Федеральное государственное бюджетное учреждение науки
Научно-исследовательский институт системных исследований РАН

ТРУДЫ НИИСИ РАН

ТОМ 2 N 1

ОБРАБОТКА ИЗОБРАЖЕНИЙ, МОДЕЛИРОВАНИЕ И ВИЗУАЛИЗАЦИЯ:

ТЕОРЕТИЧЕСКИЕ И ПРИКЛАДНЫЕ АСПЕКТЫ

МОСКВА
2012

Редакционный совет НИИСИ РАН:
В.Б. Бетелин (председатель),
О.М. Белоцерковский, Е.П. Велихов, В.А. Галатенко, В.Б. Демидович (отв. секретарь),
П.П. Кольцов, Б.В. Крыжановский, А.Г. Кушниренко, А.Г. Мадера, М.В. Михайлюк,
В.Я. Панченко, В.П. Платонов, В.Н. Решетников

Главный редактор журнала:
В.Б. Бетелин

Научные редакторы тома:
П.П. Кольцов, М.В. Михайлюк

Тематика номера:
Обработка изображений, визуализация, моделирование
и информационные технологии

Журнал публикует оригинальные статьи по следующим областям исследований: математическое и компьютерное моделирование, обработка изображений, визуализация, системный анализ, методы обработки сигналов, информационная безопасность, информационные технологии, высокопроизводительные вычисления, микроэлектронная элементная база

The topic of the issue:
Image Processing, Visualization, Modeling and
Information Technology

The Journal publishes novel articles on the following research arias: mathematical and computer modeling, system analysis, image processing, visualization, signal processing, information security, information technologies, high-performance computing, microelectronic component base

Заведующий редакцией: Ю.Н. Штейников

Издатель: Российская академия наук, НИИСИ РАН,
117218, Москва, Нахимовский проспект 36, к.1

© Федеральное государственное бюджетное учреждение науки
Научно-исследовательский институт системных исследований РАН, 2012 г.

СОДЕРЖАНИЕ

Цифровая обработка изображений

А.В.Захаров, П.П.Кольцов, Н.В.Котович, А.А.Кравченко, А.С.Куцаев, А.С.Осипов. Некоторые методы сравнительного исследования детекторов границ 4

И.В.Грибков. «Области влияния» и их использование для тестирования энергетических методов реставрации изображений 14

А.В.Захаров, П.П.Кольцов, Н.В.Котович, А.А.Кравченко, А.С.Куцаев, А.С.Осипов. Развитие метода вычисления направления касательной на основе преобразования Хафа 39

Математическое моделирование и визуализация

М.В.Михайлюк, А.М.Трушин. Моделирование динамики колес в виртуальных сценах 47

М.В.Михайлюк. Моделирование гусениц транспортных средств в системах визуализации 54

М.Г.Фуругян. Планирование вычислений в многопроцессорных системах с дополнительным ресурсом 60

Информационные технологии – моделирование и приложения

А.Г.Прилипко. Хранение истории изменения данных в реляционной базе данных 67

А.А.Бурцев, М.А.Бурцев. ДССП для троичной виртуальной машины 73

Некоторые методы сравнительного исследования детекторов границ

А.В. Захаров; П.П. Кольцов, к. ф.-м. н., доц.; Н.В. Котович; А.А. Кравченко, к. ф.-м. н.; А.С. Куцаев, к. ф.-м. н.; А.С. Осипов, к. ф.-м. н.

В статье рассматриваются основные подходы к сравнительной оценке производительности детекторов границ. При этом основное внимание уделено методам количественной оценки, использующим ground truth изображения и меры различий. Анализируются достоинства и недостатки наиболее употребительных мер различий, особенности их совместного применения. Рассматриваются нечёткие методы оценки, в том числе, подход, основанный на использовании нечётких ground truth изображений.

1. Введение

Выделение границ представляет собой один из основных шагов в задачах обработки изображений, распознавания и анализа образов, компьютерного зрения. Оно давно зарекомендовало себя в качестве метода решения задач сегментации изображений (наряду с методами кластеризации, методами растущих областей, методами с использованием гистограмм, методом водораздела и другими общеизвестными методами). Под детекторами границ понимаются различные алгоритмы, которые выделяют точки цифрового изображения, в которых резко изменяется яркость или есть другие виды неоднородностей. Следует отметить, что сам способ определения граничной точки двумерного изображения носит нечёткий характер. На практике не часто можно жёстко зафиксировать порог на то, каким должно быть изменение яркости между двумя соседними пикселями, чтобы мы могли бы утверждать, что там находится граница. Это одна из причин того, что выделение границ – нетривиальная задача. К настоящему времени разработаны десятки различных детекторов границ (при этом некоторые из них имеют различные доступные в Интернете программные реализации), что делает задачу сравнительного анализа их достоинств и недостатков и разработку универсальных критериев их оценок всё более актуальной.

Желаемым результатом выделения границ является набор связных кривых, обозначающих границы объектов, а также кривые, которые отображают изменения положения поверхностей. Однако, к сожалению, не всегда возможно выделить границы в картинах реального мира средней сложности. Границы, выделенные из таких изображений, часто имеют такие недостатки, как фрагментированность (кривые границ не соединены между собой), отсутствие границ или наличие ложных, не соответствующих исследуемому объекту границ.

Отметим, что в литературе наряду с термином детектор границ используется термин фильтр границ (edge filter). При этом, как правило, имеется в виду фильтр сглаживания (этап предобработки) в комбинации со способом вычисления силы края (edge strength), например способом вычисления градиентов интенсивности изображения в x и y направлениях, и этапом постобработки (удаления ложных границ).

Изначально методы выделения границ можно было разделить на две основные категории: методы, основанные на поиске максимумов (градиентные методы) и методы, основанные на поиске нулей (методы второго порядка). Методы, основанные на поиске максимумов, восходящие к работам Sobel, Prewitt и Roberts, выделяют границы с помощью силы края, такого как величина градиента. Затем происходит поиск локальных максимумов силы края, используя предполагаемое направление границы, обычно перпендикуляр к вектору градиента. Методы второго порядка, восходящие к работам Marr и Hildreth, ищут пересечения оси абсцисс выражения второй производной, обычно нули Лапласиана или нули нелинейного дифференциального выражения второго порядка. В настоящее время используются и статистические методы (основная идея – исследование распределения значений интенсивности в окрестности данного пикселя и определение возможности его классификации как граничного), методы с использованием нечёткой логики (fuzzy edge detectors), логических операторов (алгоритм Iverson), методы, основанные на согласованности фаз (используется частотный подход, ищутся области на изображении, где все синусоиды в частотном пространстве находятся в фазе, предполагается, что они соответствуют областям воспринимаемой границы) и другие методы.

Принято считать, что основные требования к детектору границ были впервые сформулированы J. Canny в классической работе [9]. Автору удалось сначала содержательные требования к детектору, выразить их в виде некоторой оптимизационной задачи и, наконец, решить эту задачу. Canny потребовал, чтобы детектор границ удовлетворял следующим трём критериям эффективности:

1. Хорошее отношение сигнала к шуму. Содержательно это означает, что детектор должен выделять все истинные границы и при этом не выделять ложных.
2. Хорошая локализация – точки, отмеченные как границы, должны располагаться как можно ближе к истинному положению границ.
3. Единственность отклика на границу (С одной стороны, это требование содержится в первом критерии, однако математическая формулировка первого критерия не обеспечивает выполнения данного пункта)

J. Canny изучил математическую проблему получения фильтра границ, оптимального по данным критериям. Он показал, что искомый фильтр является суммой четырёх экспонент, и что этот фильтр может быть хорошо приближен первой производной Гауссиана. Хотя работа [9] была выполнена на заре компьютерного зрения, детектор границ Canny до сих пор является одним из лучших детекторов. Кроме частных случаев трудно найти детектор, который бы работал существенно лучше, чем детектор Canny, и практически все работы по сравнительному исследованию детекторов границ содержат его в качестве своего рода эталона.

Целью настоящей работы является дать объективное представление о ряде методов (в том числе применяемых авторами) сравнительного исследования производительности детекторов границ и используемых при этом критериях оценки качества рассматриваемых алгоритмов. Предполагается, что приведённая ниже информация может помочь практическим пользователям сделать выбор того или иного детектора границ для решения своих задач более осмысленным.

2. Краткая характеристика подходов к оценке детекторов границ

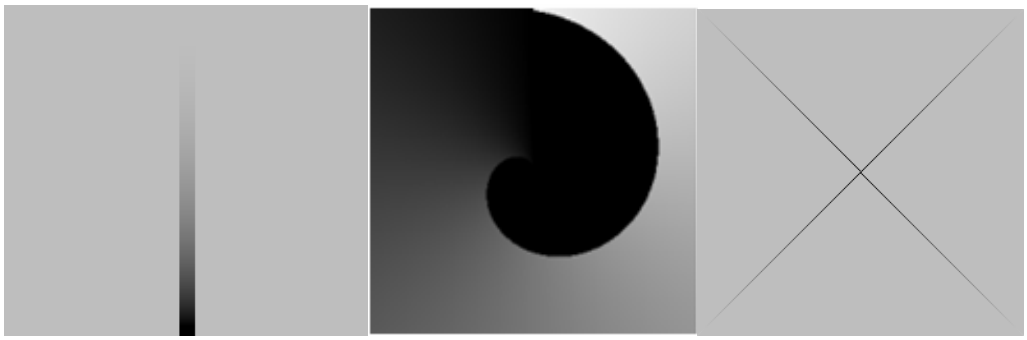
Как и в случае других классов алгоритмов обработки изображений, методы оценки производительности детекторов границ можно разделить на теоретические и эмпирические. Это зависит от того, используют они теоретический анализ или анализируют экспериментальные результаты работы того или иного алгоритма. В свою очередь, эмпирические методы подразделяются на методы, использующие ground truth оценку и т. н. методы эмпирической оценки доброкачественности (empirical goodness methods), которые не используют ground truth оценку.

Теоретические методы оценки производительности детекторов границ восходят к работам [1], [30]. В указанных работах входные данные алгоритма описываются математически, а производительность алгоритма определяется аналитически или путём симуляции. Для оценки были разработаны соответствующие меры. Недостатками этих работ являлись упрощённые математические модели, характеризующие входные сигнал и шум, и сложность применения предложенных методов к современным детекторам границ из-за сложности алгоритмов последних.

Среди работ, посвящённых эмпирическим методам оценки производительности можно выделить [19], [40], [26]. В первой из этих работ производительность алгоритмов оценивалась при помощи введённого параметра согласованности границ (edge coherence, называемого в некоторых источниках метрикой Розенфельда), которая оценивает непрерывность и толщину выделенных границ (из общих соображений понятно, что выделенные границы должны быть непрерывными и тонкими). Вторая из работ предлагает иной подход к определению и вычислению локальной согласованности границ. В третьей работе предложен метод оценки системы, где выделение границ предшествует этапу поиска линий на изображении (line detection). Была предложена мера, которая вычисляет нелинейную комбинацию носителей выделенных линий. Все эти методы оценивают либо форму выделяемых границ, либо вероятность правильного определения граничного пикселя на основе изучения яркости пикселей в его окрестности. Существенным недостатком данных методов в их практическом применении является то, что они не могут измерить смещение выделенных граничных пикселей относительно их истинного положения. В частности, эти методы могут давать неадекватные результаты на детекторах, использующих размытие изображения на этапе предобработки.

К эмпирическим методам оценки можно отнести и метод, предложенный в работе [16], где в качестве входных данных используются реальные изображения достаточно сложной структуры, при этом качество работы различных алгоритмов оценивают люди, выразившие желание участвовать в эксперименте. В качестве тестовых изображений взяты фотографии некоторых объектов (одной фотографии соответствует один объект). Экспертам в произвольном порядке предлагают набор изображений, содержащий результаты работы различных детекторов, при этом изначально неизвестно, какое изображение соответствует какому детектору. Эксперт выставляет оценку каждому изображению (насколько хорошо на них распознаются исходные объекты). На различных выборках изображений оценивается постоянство критериев оценок участников (вычисляются коэффициенты корреляции оценок по различным выборкам изображений). Предлагаются схемы включения в рассматриваемый метод новых детекторов границ. Что до практической значимости работы, то она подтвердила некоторые сложившиеся представления об известных детекторах границ (в частности, вывод о существенности предварительной настройки их входных параметров). Однако, она не решает вопрос о выборе оптимального детектора границ в решении той или иной конкретной задачи.

Наиболее обширное семейство методов оценки детекторов границ использует сравнение результатов работы исследуемых алгоритмов на тестовых изображениях с ground truth образами (как правило, созданными вручную) содержащих идеальные в понимании экспериментатора границы. Для количественной оценки результатов здесь основную роль играют меры различий (discrepancy measures). Используются и производные от этих метрик методы оценок, например анализ ROC–кривых (от Receiver Operating Characteristic). Среди ранних работ на данную тему можно упомянуть [13]. В ней детекторы оценивались путём вычисления отношения правильно определённых граничных пикселей к общему числу определённых граничных пикселей и доли граничной линии (на ground truth образе), покрытой пикселями, выделенными оцениваемым детектором. В упоминавшейся ранее работе Abdou и Pratt, была введена мера различий, включающая в себя расстояние между выделенным детектором граничным пикселем и его истинным местоположением (ниже мы рассмотрим её подробнее). Также можно упомянуть относительно раннюю работу [35], где была введена метрика, оценивающая качество выделенных границ и представляющая собой линейную комбинацию мер непрерывности границ, их гладкости (неразрывности),



a) b) c)

Рис. 1. Несколько изображений из тестового набора системы PICASSO [14]: a) Исчезающая Полоса, b) Улитка, c) Затухающий Узел.

тонкости, локализации, истинности и зашумлённости (оцениваются «лишние границы»). Примерно в это же время в работе [18] был предложен использующий ground truth метод оценки детекторов границ в системе, где данный этап предшествует этапу поиска линий.

К недостаткам упомянутых работ можно отнести то, что используемые в них ground truth образы создавались искусственно или брались несложные реальные изображения, содержащие легко идентифицируемые границы. На практике нередко встречаются достаточно трудные для распознавания ситуации, что ограничивает область применимости указанных методов. На это, в частности, указывалось в работе [16]. Также в работе [12] было показано, что небольшие изменения в используемом искусственном тестовом (и соответствующем ground truth) изображениях приводили к различным оценкам трёх известных детекторов границ (в разных ситуациях каждый из детекторов становился лучшим). Так возник вопрос о полноте набора тестовых изображений, используемых в методологии с использованием ground truth образов. Указанные недостатки были изучены и частично преодолены в последующих работах по данной тематике (в частности, это вызвало совершенствование используемых тестовых и ground truth изображений). Стали чаще использоваться изображения, моделирующие сложные для выделения границ ситуации (см., например, Рис. 1, представляющий несколько изображений из базы тестовых образов системы PICASSO [14]).

На Рис. 1 одному изображению соответствует одна сложная ситуация, связанная с наличием границы изменяющегося контраста. В работе [14] тестировалось 6 известных алгоритмов выделения границ (Canny, Rothwell, Heitger, Black, Smith and Iverson). В частности, было установлено нестабильное поведение всех алгоритмов на изображениях, содержащих подобные границы, при малых поворотах, сдвигах и растяжениях этих изображений. Использовались и упрощённые версии изображений, представленных на Рис. 1, содержащие только границы постоянного контраста, при этом поведение алгоритмов на них было относительно стабильным. Тем самым, сочетание простых и усложнённых изображений сделало тестирование более результативным.

Также в ряде работ (см., например, [8]) используются трёхзначные ground truth образы, где каждый пиксель изображения относится к одному из трёх классов: граница, фон, не важно (попадание в этот класс на оценки не влияет). Это, с одной стороны, упрощает искусственное создание ground truth образов, соответствующих реальным изображениям (можно, например, отнести к третьему классу все текстурные области на изображении), а с другой стороны делает количественные оценки более содержательными. Наконец, в работе [39] предлагается автоматический метод построения ground truth образа (т. н. estimated ground truth). Берётся несколько детекторов границ (или один детектор с разными входными параметрами) и результаты их работы на тестовом изображении. Ищется набор пикселей, оптимально соответствующий пикселям всех граничных карт, который и объявляется estimated ground truth образом.

3. Используемые меры различий

Как уже отмечалось, основными характеристиками хорошего детектора границ является высокий процент правильно выделенных граничных пикселей (высокий уровень детектирования) и высокая степень локализации (близость выделенных пикселей к соответствующим им на ground truth изображении). К настоящему времени не удалось создать меру, одинаково эффективно оценивающую эти две характеристики. Отчасти это связано с тем, что, как отмечалось в работе Canny [9], имеет место своего рода принцип неопределённости между высоким уровнем детектирования и высокой локализацией. Соответственно, для оценки данных характеристик детекторов границ используют разные меры различий. В работах [27], [37], [3], посвящённых исследованию мер различий для детекторов границ, рассматриваемые меры подразделялись на меры оценки качества детектирования (detection performance, or “statistical” measures) и меры оценки локализации (localization performance, or “distance” measures).

Приведём пример нескольких часто встречающихся мер оценки качества детектирования [27], [37], [3]. Итак, пусть имеются: X – растр изображения, состоящий из N пикселей, B – результат работы оцениваемого детектора границ (образ, состоящий из граничных точек) и A – соответствующий ground truth образ. Тогда ошибка первого рода определяется как отношение неправильно выделенных граничных пикселей к общему числу пикселей, не являющихся граничными:

$$\alpha(A,B)=\frac{n(B\setminus A)}{n(X\setminus A)},$$

где $n(.)$ – число пикселей в соответствующем множестве.

Ошибка второго рода определяется как

$$\beta(A,B)=\frac{n(A\setminus B)}{n(A)},$$

т.е. как отношение не выделенных граничных пикселей к общему числу граничных пикселей.

Общая ошибка классификации (overall misclassification rate) определяется как

$$\varepsilon(A,B)=\frac{n(A\Delta B)}{n(X)},$$

где Δ – симметрическая разность множеств.

Также на практике часто встречаются такие меры как Чувствительность (Sensitivity):

$$Se=\frac{n(B\cap A)}{n(A)}=1-\beta,$$

т.е. отношение правильно выделенных граничных пикселей к общему числу граничных пикселей, а также Специфичность (Specificity):

$$Sp=\frac{n(X/B\cup A)}{n(X/A)}=1-\alpha.$$

Чувствительность, Специфичность, ошибки первого и второго рода являются исходными величинами при построении ROC – кривых, анализ которых нашёл применение и в задаче оценки детекторов границ. Указанные меры оценки качества детектирования не случайно называются статистическими. Изначально они нашли своё применение в медицинской статистике в анализе диагностики различных заболеваний [5]. Кроме того, указанные меры являются частным случаем (при двумерной классификации) мер оценки качества классификации (classification accuracy measures), широко используемых в обработке данных дистанционного зондирования [10]. Так, например, *Sp* и *Se* представляют собой частные случаи мер точности обработки (Producer's accuracy measures) [10],[17].

К мерам оценки качества детектирования можно отнести и Евклидову среднеквадратическую метрику, применяемую при сравнении двух полутоновых изображений:

$$\| f-g \| = [\frac{1}{N}\sum_{x\in X}(f(x)-g(x))^2]^{\frac{1}{2}}$$

где $f(x)$ обозначает значение яркости изображения f в точке x. В случае детекторов границ f и g могут принимать значения 0 или 1 в зависимости от того, является ли точка соответствующего изображения граничной или нет. Сюда же относятся эквивалентные (в данной ситуации) отношения сигнала к шуму, пиковое и среднеквадратическое. Эти отношения используются и на этапе предобработки изображений (перед выделением границ) для оценки соответствующих фильтров (см. например [32]).

Указанные меры оценки качества детектирования имеют большое практическое применение, при этом неоднократно отмечались их недостатки [31]. Наиболее слабым их местом является то, что различия между изображениями A и B определяются по общему числу расхождений между ними, безотносительно к образу, который эти изображения представляют. Так, небольшой сдвиг в оцениваемой карте границ относительно ground truth изображения, затрагивающий большое число пикселей но не меняющий формы образа (т. е. к примеру, на оцениваемом изображении то же яблоко, что и на ground truth, слегка сдвинутое по отношению к последнему) приведёт к плохим значениям оценки качества детектирования. В то же время, небольшие искажения, затрагивающие незначительное число пикселей, но существенно меняющие форму изображаемого объекта (небольшие удаления линий, заполнения маленьких дырок и т. п.) дадут хорошие значения данных мер. При практическом оценивании детекторов границ с помощью этих мер следует принимать во внимание указанные недостатки. В частности, это относится к выделению границ, использующих размытие зашумлённых изображений на этапе предобработки.

Что касается мер оценки локализации, к ним можно отнести среднюю ошибку расстояния (mean error distance)[27],[37]:

$$e(A,B)=\frac{1}{n(B)}\sum_{x\in B}d(x,A),$$

где $d(X,A)=\inf \rho(x,a),\ a\in A,$ а $\rho(.,.)$ в рассматриваемой дискретной ситуации представляет собой метрику кратчайшего пути (shortest path length metric) [6],[7]. Сюда же относится среднеквадратическая ошибка расстояния (mean square error distance):

$$e^2(A,B)=\frac{1}{n(B)}\sum_{x\in B}d(x,A)^2,$$

а также упоминавшаяся ранее метрика Прэтта (Pratt's figure of merit) [1], [28]:

$$FOM(A,B)=\frac{1}{\max\{n(A),n(B)\}}\sum_{x\in B}\frac{1}{1+kd(x,A)^2},$$

где k – коэффициент масштабирования, обычно полагаемый равным 1/9, а метрика кратчайшего пути нормируется таким образом, что наименьшее ненулевое расстояние между соседними пикселями равно единице. Заметим, что:

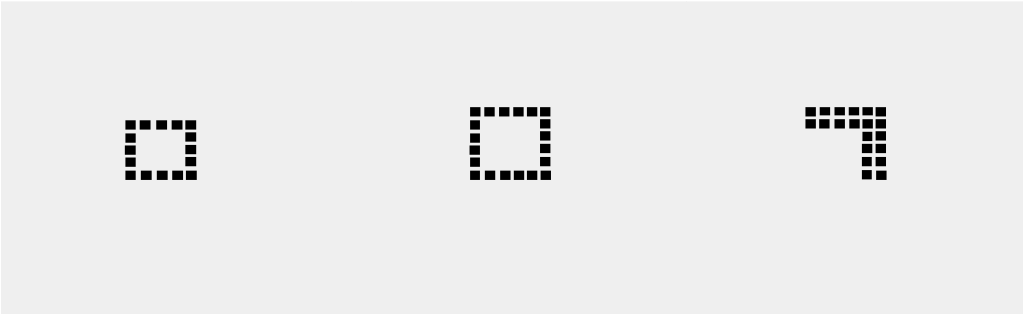
$$FOM(A,B)\neq FOM(B,A),$$

$$0<FOM(A,B)\le 1,$$

$FOM(A,B)=1$ тогда и только тогда, когда $A=B$.

К данному разряду мер оценки локализации можно отнести и метрику Хаусдорфа [3], [33]. Хотя классическая версия данной метрики обладает желательными для использования в оценке цифровых изображений топологическими свойствами, на практике она используется редко в силу своей высокой чувствительности к шуму и даже к изменениям в один пиксель на оцениваемом изображении. В свою очередь, метрика Прэтта является наиболее часто используемой из перечисленных мер.

К недостаткам перечисленных мер оценки локализации является то, что они зачастую нечувствительны к ошибкам второго рода. Например, если все ошибки оцениваемого образа B – второго рода, и $B\in A$, то



a) b) c)

Рис. 2. Пример Peli-Malah, a) – ground truth образ (квадрат 5*5), b) и c) – оцениваемые образы (результаты работы двух детекторов границ). Значение *FOM* в случаях b) и c) одинаково и равно 0.941.

значения $e=e^2=0$, а $FOM(A,B)=n(B)/n(A)=1-\beta(A,B)$. В отличие от метрики Прэтта, меры e и e^2 сильно чувствительны к фоновому шуму. К примеру, если образ B содержит даже один пиксель x, находящийся на большом расстоянии от A, значение этого расстояния даст большой вклад в значение средних ошибок расстояния. Это согласуется с известным фактом из статистики о неустойчивости среднего арифметического. В силу того, что перечисленные меры оценки локализации нормируются на переменный множитель $n(B)$ или $\max\{n(A),n(B)\}$ возможны сложности с интерпретацией их значений, (например, может быть неясно, как интерпретировать значения $FOM(A,B)$ для фиксированного A и различных B в ситуации, когда $n(B)>n(A)$). Что касается метрики Прэтта, то анализ её недостатков был проведён в работах [27],[37]. В частности, в [37] при оценке данной метрикой результатов выделения границ Лапласианом часто возникали ситуации высоких значений *FOM* на изображениях, содержащих участки разрывов границ (дырок) и участки, где получившийся граничный контур осциллирует вокруг своего истинного положения (на ground truth изображении). Это наблюдение может быть объяснено тем, что для $x \in A \cup B$ соотношение

$$FOM(A, B \cup x) < FOM(A, B) \qquad (1)$$

выполняется тогда и только тогда, когда

$$d(x, A) > k^{-1/2}(FOM(A, B)^{-1} - 1)^{1/2}.$$

В ситуации, когда $FOM(A, B) > 0.9$ и $k = 1/9$ соотношение (1) выполняется тогда и только тогда, когда $d(x, A) \geq 1$.

Также, в работе [27] был приведён получивший известность противоречащий здравому смыслу пример поведения метрики Прэтта, (т.н. Peli-Malah counterexample, Рис. 2).

Авторами были предъявлены два изображения, приведённые на Рис. 2.b) и Рис 2.c), на которых значения *FOM* будут одинаковыми, если в качестве ground truth образа взять квадрат размером 5*5 пикселей, изображенный на Рис 2.a). Значения вышеприведённых мер оценки качества детектирования будут также одинаковы. Применительно к метрике Прэтта подобные примеры означают, что она позволяет соответствовать нескольким выделенным граничным пикселям одному и тому же пикселю на ground truth изображении (в случае отсутствия взаимно однозначного соответствия пикселей значение *FOM* может быть таким же, как и в случае наличия такого соответствия). В работе [3] была предложена мера оценки локализации, представляющая собой L^p модификацию метрики Хаусдорфа:

$$\Delta_w^p = \frac{1}{n(X)}\left(\sum_{x \in X} | w(d(x, A)) - w(d(x, B)) |^p\right)^{1/p},$$

$$1 \leq p \leq \infty,$$

где в качестве w, как правило, берётся т.н. функция отсечения (cutoff transform): $w(t) = \min\{t, c\}$ для некоторого фиксированного $c>0$. Данная метрика, как и метрика Хаусдорфа, обладает желаемыми топологическими свойствами. При этом, в отличие от последней, она более устойчива к возмущениям в один или несколько пикселей (максимум в разности расстояний заменяется на L^p среднее). Ряд экспериментов показал её более адекватное поведение в сравнении с метрикой Прэтта. Так, например, в примере Peli-Malah на Рис. 2.b) и 2.c) значения Δ^2 при c=5 были 0.323 и 0.512 соответственно. В то же время, метрика Прэтта продемонстрировала более устойчивое поведение к малым возмущениям границы. В дальнейшем были предложены обобщения метрики Δ^p, позволяющие использовать её для сравнения полутоновых изображений [11], [38]. Для оценки детекторов границ данная метрика используется сравнительно редко, см., например, [11].

В силу того, что отсутствие взаимно однозначного соответствия между пикселями оцениваемой карты границ и соответствующим ground truth изображением в перечисленных мерах может привести к нереалистичным их оценкам, представляется целесообразным использовать меры, где подразумевается такое соответствие. В частности, в работе [8] было предложено проводить вычисление мер чувствительности и ошибки первого рода с использованием взаимно однозначного соответствия. Именно, берётся дистанционный порог T (в работе он полагался равным 3). Если в окрестности T пиксель оцениваемой граничной карты оказывался без соответствующих ground truth пикселей, значение ошибки первого рода увеличивалось на единицу. Если соответствующий ground truth пиксель находился, значение чувствительности увеличивалось на единицу, и этот пиксель помечался как использованный, так что его нельзя было использовать в дальнейшем. Если в окрестности T оказывалось несколько ground truth пикселей, то в качестве соответствующего пикселя брался тот, который находился на кратчайшем

расстоянии от пикселя оцениваемого изображения. В работе отмечалось, что предложенный метод слегка меняет значения ошибки первого рода и чувствительности по сравнению с их вычислением обычным способом. Однако он не влияет на общие результаты оценки производительности детекторов границ. В работе [29] данная методика была модернизирована для работы с картами границ, содержащими информацию о силе края (edge strength). Именно, если граничный пиксель имеет два соответствующих ему пикселя, находящихся на одинаковом расстоянии, то из них выбирается наиболее сходный по силе края. Метрика, сравнивающая два изображения и построенная на данном соответствии, была названа в [29] метрикой кратчайшего пути (Closest Distance Metric, *CDM*). В свою очередь, в работе [29] была предложена альтернативная метрика, основанная на поиске соответствующих пикселей на сравниваемых изображениях, т. н. метрика соответствия пикселей (Pixel Correspondence Metric, *PCM)*. Для каждой пары соответствующих пикселей строится функция затрат (cost function), вычисляемая с учётом расстояния между ними и силы края. Выбирается пара соответствующих пикселей с учётом наименьшего значения функции затрат. Общее значение метрики соответствия пикселей между изображениями f и g вычисляется по формуле.

$$PCM_T = 100(1 - \frac{C(M_{opt}(f,g))}{|f \cup g|}),$$

где $C(M_{opt}(f,g))$ - стоимость оптимального соответствия между образами, вычисляемая как сумма значений функции затрат по парам соответствующих пикселей, $|f \cup g|$ - общее число граничных пикселей в f и g. Индекс T означает окрестность, в которой ищутся соответствующие пиксели. Поиск пар пикселей по минимальному значению функции затрат представляет собой трудоёмкий с вычислительной точки зрения процесс. Для этого поиска используется алгоритм из теории двудольных графов (weighted matching algorithm), разработанный для решения сходной задачи. Данная метрика, наряду с *FOM* и *CDM*, использовалась для оценки нескольких зашумлённых классических изображений, (Lena, Peppers, Goldhill) обработанных детектором Canny, и данных изображений без добавления шума, обработанных несколькими версиями детектора Canny. Основной задачей здесь было исследовать поведение указанных мер оценки локализации при небольших сдвигах полученных границ относительно ground truth образов. Эксперименты показали в целом более аккуратное поведение метрики *PCM* по сравнению с *CDM* и *FOM*. В частности, при увеличении окрестности T увеличивается значение *PCM*, что согласуется со здравым смыслом (в данной ситуации возрастает вероятность найти соответствующий пиксель с меньшим значением функции затрат). Однако, явных преимуществ метрики *PCM* над *CDM* выявлено не было, при том, что программная реализация последней значительно проще.

Следует отметить, что при совместном применении нескольких мер различий из двух рассмотренных классов (статистических и мер оценки локализации) к оценке детекторов границ может возникнуть вопрос о согласованности их результатов и предпочтениях в их интерпретации. Например, в работе [14] для оценки шести детекторов границ, применённых к тестовым изображениям, подверженным аффинным преобразованиям (поворотам, сдвигам и растяжениям) использовались Чувствительность, Специфичность, а также метрики Прэтта и Хаусдорфа. Результаты измерений показали существенно худшее поведение двух из рассматриваемых детекторов и невозможность выявить лидера из оставшихся четырёх детекторов, опираясь исключительно на значения данных мер различий. В некоторых работах используется комбинация одной из мер различий с метрикой, не использующей ground truth образов. Так, например, в работе [2] в дополнение к метрике Прэтта использовалась метрика Розенфельда (edge coherence, [19]), определяющая, насколько хорошо граничный пиксель согласуется с расположенными по соседству другими граничными пикселями, но не характеризующая его местоположение. Можно сделать вывод, что на практике задача сочетания различных метрик сохраняет свою актуальность.

