализ сцен / Р. Дуда, П.М. Харт — Москва: Мир, 1976. — 507 с.

19. Ковтун А.А. Метод разреженного фронта для векторизации линейчатых изображений / А.А. Ковтун // ISSN 1681–6048 System Research & Information Technologies, — 2014, № 1 — С. 130-141
20. Колючкин В.Я. Алгоритмы контурной сегментации и распознавания образов объектов систем технического зрения / В.Я. Колючкин, К.М. Нгуен // Электронное научно-техническое издание «Наука и образование». Москва, 2013 №4. DOI:10.7463/0413.0548084. URL: http://technomag.stack.net/doc/548084.html.
21. Кормановский С.И. Структурно-связностная модель изображения: выделение контура и формирование признаков / С.И. Кормановский, Я.Г. Скорюкова, О. П. Мельник // Информационные технологии и компьютерная техника. 2010. № 1.
22. Котович Н.В. Распознавание скелетных образов / Н.В. Котович, О.А. Славин http://ocrai.narod.ru/skeletrecognize.html
23. Лебедько Д.С. Методы численной оценки качества выделения границ на изображениях /Лебедько Д.С., Тюкачѐв Н.А., Мохаммед Заки Хассан // Вестник Воронежского государственного университета. Сер. Системный анализ и информационные технологии. - 2015. -№ 4. С. 73-77
24. Местецкий Л.М. Непрерывный скелет бинарного изображения / Л.М. Местецкий – Доклад на конференции Графикон-99
25. Москаленко С.В., Гатчин Ю.А. Помехоустойчивый волновой алгоритм векторизации линейных растровых объектов / С.В. Москаленко, Ю.А. Гатчин – http://stanislavmoskalenko.narod.ru/articles/article_mashin3.htm.
26. Мохаммед Заки Хасан Сравнение быстродействия алгоритмов выделения контуров / Мохаммед Заки Хасан, Н.А. Тюкачев // Информатика: проблемы, методология, технологии. Материалы XV международной научно-методической конференции: в 4-х томах. - 2015 – Т.6 -Т.1. С. 53-57.
27. Мохаммед Заки Хассан О волновом алгоритме построения графа изображения для распознавании рукописных символов / Мохаммед Заки

Хассан // Российский научно-технический журнал «Мониторинг. Наука и технологии» (ISSN печатной версии 2076-7358) - №1. – 2018.

28. Мохаммед Заки Хассан. Модификация алгоритма скелетизации зонга-суня для задачи распознавания рукописных символов / Мохаммед Заки Хассан // Моделирование, оптимизация и информационные технологии. Научный журнал, - Т. 6, № 1 http://moit.vivt.ru/
29. Мохаммед Заки Хассан. Классификация символов с помощью топологических свойств графа / Мохаммед Заки Хассан, Тюкачев Н.А.// Информатика: проблемы, методология, технологии. Материалы XVI международной научно-методической конференции: в 6-х томах. - 2016 – Т.6 -Т.1. С. 101-104.
30. Мохаммед Заки Хассан. Модификация волнового алгоритма выделения линий изображения / Мохаммед Заки Хассан, Тюкачев Н.А. // Информатика: проблемы, методология, технологии. Материалы XVI международной научно-методической конференции: в 6-х томах. - 2016 – Т.6 -Т.1. С. 171-177.
31. Мохаммед Заки Хассан. Модифицированный метод жука для выделения контура изображения / Мохаммед Заки Хассан, Тюкачев Н.А. // Информатика: проблемы, методология, технологии. Материалы XVI международной научно-методической конференции: в 6-х томах. - 2016 – Т.6 -Т.1. С. 97-101.
32. Мохаммед Заки Хассан. Об алгоритмах анализа рукописных символов / Мохаммед Заки Хассан, Тюкачев Н.А. // Сборник материалов XVII международной научно-практической конференции в 5 т. /под редакцией Н.А. Тюкачева, А.А. Крыловецкого; Воронеж, Воронежский государственный университет, 9-10 февраля 2017 г. – Воронеж, Издательство «Научно-исследовательские публикации» (ООО «Велборн»). – 2017, - Т.4, - 353 с. – С. 58-68.
33. Мохаммед Заки Хассан. Сравнение алгоритмов уточнение линий / Мохаммед Заки Хассан, Тюкачев Н.А. // Сборник материалов XVII международной научно-практической конференции в 5 т. /под редакцией Н.А. Тюкачева, А.А. Крыловецкого; Воронеж, Воронежский государственный университет, 9-10

февраля 2017 г. – Воронеж, Издательство «Научно-исследовательские публикации» (ООО «Велборн»). – 2017, - Т.4, - 353 с. – С. 49-58.

34. Мохаммед Заки Хассан. Сравнение быстродействия алгоритмов выделения контуров / Мохаммед Заки Хассан, Тюкачев Н.А. // Информатика: проблемы, методология, технологии. Материалы XV международной научно-методической конференции: в 4-х томах. – 2015. – Т.4 -Т.1. С. 53-57.
35. Мохаммед Заки Хассан. Модификация волнового алгоритма выделения линий изображения / Мохаммед Заки Хассан, Тюкачев Н.А. // Информатика: проблемы, методология, технологии. Материалы XVI международной научно-методической конференции: в 6-х томах. – 2016. – Т.1. С. 171-177.
36. Помехоустойчивый волновой алгоритм векторизации линейных растровых объектов. — http://stanislavmoskalenko.narod.ru/articles/article_mashin3.htm.
37. Прэтт, У. Цифровая обработка изображений: Пер. с англ. / У. Прэтт — Москва: Мир, 1982. — Кн. 2 — 480 с.
38. Себастиан Г. Процессы принятия решения при распознавании образов / Г. Себастиан // – Киев, 1965 г., 151 с.
39. Сирота А.А. Статистические алгоритмы обнаружения границ объектов на изображениях / А.А. Сирота, А.И. Соломатин – Вестник ВГУ. Сер. Системный анализ и информационные технологии, 2008, № 1, с. 58–64.
40. Сойфер В.А. Методы компьютерной обработки изображений / В.А. Сойфер – Москва: Физматлит, 2003. – 784 с.
41. Сойфер В.А.Теоретические основы цифровой обработки изображений / В.А. Сойфер, В.В. Сергеев, С.Б. Попов, В.В. Мясников – Самара: Самарский Государственный Аэрокосмический Университет им академика С.П. Королёва, 2000 – 255 с.
42. Сорокин А.И. Разработка алгоритмов распознавания рукописных символов на основе аналитических свойств изображения: Автореферат дисс. кандидата физико-математических наук / Сорокин А.И. – Воронеж, 2010. 16 с.

43. Тюкачев Н.А. / Н.А. Тюкачев, Мохаммед Заки Хассан М.Н. Мохаммед Заки – Патент программу ЭВМ Распознавания рукописных символов. Заявка № 2018614806 от 14.05.18
44. Фисенко В. Т., Фисенко Т. Ю. Компьютерные обработки и распознавание изображений: Учеб. Пособие / В.Т. Фисенко, Т.Ю. Фисенко – СПб: СПбГУ ИТМО, 2008. – 192 с.
45. Фурман Я. А. Цифровые методы обработки и распознавания бинарных изображений / Я.А. Фурман – Красноярск: Изд-во Красноярского ун-та, 1992. – 248 с.
46. Чудовская А.К. Возможности распараллеливания алгоритмов выделения контура по технологии CUDA / А.К. Чудовская // Сб. докл. IV Межд. науч.-практич. конф. «Современная информационная Украина: информатика, экономика, философия». – Донецк, 2010, с. 67–70.
47. Яне Б. Цифровая обработка изображений (Перев. с англ.) / Б. Яне. – Москва: Техносфера, 2007. – 584 с.

Приложение 1. Шаблонная скелетизация

Процедура скелетизации, на входе список списков (после бинаризации)

```
public void Скелетизация()
        {
            if (C.twoImage != null)
            {
                bImg = new byte[C.twoImage.Width,
C.twoImage.Height];
                for (int i = 0; i < C.twoImage.Width; i++)
                    for (int j = 0; j < C.twoImage.Height; j++)
                    {
                        Color c = C.twoImage.GetPixel(i, j);
                        byte R = c.R;
                        byte G = c.G;
                        byte B = c.B;
                        double p = R * 0.3 + G * 0.59 + B * 0.11;
                        if (p > 180)
                            bImg[i, j] = 1;
                        else
                            bImg[i, j] = 0;
                    }

                int w = bImg.GetLength(0);
                int h = bImg.GetLength(1);
                int count = 1;
                while (count != 0)
          //повторять пока удалялся хотя бы один пиксель
                {
                    count = delete(w, h);
                    if (count != 0)
                        delete2(w, h);
                }

                for (int i = 0; i < C.twoImage.Width; i++)
```

```
                    for (int j = 0; j < C.twoImage.Height; j++)
                        if (bImg[i, j] == 0)
                            C.twoImage.SetPixel(i, j, Color.Black);
                        else
                            C.twoImage.SetPixel(i, j, Color.White);
            }
        }
```

```
    // удаление пикселя по основному набору, возврат кол-ва
удаленных

void delete2(int w, int h)        {
            for (int i = 1; i < h - 1; i++)
                for (int j = 1; j < w - 1; j++)
                    if (bImg[j, i] == 0)
                        if (deletable2(j, i))
                            bImg[j, i] = 1;
        }
```

```
    // получение 3*3, передача на проверку для осн.

bool deletable(int x, int y)        {
            byte[] a = new byte[9];
            for (int i = y - 1; i < y + 2; i++)
                for (int j = x - 1; j < x + 2; j++)
                    a[3 * (i - y + 1) + (j - x + 1)] = bImg[j, i];
            return check(a);
        }
```

```
    удаление пикселя по основному набору, возврат кол-ва удаленных

byte delete(int w, int h)        {
            byte count = 0;
            for (int i = 1; i < h - 1; i++)
                for (int j = 1; j < w - 1; j++)
                    if (bImg[j, i] == 0)
```

```
                    if (deletable(j, i))
                    {
                        bImg[j, i] = 1;
                        count += 1;
                    }
            return count;
        }
```

```
bool Eqv(byte[] a, byte[] b)
        {
            bool ok = true;
            int i = -1;
            while ((i < a.Length - 1) && ok)
            {
                i++;
                ok = a[i] == b[i];
            }
            return ok;
        }
```