4. О нечетких методах оценки

В настоящее время разработано большое число алгоритмов обработки изображений, в той или иной степени использующих элементы теории нечётких множеств, см. например [25], [36]. Во многих ситуациях использование таких алгоритмов представляется вполне оправданным. Например, как справедливо отмечалось в [4], «полутоновые изображения являются нечёткими по своей природе из-за неопределённости, существующей в локализации положения границы, отделяющей объект от фона». Особенно это характерно для размытых изображений. Современные методы сравнительного исследования должны как уметь работать с подобными алгоритмами, так и включать в себя элементы нечёткой логики.

Среди ранних работ, посвященных использованию нечёткой логики в задачах выделения границ можно упомянуть [24], где были рассмотрены достаточно простые нечеткие алгоритмы выделения границ, основанные на использовании классического оператора интенсификации INT, введённого основоположником нечёткой логики L. Zadeh . При этом была введена мера неопределённости границы (edge ambiguity measure), позволяющая оценивать качество работы алгоритма. Данный подход получил дальнейшее развитие в работе [15], где были введены меры, характеризующие существование, локализацию и связность выделенных граничных пикселей. При этом введенные меры могут быть использованы для оптимального выбора параметров детекторов границ (например, пороговых величин или среднее квадратическое отклонение Гауссова оператора сглаживания), не использующих нечёткой логики. Следует, однако, признать, что методика сравнительного исследования «чётких» и «нечётких» детекторов границ не получила к настоящему моменту достаточного развития. В качестве типичного примера можно привести работу [20], где результаты работы предлагаемого «нечёткого» детектора границ на нескольких тестовых изображениях сравниваются визу-

ально с результатами детектора Canny без каких-либо количественных оценок. На наш взгляд, существенным вкладом в развитие подобной методики можно признать работу [17], где было проведено обобщение нескольких известных статистических мер оценки качества на случай нечётких множеств, позволяющее, в том числе сравнивать между собой чёткие и нечёткие множества. Соответствующие меры были названы нечёткими мерами сходства (fuzzy similarity measures). Там же рассматривалась и концепция нечёткого ground truth образа. Результаты данной работы предназначались для анализа аэрокосмических снимков (определения на них областей лесов, воды, городской застройки и т. п.), но они также могут быть применены для исследования производительности различных детекторов границ. Это наблюдение нашло своё отражение в работе [23], где несколько известных («чётких») детекторов границ было протестировано с использованием нечётких мер сходства и нечётких ground truth образов. Остановимся подробнее на этих концепциях, рассмотренных в данной работе. Прежде всего, напомним несколько основных понятий из теории нечётких множеств.

Именно, пусть X есть непустое множество (в нашем случае растр изображения). Нечёткое множество C на X есть пара $<X, f_C>$, где f_C есть отображение X на [0, 1]. Значение $f_C(x)$ в точке $x \in X$ называется степенью принадлежности x множеству C, а функция f_C называется функцией принадлежности нечёткого множества. Обозначим множество всех нечётких множеств на X за $[0, 1]^X$. Для A и B из $[0, 1]^X$ нечёткое отношение включения

$A \subset B$ означает что $f_A(x) \le f_B(x)$ для всех $x \in X$.

Нечёткая мера сходства есть отображение s: $[0, 1]^X \times [0, 1]^X \to [0,1]$, сопоставляющее множествам $A,B \in [0, 1]^X$ степень сходства $s(A,B) \in [0, 1]$, удовлетворяющее условиям:

- $s(A,A) = 1$ для любого нечёткого A,
- $s(A,B) = s(B,A)$ для всех нечётких A и B (коммутативность),
- $s(A,C) \le s(A,B) \wedge s(B,C)$ при $A \subset B \subset C$,

где $p \wedge q$ обозначает минимум из p и q; максимум из p и q обозначается как $p \vee q$. Важными примерами данных мер являются:

$$s_1(A,B) = \frac{\sum_x (f_A(x) \wedge f_B(x))}{\sum_x (f_A(x) \vee f_B(x))},$$

$$s_2(A,B) = \frac{2\sum_x (f_A(x) \wedge f_B(x))}{\sum_x (f_A(x) + f_B(x))}.$$

Заметим, что обычные (чёткие) подмножества M из X включаются в данный подход, если мы будем рассматривать их как стандартные характеристические функции $1_M : X \to [0,1]$. Тем самым, меры s_1 и s_2 можно применять, когда одно из A или B или оба из них являются чёткими множествами. Как показано в [17], для чётких A или B отождествлённых со своими характеристическими функциями, s_1 и s_2 совпадают с мера-

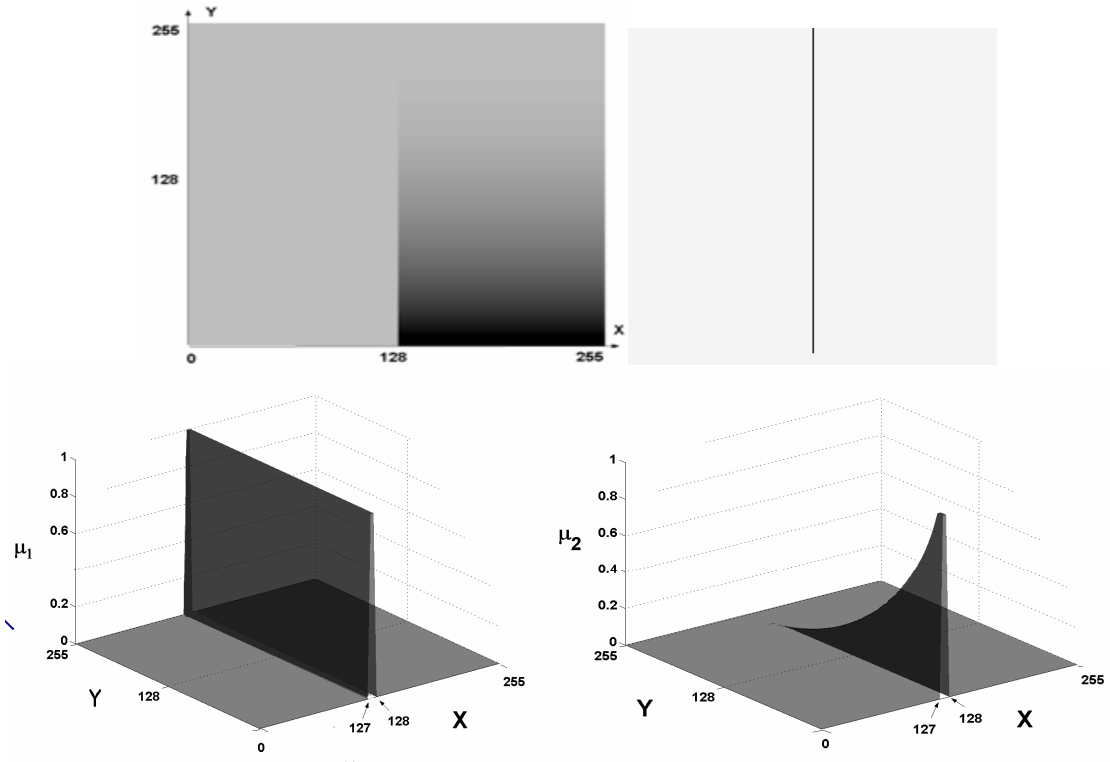


Рис. 3. a) (верхний левый) – изображение Исчезающий Уступ, b) (верхний правый) – соответствующий ground truth изображение, c) и d)(нижний левый и правый) – различные нечёткие ground truth образы (G_1 и G_2).

ми оценки качества классификации Шорта и Хеллдена соответственно (Short's and Hellden's classification accuracy measures) [10], [17].Изначально эти меры определялись только для чётких множеств, так что можно рассматривать меры s_1 и s_2 как их обобщение.

Что касается нечётких ground truth образов, не существует общих правил их построения [17]. В работе [11] было сделано предположение, что их включение в системы сравнительного тестирования детекторов границ позволит не только тестировать нечёткие детекторы, но сделать процедуру тестирования всех детекторов более содержательной. Там же предлагались различные нечёткие ground truth образы, соответствующие одному и тому же тестовому изображению. Предполагалось, что они позволят лучше протестировать те или иные свойства исследуемого детектора границ.

В качестве примера в [23] рассматривалось изображение Исчезающий Уступ (Degenerated Step), Рис. 3.a). На Рис. 3.b) изображён соответствующий ground truth образ, построенный стандартным образом (с допустимой ошибкой локализации в 1 пиксель). На Рис. 3.c) и 3.d) изображены два нечётких ground truth образа: G_1 и G_2, соответствующих Рис. 3.a). Их отличают различные функции μ_1 и μ_2 принадлежности пикселей множеству границ. Функция принадлежности μ_1 является прямоугольной, фактически образ G_1 эквивалентен стандартному образу Рис. 3.b). Функция принадлежности μ_2 учитывает относительную силу края граничных пикселей Рис. 3.a), присваивая значение 1 наиболее «сильным» пикселям в нижней части середины данного рисунка. Тестирование двух детекторов (Smith и Heitger) с использованием данных изображений показало, что поскольку функция μ_1 присваивает одинаковые значения (равные 1) сильным и слабым пикселям, её уместно использовать для проверки способности детектора выделять слабые края. В то же время, значения μ_2 для граничных пикселей зависят от их расположения (и силы), тем самым, делая значения соответствующих нечётких мер сходства чувствительным к разрывам (дыркам) на карте границ, выделенной детектором. Соответственно, образ G_2 предпочтительнее использовать для проверки свойства выделять непрерывные границы.

Другим потенциально важным приложением нечётких ground truth образов, отмеченным в [23], является возможность их использования в проверке свойства детектора границ выделять граничные точки, существенные для характеризации ограничиваемого объекта (т. н. image feature points). Например, для прямоугольника таковыми являются угловые точки. Задавая для таких точек более высокие значения функции принадлежности множеству границ в сравнении с остальными граничными точками ground truth образа можно добиться высокой чувствительности нечётких мер сходства к выделению этих существенных точек. Однако в работе [23] не содержится методики построения таких образов и использования различных нечётких мер сходства для практического тестирования детекторов границ. Тем самым, вопрос о создании методологии тестирования данных алгоритмов с использованием нечётких элементов остаётся открытым.

5. Заключение

В последние 20 лет наметился существенный рост интереса к вопросам сравнительного исследования алгоритмов обработки изображений, хотя достигнутые здесь результаты пока отстают от потребностей сегодняшнего дня. Как мы видели, не существует методики всестороннего тестирования различных детекторов границ (включающей в себя тестирование аккуратности, устойчивости, чувствительности адаптивности алгоритма и т. п.). Каждый из рассмотренных подходов имеет свои достоинства и недостатки и, как следствие, ограничения в области применимости. Тем не менее, накопленные за 20 лет результаты позволяют дать характеристику известным детекторам границ. В качестве примера приведём таблицу, взятую из работы [21], перечисляющую достоинства и недостатки ряда широко известных алгоритмов выделения границ.

Эту информацию следует принимать во внимание практическим пользователям при выборе того или иного детектора границ для решения своих задач.

Кроме того, можно сделать вывод, что если перед

Таблица 1. Достоинства и недостатки наиболее известных алгоритмов выделения границ.

Оператор выделения границ	Достоинства	Недостатки
Классический (Sobel, Prewitt, Kirsch…)	Простота реализации, детектирование граничных пикселей и их ориентаций	Чувствительность к шуму, низкая точность
Нулевые пересечения (Вторая производная по направлению градиента (Haralick))	Детектирование граничных пикселей и их ориентаций, фиксированные характеристики во всех направлениях	Чувствительность к некоторым граничным пикселям, чувствительность к шуму
Лапласиан от Гауссиана (LoG) (Marr-Hildreth)	Нахождение правильных местоположений граничных пикселей, тестирование более широких областей вокруг предполагаемого граничного пикселя по сравнению с предыдущими детекторами	Плохие результаты в угловых точках, на скругленных границах и на границах изменяющегося контраста. Невозможность определения ориентации граничного пикселя вследствие использования Лапласиана.
Гауссиан (Canny, Shen-Castan, Boie-Cox)	Использование вероятности для определения уровня ошибки и минимизации множественного отклика на фильтр, Улучшение отношения сигнала к шуму, лучшие показатели детектирования, особенно на зашумленных изображениях	Сложные вычисления, ложные нулевые пересечения, большие временные затраты.

пользователем стоит проблема выбора одного из нескольких имеющихся детекторов границ, то путь сравнительного их тестирования с использованием ground truth оценок представляется одним из наиболее перспективных. Что касается описанных выше мер различий, то можно сделать вывод, что использование мер различных классов, оценивающих как качество детектирования, так и качество локализации делает результаты тестирования более достоверными. Типичным примером сравнительного тестирования детекторов границ для решения практической задачи (исследования кальдер вулканов) является работа [2], где по тестовой фотографии вулкана вручную строится ground truth образ, затем к фотографии применяется ряд детекторов границ, а результаты сравниваются с построенным образом при помощи двух различных мер.

Другой типичный вариант практического тестирования детекторов границ содержится в работе [34], где рассматривается задача распознавания объектов различных классов (внедорожников разных моделей) на фотографиях, при этом результаты работы детекторов границ являются входными данными для алгоритма распознавания (тем самым выделение границ является частью более сложной системы анализа изображений). Оцениваются конечные результаты работы системы (используется несколько статистических мер, применяемых при оценке алгоритмов распознавания), в зависимости от которых отдаётся предпочтение тому или иному детектору границ. При этом получаются результаты, отличные от результатов непосредственного тестирования детекторов вне контекста внешней задачи (так, лучшим в задаче, рассмотренной в [34] оказывается детектор Smith). Возможность подобного расхождения результатов также следует принимать во внимание практическим пользователям.

В настоящей статье рассмотрена лишь часть методов, используемых при сравнительном изучении детекторов границ и возникающих при этом вопросах. Например, важным вопросом является выбор оптимальных входных параметров детектора границ для решения той или иной конкретной задачи. Мы планируем подробно остановиться на этом и некоторых других вопросах в следующих работах.

Литература

1. I. E. Abdou, W. K. Pratt. Quantitative design and evaluation of enhancement thresholding edge detectors // *Proceedings of IEEE,* Vol. 67, No 5, 1979, pp. 753-763.

2. D. P. Argialas, O. D. Mavrantza. Comparison of edge detection and Hough transform techniques for the extraction of geological features // *Proceedings of the XX ISPRS Congress of the International Society of Photogrammetry and Remote Sensing,* 12-23 July 2004, Istanbul, Turkey, pp. 790-795.

3. A. J. Baddeley. Errors in binary images and Lp version of the Hausdorff Metric // *Nieuw Archief voor Wiskunde,* Vol. 10, 1992, pp. 157-183.

4. M. Banerjee, M.K. Kundu. Content Based Image Retrieval with Fuzzy Geometrical Features - // Proceedings of the IEEE International Conference on Fuzzy Systems, 2003, pp. 932-937.

5. M. Bland, An introduction to medical statistics, Oxford University Press, 1987.

6. G. Borgefors. Distance transformations in arbitrary dimensions // *Computer Vision, Graphics and Image Processing,* Vol. 27, No. 3, 1984, pp. 321-345.

7. G. Borgefors. Distance transformations in digital images // *Computer Vision, Graphics and Image Processing,* Vol. 34, No. 3, 1986, pp. 344-371.

8. K.W. Bowyer, C. Kranenburg, S. Dougherty. Edge detector evaluation using empirical ROC curves // *Computer vision and Image Understanding,* Vol. 84, No. 1, 2001, pp. 77-103.

9. J. Canny. A computational approach to edge detection // *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.8, No.6, 1986, pp. 679-698.

10. R.G. Congalton. A review of assessing the accuracy of classification of remotely sensed data // *Remote Sens. of Environ.,* Vol. 37, No. 1, 1991, pp. 35-46.

11. D. Coquin, Ph. Bolon. Application of Baddeley's distance to dissimilarity measurement between gray scale images // *Pattern Recognition Letters,* Vol. 22, No. 14, 2001, pp. 1483-1502.

12. L. A. Forbes, B. A. Draper. Inconsistencies in edre detector evaluation // *Proceedings of IEEE Conf. on Computer Vision and Pattern Recognition,* 2000, pp. 398-404.

13. J. R. Fram, E. S. Deutsch. On the quantitative evaluation of edge detection schemes and their comparison with human performance // *IEEE Trans. Computers,* Vol. 24, No 6, 1975, pp. 616-628.

14. I.V. Gribkov, P.P. Koltsov, N.V. Kotovich, A.A. Kravchenko, A.S. Kutsaev, A.S.Osipov, A.V. Zakharov. Edge Detection under Affine Transformations: Comparative Study by PICASSO 2 System // *WSEAS Transactions on Signal Processing*, Issue 9, Vol. 2, September 2006, pp. 1215-1221.

15. J.H. Han, T.Y. Kim. Ambiguity distance: an edge evaluation measure using fuzziness of edges // *Fuzzy sets and systems,* Vol. 126, No 3, 2002, pp. 311-324.

16. M. D. Heath, S. Sarkar, T. Sanocki, and K. Bowyer. Robust visual method for assessing the relative performance of edge detection algorithms // *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 12, 1997, pp. 1338–1359.

17. G. Jäger, U Benz. Measures of classification accuracy based on fuzzy similarity // *IEEE Trans. on Geoscience and Remote Sensing,* Vol. 38, No. 3, 2000, pp. 1462-1467.

18. T. Kanaguro, M. Y. Jaisimha, J. Palmer, R. M. Haralick. A Methodology for quantitative performance evaluation of detection algorithms // *IEEE Trans. Image Processing,* Vol. 4, No 12, 1995, pp. 1667-1674.

19. L. Kitchen, A. Rosenfeld. Edge evaluation using local edge coherence // *IEEE Trans.Systems,Man and Cybernetics,* Vol. 11, No 9, 1981, pp. 597-605.

20. L.R. Liang, C. Looney. Competitive fuzzy edge detection // *Applied Soft Computing*, Vol.3, 2003, pp. 123-137.

21. R. Maini, H. Aggarwal. Study and comparison of various image edge detection techniques // *International Journal of Image Processing*, Vol. 3, No 1, 2009, pp. 1-11.

22. R.Medina-Carnicer, F.J.Madrid-Cuevas, N.L.Fernandez-Garcia, A.Carmona-Poyato. Evaluation of global thresholding techniques in non-contextual edge

detection // *Pattern Recognition Letters,* Vol. 26, No. 10,2005, pp. 1423-1434.

23. A. Osipov. A fuzzy approach to performance evaluation of edge detectors - // in Lecture Notes in Signal Science, Internet and Education, WSEAS Press, 2007, pp. 94-99.

24. S.K. Pal. A measure of edge ambiguity using fuzzy sets//*Pattern Recognition Letters*, V.4, No 1,1986, p.51-56.

25. S.K. Pal, D.D. Madjumadar. Fuzzy Mathematical Approach to Pattern Recognition, New York: Wiley, 1986.

26. P.L. Palmer, H. Dabis, J. Kittler. A performance measure for boundary detection algorithms // *Computer Vision and Image Understanding,* Vol. 63, No 3, 1996, pp. 476-494.

27. T. Peli, D. Malah. A study of edge detection algorithms // *Computer Graphics and Image Processing,* Vol. 20, No 1, 1982, pp. 1-21.

28. W.K. Pratt. Digital image processing, John Wiley and Sons, New York, 1977.

29. M.S. Prieto, A.R. Allen. A similarity metric for edge images // *IEEE Trans. Pattern Anal. Mach. Intell.* Vol. 25, No 10, 2003, pp. 1265-1273.

30. V. Ramesh, R.M. Haralick. Performance characterization of edge detectors // *SPIE,* Vol. 1708, 1992, pp. 252-266.

31. B.D. Ripley. Statistics, images and pattern recognition // *Canadian J. Statist.,* Vol. 14, No. 2, 1986, pp. 83-102.

32. M. Roushdy. Comparative study of edge detection algorithms applying on the grayscale noisy image using morphological filter // *Int. Journal on Graphics, Vision and Image Processing (GVIP),* V. 6, No. 4, 2006, p. 17-23.

33. J. Serra, Image analysis and mathematical morphology. Academic Press, London, 1982.

34. M.C. Shin, D. Goldgof, K. Bowyer. Comparison of edge detector performance through use in an object recognition task // *Computer Vision and Image Understanding,* Vol. 84, No 1, 2001, pp. 160-178.

35. R.N. Strickland, D.K. Cheng. Adaptable edge quality metric // *Optical Engineering,* Vol. 32, No 5, 1993, pp. 944-951.

36. H.R. Tizhoosh, Fuzzy Image Processing: Introduction in Theory and Applications, Springer-Verlag, 1997 (in German).

37. L. J. van Vliet, I.T. Young, L.D. Beckers. A nonlinear Laplace operator as edge detector in noisy images // *Computer Vision, Graphics and Image Processing,* Vol. 45, No. 2, 1989, pp. 167-195.

38. D.L. Wilson, A.J. Baddeley, R.A. Owens. A new metric for grey-scale image comparison // *Internat. J. Comput. Vision,* Vol. 24, No. 1, 1997, pp.5-17.

39. Y.Yitzhaky, E.Peli. A method for objective edge detection evaluation and detector parameter selection // *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 25, No 8, 2003, pp. 1027-1033.

40. Q. Zhu. Efficient evaluations of edge connectivity and width uniformity // *Image and Vision Computing,* Vol. 14, 1996, pp. 21-34.

On some methods of comparative study of edge detectors

P.P. Koltsov, N.V. Kotovich, A.A. Kravchenko, A.S. Kutsaev, A.S. Osipov and A.V. Zakharov

Abstract: In the present paper the main approaches to performance evaluation of edge detectors are considered. We focused on the methods of quantitative evaluation which use the ground truth images and discrepancy measures. The advantages and disadvantages of the most usable discrepancy measures and the peculiarities of their joint application are studied. The evaluation methods based on the fuzzy set theory are considered, including an approach based upon the use of fuzzy ground truth images.

“Области влияния” и их использование для тестирования энергетических методов реставрации изображений

И.В. Грибков

кандидат физико-математических наук

Одним из важных этапов обработки изображений является реставрация изображений. Мы рассматриваем энергетические методы реставрации, в которых реставрированное изображение получается как результат минимизации некоторого функционала, имеющего интегральную форму. Обилие этих методов порождает сложную задачу выбора метода, оптимально подходящего для решения конкретной задачи. Определяемые в данной статье ”области влияния” являются инструментом, позволяющим установить существенные свойства методов реставрации, в частности, их способность снижать уровень зашумления изображения и сохранять четкие границы. Методика тестирования с использованием таких областей в данной статье применяется для исследования двух известных методов реставрации изображений. В частности, находятся параметры методов реставрации, при которых эти методы работают наиболее качественно.

1. Введение

Большинство изображений, полученных оптическими, электронными, или электрооптическими средствами, обычно будет ухудшено (деградировано). Деградация может произойти из-за расфокусирования камеры, относительного движения камеры и объекта, шума в электронных датчиках, атмосферной турбулентности, и т.д. Целью восстановления изображения является получение относительно “чистого" изображения из деградированного. Для этого приходится привлекать методы фильтрации, уменьшения шума и т.д. В некоторых случаях коррекция образа на компьютере – единственный способ получения приемлемых изображений.

Реставрация изображений – искусство и наука улучшения качества образа, базирующаяся на некоторых абсолютных мерах [1]-[14]. Обычно этот процесс включает некоторые средства устранения наложенных искажений, таких как размытие и зернистость. Это не может быть сделано идеальным образом, однако в некоторых случаях возможны значительные улучшения.

Реставрация изображений может осуществляться как с использованием априорных сведений о процессе деградации, так и без таковых. Если априорные сведения имеются – например, известны оптические искажения, вносимые видеокамерой, или известны шумовые характеристики канала связи, - то для улучшения изображений чаще всего используют шумовые фильтры с заданными характеристиками, или осуществляют свертку изображения с известной корректирующей функцией. Если же таких сведений нет, то приходится применять более универсальные методы.

Такими методами являются энергетические методы реставрации изображений, рассматриваемые в этой статье. В энергетических методах новое, улучшенное изображение ищется как решение задачи минимизации некоторого функционала. Известно [8], [9], что такие методы очень хороши для решения задачи о снижении уровня шума на реставрируемом изображении. В частности, энергетические методы позволяют:

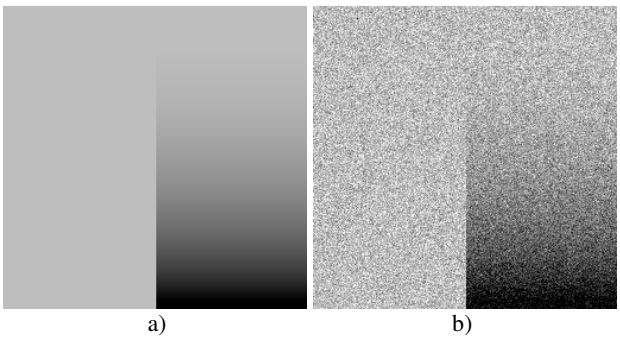
а) существенно снизить уровень шума в тех областях изображения, где яркость изображения изменяется плавно;

б) сохранить без размытия достаточно контрастные перепады яркости, то есть, границы на изображении.

Такие свойства энергетических методов, несомненно, делают их привлекательными для детального исследования их свойств и качества их работы.

В данной статье мы ограничиваемся рассмотрением шумопонижающих свойств энергетических методов реставрации изображений. Данные свойства изучаются на тестовом изображении следующего вида (Рис.1).

Изображение взято из базы данных системы PICASSO [7]. На этом изображении имеется граничная линия, причем контраст границы максимален в нижней части изображения и минимален в его верхней части. Как оказывается [8], качество работы энергетических методов реставрации очень существенно зависит от уровня контраста границы, и данное изображение позволяет эффективно исследовать особенности работы таких методов.



a) b)

Рис. 1. Неискаженное изображение (a) и зашумленное изображение $I(x,y)$ (b).

В этой статье рассматриваются следующие вопросы:

а) поведение энергетических методов реставрации в тех областях изображения, где яркость изменяется плавно (нет граничных линий);

б) поведение энергетических методов реставрации при приближении исследуемых точек к границе с переменным контрастом;

в) оценка параметров методов, при которых достигается качественное снижение шума и сохранение границ.

С этой целью вводится понятие "областей влияния", которые в данном частном случае снижения шума можно интерпретировать как адаптивные окна сглаживания изображения. Оказывается, что вдали от границ такие области велики, и в силу сглаживания внутри большой области происходит эффективное подавления шума на изображении. Вблизи границ такие области имеют малый размер, в силу чего сглаживания практически не происходит и обеспечивается сохранение резких границ. Предлагаются методики, которые позволяют визуализировать эти области. Анализ особенностей таких областей влияния позволяет подобрать параметры методов реставрации, приемлемые для решаемой конкретной задачи.

Структура статьи следующая.

В разделе 2 вводятся обозначения, принятые в данной статье и приводятся формулы функционалов двух энергетических методов. Эти функционалы обозначаются как PL (Piecewise Linear, то есть, функционал с кусочно-линейной минимизирующей функцией, [15]) и GR (Geman-Reynolds, [6], [9]) , которые далее исследуются с помощью "областей влияния".

В разделе 3 описываются два типа "областей влияния" и даются методики для их вычисления.

В разделе 4 даны результаты исследования функционала PL.

В разделе 5 даны результаты исследования функционала GR.

В разделе 6 на основании результатов разделов 5 и 6 делаются выводы выборе параметров методов PL и GR для качественного подавления шума на изображении и одновременном сохранении контрастных границ.

2. Обозначения и тестируемые функционалы

В данном разделе даются обозначения, которые далее используются в статье, приводится общий пример тестируемого энергетического метода и указывается ситуация, в которой возникает понятие "областей влияния"

2.1. Обозначения

Зафиксируем обозначения, которые будут использоваться во всей статье.

Здесь и далее мы ограничиваемся монохромными изображениями, так что яркость изображения в каждой точке полностью описывает все изображение. Координаты точки на изображении суть (x,y). Чаще всего, изображение задано в некотором прямоугольнике: $0 \le x \le T_1\text{-}1$, $0 \le y \le T_2\text{-}1$, где T_1, T_2 – размеры изображения соответственно по горизонтали и по вертикали. Тем самым, величины (x,y) принимают целочисленные значения. Значения функции яркости I обычно также целочисленные и во многих задачах обработки монохромных изображений принимают значения в диапазоне 0..255.

Далее, (x,y) – координаты точки как на исходном, так и на зашумленном и на реставрированном изображениях. Во всех случаях координаты x и y изменяются в одних и тех же пределах. $I(x,y)$ – функция яркости зашумленного исходного изображения, которое представлено на Рис.1.b. В результате реставрации изображения Рис.1.b получается изображение $u(x,y)$. Процедура реставрации состоит в том, чтобы, имея изображение $I(x,y)$, получить изображение $u(x,y)$, по возможности наиболее близкое к неискаженному изображению на Рис.1.a.

2.2. Тестируемые функционалы

Мы рассматриваем энергетические функционалы вида:

$$E(u) = w_1 \int_{\Omega} (u - I)^2 + w_2 \int_{\Omega} f(\| \nabla u \|^2), \qquad (1)$$

где $f(r) \ge 0$, $\nabla u = (u_x, u_y)$ – градиент функции изображения u, и $\| \nabla u \| = \sqrt{(u_x)^2 + (u_y)^2}$ – норма градиента. В дискретных вычислениях градиент находят путем численного дифференцирования.

Здесь $E(u)$ – минимизируемая энергия, а w_1 и w_2 – веса, с которыми оба члена выражения входят в $E(u)$. Обычно принимают $w_1 + w_2 = 1$. Оба члена, которые входят в выражение для $E(u)$, являются неотрицательными. Таким образом, на решении $u(x,y)$, минимизирующем $E(u)$, оба члена малы. Малость первого слагаемого означает, что изображение $u(x,y)$ близко к $I(x,y)$, в силу чего вообще можно говорить о реставрации изображения $I(x,y)$. Второе слагаемое в выражении для $E(u)$ накладывает ограничения на градиент $u(x,y)$. Какого рода эти ограничения – целиком зависит от функции f.

2.3. Конкретные формулы тестируемых функционалов

Для выполнения реставрации $I(x,y)$ мы рассматриваем методы, которые называются энергетическими. В энергетических методах реставрированное изображение $u(x,y)$ находится как решение задачи минимизации функционала вида (1), который представлен в интегральной форме. В данной статье мы рассматриваем два вида функционалов:

а) функционал с кусочно-линейной минимизирующей функцией; далее он обозначается как PL (Piecewise Linear), [15];

б) модифицированный функционал Гемана-Рейнольдса; далее он обозначается как GR [6], [9].