Определение принадлежности 3*3 к основным шаблонам

```
        bool check(byte[] a)
        {
            byte[] t123457 = { 1, 1, 0, 0, 1, 0 };
            byte[] t013457 = { 1, 1, 1, 0, 0, 0 };
            byte[] t134567 = { 0, 1, 0, 0, 1, 1 };
            byte[] t134578 = { 0, 0, 0, 1, 1, 1 };
            byte[] t0123457 = { 1, 1, 1, 0, 0, 0, 0 };
            byte[] t0134567 = { 1, 0, 1, 0, 0, 1, 0 };
            byte[] t1345678 = { 0, 0, 0, 0, 1, 1, 1 };
            byte[] t1234578 = { 0, 1, 0, 0, 1, 0, 1 };

            bool ok = false;

            byte[] t = { a[1], a[2], a[3], a[4], a[5], a[7] };
```

```
            if (Eqv(t, t123457))
                ok = true;
            else
            {
                byte[] t1 = { a[0], a[1], a[3], a[4], a[5], a[7] };
                if (Eqv(t1, t013457))
                    ok = true;
                else
                {
                    byte[] t2 = { a[1], a[3], a[4], a[5], a[6], a[7]
};
                    if (Eqv(t2, t134567))
                        ok = true;
                    else
                    {
                        byte[] t3 = { a[1], a[3], a[4], a[5], a[7],
a[8] };
                        if (Eqv(t3, t134578))
                            ok = true;
                        else
                        {
                            byte[] t4 = { a[0], a[1], a[2], a[3],
a[4], a[5], a[7] };
                            if (Eqv(t4, t0123457))
                                ok = true;
                            else
                            {
                                byte[] t5 = { a[1], a[3], a[4],
a[5], a[6], a[7], a[8] };
                                if (Eqv(t5, t1345678))
                                    ok = true;
                                else
                                {
```

```
                                    byte[] t6 = { a[0], a[1], a[3],
a[4], a[5], a[6], a[7] };
                                    if (Eqv(t6, t0134567))
                                        ok = true;
                                    else
                                    {
                                        byte[] t7 = { a[1], a[2],
a[3], a[4], a[5], a[7], a[8] };
                                        if (Eqv(t7, t1234578))
                                            ok = true;
                                    }
                                }
                            }
                        }
                    }
                }
                return ok;
        }
```

```
    //определение принадлежности 3*3 к шумам
bool fringe(byte[] a)
        {
            byte[][] t = new byte[13][]
            {
                new byte[] {1,1,1,1,0,1,1,1,1},

                new byte[] {1,1,1,1,0,1,1,0,0},
                new byte[] {1,1,1,0,0,1,0,1,1},
                new byte[] {0,0,1,1,0,1,1,1,1},
                new byte[] {1,1,0,1,0,0,1,1,1},

                new byte[] {1,1,1,1,0,1,0,0,1},
                new byte[] {0,1,1,0,0,1,1,1,1},
```

```
            new byte[] {1,0,0,1,0,1,1,1,1},
            new byte[] {1,1,1,1,0,0,1,1,0},

            new byte[] {1,1,1,1,0,1,0,0,0},
            new byte[] {0,1,1,0,0,1,0,1,1},
            new byte[] {0,0,0,1,0,1,1,1,1},
            new byte[] {1,1,0,1,0,0,1,1,0}
        };
        bool ok = false;
        int i = -1;
        while ((i < t.GetLength(0) - 1) && !ok)
            ok = Eqv(a, t[++i]);
        return ok;
    }
```

Приложение 2. Упрощение графа

```
    Удалить узлы с короткими ребрами.
// удалить узлы с короткими реброми
public static void удалитьУзлыСкороткимиРебрами()
{
    for (int k = 0; k < вершины.Count; k++)
    {
        int j = 0;
        while (j <= вершины[k].Узлы.Count - 1)
        {
            Узел узел = вершины[k].Узлы[j];

            // удалить короткие, которые начинаются в узле
            if (узел.Ребра.Count >= 2)
            {
                int i = 0;
                while (i < узел.Ребра.Count - 0)
                {
                    if      ((узел.Ребра[i].ТочкиСкелета.Count      <
короткоеРебро) &&
                        ((узел.Ребра[i].КонечныйУзел == null))
                        )
                        узел.Ребра.RemoveAt(i);
                    else
                        i++;
                }
                if (узел.Ребра.Count == 0)
                    вершины[k].Узлы.Remove(узел);
                else
                    j++;
            }

            // удалить узел, у которого одно короткое ребро
```

```
            if ((узел.Ребра.Count == 1) &&
                (узел.Ребра[0].ТочкиСкелета.Count < короткоеРебро))
            {
                //for (int j1 = 0; j1 < вершины[k].Узлы.Count; j1++)
                int j1 = 0;
                while (j1 < вершины[k].Узлы.Count)
                {

if((j!=j1)&&(вершины[k].Узлы[j1].Ребра.IndexOf(узел.Ребра[0])!=-1)
                        && (вершины[k].Узлы[j1].Ребра.Count != 1))

вершины[k].Узлы[j1].Ребра.Remove(узел.Ребра[0]);
                    else
                        j1++;
                }
                вершины[k].Узлы.Remove(узел);
            }
            else
                j++;
        }
    }
}


    Найти узлы близкие к концам ребер.
public static void удалитьУзлыСкороткимиРебрами1()
{
    for (int k = 0; k < вершины.Count; k++)
    {
        int j = 0;
        while (j <= вершины[k].Узлы.Count - 1)
        {
            Узел узел = вершины[k].Узлы[j];
```

```
            // удалить узел, у которого одно короткое ребро
            if        ((узел.Ребра.Count        ==        1)        &&
(узел.Ребра[0].ТочкиСкелета.Count == 1))
            {
                for (int j1 = 0; j1 < вершины[k].Узлы.Count; j1++)
                    if                      ((j                     !=
j1)&&(вершины[k].Узлы[j1].Ребра.IndexOf(узел.Ребра[0])!=-1))

вершины[k].Узлы[j1].Ребра.Remove(узел.Ребра[0]);
                вершины[k].Узлы.Remove(узел);
            }
            else
                j++;
        }
    }
}
```

Удалить узлы с одним ребром.

```
public static void удалитьУзлы_с_1_ребром()
{
    for (int k = 0; k < вершины.Count; k++)
    {
        int j = 0;
        Вершины w = вершины[k/*Ind[k]*/];
        while (j < w.Узлы.Count - 0)
            if       ((w.Узлы[j].Ребра.Count       ==       1)       &&
(w.Узлы[j].Ребра[0].КонечныйУзел != null))
            {
                Узел u = w.Узлы[j].Ребра[0].КонечныйУзел;

                //if  (u.Ребра.IndexOf(w.Узлы[j].Ребра[0])  ==  -1)  //
ребра нет
                {
```

```
                    w.Узлы[j].Ребра[0].НачальныйУзел = null;
                    u.Ребра.Add(w.Узлы[j].Ребра[0]);
                }
                if (!w.Узлы[j].Ребра[0].замкнуто)
                    w.Узлы.Remove(w.Узлы[j]);
                else
                    j++;
            }
            else
                j++;
    }
}
```