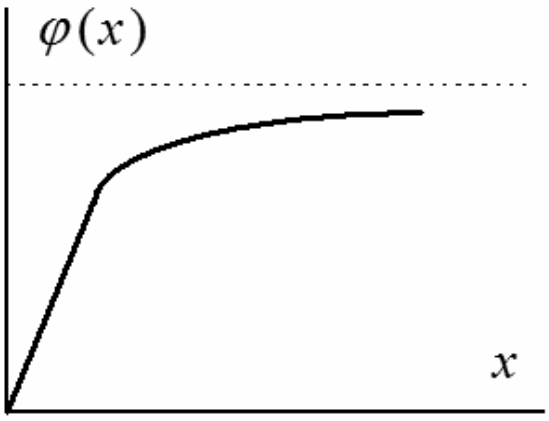


Рис. 2. Гладкая функция φ.

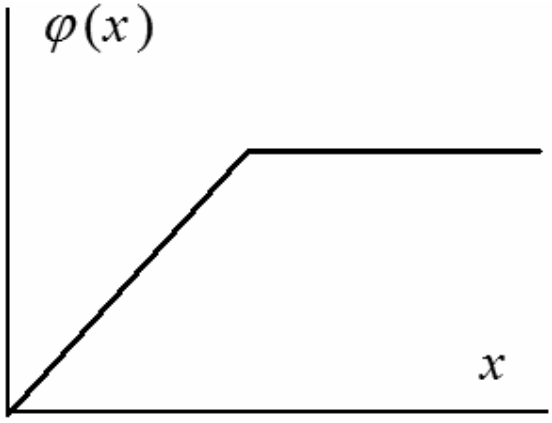


Рис. 3. Кусочно-линейная функция φ.

Для дальнейшего нам будет удобно переписать формулу (1) в виде:

$$E(u) = (1-\lambda)\int_{\Omega}(u-I)^2 + \lambda\int_{\Omega} c^2\varphi\left(\left(\frac{\|\nabla u\|}{c}\right)^2\right), \quad (2)$$

где $c^2 = \mu^2 \max_{(x,y)\in\Omega} \|\nabla I(x,y)\|^2$ и $0 \le \varphi(\mathrm{r}) \le 1$.

При такой записи первый параметр $0 < \lambda < 1$ регулирует вес, с которым в формулу (2) входит каждый из интегральных членов.

Для корректных вычислений требуется, чтобы оба интегральных члена в (2) имели приблизительно одинаковый порядок величины по отношению к изменению контраста исходного изображения *I*. Это обеспечивается выбором коэффициента *c*, который линейно зависит от контраста *I*. Таким образом, константа $0 < \mu \le 1$ является вторым параметром задачи минимизации.

Функционал PL (Piecewise Linear). Функционалы вида (2) рассматривались в нашей работе [8] при тестировании методов реставрации изображений. Свойства таких функционалов существенно зависят от вида функции φ. В работе [15] рассматривались функции φ вида Рис.2, 3.

Такие функции быстро возрастают при малых *r*, что приводит к эффекту сглаживания изображения в силу ограничения величины градиента при решении минимизационной задачи. При больших *r* функции φ на Рис.2, 3 постоянны или медленно возрастают. Это приводит к эффекту сохранения резких границ, поскольку при больших значениях градиента не вносится существенного вклада в минимизируемый член в (2). При малых *r* функции возрастают быстрее, что ограничивает рост градиента изображения *u*. Поэтому происходит сглаживание изображения. В дальнейшем мы будем решать задачу (2) с кусочно-линейной функцией φ в виде (Рис.3):

$$\begin{aligned} &\varphi(r) = r, \quad \text{если} \quad 0 \le r \le 1, \\ &\varphi(r) = 1, \quad \text{если} \quad r > 1. \end{aligned} \quad (3)$$

Функционал GR (Geman-Reynolds). В работах [6], [9] рассматривались функционалы (2) с функцией

$$\varphi(r) = r/(1+r). \quad (4)$$

Как показано в [9], функционал (2) с такой функцией является модификацией известного функционала, разработанного Геманом и Рейнольдсом [6].

Минимизационная задача (2) решается стандартным методом итераций (например, [9]). В случае малых значений λ реставрированное изображение *u* близко к исходному изображению *I*. В случае больших значений λ реставрированное изображение *u* более гладкое. Однако, как оказалось в наших экспериментах, параметр λ в основном влияет на качество сходимости итерационного процесса. Хорошая сходимость получалась при λ=0.8 и λ=0.9. Поэтому далее в тестируемых методах мы ограничились такими значениями параметра λ. В то же время, значение параметра μ очень существенно влияет на результат вычислений *u*. Типичная ситуация такова: если μ мало (например, μ = 0.1..0.2), то хорошо сохраняются границы на зашумленном изображении, а подавление шума – слабое. Если μ велико (например, μ = 0.8..0.9), то подавление шума хорошее, но на реставрированном изображении сохраняются только наиболее контрастные границы. Это показано на Рис.4, где приведены результаты реставрации изображения Рис.1.b при λ=0.8 и различных значениях μ от 0.1 до 1.0 с шагом 0.1. Видно, что уровень зашумления изображения значительно снижен. Черная стрелка указывает (приблизительно) на ту точку, ниже которой сохраняется четкая, контрастная граница, а выше которой менее контрастная граница размывается. В частности, при μ=1 вся граница размывается полностью. Таким образом, значение μ=1 не представляет интереса для исследования, и далее тестируемые методы изучаются для значений μ от 0.1 до 0.9.

Области изображения на Рис.4 разделяются на три части. К первой области относятся те, на которые указывают черные стрелки (в сущности, это точки). Далее будем говорить, что в этих областях изображение имеет критический уровень контраста. Если уровень контраста ниже критического, то на реставрированном изображении четкая граница не сохраняется, а если выше – сохраняется. Далее будет видно (Разделы 4, 5), что при критических значениях контраста методы реставрации ведут себя очень неустойчиво. К двум другим областям Рис.4 относятся те, которые имеют уровень контраста выше и, соответственно, ниже критического. Как правило, в этих случаях методы реставрации ведут

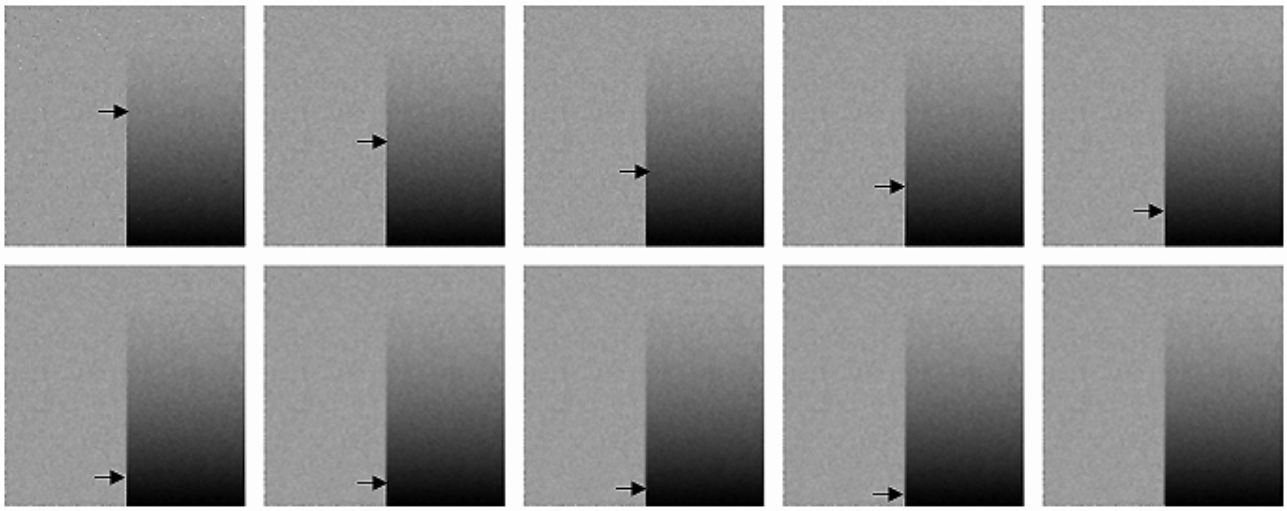


Рис. 4. Метод PL. Обработка изображения Рис.1.b. Здесь выбрано λ=0.8. Слева направо и сверху вниз: параметр μ изменяется от 0.1 до 1.0 с шагом 0.1.

себя предсказуемым образом: граница размывается в случае малого контраста и сохраняется в случае большого. Определяемая далее методика с использованием областей влияния позволяет достаточно точно найти критические значения контраста изображения.

3. Области влияния и методики их вычисления

Пусть на исходном изображении (Рис.1.b) имеется точка с координатами (x_0,y_0) и с яркостью в этой точке $I(x_0,y_0)$.

После выполнения процедуры реставрации мы получаем изображение того же размера, что и исходное, на котором также имеется точка с координатами (x_0,y_0). Значение яркости в этой точке равно $u(x_0,y_0)$. Естественно предположить, что основной вклад в вычисление величины $u(x_0,y_0)$ внесла точка (x_0,y_0) на исходном изображении Рис.5.a с яркостью $I(x_0,y_0)$. Возможно, некоторый вклад в вычисление величины $u(x_0,y_0)$ на Рис.5.b внесла «близкая» точка (x_1,y_1) с яркостью $I(x_1,y_1)$. В то же время, представляется крайне сомнительным, чтобы «удаленная» точка, такая, как (x_2,y_2) с яркостью $I(x_2,y_2)$ на Рис.5.a, могла бы внести существенный вклад в вычисление значения $u(x_0,y_0)$ на Рис.5.b. В самом деле, если бы это было так, то реставрированное изображение, скорее всего, просто не было бы похоже на исходное. С другой стороны, процедура реставрации с помощью интегрального функционала является решением глобальной минимизационной задачи, а решение $u(x,y)$ находится как минимум интеграла по всему исходному изображению. Тем самым, как «близкая» точка (x_1,y_1), так и «удаленная» точка (x_2,y_2) формально находятся в равном положении по отношению к точке (x_0,y_0) и в равной степени могут влиять на значение яркости $u(x_0,y_0)$. Однако этого не случается. Что же происходит во время процедуры реставрации?

В данной ситуации возникает естественная гипотеза: несмотря на то, что интегральный функционал глобален, реальное значение для вычисления $u(x_0,y_0)$ имеют только точки, близкие к (x_0,y_0). Иными словами, на значение $u(x_0,y_0)$ влияют точки, расположенные в некоторой окрестности точки (x_0,y_0) на изображении I. Естественно поставить вопрос о том, сколь велика такая окрестность и как она выглядит.

Далее окажется, что такая окрестность действительно существует, и внутри нее происходит сглаживание изображения – за счет чего происходит снижение уровня зашумления изображения. Однако такая окрестность адаптивна. Вне граничных перепадов яркости эта окрестность велика, а вблизи границ она мала. В результате, происходит эффективное снижение уровня шума в областях с медленно изменяющейся яркостью и одновременно с этим сохранение четких границ, если только они достаточно контрастны.

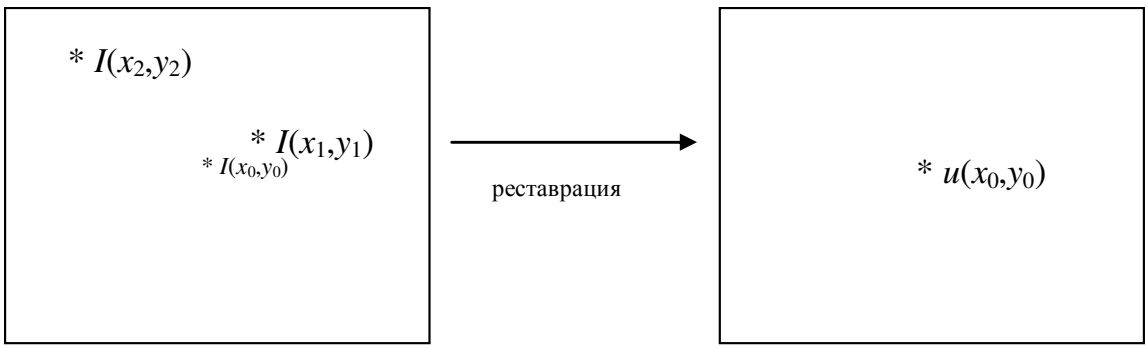


Рис. 5.a Исходное изображение

Рис. 5.b Реставрированное изображение

Окрестности, о которых говорилось выше, мы будем называть «областями влияния в энергетических методах реставрации», или просто «областями влияния». В дальнейшем этот термин будет употребляться без кавычек.

С нашей точки зрения, задача об определении подобной окрестности распадается на две части.

а) НА КАКИЕ ТОЧКИ реставрированного изображения u в окрестности (x_0,y_0) на u оказывает влияние изменение яркости в одной точке (x_0,y_0) исходного изображения I? Очевидно, что при некотором изменении значения $I(x_0,y_0)$ (после повторения процедуры реставрации), изменится яркость не только $u(x_0,y_0)$, но и некоторых точек вблизи (x_0,y_0) на u. Эти точки образуют окрестность точки (x_0,y_0) на u. *Данная окрестность располагается на реставрированном изображении u,* и мы ее будем называть *областью влияния первого рода*.

Отметим, что для решения этой задачи требуется иметь исходное и реставрированное изображения, а также измененное (в одной точке (x_0,y_0) исходное изображение) и, соответственно, реставрацию измененного исходного изображения. Таким образом, процедура реставрации производится два раза.

б) КАКИЕ ТОЧКИ исходного изображения I окрестности (x_0,y_0) на I оказывают влияние на изменение яркости в одной точке (x_0,y_0) реставрированного изображения u? Очевидно, что при некотором изменении значения яркости хотя бы одной точки в небольшой окрестности (x_0,y_0) на I, произойдет и изменение яркости $u(x_0,y_0)$ на u. Такие точки образуют окрестность точки (x_0,y_0) на I. *Данная окрестность располагается на исходном изображении I*, и мы ее будем называть *областью влияния второго рода*.

Отметим, что для решения этой задачи требуется иметь исходное и реставрированное изображения, а также много измененных исходных изображений в каждой точке окрестности (x_0,y_0) на I. Соответственно, реставрацию измененного исходного изображения необходимо проводить много раз, и в данном случае объем вычислений получается существенно больше, чем для вычисления области влияния первого рода.

3.1. Оценка областей влияния

Значения яркости исходного зашумленного изображения I обычно имеют целочисленные значения, в наиболее типичном случае принимающие значения от 0 до 255. Реставрированное изображение находится в процессе минимизации энергетического функционала (1), для чего необходимо решить дифференциальное уравнение численными методами. Решение находится при помощи итерационного процесса. Чтобы это было возможно, в первую очередь значения яркости I переводятся в вещественный формат, и все дальнейшие вычисления производятся в вещественном формате. В таком же формате получается и реставрированное изображение u. Далее, для изображения результата на экране компьютера, значения яркости u округляются.

Для оценки размера и формы таких областей влияния предлагается следующие методики. Пункты 1-3 относятся к вычислению областей влияния как первого, так и второго рода.

1. Выбирается зашумленное изображение, имеющее границы. В данной статье используется изображение Рис.1.b. Это изображение обозначим как I.
2. Производится реставрация этого изображения с помощью функционала (1). Полученное изображение обозначим как u_0.
3. Выбирается и фиксируется точка с координатами (x_0,y_0) как на I, так и на u_0.

3.1.1. Вычисление областей влияния первого рода

Напомним, что эти области находятся на реставрированном изображении u.

4. Производится изменение изображения I в точке (x_0, y_0). Именно, I заменяется на I_δ , где $I_\delta(x_0,y_0) = I(x_0,y_0) + \delta$, δ – некоторая малая вещественная величина, а в остальных точках I_δ совпадает с I.
5. Производится реставрация изображения I_δ. Полученное изображение обозначим как u_δ.
6. Строится изображение $D_1(x,y) = u_\delta(x,y) - u_0(x,y)$.

Полученное изображение D_1 является изображением области влияния первого рода. Значение яркости $D_1(x,y)$ соответствует степени влияния изменения изображения I в точке (x_0,y_0) на результат реставрации. Формально изображение D_1 имеет такой же размер, как изображение I_δ или реставрированное изображение u_δ. Однако в действительности I_δ изменяет значение яркости $u_\delta(x,y)$ лишь в малой окрестности (x_0,y_0) на u В силу этого далее из D_1 вырезается часть в окрестности (x_0,y_0), содержащая реальные различия между u_δ и u_0.

На Рис.6 показан пример вычисления областей влияния первого рода для метода PL. На всех рисунках значение $y_0 = 27$, а x_0, как показано вверху рисунков, принимает значения от 123 до 133. Черные стрелки на этих рисунках указывают на вычисленные области влияния, которые обозначены белыми точками. Все

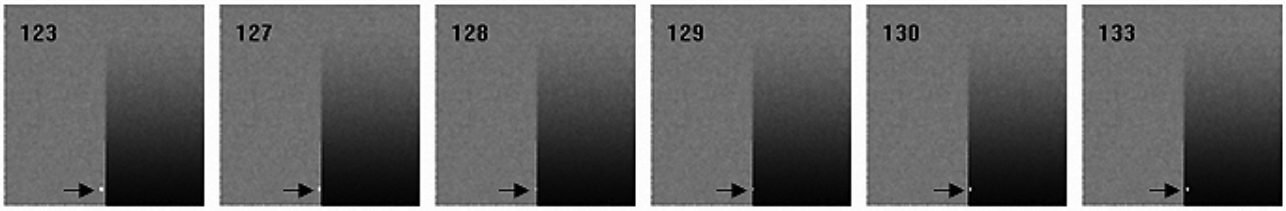


Рис. 6. Метод PL. $\lambda = 0.8$. $\mu = 0.3$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$. Черная стрелка указывает на “область влияния”.

рисунки имеют размер 256x256 пикселов (на Рис.6 уменьшено), граница изображений находится между 128 и 129 пикселами.

3.1.2. Вычисление областей влияния второго рода

Напомним, что эти области находятся на исходном изображении *I*.

4. Производится серия изменений изображения *I*. Именно, исходное изображение *I* заменяется на новые изображения $I_{\Delta x,\,\Delta y}$, где $I_{\Delta x,\,\Delta y}(x_0+\Delta x, y_0+\Delta y) = I(x_0+\Delta x, y_0+\Delta y)+\delta$, δ – некоторая малая вещественная величина, а в остальных точках $I_{\Delta x,\,\Delta y}$ совпадает с *I*. Значения Δx и Δy являются целыми числами с шагом 1.
5. Производится реставрация изображения $I_{\Delta x,\,\Delta y}$. Полученное изображение обозначим как $u_{\Delta x,\,\Delta y}$.
6. Вычисляется значение $D_2(\Delta x,\ \Delta y) = u_{\Delta x,\,\Delta y}(x_0,y_0) - u_0(x_0,y_0)$.

Для каждого значения $(\Delta x,\ \Delta y)$ производится изменение изображения *I* с одним и тем же значением δ и выполняется реставрация, после чего вычисляется $D_2(\Delta x,\ \Delta y)$. Затем строится изображение $D_2(\Delta x,\ \Delta y)$, в котором величины Δx и Δy являются координатами, причем значения $\Delta x=0$, $\Delta y=0$ соответствуют центру изображения D_2.

Полученное изображение D_2 является изображением области влияния второго рода. В нем отмечены все точки, изменение яркости в которых оказывали влияние на яркость реставрированного изображения в точке (x_0,y_0). Значение яркости $D_2(\Delta x,\ \Delta y)$ соответствует степени влияния изменения в точке $(x_0+\Delta x, y_0+\Delta y)$ на исходном изображении *I* на результат реставрации *u*.

3.2. Изображение областей влияния

Если нас интересует область влияния в другой точке (x_0,y_0), то вся процедура 1-6 повторяется для другой точки, и строится новое изображение D_1 или D_2. Поскольку удаленные точки не оказывают влияния на значение $u(x_0,y_0)$, достаточно рассматривать эти области в пределах нескольких единиц пикселов. Как оказалось в наших экспериментах, в большинстве случаев достаточно рассматривать окрестности точки (x_0,y_0) размером 11x11 пикселов для области влияния как первого, так и второго рода.

Значение δ выбирается из следующих соображений.

а) Изменение исходного изображения не должно существенно влиять на результат реставрации. Поэтому значение δ выбирается таким, чтобы при его уменьшении в несколько раз не происходили бы существенные изменения ни в размере, ни в форме областей влияния. Как показали наши эксперименты, при выборе $\delta = 1.0$ и $\delta = 0.1$ размер и форма найденных областей влияния практически не различаются, то есть, такие значения величины δ являются приемлемыми. Далее используется $\delta = 0.1$.

б) Исходное и измененное изображения должны обрабатываться по одной и той же схеме, чтобы поиск различий между двумя реставрированными изображениями был корректным. Так как реставрированное изображения ищется методом итераций, это означает, что в обоих случаях количество итераций должно быть одинаковым. При выборе $\delta = 0.1$ так и было во всех последующих измерениях.

Как показано на Рис.6, области влияния имеют очень малый размер. Для того, чтобы изобразить области влияния в приемлемом виде, их необходимо увеличить. Следующие пункты относятся к обработке областей влияния, как первого, так и второго рода.

7. Вычисляется функция $D_1(x,y)$ или $D_2(\Delta x,\ \Delta y)$. В случае $D_1(x,y)$ вырезается область размером 11x11 пикселов с центром в (x_0,y_0). В случае $D_2(\Delta x,\ \Delta y)$ берется область $-5 \le \Delta x,\ \Delta y \le 5$ пикселов.
8. Эта функция затягивается двумерным вещественным сплайном. После такой процедуры величины Δx, Δy могут принимать вещественные значения.
9. Затянутая сплайном функция D_1 или D_2 масштабируется так, чтобы диапазон изменения ее аргументов составлял 0..127. На всех последующих рисунках изображены области влияния, заключенные в квадраты размером 128x128 пикселов.
10. Берется негативное изображение $255 - D_{1,2}$. В этом случае те точки, которые вносят наибольшее воздействие на результат реставрации, выглядят черными, а наименьшее – белыми.
11. Яркости изображений $D_{1,2}$ шкалируются так, чтобы реальный диапазон яркостей при неотрицательных значениях $D_{1,2}$ составлял 0..255.
12. После масштабирования по размеру и шкалирования по яркости, значения $D_{1,2}$ округляются до целых. Оконтуриваются границы полученного изображения по равенству целых чисел нулю.
13. В центре изображения ставится маркерная точка для локализации центра.

3.3. Примеры

На Рис.7 показан типичный пример из серии рисунков, которая будет продемонстрирована далее.

Здесь и далее рисунок состоит из двух строк. В верхней строке показана область влияния D_1 первого рода. Как указано выше, для ее вычисления делается малое изменение исходного изображения в точке (x_0,y_0) на исходном изображении, и отслеживается, как это изменение отражается на реставрированном изображении. В данном примере выбран метод PL, при параметрах . $\lambda = 0.9$, $\mu = 0.3$. Исходное изображение зашумлено гауссовым шумом с девиацией 10. Здесь $y_0 = 27$, а x_0 принимают пять значений от 123 до 133, как и на Рис.6. Каждое изображение на Рис.7 имеет размер 11x11 пикселов, однако растянуто до размера 128x128 пикселов при помощи сплайна. Наибольший интерес представляет динамика областей влияния при приближении к границе, между пикселами 128 и 129 по оси абсцисс. Как видно из Рис.7, если тестируемая точка (например, (123, 27), верхнее левое изображение) дос-

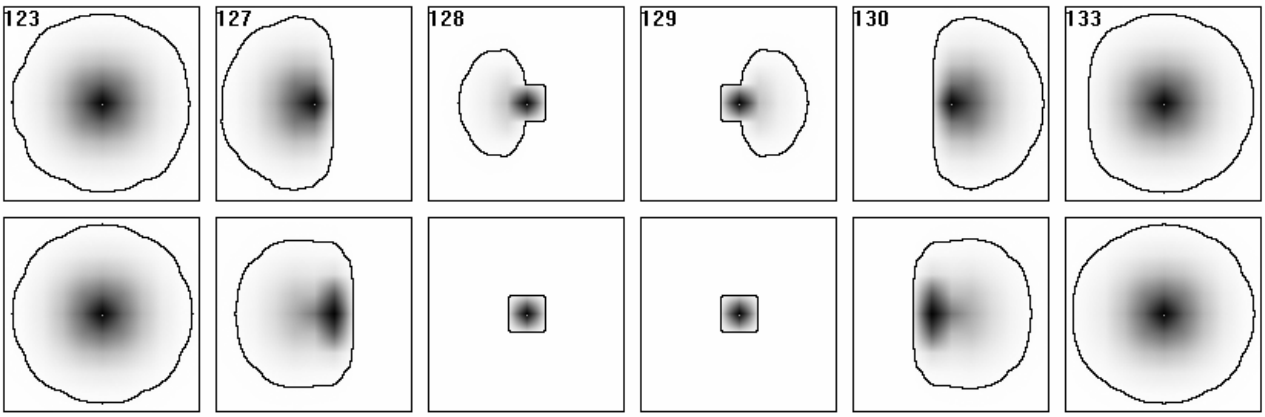


Рис. 7. Пример вычисления областей влияния для метода PL. $\lambda = 0.9$. $\mu = 0.3$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

таточно удалена от границы ($x_0 = 128$), то такая область влияния велика. По мере приближения к границе она уменьшается.

В нижней строке Рис.7 показана область влияния D_2 второго рода. Значения x_0 такие же, как в верхней строке. Как указано выше, для вычисления D_2 делается серия изменений исходного изображения в окрестности точки (x_0,y_0) на исходном изображении, и отслеживается, как эти изменения отражаются на одной точке (x_0,y_0) реставрированного изображения. Как видно из Рис.7, на значение яркости $u(x_0,y_0)$ оказывают влияние точки из $I(x_0,y_0)$, находящиеся в большой (практически 11x11 пикселов) окрестности точки (x_0,y_0) на исходном изображении (нижнее левое изображение). В то же время, при приближении к границе (x_0 = 128, 129) такая окрестность существенно уменьшается.

Благодаря такому явлению, в тех областях изображения, где нет граничных линий, происходит размытие изображения, и тем самым уменьшается уровень шума. Там же, где есть граничные линии, такого размытия не происходит, и сохраняются контрастные границы. Увеличение яркости в точке (x_0,y_0) на реставрированном изображении при увеличении яркости в той же точке или ее окрестности на исходном изображении выглядит естественным. В этом случае обеспечивается хорошее подавление шума и сохранение границ без искажения изображения. Однако реставрированное изображение получается как результат решения нелинейного дифференциального уравнения, и очевидный результат получается далеко не всегда. Приведем еще один пример (см. Рис.8).

Отличие от предыдущего примера состоит в том, что изменено значение параметров λ, μ и координата тестируемой точки y_0. Как выяснится в следующем разделе, незначительные изменения λ и y_0 не очень существенны, зато изменение μ вносит радикальные изменения в процесс реставрации. Серые области на Рис.8 означают, что изменение исходного изображения в окрестности точки (x_0,y_0) (увеличения яркости на величину $\delta = 0.1$) приводят не к увеличению яркости в соответствующих областях реставрированного изображения, а к ее уменьшению. Данное явление означает, что в некоторых точках реставрированное изображение искажается, и это можно отнести к ”артефактам реставрации” (далее будем называть эти серые области областями артефактов). Таким образом, при значении параметра $\mu = 0.1$ метод PL ведет себя неустойчиво, и данное значение параметра μ не следует использовать при реставрации изображений.

Из приведенных примеров можно сделать следующие выводы.

а) Определенные выше области влияния можно рассматривать как инструмент, позволяющий изучать

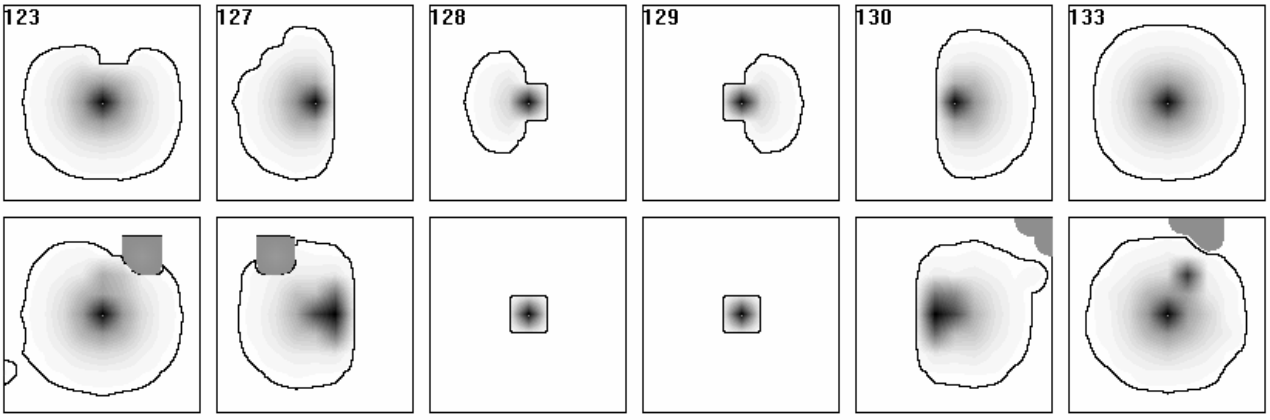


Рис. 8. Пример вычисления областей влияния для метода PL. $\lambda = 0.8$. $\mu = 0.1$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 25$. Серые области в нижнем ряду вне (x_0,y_0) указывают на артефакты реставрации.

особенности сложных энергетических методов реставрации изображений.

б) Анализ вида и размера областей влияния позволяет оценить пределы изменения параметров методов реставрации изображений, при которых такие методы хорошо подавляют шум, сохраняют контрастные границы на изображении и вносят минимальные искажения в изображение.

В следующих двух разделах более подробно исследуются методы PL и GR с помощью использования областей влияния первого и второго рода.

4. Метод PL

В этом разделе изучается метод PL (Piecewise Linear) с кусочно-линейной минимизирующей функцией, представленной на Рис.3 и функционалом (2). Эта функция задается формулой (3). Применение методики областей влияния дает возможность подробно исследовать свойства этого метода, в частности, найти значения его параметров, при которых он вносит наименьшие искажения в реставрируемое изображение.

Метод PL зависит от двух параметров, заданных в формуле (2): λ и μ. Если от параметра λ в основном зависит плавность сходимости метода, то от параметра μ в значительной степени зависят свойства понижения шума на изображении и сохранение контрастных границ. Поэтому далее рассматриваются значения $\lambda = 0.8$ и $\lambda = 0.9$, которые оказались оптимальными для обеспечения хорошей сходимости данного метода при реставрации с помощью итерационного процесса, и значения μ от 0.1 до 0.9.

Исходное изображение зашумлено по Гауссу. Далее мы рассматриваем разные степени зашумления, с девиацией шума от 0 до 30. В качестве "идеального" изображения, к которому далее добавлялся нормированный шум, было выбрано изображение Рис.1.a. Однако, для демонстрации методики областей влияния для тестирования выбранных функционалов, наиболее показателен уровень зашумления с девиацией, равной 10.

Все приводимые далее рисунки состоят из двух строк изображений. В верхней строке показаны области влияния первого рода. Эти области располагаются на реставрированном изображении и дают отклик реставрированного изображения на изменение исходного изображения в некоторой точке (x_0,y_0). В нижней строке показаны области влияния второго рода. Эти области располагаются на исходном изображении и показывают отклик реставрированного изображения в точке (x_0,y_0) при изменении исходного изображения в окрестности точки с теми же координатами. В наших экспериментах все области влияния ограничены размером 11x11 пикселов, однако для удобства визуализации их изображения растянуты до размера 128x128 пикселов.

Наибольший интерес представляет поведение методов реставрации, когда:

а) точки (x_0,y_0) приближаются к границе, расположенной между 128 и 129 пикселами изображения Рис.1.b. Эти точки выбраны так, что они находятся на горизонтальной линии и приближаются к границе. Их координаты x_0 соответственно равны 123, 127, 128, 129, 130 и 133. Значения этих координат указаны на рисунках;

б) при увеличении яркости исходного изображения в точке (x_0,y_0) происходит уменьшение яркости в окрестности точки реставрированного изображения. Это говорит об искажениях при реставрации;

в) точки (x_0,y_0) приближаются к области, отмеченной стрелкой на Рис.4. Ниже области хорошо сохраняются границы, выше – полностью размываются. Оказывается, и в этой области методы ведут себя неустойчиво.

4.1. Метод PL при малом значении параметра μ

На Рис.9-15 приводится серия изображений областей влияния при $\mu = 0.1$ и различных значениях λ и y_0. Искажение формы областей влияния по сравнению с областями на Рис.7 говорит об искажениях реставрированного изображения. Значительное количество точек, в которых уменьшается яркость (на Рис.9-15 закрашены серым цветом), также указывает на искажения границ. Можно признать, что при $\mu = 0.1$ метод PL неустойчив.

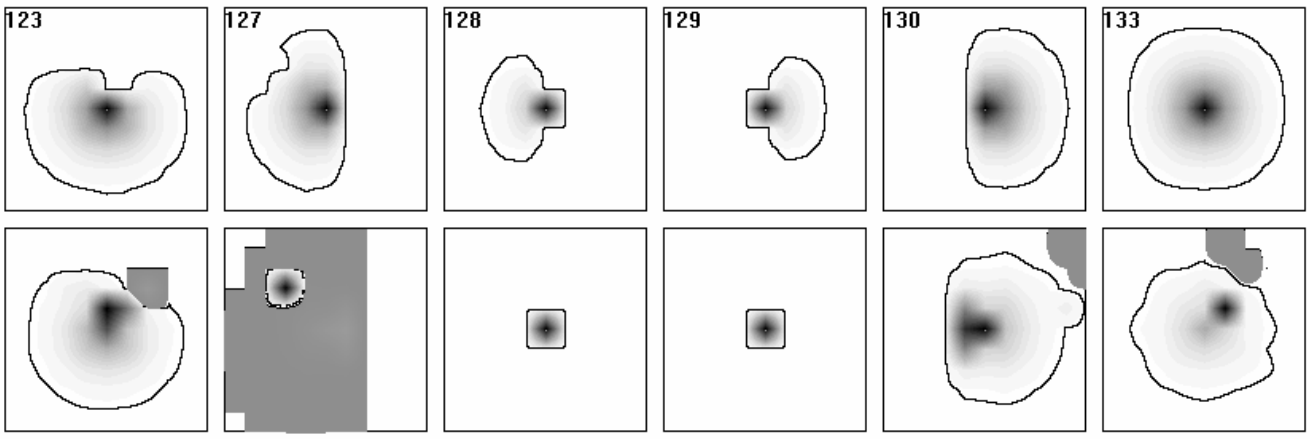


Рис. 9. Метод PL. $\lambda = 0.8$. $\mu = 0.1$. Девиация шума = 10. Ордината тестируемой точки $y_0 = 26$.

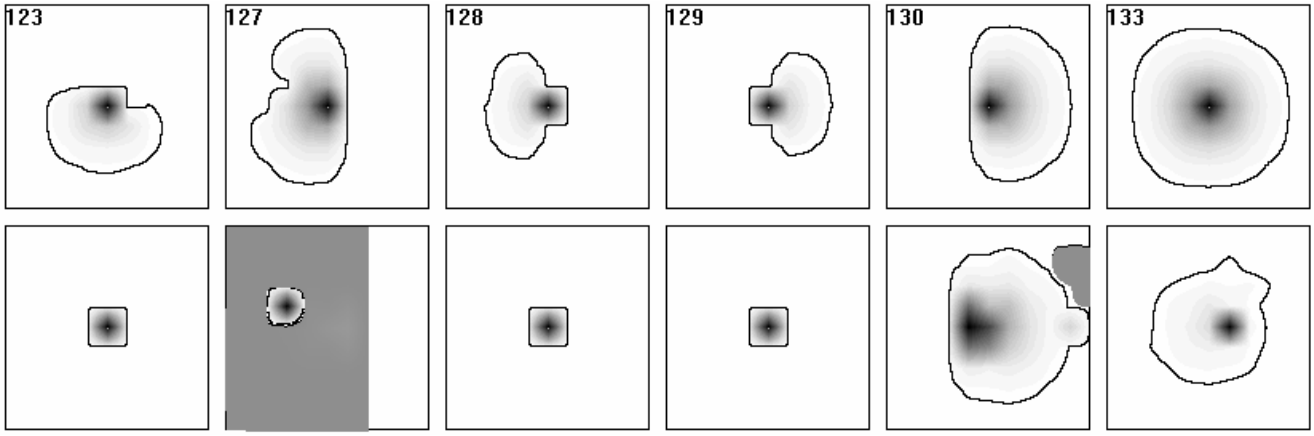


Рис. 10. Метод PL. $\lambda = 0.8$. $\mu = 0.1$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

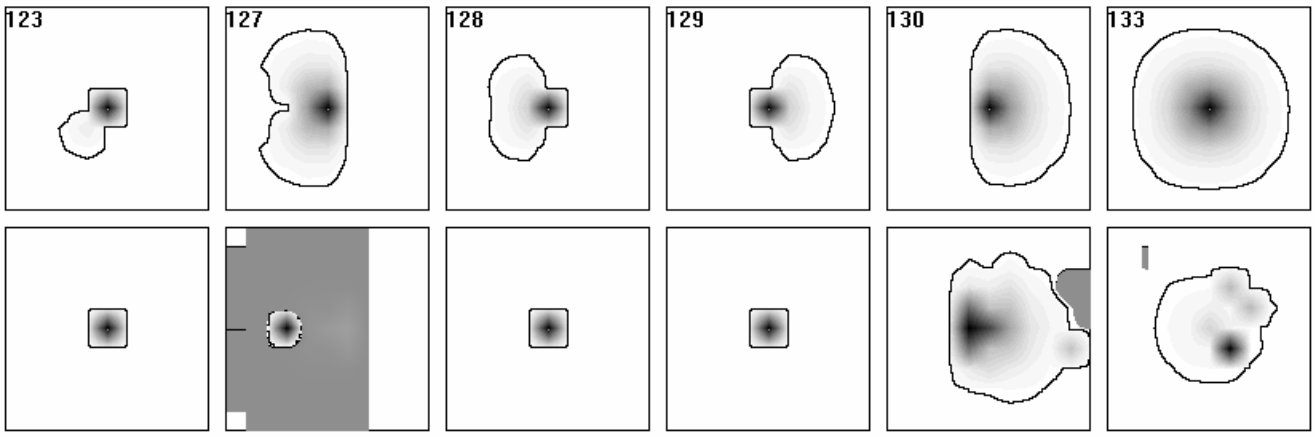


Рис. 11. Метод PL. $\lambda = 0.8$. $\mu = 0.1$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 28$.

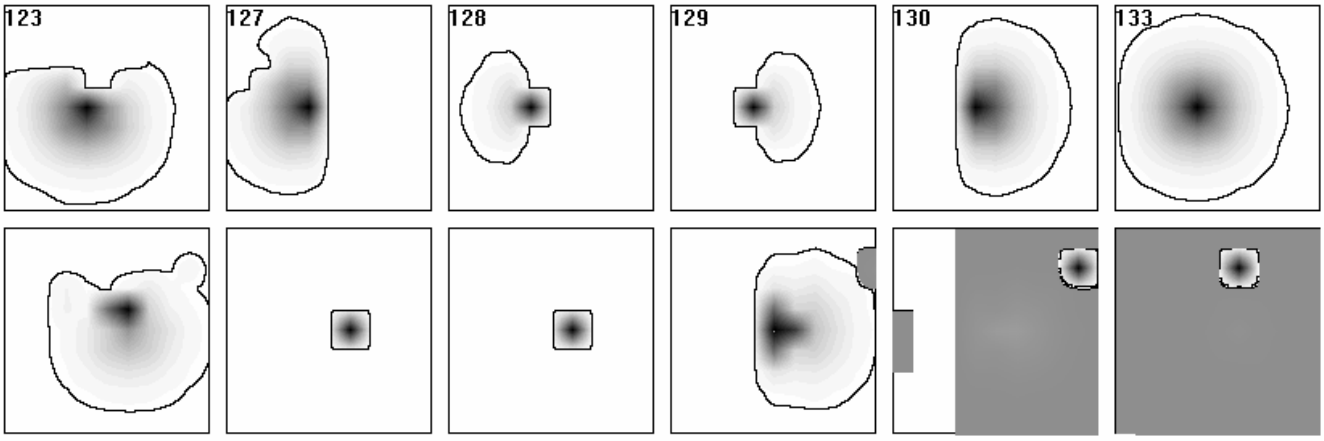


Рис. 12. Метод PL. $\lambda = 0.9$. $\mu = 0.1$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 26$.

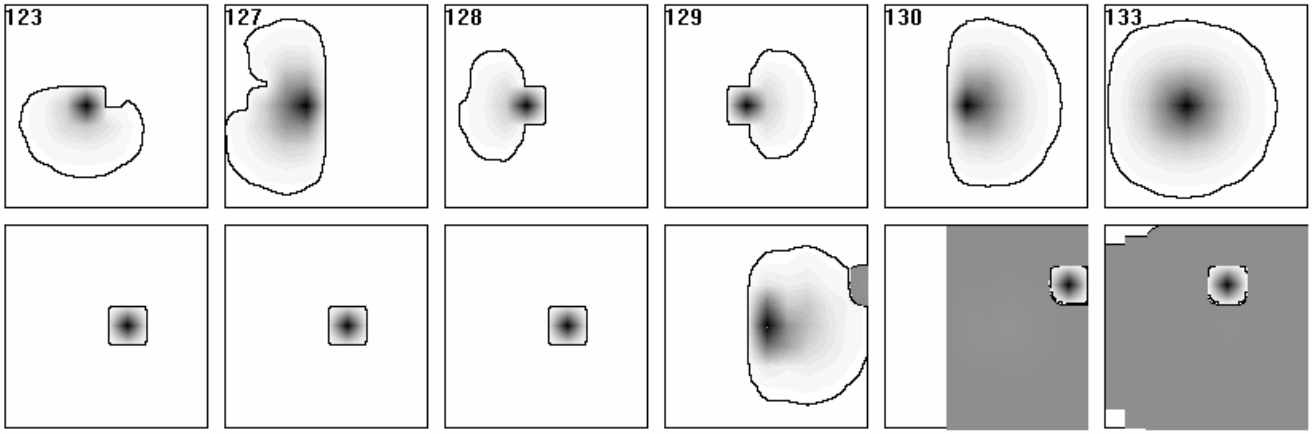


Рис. 13. Метод PL. $\lambda = 0.9$. $\mu = 0.1$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

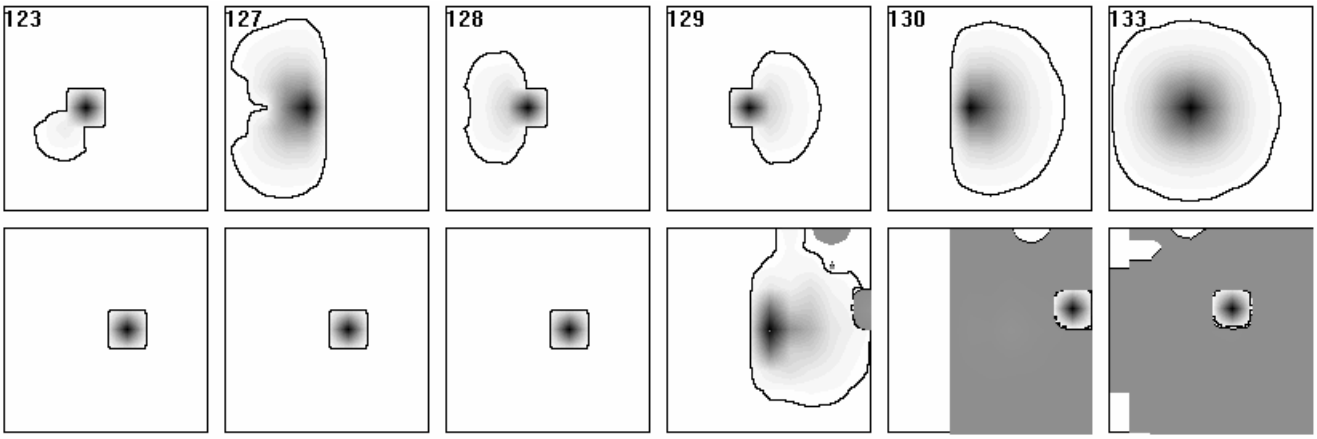


Рис. 14. Метод PL. $\lambda = 0.9$. $\mu = 0.1$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 28$.

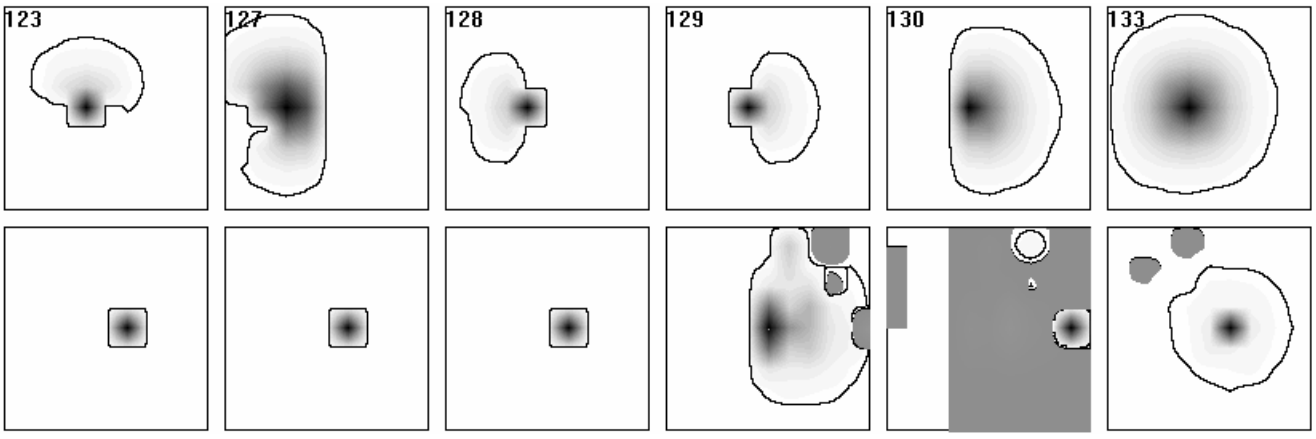


Рис. 15. Метод PL. $\lambda = 0.9$. $\mu = 0.1$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 29$.

4.2. Метод PL при средних значениях параметра μ

Средними являются значения μ = 0.2…0.8. Значения координаты x_0 выбраны такими же, как и ранее. При задании координаты y_0 полезно иметь в виду Рис.4, где показано, при каких значениях y_0 прекращается сохранение контрастной границы, в зависимости от параметра μ. На Рис.16-26 значение y_0 выбиралось равным 27 и 57 (Рис.17, 20). Как видно, области влияния, полученные для разных y_0, мало различаются. В соответствии с Рис.4, оба значения координаты y_0 находятся в той области, где сохраняется контрастная граница.

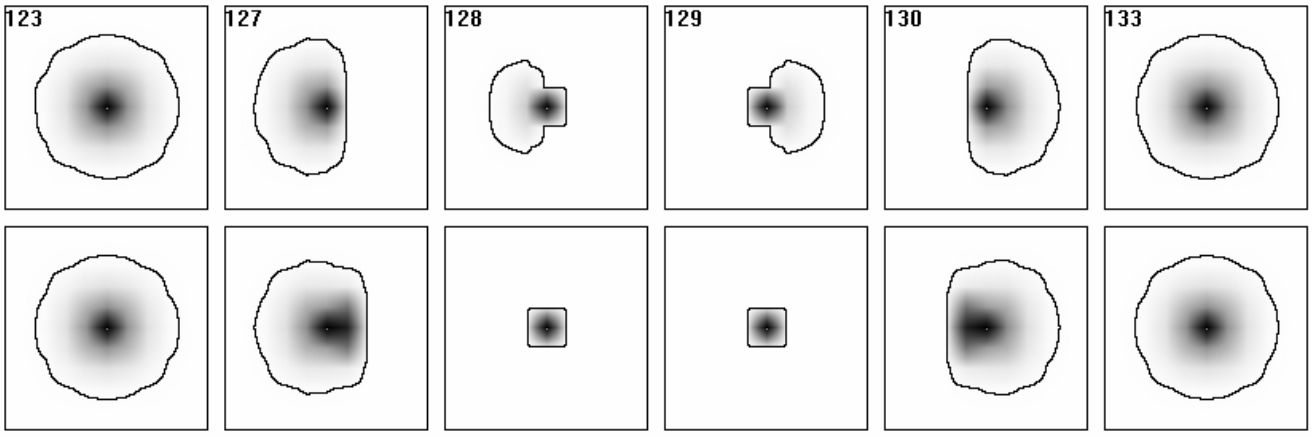


Рис. 16. Метод PL. λ = 0.8. μ = 0.2. Девиация гауссова шума = 10. Ордината тестируемой точки y_0 = 27.

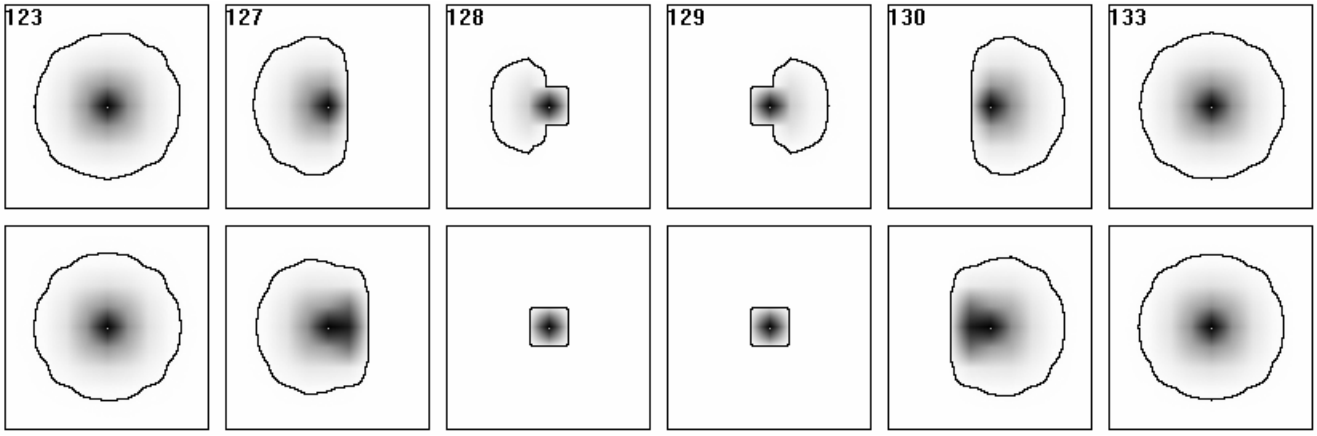


Рис. 17. Метод PL. λ = 0.8. μ = 0.3. Девиация гауссова шума = 10. Ордината тестируемой точки y0 = 57.

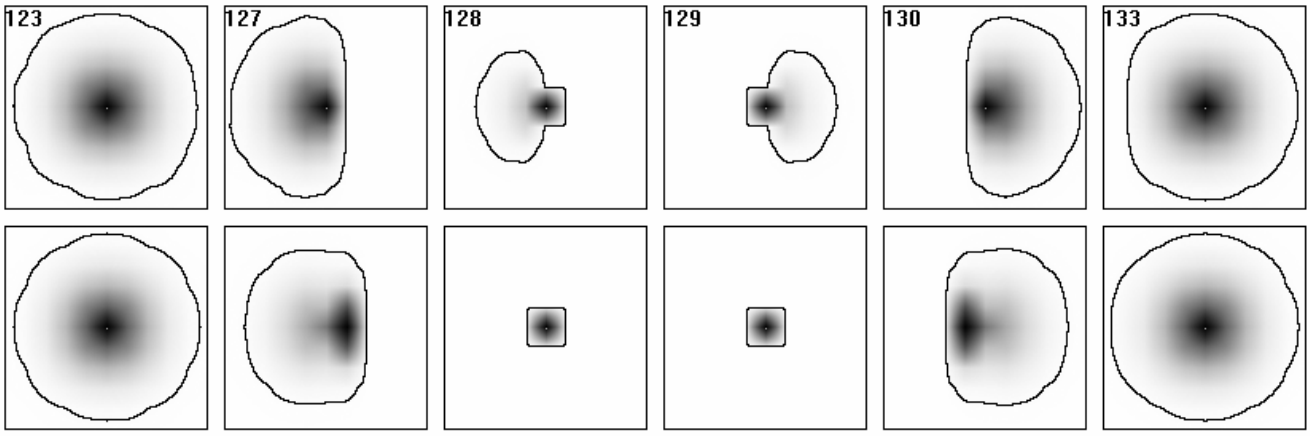


Рис.18. Метод PL. λ = 0.9. μ = 0.3. Девиация гауссова шума = 10. Ордината тестируемой точки y_0 = 27.

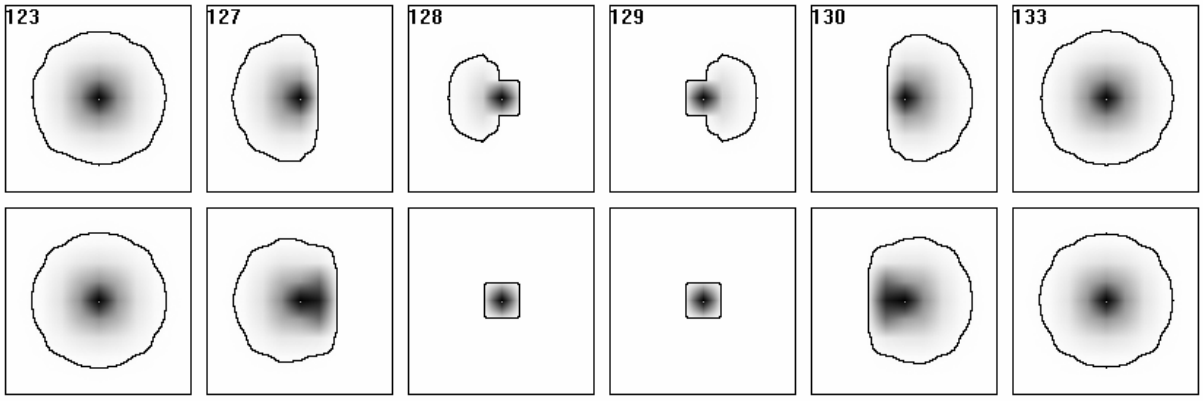


Рис. 19. Метод PL. $\lambda = 0.8$. $\mu = 0.3$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

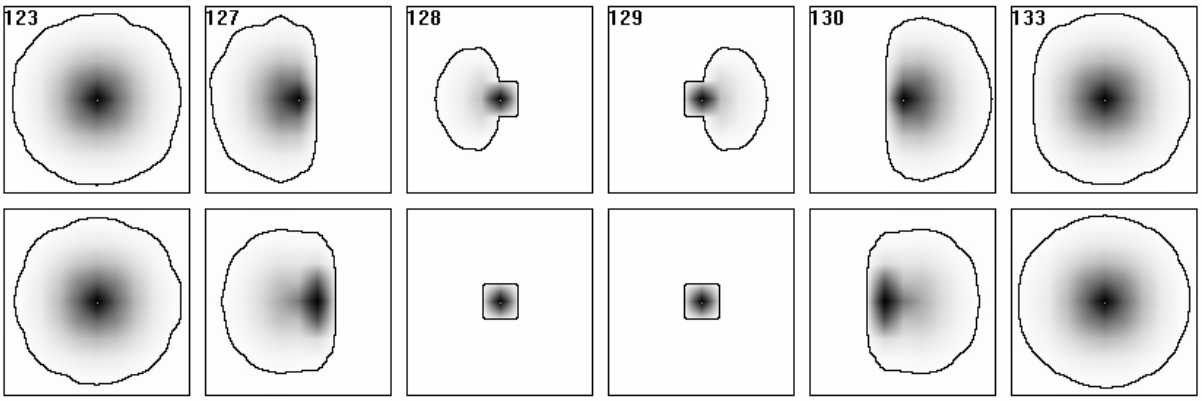


Рис. 20. Метод PL. $\lambda = 0.9$. $\mu = 0.3$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 57$.

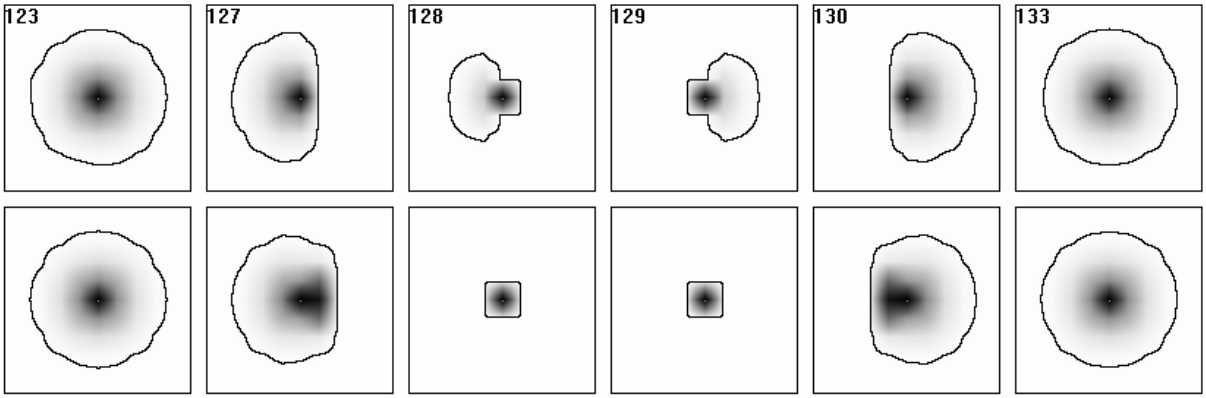


Рис.21. Метод PL. $\lambda = 0.8$. $\mu = 0.4$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

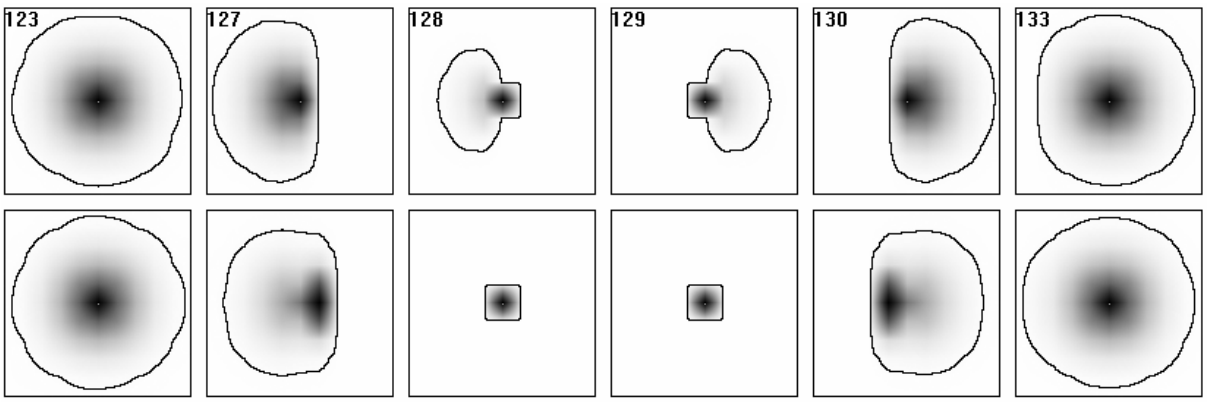


Рис.22. Метод PL. $\lambda = 0.9$. $\mu = 0.4$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

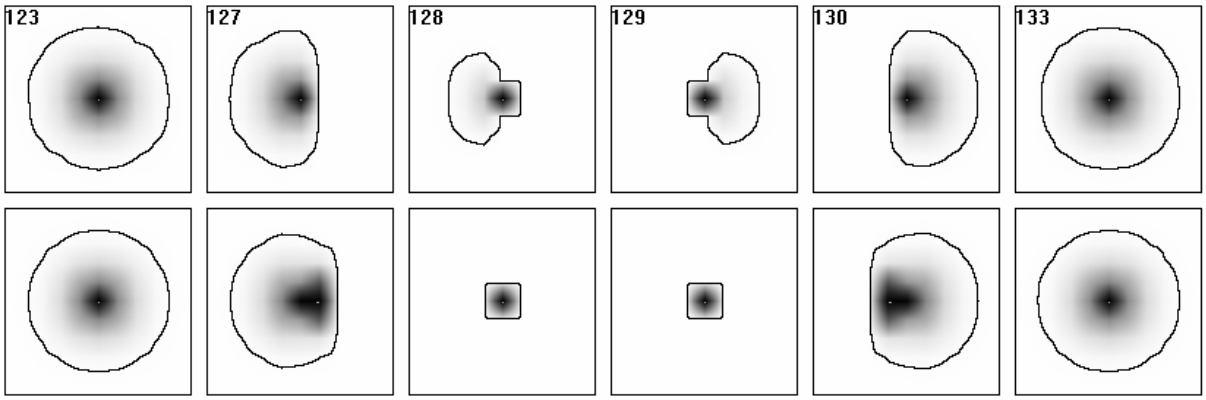


Рис. 23. Метод PL. $\lambda = 0.8$. $\mu = 0.5$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

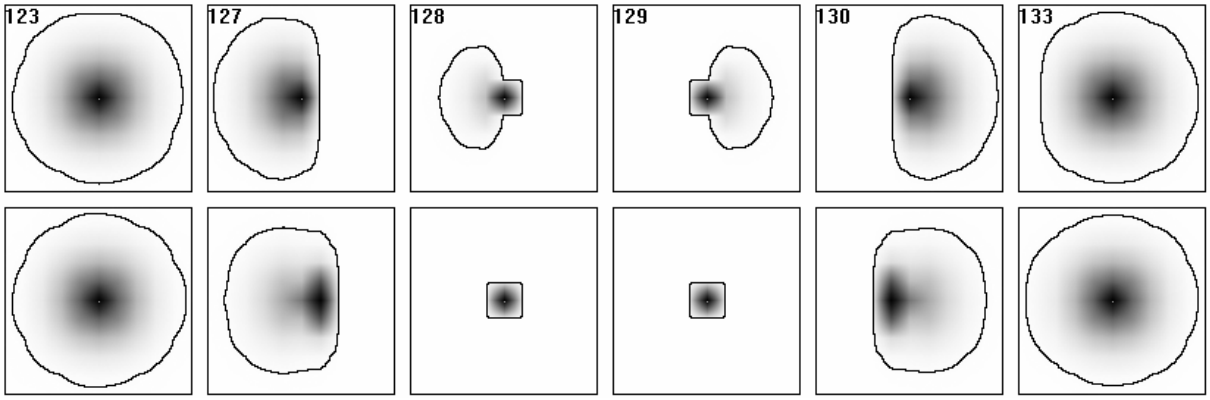


Рис. 24. Метод PL. $\lambda = 0.9$. $\mu = 0.5$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

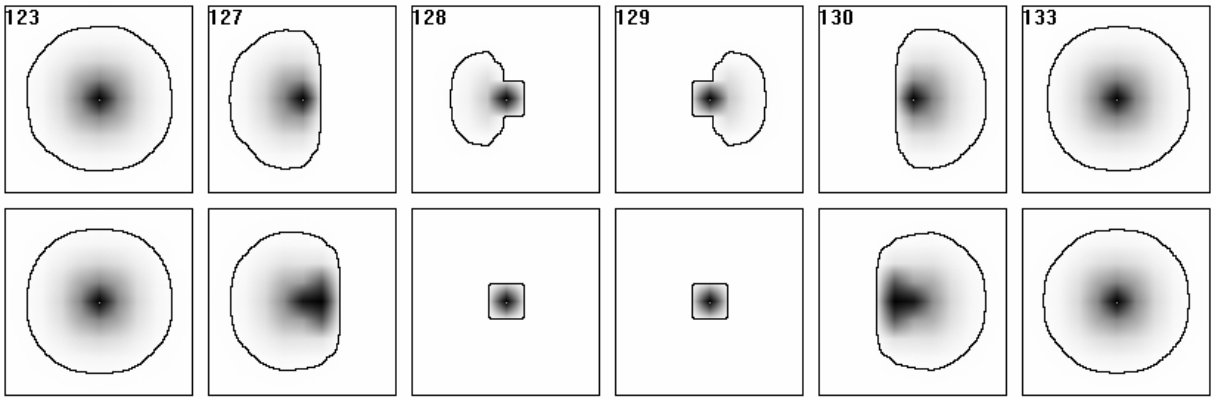


Рис. 25. Метод PL. $\lambda = 0.8$. $\mu = 0.6$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

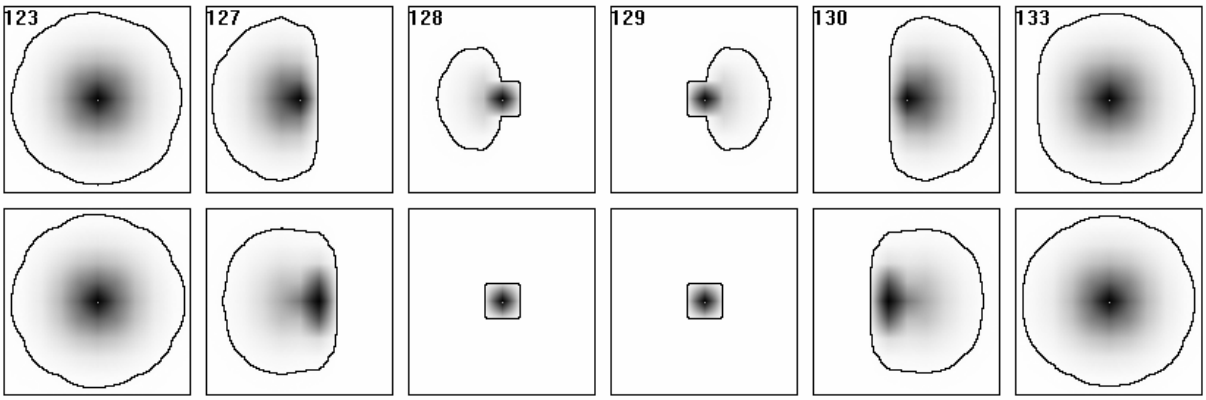


Рис. 26. Метод PL. $\lambda = 0.9$. $\mu = 0.6$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

Рассмотрим подробнее Рис.27-30. Для параметра $\mu = 0.7$ значение $y_0 = 27$ является как раз тем значением ординаты, при котором прекращается сохранение контрастной границы (см. Рис.4). Как видно из Рис.27, в этом случае метод PL дает резкий сбой. По виду областей влияния первого рода можно заключить, что происходит фактически размытие изображения, а области влияния второго рода (нижняя строка рисунков) вообще имеют практически произвольную форму. В случае значения $\lambda = 0.9$ при том же значении $\mu = 0.7$ происходит просто размытие изображения, то есть, (Рис.28), при этих параметрах y_0 находится выше того уровня, при котором контрастная граница еще сохраняется.