Удалить узлы с двумя ребрами.

```
// удалить узлы с двумя ребрами
public static void удалитьУзлыСдвумяРебрами()
{
    for (int k = 0; k < вершины.Count; k++)
    {
        int j = 0;
        Вершины w = вершины[k];
        while (j < w.Узлы.Count)
        //for (int j = 0; j < вершины[k].узлы.Count; j++)
        {
            Узел узел = w.Узлы[j];

            bool ok1 = (узел.Ребра.Count == 2);
            if (ok1) // одно из ребер замкнуто
                ok1        =        !(узел.Ребра[0].замкнуто        |
узел.Ребра[1].замкнуто);

            if (ok1)
            {
```

```
                bool    ok    =    (узел.Ребра[0].НачальныйУзел    ==
узел.Ребра[0].КонечныйУзел) |
                    (узел.Ребра[1].НачальныйУзел                    ==
узел.Ребра[1].КонечныйУзел);
                if (!ok)
                {
                    bool ok2 = true;
                    int numA, numB;
                    if (узел.Ребра[0].КонечныйУзел != null)
                        numA = узел.Ребра[0].КонечныйУзел.num;
                    else
                        numA = -1;
                    if (узел.Ребра[1].КонечныйУзел != null)
                        numB = узел.Ребра[1].КонечныйУзел.num;
                    else
                        numB = -1;
                    int numC = узел.num;

                    Ребро A = узел.Ребра[0], B = узел.Ребра[1];
                    if ((numA == numC) & (numB == numC))  // 1   ---
> C <---  = ------>
                    {
                        for (int i = B.ТочкиСкелета.Count - 1; i >
0; i--)
                            A.ТочкиСкелета.Add(B.ТочкиСкелета[i]);
                        A.КонечныйУзел = B.НачальныйУзел;
                    }

                    if ((numA == numC) & (numB != numC)) // 2   --->
C --->  = ------>
                    {
                        for (int i = 0; i < B.ТочкиСкелета.Count;
i++)
                            A.ТочкиСкелета.Add(B.ТочкиСкелета[i]);
```

```
                    A.КонечныйУзел = B.КонечныйУзел;
                }

                if ((numA != numC) & (numB == numC)) // 3   <---
C <---  = <------
                {
                    for (int i = 0; i < A.ТочкиСкелета.Count;
i++)
                        B.ТочкиСкелета.Add(A.ТочкиСкелета[i]);
                    A.ТочкиСкелета = B.ТочкиСкелета;
                    // в A.КонечныйУзел найти A и заменить на B

                    if (A.КонечныйУзел != null)
                    {
                        int n = A.КонечныйУзел.Ребра.IndexOf(A);
                        if (n != -1)
                            A.КонечныйУзел.Ребра[n] = B;
                    }

                    B.КонечныйУзел = A.КонечныйУзел;
                }

                if ((numA != numC) & (numB != numC)) // 4   <---
C --->  = ------>
                {
                    PointF tmp;

                    int L, n;

                    //if (())

                    if    ((B.КонечныйУзел    !=    null)    &&
(A.КонечныйУзел != null)) // u ----> u
                    {
```

```
                            L = A.ТочкиСкелета.Count;
                            for (int i = 0; i < L / 2; i++)
                            {
                                tmp = A.ТочкиСкелета[i];
                                A.ТочкиСкелета[i] = A.ТочкиСкелета[L
- i - 1];
                                A.ТочкиСкелета[L - i - 1] = tmp;
                            }

                            for (int i = 0; i <
B.ТочкиСкелета.Count; i++)

A.ТочкиСкелета.Add(B.ТочкиСкелета[i]);

                            n = B.КонечныйУзел.Ребра.IndexOf(B);
                            if (n != -1)
                                B.КонечныйУзел.Ребра[n] = A;

                            A.КонечныйУзел = B.КонечныйУзел;
                        }
                        else
                            if ((B.КонечныйУзел == null) &&
(A.КонечныйУзел != null)) // u ----> 0
                            {
                                L = A.ТочкиСкелета.Count;
                                for (int i = 0; i < L / 2; i++)
                                {
                                    tmp = A.ТочкиСкелета[i];
                                    A.ТочкиСкелета[i] =
A.ТочкиСкелета[L - i - 1];
                                    A.ТочкиСкелета[L - i - 1] = tmp;
                                }
```

```
                                for (int i = 0; i <
B.ТочкиСкелета.Count; i++)

A.ТочкиСкелета.Add(B.ТочкиСкелета[i]);

                                A.КонечныйУзел = null;
                            }
                            else
                                if
((B.КонечныйУзел!=null)&&(A.КонечныйУзел==null)) //0-> u
                                {
                                    L = B.ТочкиСкелета.Count;
                                    for (int i = 0; i < L / 2; i++)
                                    {
                                        tmp = B.ТочкиСкелета[i];
                                        B.ТочкиСкелета[i]          =
B.ТочкиСкелета[L - i - 1];
                                        B.ТочкиСкелета[L - i - 1] =
tmp;
                                    }

                                    for (int i = 0; i <
A.ТочкиСкелета.Count; i++)

B.ТочкиСкелета.Add(A.ТочкиСкелета[i]);


                                    //n                          =
B.КонечныйУзел.Ребра.IndexOf(B);
                                    //if (n != -1)
                                    //     B.КонечныйУзел.Ребра[n] =
B;
                                    B.НачальныйУзел                =
B.КонечныйУзел;
```

```
                                    B.КонечныйУзел = null;
                                }
                                else
                                    if ((B.КонечныйУзел == null) &&
                                        (A.КонечныйУзел == null)) //
0 ----> 0
                                    {
                                        L = A.ТочкиСкелета.Count;
                                        for (int i = 0; i < L / 2;
i++)
                                        {
                                            tmp = A.ТочкиСкелета[i];
                                            A.ТочкиСкелета[i]       =
A.ТочкиСкелета[L - i - 1];
                                            A.ТочкиСкелета[L  -  i  -
1] = tmp;
                                        }

                                        for   (int   i   =   0;   i   <
B.ТочкиСкелета.Count; i++)

A.ТочкиСкелета.Add(B.ТочкиСкелета[i]);

                                        A.КонечныйУзел = null;

A.НачальныйУзел.Ребра.Remove(B);
                                        A.НачальныйУзел.p           =
A.ТочкиСкелета[0];
                                        ok2 = false;
                                        j++;
                                    }
                        }
                        if (ok2)
                            вершины[k].Узлы.Remove(узел);
```

```
                }
                else
                    j++;
            }
            else
                j++;
        }
    }
}
```