Если выбрать меньшее значение $y_0 = 12$, попадающее в область сохранения контрастных границ, то области влияния принимают ”традиционный” вид (Рис.29,30).

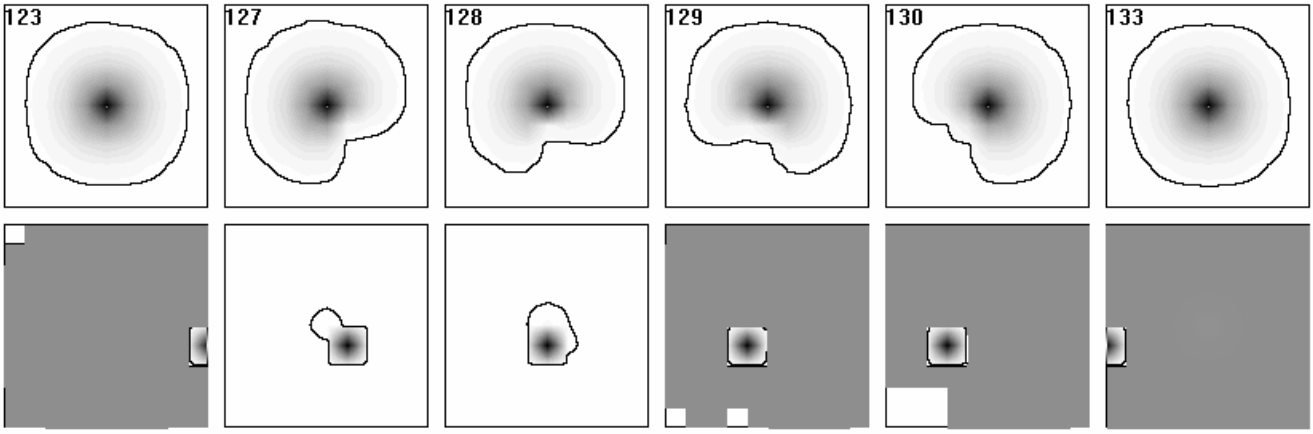


Рис. 27. Метод PL. $\lambda = 0.8$. $\mu = 0.7$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

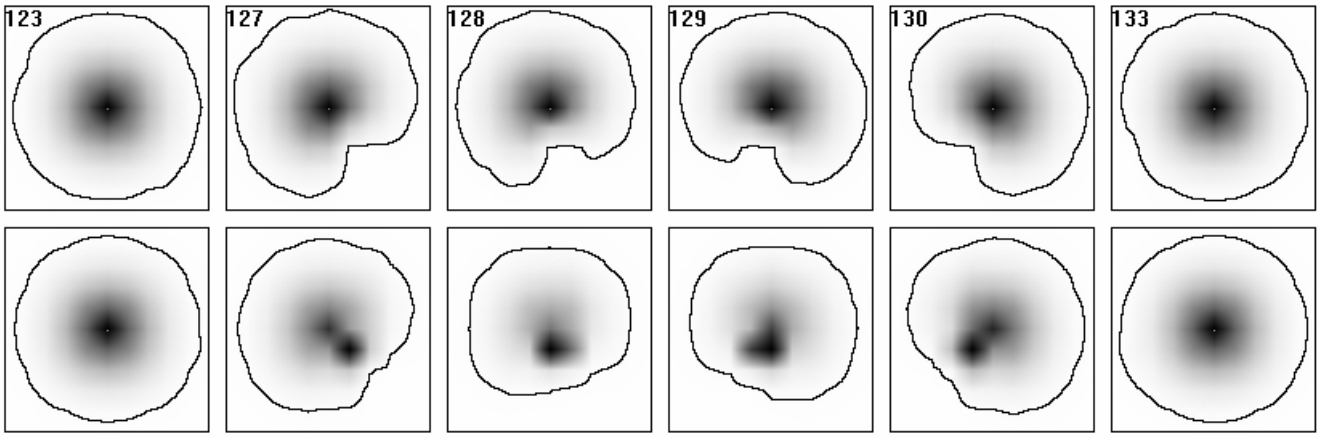


Рис.28. Метод PL. $\lambda = 0.9$. $\mu = 0.7$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

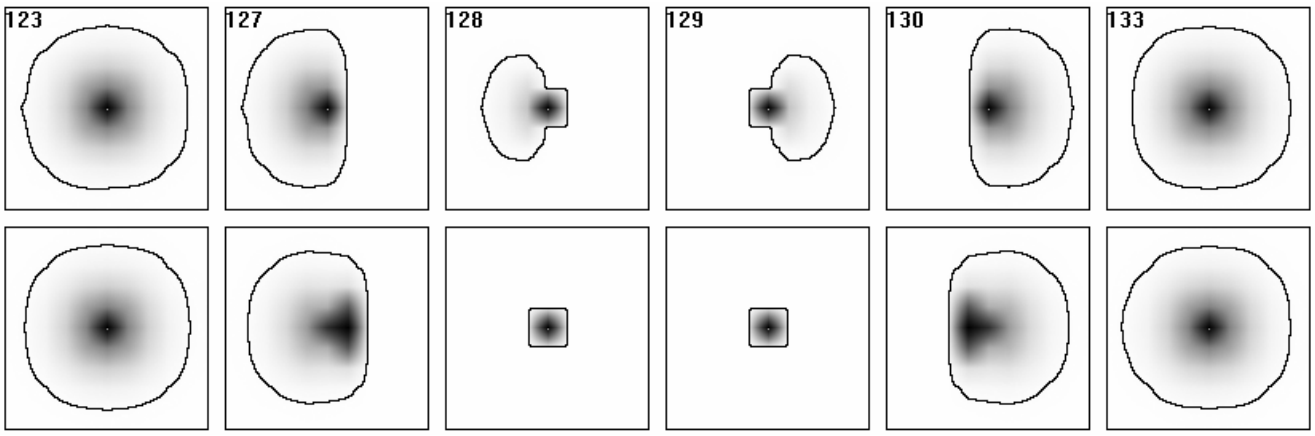


Рис.29. Метод PL. $\lambda = 0.8$. $\mu = 0.7$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 12$.

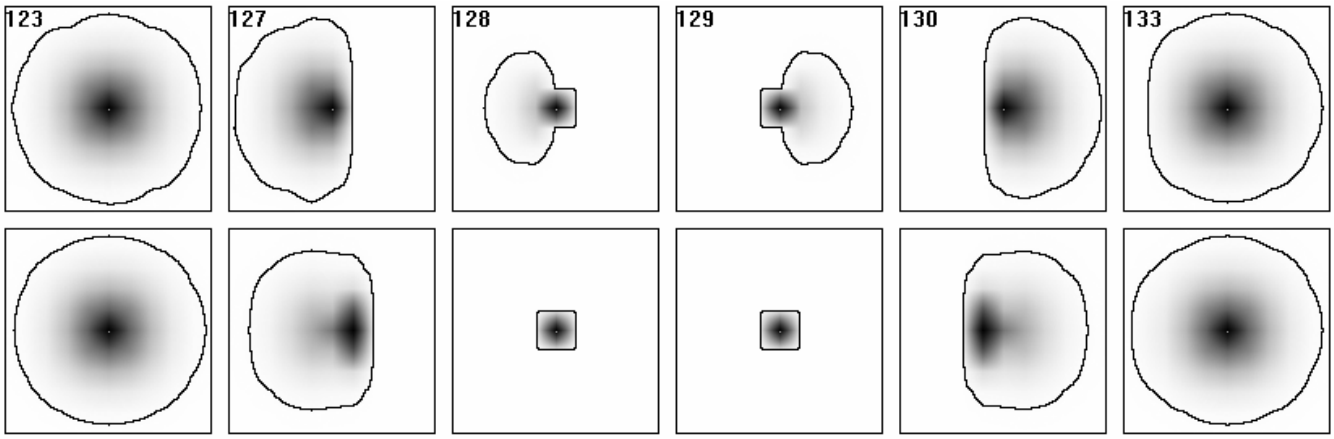


Рис. 30. Метод PL. $\lambda = 0.9$. $\mu = 0.7$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 12$.

Рассмотрим Рис.31-37. Для метода PL при $\mu = 0.8$ и $y_0 = 27$, y_0 попадает в область, где контрастные границы не сохраняются. На Рис.31, 32 показаны области влияния, внутри которых есть только размытие изображения. Ситуация меняется, если взять меньшее значение $y_0 = 14$. Однако такое значение y_0 является пограничным, выше которого внутри области влияния происходит полное размытие изображения, а ниже - еще сохраняются контрастные границы (см. Рис.4). Как и на Рис.27, здесь происходит искажение изображения. См. также Рис.36.

В случае выбора еще меньшего значения $y_0 = 10$, метод PL дает приемлемый результат (Рис.34,35).

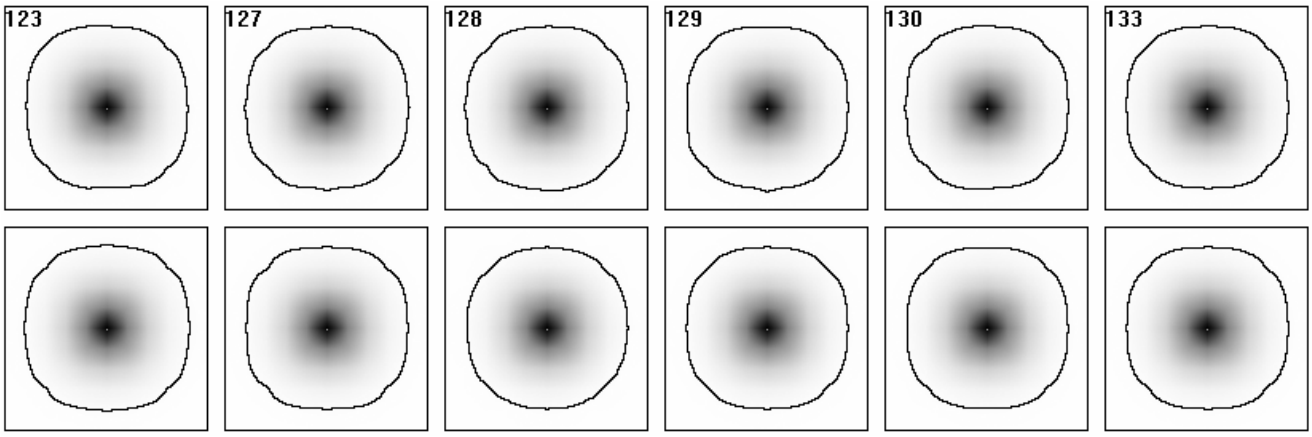


Рис. 31. Метод PL. $\lambda = 0.8$. $\mu = 0.8$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

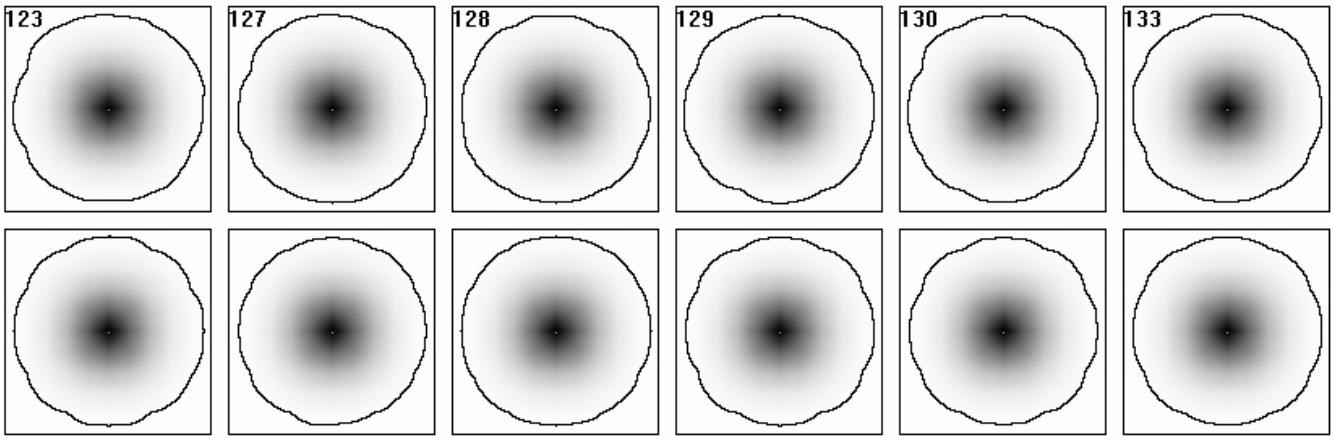


Рис. 32. Метод PL. $\lambda = 0.9$. $\mu = 0.8$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

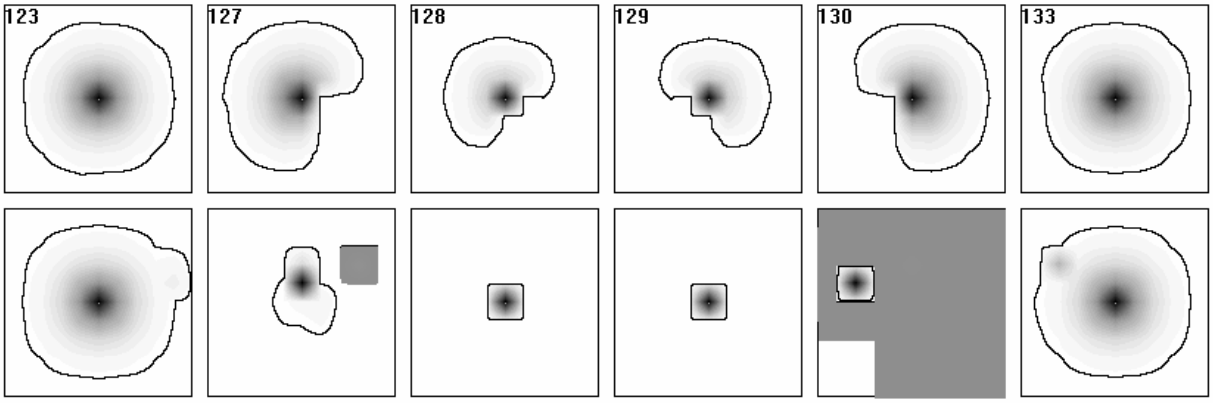


Рис. 33. Метод PL. $\lambda = 0.8$. $\mu = 0.8$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 14$.

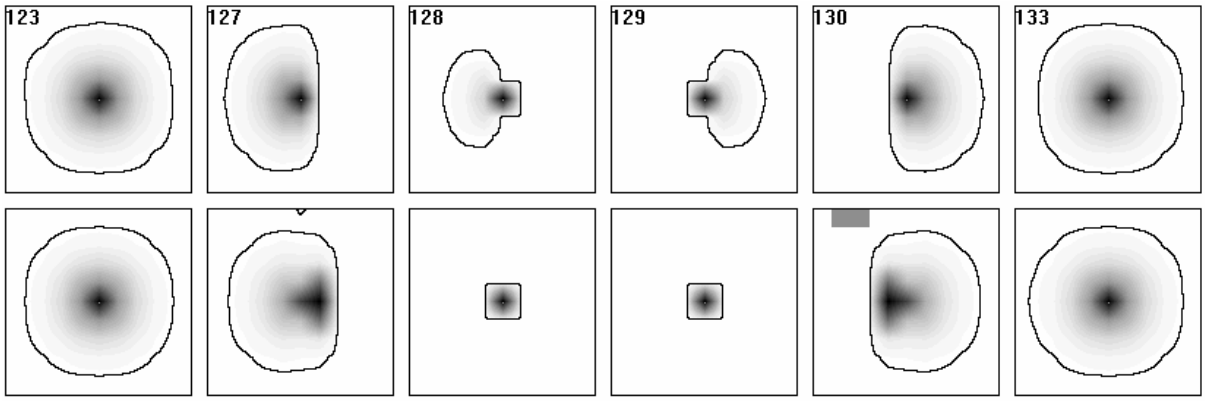


Рис. 34. Метод PL. $\lambda = 0.8$. $\mu = 0.8$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 10$

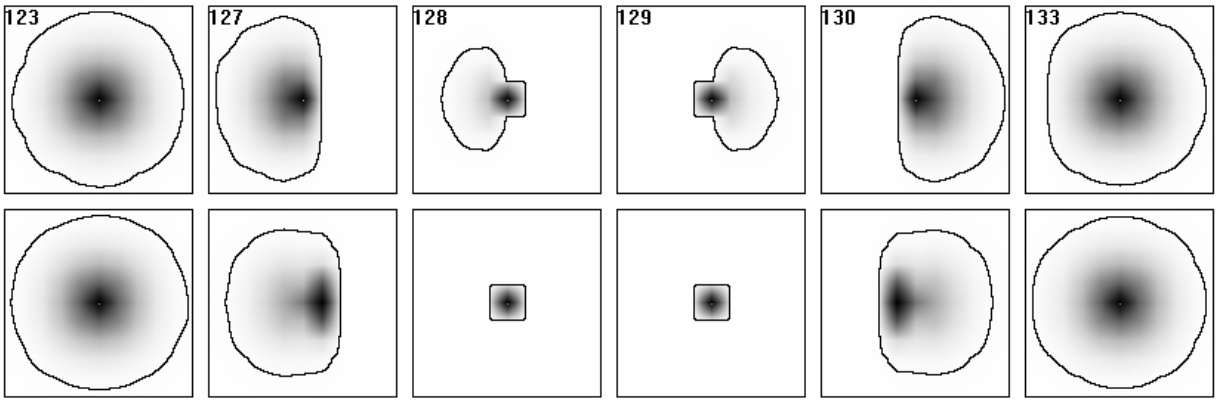


Рис. 35. Метод PL. $\lambda = 0.9$. $\mu = 0.8$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 10$.

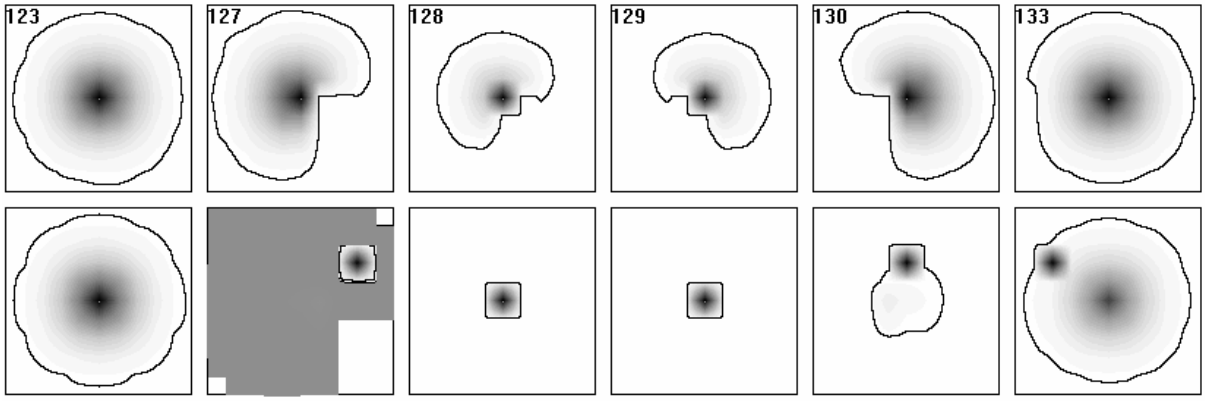


Рис. 36. Метод PL. $\lambda = 0.9$. $\mu = 0.8$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 14$.

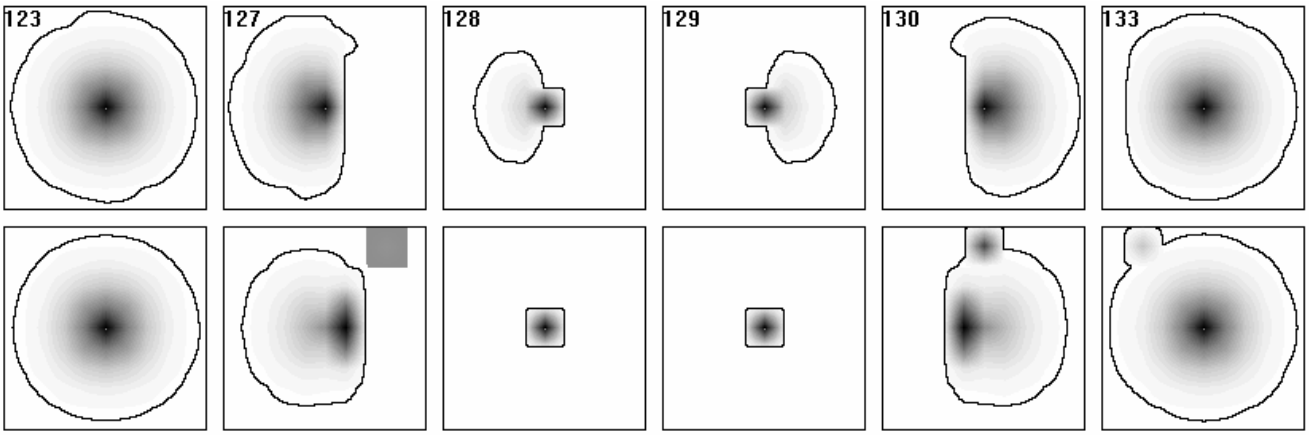


Рис. 37. Метод PL. $\lambda = 0.9$. $\mu = 0.8$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 12$.

Как видно из Рис.36, 37, метод использования областей влияния позволяет достаточно точно определить значение y_0, при котором прекращается сохранение контрастных границ. В частности, при $y_0 = 14$ метод PL ведет себя неустойчиво, а при $y_0 = 12$ получается приемлемый результат (Рис.37). Данные измерения относятся к тестовому изображению Рис.1, однако по значению y_0 с помощью этого изображения можно определить тот уровень контраста границы, выше которого граница сохраняется, а ниже которого – размывается.

4.3. Метод PL при больших значениях параметра μ

К большим значениям относится величина $\mu = 0.9$. Следует иметь в виду, что при таком значении μ метод PL способен сохранять только наиболее контрастные границы.

Пример приведен на Рис.38. Несмотря на то, что в этом примере выбрано $y_0 = 12$, то есть тестируемая точка находится в области границы с высоким контрастом (см. Рис,1,4), размер областей влияния велик, и граничная линия размывается.

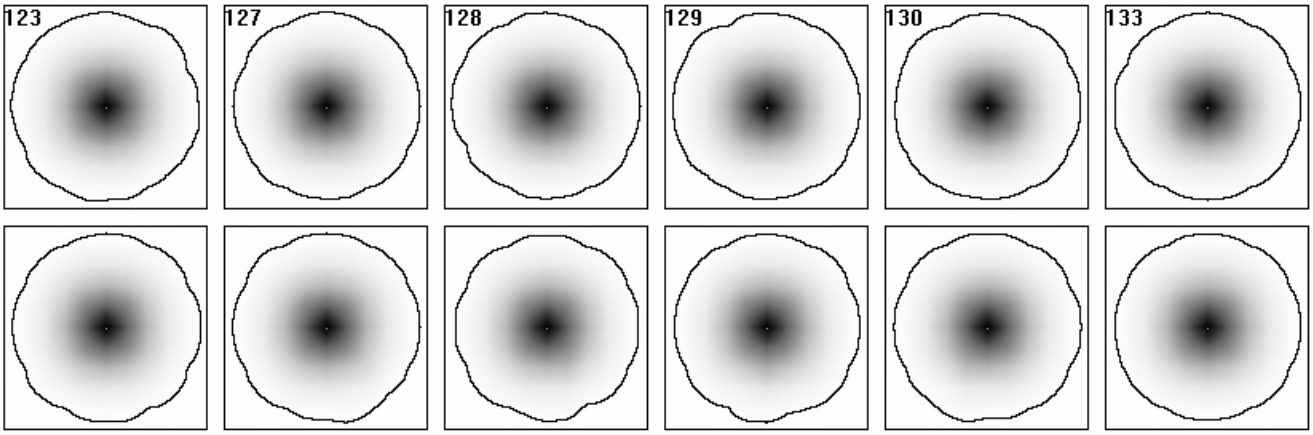


Рис. 38. Метод PL. $\lambda = 0.9$. $\mu = 0.8$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 12$.

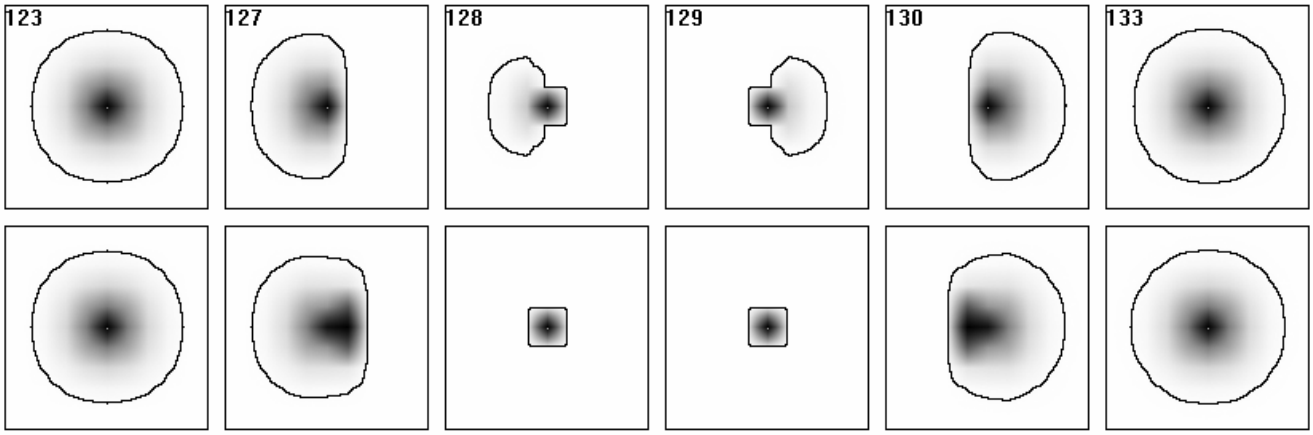


Рис. 39. Метод PL. $\lambda = 0.9$. $\mu = 0.3$. Девиация гауссова шума = 0. Ордината тестируемой точки $y_0 = 27$.

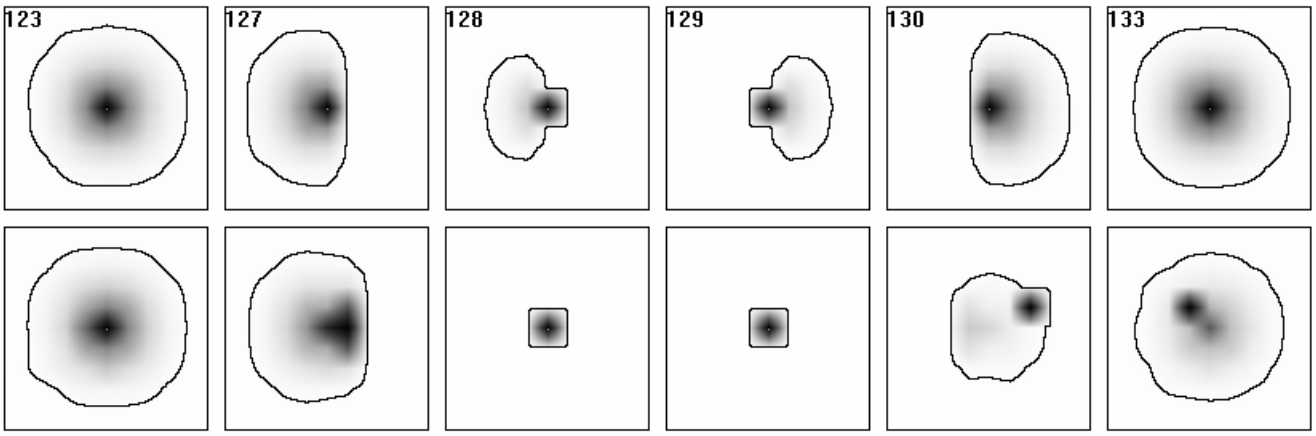


Рис. 40. Метод PL. λ = 0.9. μ = 0.3. Девиация гауссова шума = 30. Ордината тестируемой точки y_0 = 27.

4.4. Другие случаи

К другим случаям относится зашумление с девиацией 0 и 30. На Рис.39,40 представлены результаты обработки незашумленного изображения и изображения с высоким уровнем шума. В этих случаях вид и размер областей влияния несколько отличается от тех, что были представлены на предыдущих рисунках. Однако отличие небольшое. Это говорит о том, что выбор изображения с девиацией шума, равной 10, был адекватным.

5. Метод GR

В этом разделе изучается метод GR (Geman-Reynolds) с функционалом (2). Минимизирующая функция задается формулой (4).

Аналогично случаю метода PL, параметр λ в основном влияет на качество сходимости итерационного процесса при вычислении реставрированного изображения. Как показали наши эксперименты, быстрая сходимость с гладкими графиками всех контролируемых величин получается при λ = 0.8..0.95. Поэтому в дальнейших примерах мы ограничиваемся значением λ = 0.85. Выбор другого значения λ несколько изменяет вид областей влияния, однако эти изменения незначительны. В то же время, выбор второго параметра μ очень существенно влияет на работу метода GR.

При тестировании метода GR, как и прежде, выбирались точки (x_0,y_0) вблизи границы изображения Рис.1. На последующих рисунках в верхнем ряду показаны области влияния первого рода, в нижнем ряду – второго рода. Значения x_0 указаны непосредственно на рисунках, а значения y_0 – в подписях под рисунками. От значения y_0 зависит контраст границы, к которой приближается исследуемая точка: при y_0 = 255 контраст минимален, при y_0 = 0 он максимален (см. Рис.1).

Зафиксируем некоторое среднее значение μ, именно, μ = 0.5 и рассмотрим эволюцию областей влияния при изменении ординаты y_0. Примеры показаны на Рис.41-45. Здесь ордината y_0 последовательно снижается с 137 до 17, соответственно, возрастает контраст границы, к которой приближаются тестируемые точки.

На Рис.41 взята ордината y_0 = 137, при которой все точки (x_0,y_0) находятся в области слабого контраста границы Рис.1. Области влияния показывают, что тогда при реставрации граница полностью размывается.

На Рис.42 взята y_0 = 97. Серые области вне центра («области артефактов» реставрации) означают, что при увеличении яркости на величину δ в точке (x_0,y_0), или окрестности ее на исходном изображении, яркость в окрестности той же точки (x_0,y_0) на реставрированном изображении не увеличивается, а уменьшается.

Это явление приводит к искажениям границы на реставрированном изображении. То же самое относится и к Рис.43-45. Как видно из рисунков, метод GR искажает границу значительно сильнее, чем метод PL.

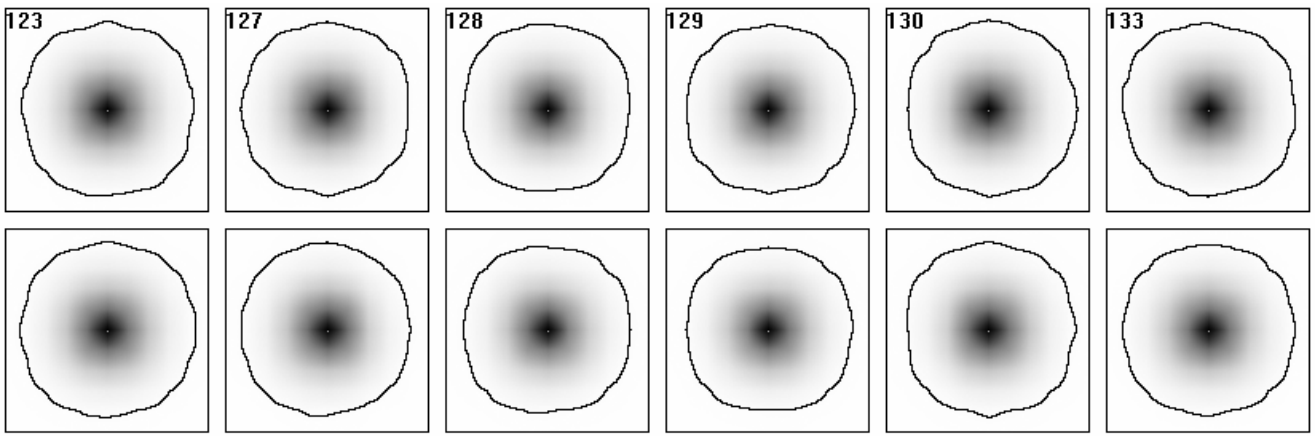


Рис. 41. Метод GR. λ = 0.85. μ = 0.5. Девиация гауссова шума = 10. Ордината тестируемой точки y_0 = 137.

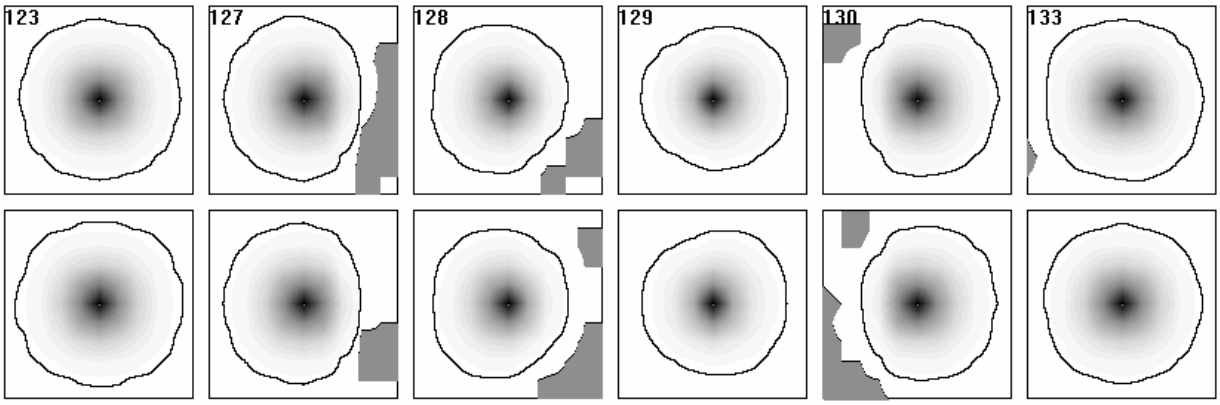


Рис. 42. Метод GR. $\lambda = 0.85$. $\mu = 0.5$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 97$.

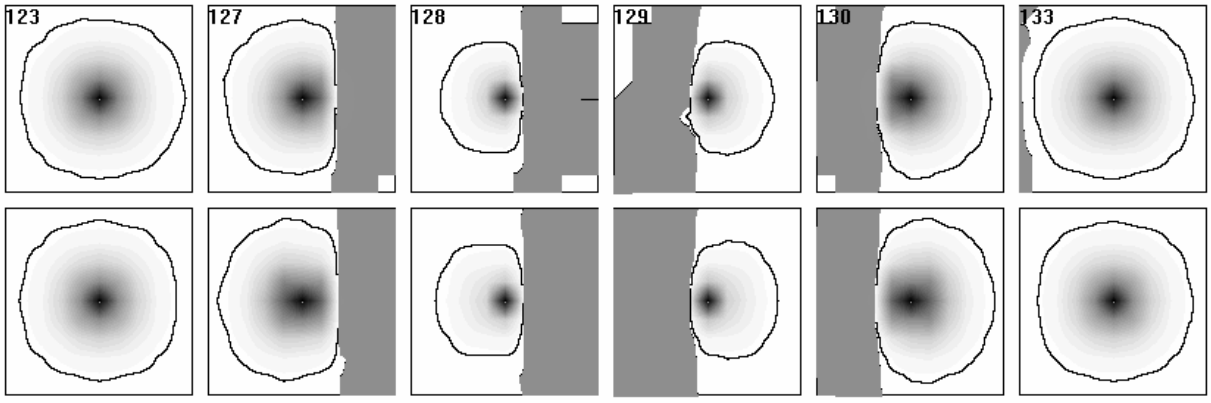


Рис. 43. Метод GR. $\lambda = 0.85$. $\mu = 0.5$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 57$.

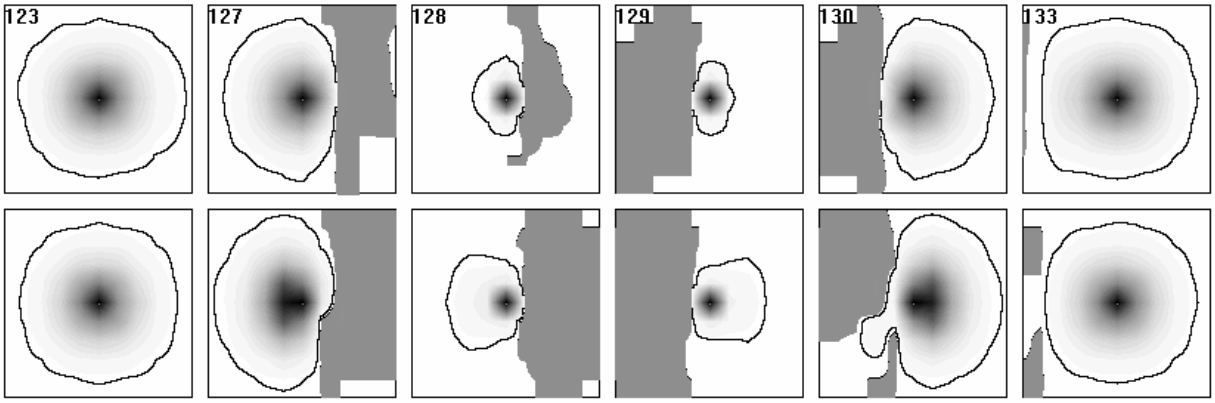


Рис. 44. Метод GR. $\lambda = 0.85$. $\mu = 0.5$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

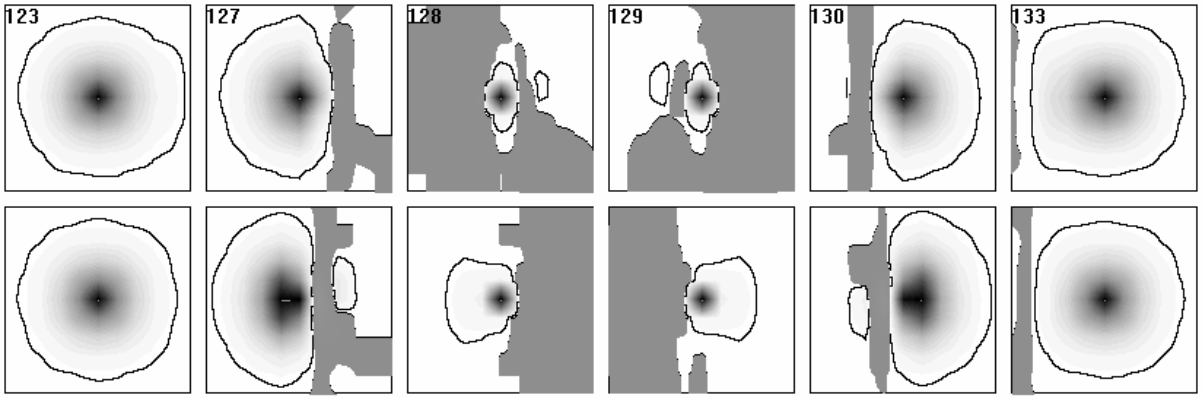


Рис. 45. Метод GR. $\lambda = 0.85$. $\mu = 0.5$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 17$.

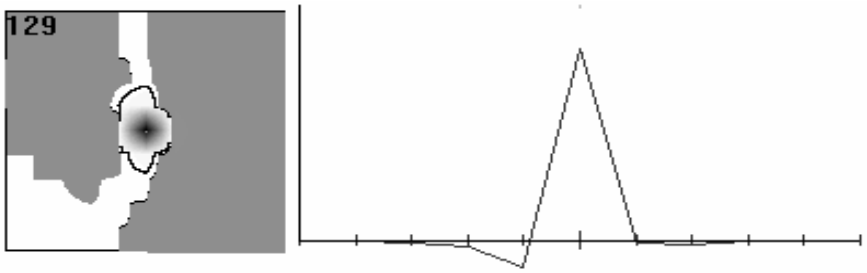


Рис. 46. Метод GR. $\lambda = 0.85$. $\mu = 0.5$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 12$.

Чтобы оценить, насколько велики искажения границы, рассмотрим Рис.46. Слева показана область влияния первого рода при выбранных значениях параметров, а справа – сечение этой области по горизонтали, проходящее через центр. На этом сечении отмечены значения яркости 11 пикселов изображения D_1 в окрестности центральной точки. Для получения Рис.46 в точке (x_0,y_0) исходного изображения яркость увеличивалась на величину $\delta = 0.1$, и затем вычислялась область влияния D_1.

Как видно на правом графике Рис.46, в точке (x_0,y_0) имеется пик возрастания яркости, однако в соседних точках яркость уменьшилась, и появился отрицательный пик, что говорит о некотором искажении границы. Однако величина отрицательного пика не превышает нескольких процентов от величины положительного пика, в целом качество сохранения резкого граничного перепада на реставрированном изображении можно считать приемлемым.

5.1. Метод GR при малых значениях параметра μ

В данном случае малыми значениями являются $\mu = 0.1$ и $\mu = 0.2$. При этих значениях μ наблюдаются сильные искажения областей влияния как первого рода D_1 (верхний ряд), так и второго рода D_2 (нижний ряд), а также множество участков с отрицательными значениями D_1 и D_2. На Рис.47 показаны области влияния для $\lambda = 0.85$, а на Рис.48 – для $\lambda = 0.95$. Результаты приблизительно одинаковые. Поэтому далее используется только значение $\lambda = 0.85$. На Рис.49 показаны аналогичные изображения для $\mu = 0.2$. Обратим внимание на то, что вблизи самой границы ($x_0 = 128, 129$) области влияния второго рода D_2 малы. Это значит, что сохранение контрастной границы очень хорошее. Таким образом, метод GR обладает той особенностью, что он вносит больше искажений в те области изображения, где яркость изменяется плавно, чем в те области, где есть границы.

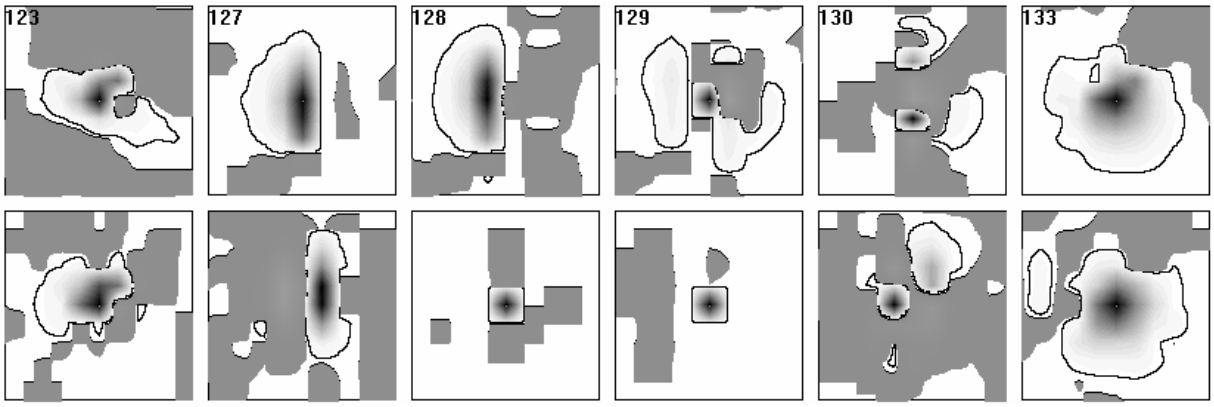


Рис. 47. Метод GR. $\lambda = 0.85$. $\mu = 0.1$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

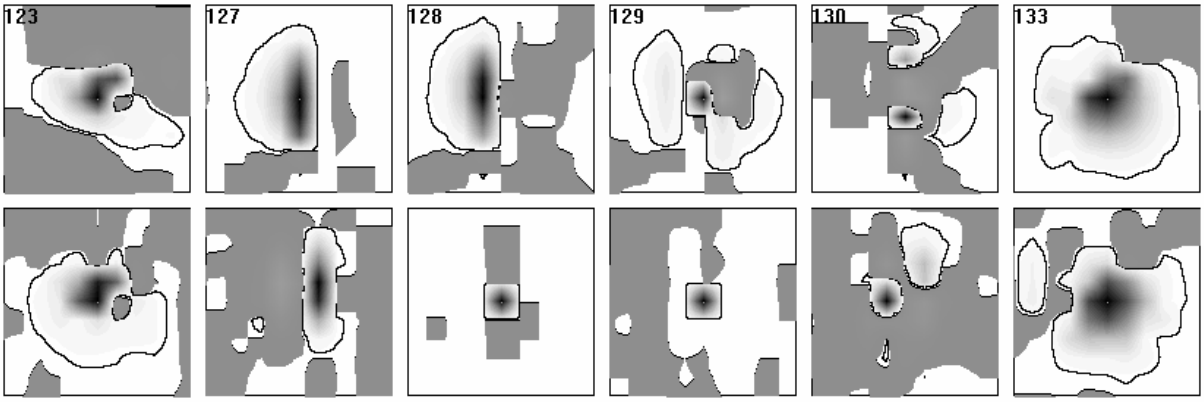


Рис. 48. Метод GR. $\lambda = 0.95$. $\mu = 0.1$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

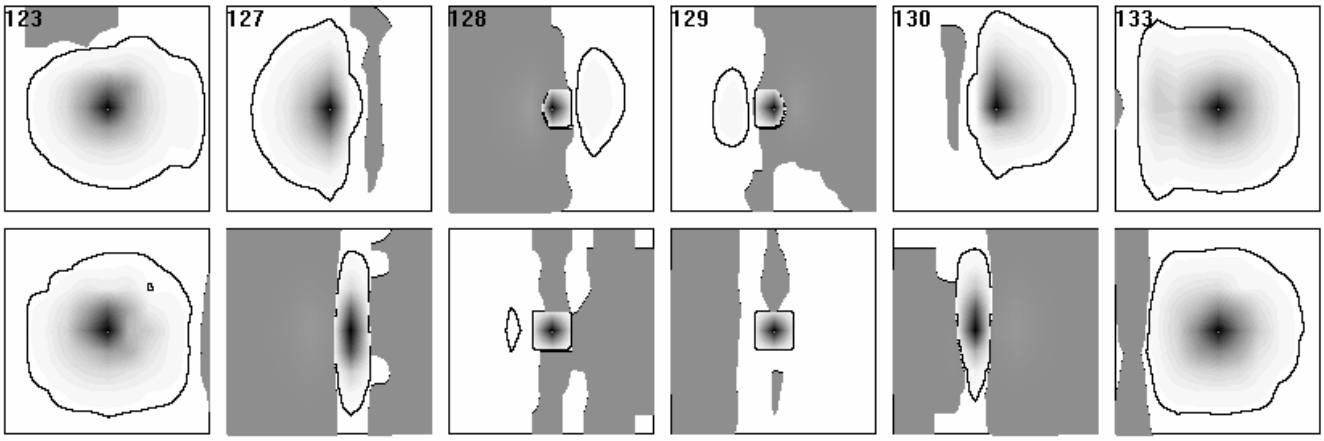


Рис. 49. Метод GR. λ = 0.85. μ = 0.2. Девиация гауссова шума = 10. Ордината тестируемой точки y_0 = 27.

5.2. Метод GR при средних значениях параметра μ

Здесь к средним относятся значения μ от 0.4 до 1.0. Соответствующие области влияния при y_0=27 показаны на Рис.50-56. Видно, что в удаленных от границы точках (x_0=123, 133) области влияния велики, и в окрестности точек происходит размытие изображения, что приводит к его сглаживанию и уменьшению уровня шума. При выборе тестируемой точки вблизи границы Рис.1, типичными являются изображения с x_0=128, 129.

На Рис.50-51 области влияния вблизи границы малы, но на Рис.52-56 они увеличиваются. Это говорит о том, что для μ = 0.5 контраст, задаваемый ординатой y_0 = 27 на Рис.1, является критическим и граница размывается. При уменьшении значения y_0 размер областей влияния уменьшается. Это явление аналогично тому, которое приведено на Рис.29,30 для метода PL.

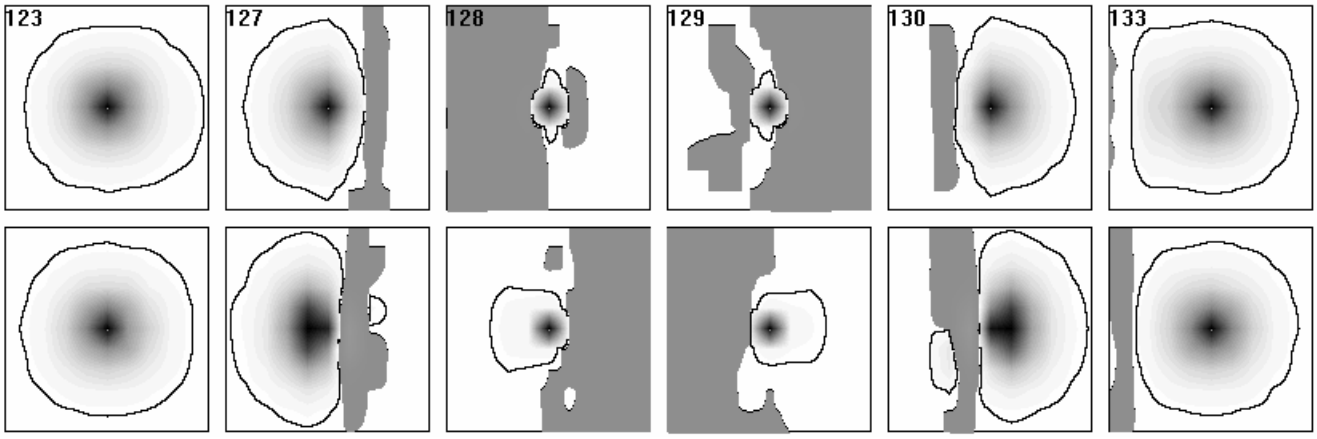


Рис. 50. Метод GR. λ = 0.85. μ = 0.4. Девиация гауссова шума = 10. Ордината тестируемой точки y_0 = 27.

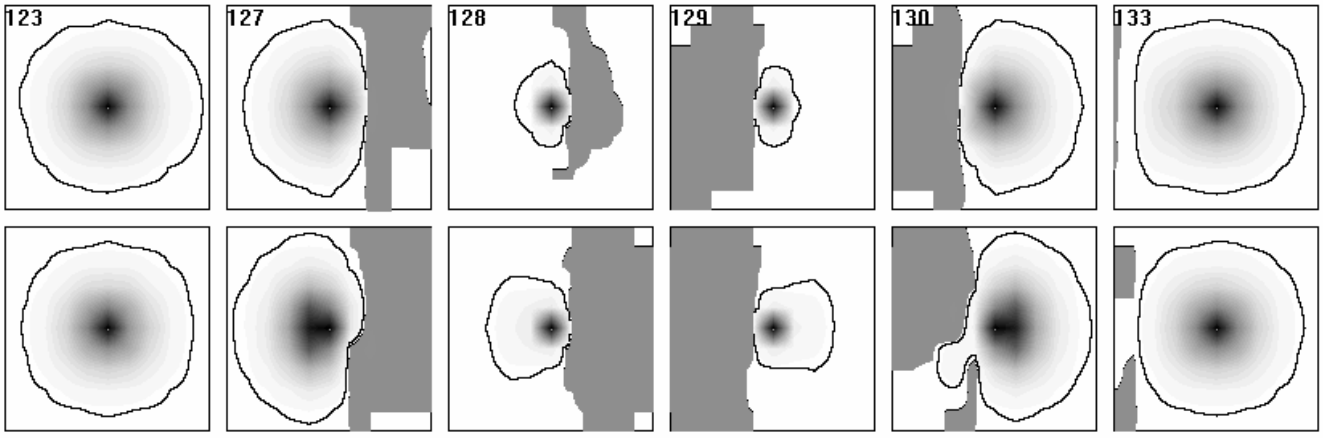


Рис. 51. Метод GR. λ = 0.85. μ = 0.5. Девиация гауссова шума = 10. Ордината тестируемой точки y_0 = 27.

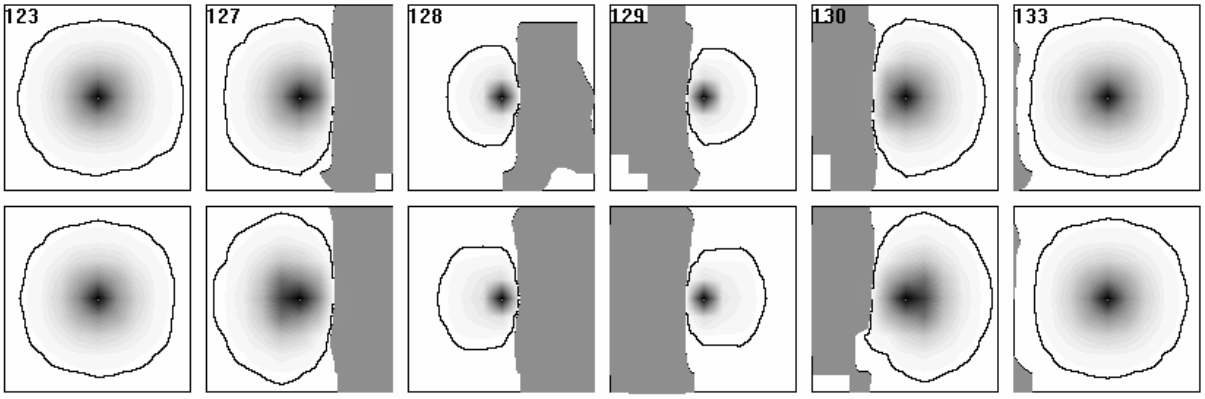


Рис. 52. Метод GR. λ = 0.85. μ = 0.6. Девиация гауссова шума = 10. Ордината тестируемой точки y_0 = 27.

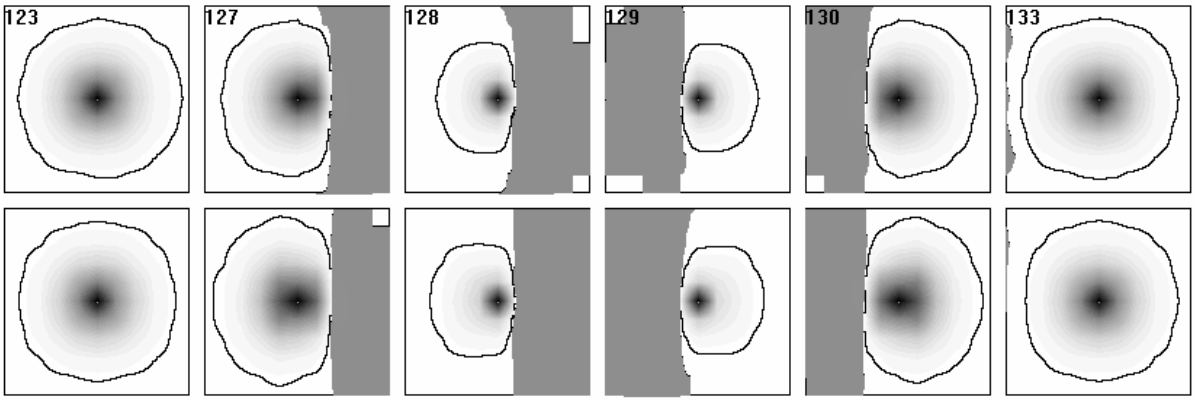


Рис. 53. Метод GR. λ = 0.85. μ = 0.7. Девиация гауссова шума = 10. Ордината тестируемой точки y_0 =27.

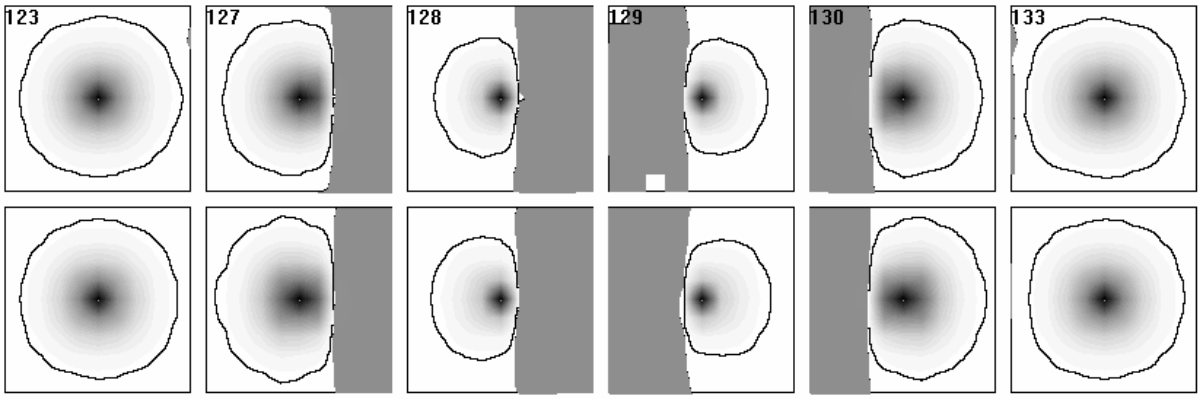


Рис. 54. Метод GR. λ = 0.85. μ = 0.8. Девиация гауссова шума = 10. Ордината тестируемой точки y_0 = 27.

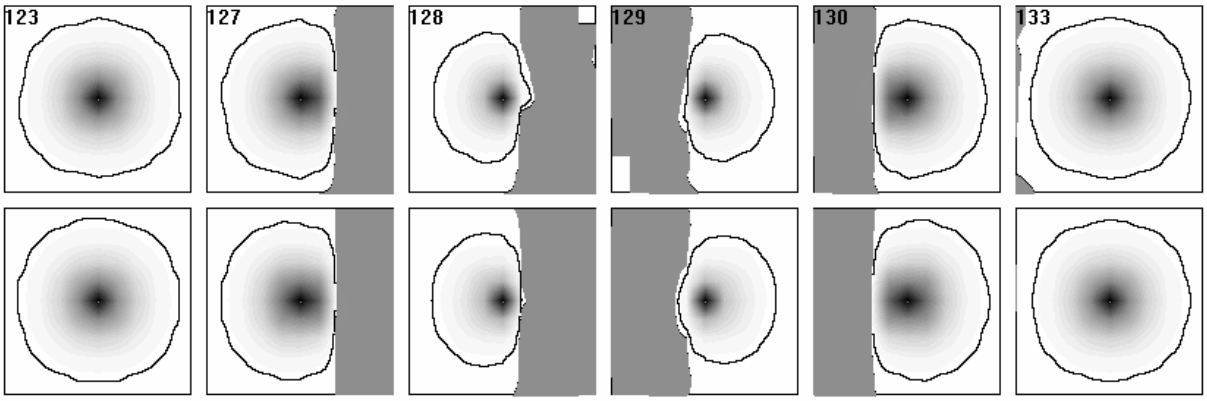


Рис. 55. Метод GR. λ = 0.85. μ = 0.9. Девиация гауссова шума = 10. Ордината тестируемой точки y_0 = 27.