Найти точки близкие к узлам.

```
public static void найтиТочкиБлизкиеКузлам()
{
    for (int k = 0; k < вершины.Count; k++)
    {
        for (int j = 0; j < вершины[k].Узлы.Count; j++)
            if (вершины[k].Узлы[j].Ребра.Count == 1)
            {
                // для всех узлов
                Узел узел = вершины[k].Узлы[j];
                //for (int k1 = 0; k1 < вершины.Count; k1++)
                for (int j1 = 0; j1 < вершины[k].Узлы.Count; j1++)
                {
                    int m = -1; bool ok = false;
                    while ((m < узел.Ребра.Count - 1) && !ok)
                    {
                        m++;
                        ok     =     узел.Ребра[m].КонечныйУзел     ==
вершины[k].Узлы[j1];
                    }
                    if (!ok & ((узел != вершины[k].Узлы[j1])))
                    {
                        //PointF p = узел.p;
```

```
                        for (int i1 = 0; i1 < вершины[k].Узлы[j1].Ребра.Count; i1++)
                        {
                            Ребро ребро = вершины[k].Узлы[j1].Ребра[i1];
                            double min = int.MaxValue;
                            int nMin = -1;
                            int kMin = -1;
                            int iMin = -1;
                            int jMin = -1;
                            for (int e = 0; e < ребро.ТочкиСкелета.Count; e++)
                            {
                                double d = Math.Abs(узел.p.X - ребро.ТочкиСкелета[e].X) +
                                    Math.Abs(узел.p.Y - ребро.ТочкиСкелета[e].Y);
                                if (d < min)
                                {
                                    nMin = e;  // точка
                                    //kMin = k1; // вершина
                                    min = d;
                                    iMin = i1; // ребро
                                    jMin = j1; // узел
                                }
                            }
                            if ((min > 0) && (min < Вершины.отКонцаРебраДоУзла))
                            {
                                // ребро - найденное
                                Ребро newРебро = new Ребро(узел);
                                for (int u = nMin; u < ребро.ТочкиСкелета.Count; u++)
```

```
newРебро.ТочкиСкелета.Add(ребро.ТочкиСкелета[u]);
                                ребро.ТочкиСкелета.RemoveRange(nMin
+ 1,
                                    ребро.ТочкиСкелета.Count  -  nMin
- 1);
                                newРебро.КонечныйУзел               =
ребро.КонечныйУзел;
                                ребро.КонечныйУзел = узел;
                                узел.Ребра.Add(ребро);
                                узел.Ребра.Add(newРебро);

вершины[k].Узлы[j].Ребра[0].ТочкиСкелета.Insert(0,
                                    ребро.ТочкиСкелета[nMin]);
                                вершины[k].Узлы[j].p                =
ребро.ТочкиСкелета[nMin];
                                return;

                            }
                        }
                    }
                }
            }
    }
}
```

Удалить узлы с одним ребром.

```
public static void удалитьУзлы_с_1_ребром()
{
    for (int k = 0; k < вершины.Count; k++)
    {
        int j = 0;
        Вершины w = вершины[k/*Ind[k]*/];
```

```
        while (j < w.Узлы.Count - 0)
            if       ((w.Узлы[j].Ребра.Count       ==       1)       &&
(w.Узлы[j].Ребра[0].КонечныйУзел != null))
            {
                Узел u = w.Узлы[j].Ребра[0].КонечныйУзел;

                //if  (u.Ребра.IndexOf(w.Узлы[j].Ребра[0])  ==  -1)  //
ребра нет
                {
                    w.Узлы[j].Ребра[0].НачальныйУзел = null;
                    u.Ребра.Add(w.Узлы[j].Ребра[0]);
                }
                if (!w.Узлы[j].Ребра[0].замкнуто)
                    w.Узлы.Remove(w.Узлы[j]);
                else
                    j++;
            }
            else
                j++;
    }
}
```