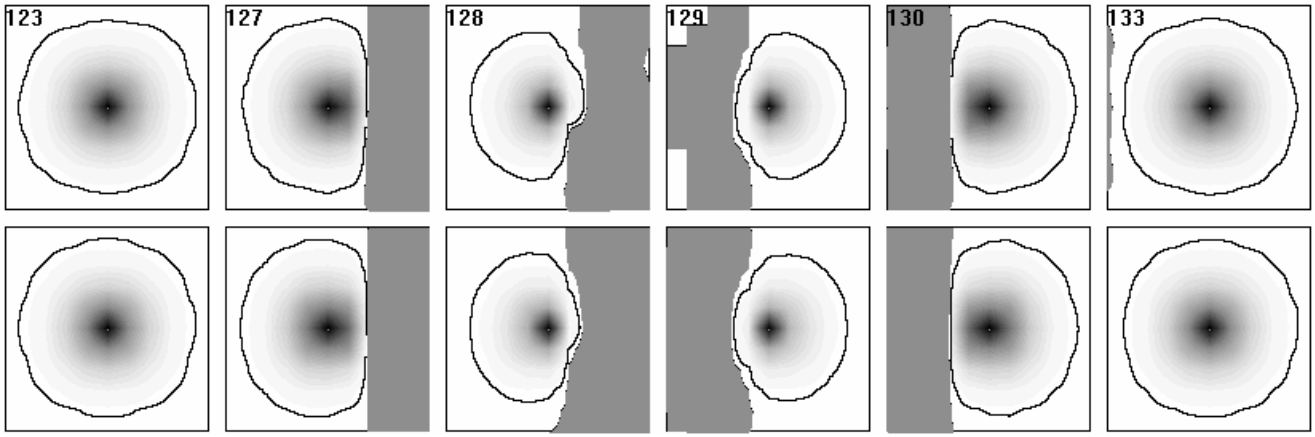


Рис. 56. Метод GR. $\lambda = 0.85$. $\mu = 1.0$. Девиация гауссова шума = 10. Ордината тестируемой точки $y_0 = 27$.

5.3. Другие случаи

К другим случаям относится зашумление с девиацией 0 и 30 при $\lambda = 0.9$, $\mu = 0.3$ и с девиацией 20 при $\lambda = 0.8$, $\mu = 0.3$. На Рис.57-59 представлены результаты обработки незашумленного изображения и изображения с высоким уровнем шума (девиация 20 и 30). Ордината тестируемой точки, то есть, уровень контраста границы, во всех случаях выбрана одной и той же: $y_0 = 27$. Как видно из рисунков, области влияния различны. Наибольшее искажение областей влияния происходит при больших уровнях шума на изображении (Рис.58, 59).

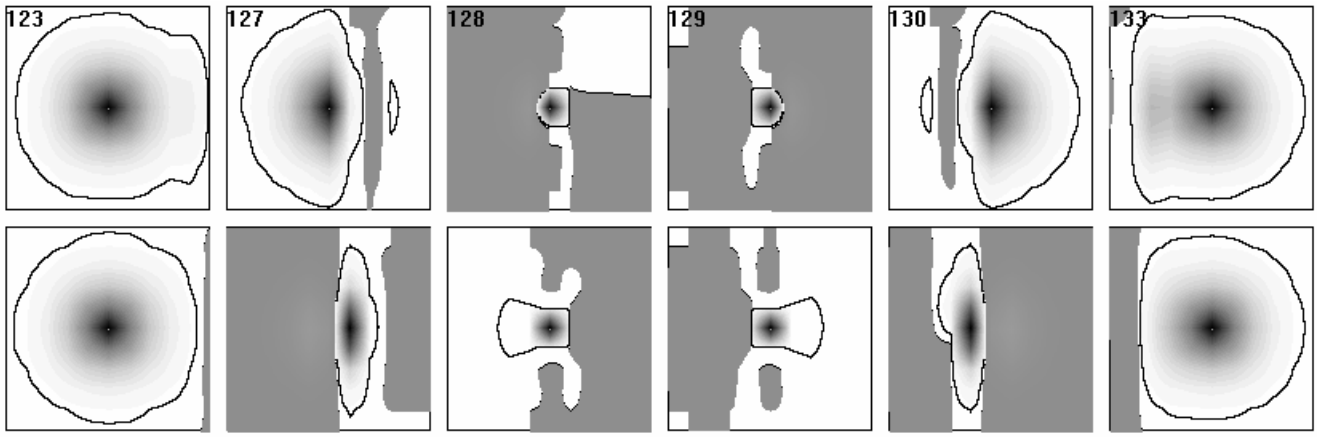


Рис. 57. Метод GR. $\lambda = 0.9$. $\mu = 0.3$. Девиация гауссова шума = 0. Ордината тестируемой точки $y_0 = 27$.

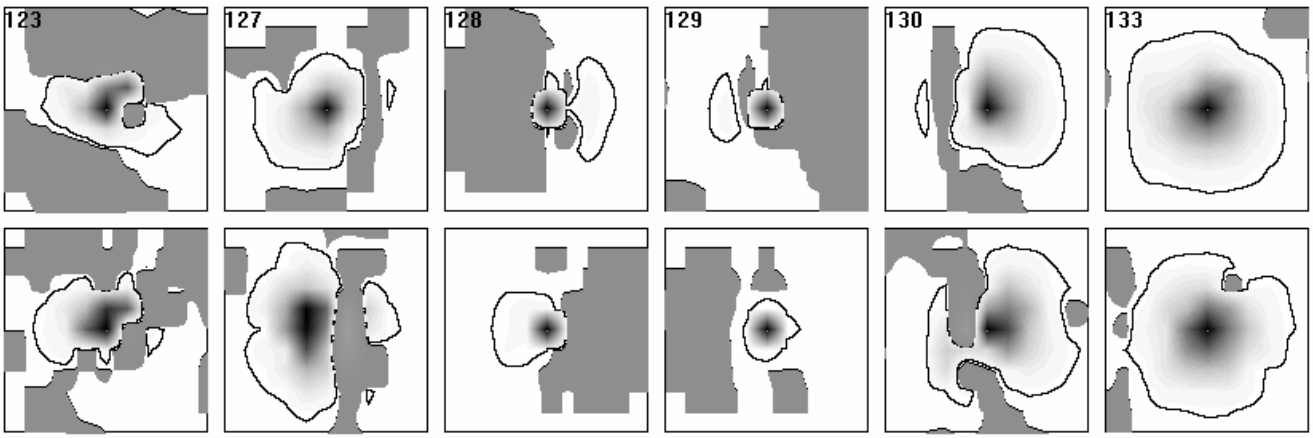


Рис. 58. Метод GR. $\lambda = 0.9$. $\mu = 0.3$. Девиация гауссова шума = 30. Ордината тестируемой точки $y_0 = 27$.

Рис. 59. Метод GR. $\lambda = 0.8$. $\mu = 0.3$. Девиация гауссова шума = 20. Ордината тестируемой точки $y_0 = 27$.

6. Заключение и выводы

В статье разработана методика вычисления областей влияния, характеризующих некоторые свойства методов реставрации изображений. Эта методика иллюстрируется на примере двух энергетических методов реставрации, PL и GR. Изучались свойства понижения уровня зашумления изображения и одновременно сохранения контрастных границ. При этом метод реставрации рассматривался как некоторый "черный ящик", о котором априори ничего неизвестно, кроме того, что у метода есть параметры, от которых зависит качество его работы.

1. Оказалось, что есть критические значения контраста, при которых метод реставрации ведет себя неустойчиво и граница реставрированного изображения искажается.

2. Для метода PL оптимальными являются значения параметров от $\lambda = 0.8$ до $\lambda = 0.9$ и значения $\mu = 0.2..0.8$. При меньших значениях μ наблюдаются искажения изображения, при больших – практически не происходит сохранения контрастных границ. Искажения изображения также наблюдаются при критических значения контраста, которые свои для каждого значения μ.

3. Метод GR практически всегда вносит искажения в реставрированное изображение. При увеличении яркости в некоторой точке исходного изображения, яркость в окрестности точки реставрированного изображения может падать. Однако это не следует считать недостатком метода, поскольку такой эффект может приводить к усилению контраста размытых границ, то есть, к повышению четкости изображения. По нашим оценкам, для метода GR оптимальными являются значения параметра λ от $\lambda = 0.8$ до $\lambda = 0.95$. Если требуется сохранение наиболее контрастных границ на изображении, то следует выбирать $\mu = 0.1..0.2$. Если предпочтительно наибольшее подавление шума, то оптимальны значения $\mu = 0.3..1.0$.

Поскольку энергоминимизирующие методы используются также для сегментации изображений (см. например [8],[9], [15]), изложенная здесь методика может быть использована для тестирования качества сегментации размытых изображений такими методами. Заметим, что представленные выше результаты относятся к только к сохранению границ типа ступеньки (Рис.1). Тем самым, задача всестороннего исследования свойства сохранения границ остаётся открытой. Мы считаем, что набор тестовых изображений системы PICASSO [7]-[8] будет полезен для этой цели.

Литература

1. K.W.Bowyer and P.J.Phillips. Overview of Work in Empirical Evaluation of Computer Vision Algorithms. // Empirical Evaluation Techniques in Computer Vision, K.W.Bowyer and P.J.Phillips (Eds.), IEEE Computer Press, 1998, pp. 1-11.

2. Y.Boykov, V.Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. // International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR) 2001, pp. 359-374.

3. P.Courtney and N.A. Thacker. Performance Characterisation in Computer Vision: The Role of Statistics in Testing and Design. // Imaging and Vision Systems: Theory, Assessment and Applications, J. Blanc-Talon and D. Popescu (Eds.), NOVA Science Books, 2001.

4. S.Geman and D.Geman. Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images. // *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6, № 6, pp. 721-741, 1984.

5. D.Geman, S.Geman, C.Graffigne, and P.Dong. Boundary detection by constrained optimization. // *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, № 7, pp. 609-628, 1990.

6. D.Geman and G.Reynolds. Constrained restoration and the recovery of discontinuites. // *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, №3 pp. 376-383, Mar. 1992.

7. I.V. Gribkov, P.P. Koltsov, N.V. Kotovich, A.A. Kravchenko, A.S. Kutsaev, V.K. Nikolaev, A.V. Zakharov. PICASSO – Edge Detectors Evaluation System Based on the Comprehensive Set of Artificial Images. // Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics, Vol. 9, 2002, pp 88-93.

8. I.V. Gribkov, P.P. Koltsov, N.V. Kotovich, A.A. Kravchenko, A.S. Koutsaev, V.K. Nikolaev, A.V. Zakharov. Testing of Energy Minimizing Methods in

Image Preprocessing Using the PICASSO System. // Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics, Vol. 6, 2004, pp 233-238.

9. G.A.Hewer, C.Kenney, and B.S.Manjunath. Variational Image Segmentation Using Boundary Functions. // *IEEE Transactions on Image Processing*, Vol.7, No. 9, 1998, pp.1269-1282.

10. D.Mumford and J.Shah. Boundary detection by minimizing functionals. // Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Vol. 17, 1985, pp. 22-26.

11. D.Mumford and J.Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *// Communications on Pure and Applied Mathematics* , Vol. 42, No 5, 1989, pp. 577-685.

12. T.Poggio, H.Voorhees and A.Yuille. A regularized solution to edge detection. – M.I.T. Artificial Intelligence Laboratory, Cambridge, MA, Report AIM-833, May 1985.

13. V.Torre and T.Poggio. On edge detection. – M.I.T. Artificial Intelligence Laboratory, Cambridge, MA, Report AIM-768, August 1984.

14. C.Samson, L.Blanc-Feraud, G.Aubert, and J.Zerubia. A Variational Model for Image Classification and Restoration. // *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 5, 2000, pp. 460-472.

15. O.Veksler. Efficient Graph-Based Energy Minimization Методs in Computer Vision. – PhD thesis, Cornell University, July 1999.

The usage of “influence areas” for testing of energy minimizing image restoration methods

I.V.Gribkov

Abstract: Image restoration is one of important stages in image processing. We consider the energy methods for image restoration, when a restored image is a result of minimization of a certain integral functional. The wide variety of such methods makes it difficult to find the optimal method to solve a concrete task. The “influence areas” introduced in this paper is a tool which allows one to find essential properties of the restoration methods, such as the ability to reduce the noise level and to preserve contrast boundaries on the restored image. The whole testing technique based upon this approach is applied to study a couple of known image restoration methods. In particular, the parameters of these methods, ensuring their better performance, are found.

Развитие метода вычисления направления касательной на основе преобразования Хафа

А.В. Захаров; П.П. Кольцов, к. ф.-м. н., доц.; Н.В. Котович; А.А. Кравченко, к. ф.-м. н.; А.С. Куцаев, к. ф.-м. н.; А.С. Осипов, к. ф.-м. н.

Рассматривается метод оценки угла наклона касательной к изображению кривой, основанный на преобразовании Хафа. Анализируется выбор оптимальных размеров вычислительного окна. Проведено сравнение аппроксимации окружностями и сплайнами при оценках угла по тройке пикселов. Предложено уточнение процедуры голосования в методе Хафа, учитывающее разброс единичных оценок угла. Диапазон изменения значений счетчиков при этом растет, что позволяет при помощи порогового значения отбирать решения в пределах заданной погрешности. Рассмотрено влияние уровня шума на рост погрешности вычисления направления касательной.

1. Введение

Детекторы границ извлекают из изображения существенную часть содержащейся в нем геометрической информации. В связи с распознаванием объектов возникает ряд задач дальнейшей обработки выделенных границ, в которых необходимо оценивать кривизну и направление касательной к дискретной кривой, таких как выделение угловых точек и точек излома. При анализе аффинных преобразований изображений полезно группировать точки с одинаковым направлением касательной, так как параллельность сохраняется при аффинном преобразовании. Методы оценки кривизны и направления касательной обычно требуют представления кривой в виде цепочки 8-связных пикселов. Далее в них может использоваться подход наименьших квадратов и аппроксимация кривой с помощью окружности или сплайна [1]. В [2] предложен метод разностной медианной фильтрации, в котором направление касательной в выбранной точке оценивается при помощи медианы для множества направлений секущих дискретной кривой, проведенных из выбранной точки. В последние годы развиваются методы определения касательных к дискретным кривым, основанные на выделении в них отрезков дискретных прямых наибольшей длины [3, 4].

Простой и устойчивый метод определения направления касательной, не требующий выделения цепочек пикселов и основанный на преобразовании Хафа, был предложен в [5]. Этот метод использовался в задаче определения аффинного преобразования шаблона кривой в изображение и показал хорошие результаты на примере изображений дуг эллипсов. Однако в случае произвольных кривых этот метод не всегда дает точность, необходимую для прикладной задачи. Данная работа посвящена исследованию возможностей метода, предложенного в [5], и выбору оптимальных параметров для получения надежной и точной оценки.

Далее статья имеет следующую структуру. В Главе 2 описан метод оценки направления касательной при помощи преобразования Хафа. В Главе 3 подробно исследуются факторы, влияющие на выбор размера локального окна. Вводится многопараметрическое семейство тестовых кривых, позволяющее получать устойчивые оценки погрешности. Глава 4 описывает замену окружностей сплайнами при промежуточных оценках угла наклона касательной. В Главе 5 описаны проблемы, связанные с наличием разрывов на изображениях кривых. В Главе 6 рассматривается модификация процедуры голосования в методе Хафа, основанная на учете величины разброса оценок угла. В Главе 7 описывается модель шума для дискретных кривых и рассматривается влияние шума на погрешность вычисления направления касательной. В Главе 8 дана краткая сводка результатов.

2. Оценка направления касательной при помощи преобразования Хафа

Метод определения направления касательной к изображению кривой, описанный в [5], основан на преобразовании Хафа. Пикселы в нем считаются конечными квадратами. Чтобы определить угол касательной в выбранном пикселе кривой, на изображении рассматривается локальное окно размерами $W \times W$ пикселов с центром C в центре выбранного пиксела. В локальном окне перебираются все пары пикселов кривой **a** и **b**, отвечающие ограничениям на взаимное положение (см. ниже). Для каждой пары пикселов рассматривается множество аппроксимирующих окружностей, проходящих через тройки точек A, B, C, где A и B - произвольные вершины пикселов **a** и **b**, соответственно. По всем этим окружностям вычисляется минимум и максимум угла наклона касательной к окружности в точке C (Рис. 1).

Нужно заметить, что использование вершин пикселов **a** и **b** в качестве точек A и B не является единственно возможным. Можно использовать середины сторон или другие характерные точки, ближе или дальше от центра пиксела. От этого зависит главным образом длина отрезков изменения угла: чем ближе точки A и B к центрам пикселов, тем короче эти отрезки и меньше степень их наложения. Опыт показывает, что при этом чаще всего уменьшается точность определения пика угла при преобразовании Хафа, так что другие варианты выбора точек A и B в пикселах далее не рассматриваются.

Допустимые пары пикселов отбираются в предположении, что они принадлежат изображению кривой с достаточно малой кривизной. Критерием отбора служит величина угла ACB между направлениями на центры пикселов **a** и **b** из центра окна. Угол ACB не должен быть меньше порогового значения (далее используется пороговое значение 120°). При этом центр окна, в котором определяется угол касательной, лежит между пикселами **a** и **b**. Далее для кривых с разрывами мы будем дополнительно рассматривать пары пикселов, расположенные по одну сторону от центра окна.

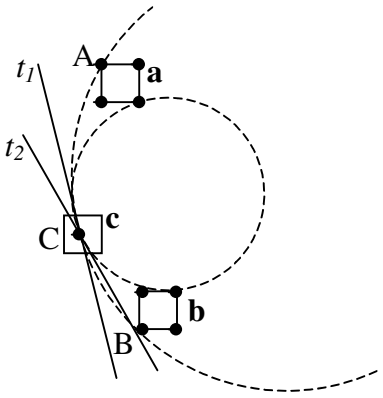


Рис. 1. Определение интервала изменения угла наклона касательной.

Чтобы свести вместе результаты по всем допустимым парам пикселов локального окна, в [5] используется одномерное преобразование Хафа. Массив счетчиков, отвечающих углам от 0 до 180° с некоторым шагом, предварительно обнуляется. При голосовании для каждой пары пикселов все счетчики, отвечающие интервалу изменения угла, увеличиваются на некоторое число (в простейшем случае 1). После голосования результат определяется по счетчику, имеющему максимальное значение. Если таких счетчиков несколько, берется середина отрезка, на котором они расположены. Если максимальное значение счетчика меньше порога, считается, что данных для определения угла недостаточно. Порог зависит от размеров окна; в [5] для окна размером W=13 он составлял 25.

Чтобы вычислить угол наклона касательной к окружности, проходящей через точки A, B, C, поместим начало координат в C и обозначим координаты точек A и B через (x_a, y_a) и (x_b, y_b), соответственно. Уравнение такой окружности с центром в точке (f, g) можно представить в виде:

$$(x-f)^2+(y-g)^2=f^2+g^2 \ . \qquad (1)$$

Угол θ между касательной к окружности в точке C и осью x равен:

$$\theta=arctg\ (-f/g) \ . \qquad (2)$$

Чтобы найти f и g, подставим в (1) координаты точек A и B и решим полученную систему двух уравнений. Подставляя решение в (2), получим:

$$\theta=arctg\left(\frac{r_b^2\cdot y_a-r_a^2\cdot y_b}{r_b^2\cdot x_a-r_a^2\cdot x_b}\right) , \qquad (3)$$

где $r_a^2 = x_a^2 + y_a^2$, $r_b^2 = x_b^2 + y_b^2$. Когда детерминант системы обращается в 0, точки A, B, C лежат на прямой, наклон которой также вычисляется по (3).

3. Выбор размера локального окна

Размер локального окна зависит от необходимой точности и надежности оценки. В [5] при размере окна W = 13 исследовались дуги эллипсов с полуосями $10 \le a \le 30$, b = 10 пикселов. Средняя погрешность составляла 3°, а максимальная ошибка росла от 5° до 13.5° вместе с большой полуосью эллипса a. Такая точность была достаточной для прикладной задачи, однако на более широком классе кривых потребовалось уточнение параметров метода, в первую очередь - размеров окна. При этом возможный выбор крайне невелик, так как окна с W = 9 и ниже дают слишком низкую точность.

Порядок величины погрешностей можно оценить, если рассмотреть в окне всевозможные отрезки дискретных 8-связных прямых, включающие центральный пиксел окна. В окне размером W = 13 можно выделить 892 таких отрезков, которым соответствует всего 184 различных угла наклона. Средняя погрешность определения угла наклона прямой по этим данным может составить около 0.25°, а максимальная 1.25°. Это нижние границы погрешности, но для произвольных цифровых кривых погрешность гораздо выше. Чем дальше пиксел от центра окна, тем меньше его положение связано со свойствами кривой в центре. В то же время периферийные пикселы дают наименьший разброс при определении направления касательной. Это один из главных недостатков описываемого подхода.

На Рис. 2 показан пример изображения кубической параболы $y = 4\ (x/11)^3$ и прямой $y = x/11$ в локальном окне размером W = 13 с центром в начале координат. Оценка угла касательной методом, описанным выше, составляет 5°, что близко к точному наклону прямой. Однако касательная к кубической параболе имеет в центре нулевой наклон, и для нее 5° - это величина погрешности. Погрешность можно снизить, если имеется дополнительная информация о свойствах и параметрах изображенных кривых. Если такой информации нет, то повышение точности за счет увеличения размера окна и включения периферийных пикселов невозможно. Если же периферийные пикселы вообще не принадлежат данной кривой, то их включение может только ухудшить оценку. Поэтому выбор оптимального размера окна очень важен.

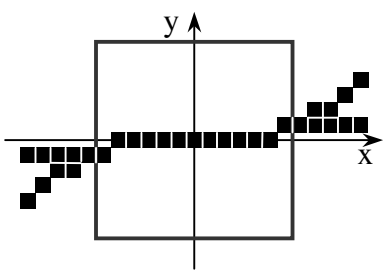


Рис. 2. Представление кубической параболы и прямой в локальном окне.

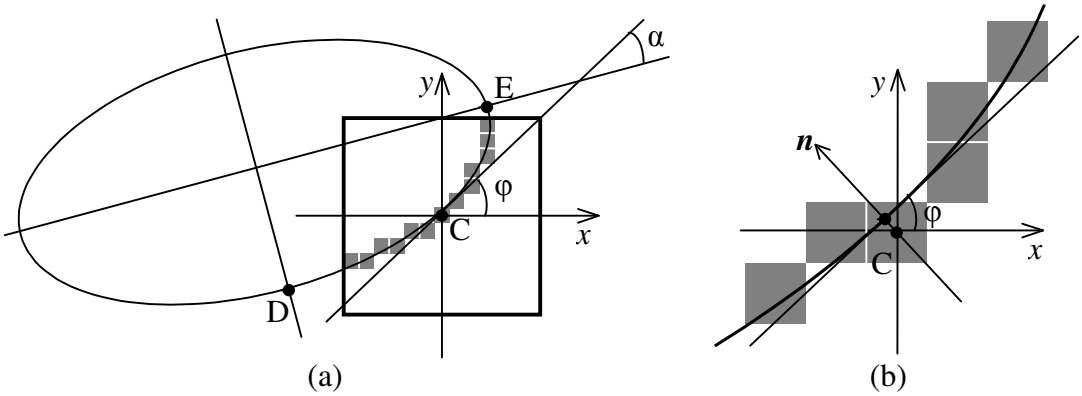


Рис. 3. Построение изображения дуги эллипса в локальном окне. (a) Смещение по нормали отсутствует. (b) Добавлено смещение по нормали.

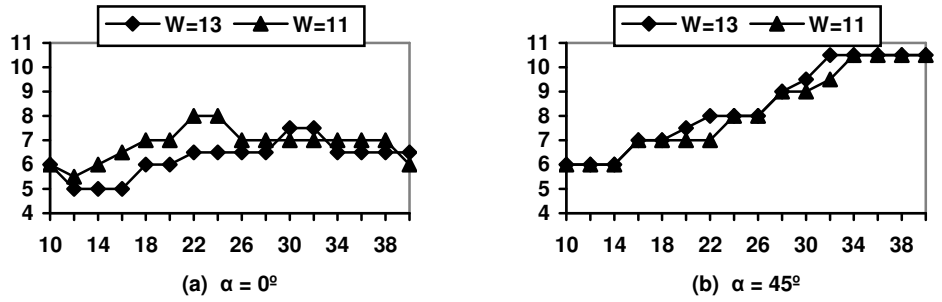


Рис. 4. Максимальная погрешность $E_M(\rho)$ для двух размеров окна в зависимости от радиуса кривизны в характерной точке эллипса D

Для сравнения параметров метода определения направления касательной будем использовать семейство дуг эллипсов, построение которых показано на Рис. 3.

На исходном эллипсе $(x/a)^2 + (y/b)^2 = 1$ выбирается точка C, касательная в которой наклонена к оси x под углом α. Исходный эллипс поворачивается вокруг точки C на угол $\varphi - \alpha$, чтобы угол наклона касательной в точке C был равен заданному углу φ. Затем эллипс перемещается так, чтобы точка C совпала с центром локального окна (Рис. 3a). Еще одно перемещение на небольшое расстояние d делается в направлении нормали к кривой в точке C. После этого кривая изображается в локальном окне.

Перемещение в направлении нормали показано в увеличенном виде на Рис. 3b. Это важное дополнение, которое позволяет полнее учесть шум, возникающий при растеризации. Без этого дополнения в небольшой окрестности центра окна C изображение будет иметь лишнюю симметрию, повышающую точность вычисления угла. Это видно на Рис. 3a, где показан случай $d = 0$. На Рис. 3b, где $d > 0$, изображение не имеет симметрии в C и выглядит менее гладким.

Для каждого набора параметров a, b, α вычислялся угол наклона касательной для окон размером $W = 13$ и 11 при заданном угле φ от 0 до 180° с шагом 1°, а также для нескольких значений поперечного сдвига d. Определялась средняя и максимальная погрешность угла наклона при изменении φ и d. В сериях вычислений параметр b был зафиксирован ($b = 10$), параметр a изменялся. Результаты представлены в зависимости от значений радиуса кривизны ρ в точке D на малом диаметре (Рис. 3a), для которой $\rho = a^2/d$.

На Рис. 4 приведена зависимость максимальной погрешности оценки угла наклона касательной E_M от ρ в градусах для размеров локального окна $W = 13$ и 11 при различных значениях угла α. Для каждого значения α зависимости довольно близки. При $\alpha = 0$ дуга эллипса симметрична, и оценки в большем окне точнее. С ростом α точка C приближается к максимуму кривизны эллипса (точка E на Рис. 3), асимметрия дуги растет, и преимущество большего окна пропадает. Кроме того, при $\alpha = 30°$ и $\alpha = 45°$ наблюдается рост

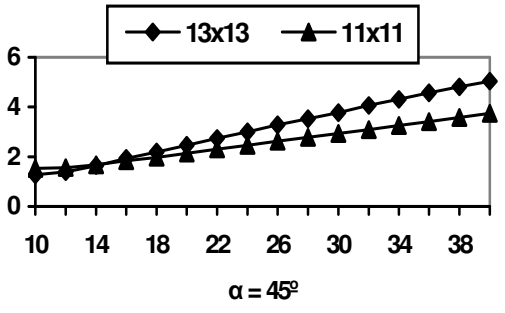
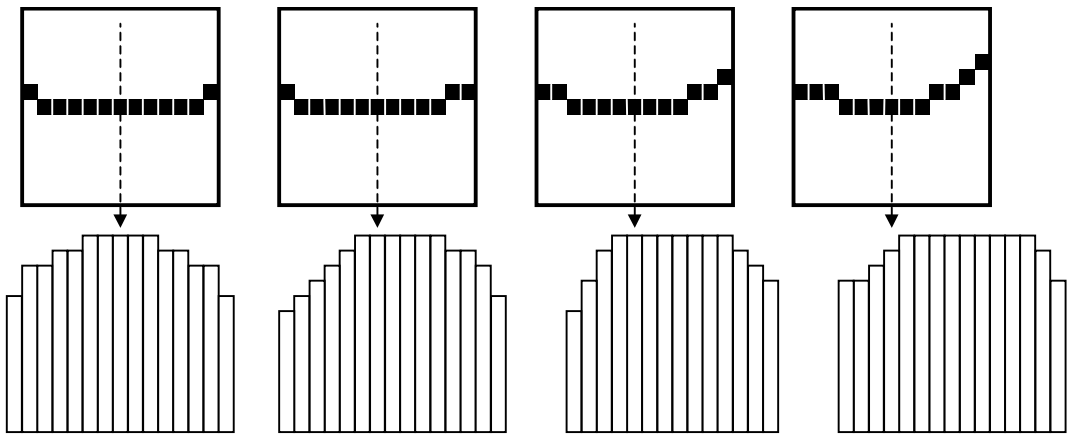


Рис. 5. Средняя погрешность E_A для двух размеров окна в случае большой асимметрии.



(a) α = 0º, E = 0º (b) α = 15º, E = 1.5º (c) α = 30º, E = 2.5º (d) α = 45º, E = 5º

Рис. 6. Изображения кривых и диаграммы счетчиков для окна W = 13 и различных значений показателя асимметрии α. Шаг диаграммы 1º, стрелки указывают точное значение угла наклона касательной

погрешности вместе с ρ.

Средняя погрешность оценки угла наклона касательной E_A (Рис. 5) ведет себя аналогично. Начиная с α = 30°, E_A для большего окна оказывается выше и растет быстрее. Это показывает, что для асимметричной кривой использование пикселов, далеких от центра окна, снижает точность оценки.

Влияние асимметрии кривых можно подробно рассмотреть на примере. На Рис. 6 показаны изображения кривых, диаграммы счетчиков и погрешность E в методе Хафа при размерах окна W = 13, значении ρ = 30 (a = 17.1, b = 10) и различных значениях показателя асимметрии α. Угол касательной везде задавался нулевым.

На Рис. 6a-d стрелки указывают середину локального окна и позицию в диаграмме счетчиков, отвечающую заданному значению угла наклона касательной φ = 0. Еще один показатель разброса оценки - ширина верхней части диаграммы или количество счетчиков с максимальным значением, которое также растет с α от 5 (Рис. 6a) до 9 (Рис. 6d). На этом примере видно, что на изображении достаточно гладкой кривой без дефектов погрешность 5° только за счет асимметрии является обычной.

Еще одна причина роста погрешности также связана с приближением к участку с большой кривизной. В приведенных зависимостях (Рис. 4, 5) с ростом показателя асимметрии α. центр окна C приближается к максимуму кривизны. При этом со стороны максимума в локальное окно попадает больше пикселов, что приводит к смещению оценки угла касательной. Этот эффект подробно показан на Рис. 7.