Найти точки близкие к концам.

```
public static void найтиТочкиБлизкиеКконцамРебер()
{
    for (int k = 0; k < вершины.Count; k++)
    {
        for (int j = 0; j < вершины[k].Узлы.Count; j++)
        {
            // для всех узлов
            Узел узел = вершины[k].Узлы[j];
            for (int i = 0; i < узел.Ребра.Count; i++)
                if (узел.Ребра[i].КонечныйУзел == null)
```

```
                {
                    int L = узел.Ребра[i].ТочкиСкелета.Count;
                    PointF              последняяТочка              =
узел.Ребра[i].ТочкиСкелета[L - 1];

                    for  (int  j1  =  0;  j1  <  вершины[k].Узлы.Count;
j1++)
                        if (узел != вершины[k].Узлы[j1])
                        {
                            for    (int    i1    =    0;    i1    <
вершины[k].Узлы[j1].Ребра.Count; i1++)
                            {
                                Ребро             ребро             =
вершины[k].Узлы[j1].Ребра[i1];
                                double min = int.MaxValue;
                                int nMin = -1;
                                for    (int    e    =    0;    e    <
ребро.ТочкиСкелета.Count; e++)
                                {
                                    double             d             =
Math.Abs(последняяТочка.X –
                                        ребро.ТочкиСкелета[e].X) +
                                        Math.Abs(последняяТочка.Y –
                                        ребро.ТочкиСкелета[e].Y);
                                    if (d < min)
                                    {
                                        nMin = e;
                                        min = d;
                                    }
                                }
                                if               (min               <
Вершины.отКонцаРебраДоУзла)
                                {
                                    Узел узелNew =
```

```
                                        new                 Узел(null,
ребро.ТочкиСкелета[nMin]);
                                    Ребро     newРебро     =     new
Ребро(узелNew);
                                    newРебро.КонечныйУзел          =
ребро.КонечныйУзел;
                                    ребро.КонечныйУзел = узелNew;
                                    for   (int   u   =   nMin;   u   <
ребро.ТочкиСкелета.Count; u++)

newРебро.ТочкиСкелета.Add(ребро.ТочкиСкелета[u]);

ребро.ТочкиСкелета.RemoveRange(nMin + 1,
                                       ребро.ТочкиСкелета.Count     -
nMin - 1);
                                    узелNew.Ребра.Add(ребро);
                                    узелNew.Ребра.Add(newРебро);

узелNew.Ребра.Add(узел.Ребра[i]);
                                    узел.Ребра[i].КонечныйУзел     =
узелNew;
                                    узел.Ребра[i].ТочкиСкелета.

Add(ребро.ТочкиСкелета[nMin]);
                                    вершины[k].Узлы.Add(узелNew);
                                    return;
                                }
                            }
                        }
                    }

        }
    }
}
```

Удалить одинаковые ребра.

Приложение 3. Построение графа

```
public static void AddTreeNew()
{
    NodeC p;
    int i, j;
    bool ok;
    List<Ребро> R = new List<Ребро>();

    Tree.top = new NodeC(200, 80, null);
    Tree.top.typeNode = TypeNode.вершина;

    for (int k = 0; k < вершины.Count; k++)
    {
        char ch;
        if (k <= 25)
            ch = (char)(1072 + k);
        else
            if (k == 26)
                ch = '|';
            else
                ch = (char)(1072 + k - 1);

        Вершины вершина = вершины[Волна.Ind[k]];
        p = Tree.top;

        // составляем массив ребер
        R.Clear();
        for (j = 0; j < вершина.Узлы.Count; j++)
        {
            for (i = 0; i < вершина.Узлы[j].Ребра.Count; i++)
                if (R.IndexOf(вершина.Узлы[j].Ребра[i]) == -1)
                    R.Add(вершина.Узлы[j].Ребра[i]);
        }
```

```
        i = -1; ok = false; // ищем 0-диакритический
        while ((i < p.childs.Count - 1) & !ok)
        {
            i++;
            ok = p.childs[i].count == вершина.Диакритические.X;
        }
        if (!ok)
        {
            NodeC q = new NodeC(p.xc + 200 * p.childs.Count, p.yc +
80, p);
            q.typeNode = TypeNode.диакритических;
            q.count = (byte)вершина.Диакритические.X;
            q.Диакритические = вершина.Диакритические;
            p.childs.Add(q);
            p = q;
        }
        else
            p = p.childs[i]; // p - диакритический


        if (вершина.Диакритические.X != 0)       // диакритический <>
0 найти положение
        {
            i = -1; ok = false;
            while ((i < p.childs.Count - 1) & !ok)
            {
                i++;
                ok = p.childs[i].count == вершина.Диакритические.Y;
            }
            if (!ok)
            {
                NodeC  q  =  new  NodeC(p.xc  +  200  *  p.childs.Count,
p.yc + 80, p);
                q.typeNode = TypeNode.положение;
```

```
                q.count = (byte)вершина.Диакритические.Y;
                //q.Диакритические = вершина.Диакритические;
                p.childs.Add(q);
                p = q;
            }
            else
                p = p.childs[i];
        }

        // ищем подходящее число узлов
        i = -1; ok = false;
        while ((i < p.childs.Count - 1) & !ok)
        {
            i++;
            ok = p.childs[i].count == вершина.Узлы.Count;
        }
        if (!ok)
        {
            NodeC q = new NodeC(p.xc + p.childs.Count * 200, p.yc +
80, p);
            q.typeNode = TypeNode.числоУзлов;
            q.count = (byte)вершина.Узлы.Count;
            p.childs.Add(q);
            p = q;
        }
        else
            p = p.childs[i];

        // ищем подходящее число ребер
        i = -1; ok = false;
        while ((i < p.childs.Count - 1) & !ok)
        {
            i++;
            ok = p.childs[i].count == R.Count;
```

```
        }
        if (!ok)
        {
            NodeC q = new NodeC(p.xc + p.childs.Count * 200, p.yc +
80, p);
            q.typeNode = TypeNode.числоЛиний;
            q.count = (byte)R.Count;
            p.childs.Add(q);
            p = q;
        }
        else
            p = p.childs[i];

        if (p.childs == null) // p.childs[i] - подходящее число
ребер
            p.childs = new List<NodeC>();
        int s = p.childs.Count;

        NodeC v = new NodeC(p.xc + s * 10, p.yc + 80 + s * 10, p);
        v.typeNode = TypeNode.символ;
        v.lines = new List<TypeLine>();
        v.symbols = "" + ch;
        v.W = вершина.W;
        v.H = вершина.H;
        v.Диакритические = вершина.Диакритические;

        for (j = 0; j < R.Count; j++)
        {
            TypeLine t = new TypeLine();

            t.typeLine2 = TypeLine2.концов2;
            if ((R[j].КонечныйУзел == null) | (R[j].НачальныйУзел ==
null))
                t.typeLine2 = TypeLine2.концов1;
```

```
if (R[j].КонечныйУзел == R[j].НачальныйУзел)
    t.typeLine2 = TypeLine2.кольцо;

byte b = ЧислоЕдиниц(R[j].bEdge);
if (b <= 2)
    t.typeLine1 = TypeLine1.прямая;
else
    if (b == 8)
        t.typeLine1 = TypeLine1.кольцо;
    else
        t.typeLine1 = TypeLine1.дуга;
t.sector = R[j].bEdge;
t.R = R[j].R;
t.xc = R[j].xc;
t.yc = R[j].yc;

t.ТочкиСкелета = new List<PointF>();
for (int d = 0; d < R[j].ТочкиСкелета.Count; d++)
    t.ТочкиСкелета.Add(R[j].ТочкиСкелета[d]);

R[j].СоздатьЛинию Производной();

t.линияПроизводной = new List<PointF>();
for (int d = 0; d < R[j].линияПроизводной.Count; d++)
    t.линияПроизводной.Add(R[j].линияПроизводной[d]);

t.lines = new List<List<Lines>>();
for (int d = 0; d < R[j].lines.Count; d++)
{
    List<Lines> tmp = new List<Lines>();
    for (int f = 0; f < R[j].lines[d].Count; f++)
    {
        tmp.Add(R[j].lines[d][f]);
    }
```

```
                t.lines.Add(tmp);
            }

            t.PolyLine = new List<PointF>();
            for (int d = 0; d < R[j].PolyLine.Count; d++)
                t.PolyLine.Add(R[j].PolyLine[d]);

            v.lines.Add(t);
        }

        p.childs.Add(v);
    }
}
```

Приложение 4. Арабские символы

Таблица 2.

Классификация арабских символов по топологическим признакам.

№					Диакрит	Узлов	Ребер
1	ا		ا			1	
6	ح	ـحـ	حـ	ح	0	1,2	1-4
8	ـد		د		0	1	1-3
10	ـر		ر		0	1	1
12	س	ـسـ	سـ	س	0	1	1
14	ص	ـصـ	صـ	ص	0	1,2	
16	ط	ط	ط	ط	0	1,2	

18	ع	ـعـ	ـع	ع	0	1	
23	ل	ـلـ	ـل	ل	0	1,2	
24	م	ـمـ	ـم	م	0	1,2	
26	ه	ـهـ	ـه	ه	0	1,2	
27	و		ـو	و	0	1	
2	ب	ـبـ	ـب	ب	1	1	
5	ج	ـجـ	ـج	ج	1	1,2	
7	خ	ـخـ	ـخ	خ	1	1,2	

9	ذ		ذ		1	1	
11	ز		ز		1	1	
15	ض	ضـ	ضـ	ض	1	1,2	
17	ظ	ظ	ظ	ظ	1	1,2	
19	غ	غ	غ	غ	1		
20	ف	فـ	فـ	ف	1	1	
22	ك	كـ	كـ	ك	1	1,2	
25	ن	نـ	نـ	ن	1	1	

3	ـت	ـتـ	تـ	ت	2	1	
21	ـق	ـقـ	قـ	ق	2	1	
28	ـي	ـيـ	يـ	ي	2	1	
4	ـث	ـثـ	ثـ	ث	3	1	
13	ـش	ـشـ	شـ	ش	3	1	