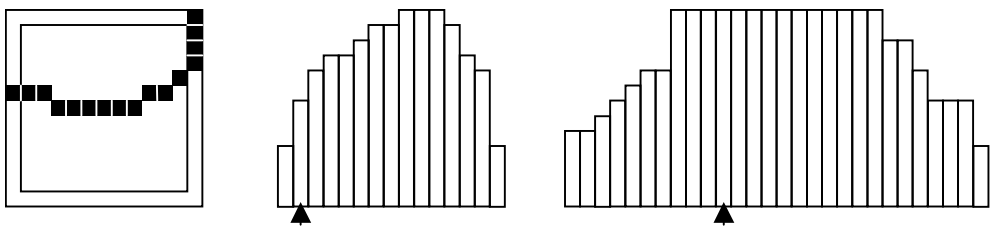
На Рис. 7a изображена дуга эллипса в локальном окне для параметров a = 20, b = 10, α= 45°, направление касательной φ = 0. На Рис. 7b показана диаграмма счетчиков для окна W = 13, заданное значение угла φ отмечено стрелкой. Из-за того, что в окне W = 13 справа на три пиксела кривой больше, погрешность угла составила 8°, несмотря на узкий пик на диаграмме, как раз и обусловленный правыми пикселами.

На Рис. 7c показана диаграмма счетчиков для той же кривой, но при использовании окна W = 11. Здесь ширина максимума счетчиков значительно больше, а высота пика в 2 раза меньше, чем в предыдущем случае. Тем не менее, поскольку окно W = 11 отсекает "лишние" пикселы, оценка угла составляет 3.5°, что гораздо ближе к заданному значению.

Из этих примеров можно сделать вывод, что точность оценки угла касательной зависит от изменения кривизны вдоль кривой. Асимметрия дискретной кривой в локальном окне увеличивает погрешность. При этом окно W = 13 дает примерно ту же точность, что и W = 11. Вблизи резких поворотов кривой окно W = 11 в среднем дает более точные результаты, а максимальная погрешность для этих окон примерно одинаковая.

4. Использование сплайнов для оценки угла

Аппроксимация окружностью, проходящей через точки кривой A, B, C, хорошо подходит для оценки угла касательной к кривой в точке C, когда C расположена между A и B. Если кривая достаточно плавная и



(a) Изображение (b) Счетчики, W = 13 (c) Счетчики, W = 11

Рис. 7. Оценка угла касательной около поворота с большой кривизной.

не имеет разрывов, нет необходимости использовать другие способы аппроксимации.

Вместе с тем окружность, имеющая постоянную кривизну, не позволяет учесть изменение кривизны вдоль кривой. Кроме того, при наличии разрывов появляется потребность использовать пары пикселов **a** и **b**, расположенные по одну сторону от центра. Здесь постоянная кривизна может быть причиной большого разброса оценок (см. далее пример на Рис. 10).

Из кривых переменной кривизны для оценки угла касательной удобно использовать обычный кубический сплайн $y = s(x)$. Для его построения задаются опорные точки (x_j, y_j), $j = 0, 1, ..., n$. Вторая производная сплайна непрерывна и принимает значения S_j.в точках x_i. На интервале (x_{j-1}, x_j) сплайн $s(x)$ задается выражением:

$$s(x) = S_j \frac{(x-x_{j-1})^3}{6h_j} - S_{j-1}\frac{(x-x_j)^3}{6h_j} + \\ + (x-x_{j-1})\left(\frac{y_j}{h_j} - \frac{h_j S_j}{6}\right) - \\ - (x-x_j)\left(\frac{y_{j-1}}{h_j} - \frac{h_j S_{j-1}}{6}\right) \quad (4)$$

где $h_j = x_j - x_{j-1}$.

Пусть дано локальное окно с центром C и двумя точками, A и B (Рис. 8). Проведем ось x вспомогательной системы координат через крайние точки тройки A, B, C. В первом случае (Рис. 8a) точка C расположена между A и B, во втором (Рис. 8b) A и B лежат по одну сторону от C. Во втором случае без ограничения общности можно принять, что точка A расположена ближе к C, т. е. AC < BC.

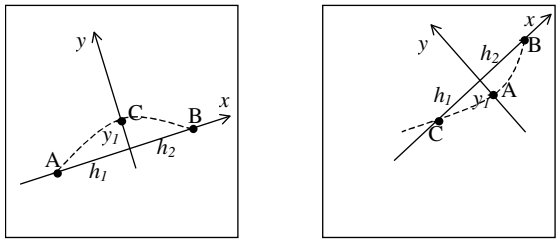


Рис. 8. Построение кубического сплайна для оценки угла касательной.

Построим сплайн для $n = 2$ на опорных точках A, B, C во вспомогательной системе координат, где $y_0 = y_2 = 0$. Сплайн можно представить себе в виде гибкой линейки, опирающейся на стержни в заданных точках. Пусть в крайних точках нет приложенных моментов сил. Тогда можно задать в них вторую производную $S_0 = S_2 = 0$. Из условия непрерывности первой производной в средней точке получим:

$$S_1 = -\frac{3y_1}{h_1 h_2} \ . \quad (5)$$

Первая производная $s'(x)$ в точке C в первом и втором случае (Рис. 8a и 8b) выражается в виде (6) и (7), соответственно.

$$s'(x_1) = y_1 \cdot (1/h_1 - 1/h_2) \ , \quad (6)$$

$$s'(x_0) = y_1 \cdot (1/h_1 + 1/2h_2) \ . \quad (7)$$

Величины y_1, h_1 и h_2 в обоих случаях выражаются через координаты точек A и B. Подставляя выражения для y_1, h_1 и h_2 в (6) и (7) и добавляя угол наклона вспомогательной оси x, получим для угла наклона касательной θ в первом и втором случае выражения (8) и (9), соответственно.

$$\theta = arctg\left(\frac{P(r_b^2 - r_a^2)}{(r_b^2 - G)(r_a^2 - G)}\right) + arctg\left(\frac{y_b - y_a}{x_b - x_a}\right) , \quad (8)$$

$$\theta = arctg\left(\frac{P(G - 2r_b^2)}{2G(r_b^2 - G)}\right) + arctg\left(\frac{y_b}{x_b}\right) , \quad (9)$$

где $r_a^2 = x_a^2 + y_a^2$, $r_b^2 = x_b^2 + y_b^2$ – скалярные квадраты векторов CA и CB, $G = x_a x_b + y_a y_b$, $P = x_a y_b - x_b y_a$.

Сравним среднюю погрешность оценки угла касательной при аппроксимации с помощью окружностей и сплайнов. Так как погрешности для этих случаев очень близки, будем изображать погрешность для сплайна в процентах от погрешности для окружности. Оценки делаются для семейства дуг эллипсов, описанного в п. 3, и размера локального окна $W = 11$.

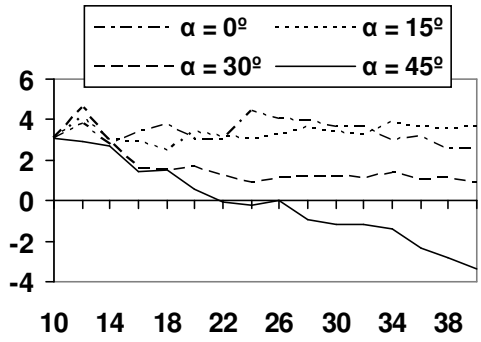


Рис. 9. Относительная погрешность F при аппроксимации окружностями и сплайнами.

На Рис. 9 приведены графики функции относительной средней погрешности F для параметра α.= 0, 15°,.30° и 45°. Функция $F = (E_{A,S}$ и $E_{A,C})\cdot 100\%$, где $E_{A,C}$ и $E_{A,S}$ - средняя погрешность при аппроксимации окружностями и сплайнами, соответственно. Все графики имеют общую точку при $\rho = a = b = 10$, т. е. когда эллипс является окружностью. При малой асимметрии средняя погрешность для сплайнов примерно на 3% выше, чем для окружностей. С ростом асимметрии функция F становится убывающей. При наибольшей асимметрии и больших ρ она отрицательна, т. е. сплайны дают меньшую погрешность. Здесь проявляется способность сплайна более точно повторять форму асимметричной кривой. Однако для симметричных кривых аппроксимация сплайном дает менее точные результаты

5. Разрывы на кривых

При оценке направления касательной методом Хафа значение счетчика зависит от количества пар пикселов, которые увеличили данный счетчик Если максимальное значение счетчиков не ниже порогового

значения, то угол оценивается по счетчикам, имеющим максимальное значение. Иначе считается, что данных недостаточно и угол не определен.

Порог счетчиков зависит от размера локального окна W. Пусть $W = 2n + 1$, а инкремент счетчиков при голосовании равен 1. Изображение гладкой кривой без разрывов, включающее центр окна, содержит не менее n пикселов с каждой стороны от центра. В допустимой паре пикселы расположены по разные стороны от центра, и число таких пар не меньше n^2. Если все они дают вклад в один и тот же счетчик, то его значение и будет максимальным. В [5] для $W = 13$ ($n = 6$) было выбрано пороговое значение $(n\text{-}1)^2 = 25$. Оно взято с запасом, чтобы разрешить отсутствие 1-2 пикселов или наличие недопустимых пар при крутом повороте кривой. Аналогично, для окна $W = 11$ ($n = 5$) можно взять пороговое значение $(n\text{-}1)^2 = 16$.

Отсутствие одного пиксела кривой обычно не мешает определению угла в соседних пикселах. Однако если ширина разрыва составляет 2 пиксела, то из n^2 пар останется $n(n\text{-}2)$, что на единицу меньше указанных порогов. Это значит, что в n-1 пикселах кривой с каждой стороны от края разрыва угол не определен. Вместе с самим разрывом общий размер фрагмента, где угол не определен, составит $2n$ пикселов. Это является серьезной помехой при обработке изображения.

Чтобы уменьшить влияние разрыва, можно уменьшить порог и/или попытаться извлечь больше информации из имеющихся данных. Например, направление касательной можно оценить по паре пикселов кривой, расположенных по одну сторону от центра окна. При этом можно использовать аппроксимацию окружностями (2.3) или сплайнами (4.4b). Чтобы отобрать допустимые пары пикселов **a** и **b**, можно использовать прежнее ограничение на угол CAB (здесь снова C - центр окна, A и B - центры пикселов **a** и **b**, соответственно, AC < BC). Однако этого мало, так как у соседних пикселов **a** и **b** некоторые вершины совпадают, поэтому такие пары использовать нельзя. Также нежелательно использование пикселов, соседних с центральным. В двусторонних парах близость пикселов влияет меньше и подобные ограничения не нужны.

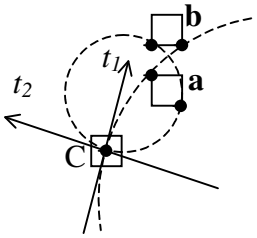


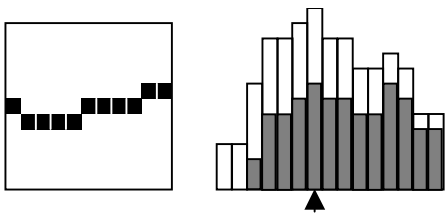
Рис. 10. Разброс односторонней оценки угла касательной при аппроксимации окружностью.

Еще один пример неудачной пары пикселов показан на Рис. 10. Хотя ограничение для угла CAB выполнено, близость пикселов и аппроксимация окружностью дают слишком большой интервал направлений касательных t_1 и t_2. Такие пары просто увеличивают фоновое значение счетчиков, но могут и ухудшить оценку. Расчеты показывают, что при использовании односторонних пар аппроксимация окружностью всегда уступает по точности аппроксимации сплайном, и поэтому далее она не рассматривается.

Разрывы кривых позволяют исследовать границы применимости метода. В качестве исходных данных будем использовать семейство дуг эллипсов, описанное в п. 3, и локальное окно размером $W = 11$. Разрыв образуется путем удаления заданного числа пикселов с одной стороны от центра. Положение разрыва определяется параметром z, равным смещению разрыва вдоль кривой относительно центра окна. Положительные значения z указывают, что разрыв смещается в заданном направлении касательной, отрицательные - в противоположном. Зависимость погрешности оценки угла касательной от параметров метода приводится в следующей главе, после внесения необходимых изменений в метод.

6. Учет ширины интервалов углов

Обычно в методе Хафа счетчики увеличиваются на 1. При определении направления касательной длина интервала углов определяется разбросом оценок угла для данной пары пикселов. Более короткие интервалы формируют пик, по которому вычисляется результат. Более длинные интервалы чаще всего увеличивают фоновое значение счетчиков и не влияют на результат. Однако, если конец длинного интервала попадает на пик, результат изменяется. Чтобы уменьшить влияние больших интервалов, можно ввести зависимость инкремента счетчиков Δ_c от длины интервала L. Ряд экспериментов показал, что зависимость $\Delta_c = C / L$, где C - константа, снижает среднюю погрешность. При этом снижение оказалось примерно одинаковым для аппроксимации окружностями и сплайнами: около 4% для малой асимметрии кривых и от 1 до 1.5% - при средней и большой асимметрии. Константу C можно подбирать для различных способов аппроксимации; здесь использовалось $C = 130$ для окружностей и $C = 110$ для сплайнов.



(a) Окно (b) Диаграммы счетчиков

Рис. 11. Пример учета влияния ширины интервала углов.

На Рис. 11 показан пример улучшения оценки направления касательной при учете ширины интервала углов. На Рис. 11a приведено изображение в локальном окне, а на Рис. 11b - диаграммы счетчиков с учетом ширины интервала (светлые полосы) и без учета (темные полосы). В обоих случаях есть второй пик, создаваемый концами длинных интервалов, однако в первом случае этим интервалам соответствует меньший инкремент, и истинный пик оказывается выше.

Чтобы отделить случаи, когда для определения направления касательной недостаточно данных, используется пороговое значение для максимума счетчиков.

Если инкремент счетчиков зависит от длины интервала, то значение счетчика - это количество пар пикселов, которые увеличили данный счетчик, с весами. Вес для более короткого интервала выше, поэтому появляется возможность отбирать решения, полученные при меньшем разбросе оценок угла, т. е. более точные.

Пороговое значение счетчиков удобно подбирать при помощи изображений кривых с разрывами, описанных в п. 5. Потребуем, чтобы для разрывов длины 2 не менее чем 80% решений превышали порог и имели приемлемую погрешность. В случае разрывов длины 3 данных для определения направления явно недостаточно, поэтому доля решений не имеет значения. Однако отобранные решения в этом случае также должны иметь приемлемую погрешность.

Подбор на массиве из 90×16×5 = 7200 кривых в сочетании с 8 положениями разрыва дал следующие пороговые значения: 46 для аппроксимации окружностями, 39 и 45 для аппроксимации сплайнами при использовании односторонних пар пикселов и без него, соответственно. Доля неопределенных углов для разрывов длины 2 составляет при этом от 15 до 19%. Максимальная погрешность угла везде не превышает 12.5°. Отметим, что если пороговое значение задавать для пар пикселов, такой отбор решений будет просто невозможен, так как интервал допустимых значений порога очень узок.

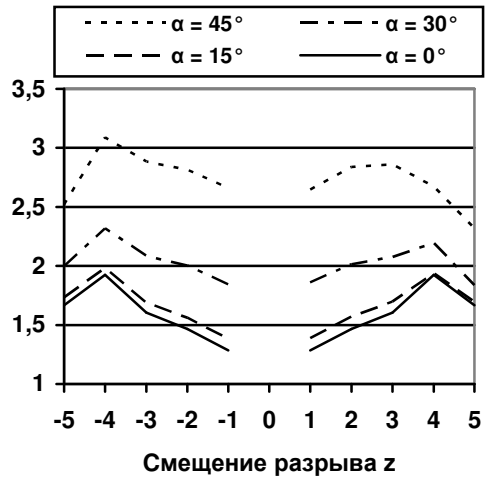


Рис. 12. Средняя погрешность определения угла касательной в зависимости от смещения разрыва длины 2 при различных значениях α, показателя асимметрии кривой.

На Рис. 12 показаны результаты вычисления направления касательной при наличии разрыва кривой при аппроксимации окружностями и с применением полученного выше значения порога. Приведена зависимость средней погрешности определения угла от положения разрыва относительно центра окна (параметра z) для различной асимметрии кривой (параметра α). Здесь берется среднее значение погрешности как по углам наклона касательной, так и по значениям радиуса кривизны ρ от 10 до 40. Максимальная абсолютная погрешность меняется с ростом α от 8° до 12.5°.

На Рис. 12 можно видеть, что с ростом асимметрии кривой растет и погрешность. При значениях z = -4 и 4 в разрыв попадают пикселы кривой, наиболее удаленные от центра, поэтому точность здесь ниже всего. При z = -5 и 5 разрыв частично выходит за пределы окна, и точность растет.

Аппроксимация сплайнами дает примерно те же результаты. Максимальная погрешность отличается на 0.5° в ту или другую сторону, так что говорить о превосходстве какого-либо подхода нельзя.

7. Оценка влияния шума

Как указывается в [2], для дискретных кривых модель шума, основанная на наложении некоррелированного гауссова шума, очень несовершенна. Шум может проявляться в искажении самой кривой и/или изменениях фона. В обоих случаях шум моделирует искажение исходного изображения, прошедшее через детектор границ либо иное средство получения кривой.

Для оценки влияния шума на вычисление направления касательной в данной работе использовано только искажение кривой как наиболее простое. Единичное искажение кривой - это перенос одного пиксела кривой в соседнюю позицию, при котором кривая сохраняет связность (разрывы при этом не рассматриваются). Уровень шума задается числом единичных искажений, последовательно примененных к дискретной кривой в локальном окне. Выбор пиксела на каждом шаге происходит равновероятно из всех доступных возможностей.

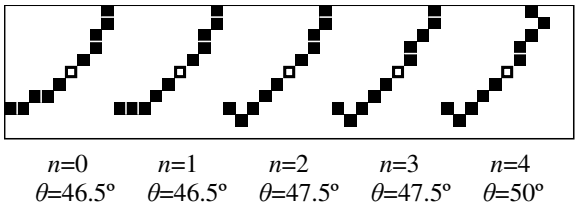


Рис. 13. Последовательное добавление шума к дискретной кривой.

На Рис. 13 показан пример последовательного добавления шума к дискретной кривой. Исходная кривая - дуга эллипса с a = 14.14, b = 10, центром при α = 30° и заданным наклоном касательной φ. = 45°. Центральный пиксел окна везде помечен белой точкой. Вычисленный наклон касательной меняется от θ = 46.5° (нулевой шум) до θ = 50° при уровне шума n = 4.

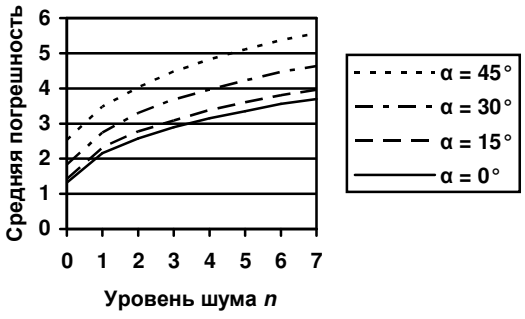


Рис. 14. Влияние уровня шума на среднюю погрешность вычисления наклона касательной.

На Рис. 14 показаны зависимости средней погрешности от уровня шума для различной асимметрии кри-

вой. Видно, что кривые отличаются в основном масштабом. Это указывает на определенную независимость влияния асимметрии исходной кривой и уровня шума на погрешность определения направления касательной.

8. Заключение

Рассмотренный метод оценки угла касательной к изображению кривой является мощным и надежным средством анализа границ, полученных при обработке исходного изображения, при условии правильного выбора параметров метода. Анализ размеров локального окна, используемого при оценке угла, показал, что размер $W = 11$ является наиболее приемлемым в общем случае, когда о классе кривых известно мало. Сравнение аппроксимации при помощи окружностей и сплайнов, в том числе с односторонней аппроксимацией, показало, что преимущества сплайнов проявляются не всегда. Модификация процедуры голосования в методе Хафа приводит к более точным результатам и дает возможность точнее отбирать решения при помощи порогового значения счетчиков. Оценки влияния шума показали скорость роста погрешности при увеличении уровня шума.

Литература

1. T.Pavlidis. Application of splines to shape description. In Visual Forms - Analysis and Recognition, Proceedings of InternationalWorkshop on Visual Form, 1991, 431–441.

2. J.Matas, Z.Shao, J.Kittler. Estimation of Curvature and Tangent Direction by Median Filtered Differencing. In Proceedings of ICIAP, 1995, 83-88.

3. J.-O.Lachaud, F.de Vieilleville. Comparison and improvement of tangent estimators on digital curves. Pattern Recognition, Vol. 42, No 8, 2009, 1693-1707.

4. Ciu, Qian; Wang, Lin. Tangent Estimation Based on Digital Straight Lines. CISP '08. Congress on Image and Signal Processing, 2008, 830-833.

5. A.Kimura, T.Watanabe. Generalized Hough Transform to Be Extended as an Affine-Invariant Detector of Arbitrary Shapes. Electronics and Communications in Japan, Part 2, Vol. 87, No. 6, 2004.

Some improvements of a tangent estimator based on Hough transform

P.P. Koltsov, N.V. Kotovich, A.A. Kravchenko, A.S. Kutsaev, A.S. Osipov and A.V. Zakharov

Abstract: A method of calculation of a tangent to a digital curve, based on Hough transform is considered. The choice of the optimal size of a computing window is analyzed. A more exact voting procedure in the Hough method is proposed that takes into account the scatter of individual angle estimations. The range of counters values thus grows that allows a more precise thresholding for a fixed error limit in the presence of curve breaks. The influence of noise level on accuracy of tangent estimation is also estimated.

Моделирование динамики колес в виртуальных сценах

М.В. Михайлюк[1], **А.М. Трушин**

1 – доктор физико-математических наук

Рассматривается задача моделирования динамики колес в виртуальных трехмерных сценах компьютерных тренажерных комплексов. Предлагается физическая модель расчета динамики колеса, учитывающая силы, действующие на колесо (силы трения, сила от двигателя, сила тяжести и т.д.). Описывается механизм определения соприкосновения колеса с подстилающей поверхностью. Описывается понятие подвески колеса и дается описание одной из типов подвески колеса и механизма ее работы.

Введение

В тренажерных комплексах часто используются механические средства передвижения, осуществляющие перемещения на колесной базе. Колеса прикрепляются к корпусу, и за счет их вращения происходит механическое перемещение всего подвижного транспортного средства (далее ПТС). В рассматриваемой модели все колеса привязываются к корпусу (колесной базе) ПТС, характеризующемуся положением его центра масс. Конечной задачей является расчет силы, создаваемой колесом за счет его вращения с учетом силы трения о поверхность, на которой колесо находится в данный момент, которая в итоге прикладывается к центру масс корпуса ПТС и благодаря этому осуществляется его передвижение. Моделирование динамики колес является практически неотъемлемой частью большинства виртуальных тренажерных систем. Настоящая работа выполнена при поддержке РФФИ (грант № 10-07-00404а).

1. Динамика колеса

В настоящей модели колесо имеют следующие характеристики:

– радиус R и ширина H колеса;

– масса m, приходящаяся на колесо от ПТС;

– инертная масса I_y колеса относительно и момент J инерции колеса относительно оси вращения;

– значение M_{Tp} момента силы трения на оси колеса;

– материал протектора, из которого по таблице определяются коэффициенты $k_{ск}$ и $k_{кач}$ трения (скольжения и качения) колеса о поверхность дороги;

– коэффициенты k жесткости рессоры и c затухания колебания в рессоре;

– наличие (или отсутствие) двигателя, поворотного механизма для управления от руля и тормозной муфты;

– тип подвески.

Расчет динамики колеса производится для каждого кадра визуализации, т.е. промежуток dT времени между текущим и следующим моментами в расчете равен промежутку времени между кадрами. Целью расчетов является вычисление новых значений силы тяги, приложенной к колесу от двигателя, сил трения колеса о поверхность земли, а также текущий угол α поворота колеса вокруг своей оси.

Систему координат в колесе выберем следующим образом (см. рис. 1). Центр О системы координат поместим в центре внешней стороны колеса, ось x направим внутрь колеса вдоль оси его вращения, ось y по направлению движения ПТС вперед, а ось z –

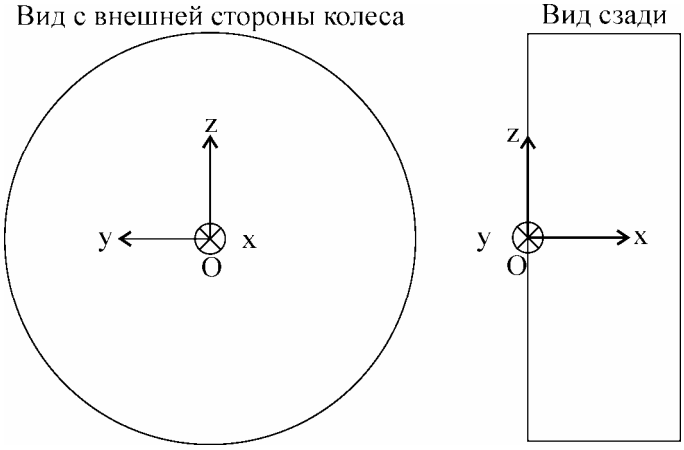


Рис. 1. Расположение системы координат в колесе

вверх. Для левого колеса эта система координат будет правосторонней (для правого – левосторонней).

Рассмотрим все силы, действующие на колесо вдоль осей его системы координат (см. рис. 2). Вдоль

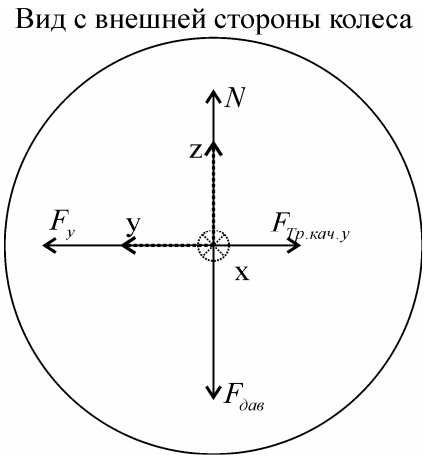


Рис. 2. Силы, действующие на колесо

оси z колесо давит на поверхность, на которой оно в данный момент находится, с силой $F_{дав}$ давления. От поверхности в свою очередь на колесо действует сила N реакции опоры (см. рис. 3). Если колесо движется, то на него действуют также силы трения, а именно сила $F_{Тр.кач.y}$ трения качения вдоль оси y противоположно направлению движения и сила $F_{Тр.ск.x}$ поперечного трения скольжения, направленная вдоль оси x. Кроме того, если колесо имеет двигатель, то от последнего на колесо действует также сила F_y тяги. Вычислим теперь все эти силы.

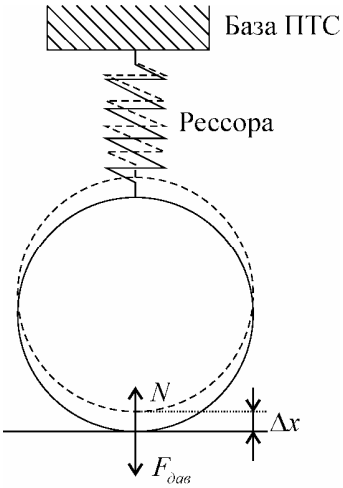


Рис. 3. Сила реакции опоры

Сила $F_{дав}$ давления на колесо включает в себя силу тяжести массы m, приходящейся на это колесо от ПТС, текущую силу сжатия рессоры и силу, вызывающую затухание ее колебаний за счет трения среды. Таким образом, получаем:

$$F_{дав} = mg + k\Delta x + c\Delta v \qquad (1)$$

где m (масса), k (жесткость рессоры) и c (коэффициент затухания колебаний в рессоре) задаются в параметрах колеса, Δx – изменение длины рессоры (при сжатии/растяжении) по сравнению с предыдущим шагом расчета, а $\Delta v = \frac{\Delta x}{dT}$ – скорость сжатия/растяжения рессоры.

Рассмотрим два случая: когда колесо находится на земле и когда колесо не касается земли.

1.1. Колесо находится на земле

В этом случае сила реакции опоры равна по величине и обратна по направлению силе $F_{дав}$ из (1):

$$N = -F_{дав} \qquad (2)$$

На колесе может находиться тормозная муфта, которая имеет два состояния: полностью зажата или полностью отпущена. В первом случае колесо не вращается и силу $F_{Тр.кач.y}$ трения вдоль оси y устанавливаем равной бесконечности (очень большой величине). Во втором случае колесо вращается, и сила трения возникает как от поверхности, по которой катится колесо, так и от трения в оси колеса. Для сил трения получаем следующие формулы:

$$F_{Тр.ск.x} = k_{ск} N$$

$$F_{Тр.кач.y} = \frac{k_{кач} N}{R} + \frac{M_{Тр.}}{R},$$

где N – сила реакции опоры из (2), R – радиус колеса, $k_{ск}$ и $k_{кач}$ – коэффициенты трения (скольжения и качения) колеса о поверхность дороги, $M_{Тр}$ – момент силы трения в оси колеса (задается как параметр колеса). Коэффициенты $k_{ск}$ и $k_{кач}$ берутся из таблицы, исходя из материала протектора колеса и материала поверхности, по которой колесо в данный момент движется.

Для вычисления силы F_y тяги рассмотрим случай, когда колесо имеет двигатель и тормозной муфты нет или она не зажата. Тогда сила F_y будет определяться дополнительным моментом силы M, создаваемым двигателем на колесе. Можно считать, что сила приложена к ободу колеса и тогда

$$F_y = \frac{M}{R},$$

где R – радиус колеса, а дополнительный момент вычисляется по формуле

$$M = \left(U - \frac{\omega}{\omega_{xx}}\right) M_{П} \qquad (3)$$

В этой формуле (см. [1]) U – напряжение, подаваемое на двигатель пользователем с пульта управления, ω – текущая угловая скорость вращения вала двигателя, ω_{xx} – угловая скорость холостого хода, а $M_{П}$ – пусковой момент.

Если зажата тормозная муфта или на колесе нет двигателя, то сила тяги отсутствует, т.е. $F_y = 0$.

Рассмотрим вычисление величины α угла поворота колеса вокруг своей оси. Напомним, что мы рассматриваем случай, когда колесо находится на земле. Если рассматривать колесо само по себе, т.е. не

связанным с остальными частями транспортного средства, то его линейная скорость $\upsilon_к$ определяется угловой скоростью вращения и ускорением, обусловленным силой F_y тяги. Для вычисления можно использовать следующую формулу:

$$\upsilon_к = \omega R + \frac{F_y dT}{I_y},$$

где I_y – инертная масса колеса, которая вычисляется по значениям параметров колеса и двигателя на нем. Базовый (основной) объект в общем случае может иметь другую линейную скорость V_y вдоль оси y. В общем случае основной объект – это платформа, к которой крепятся колеса, вместе со всеми конструкциями на ней. Для выравнивания скорости колеса и базового объекта необходима сила

$$F_{выр} = \frac{\left|\upsilon_к - V_y\right| m}{dT}$$

Если эта сила будет превышать силу $F_{Тр.ск.y}$ продольного трения скольжения колеса, то колесо будет проскальзывать, т.е. его угловая скорость будет определяться линейной скоростью $\upsilon_к$, а сила тяги – силой $F_{Тр.ск.y}$. Поэтому можно записать:

$F_{Тр.ск.y} = k_{ск} N$

Если $F_{выр} \geq F_{Тр.ск.y}$, то

$\omega = \frac{\upsilon_к}{R}$

$\alpha = \alpha + \omega \cdot dT$

$F_{Тр.кач.,y} = 0$

$F_y = F_{Тр.ск.y}$

Конец Если

1.2. Колесо не касается земли

Если колесо не касается земли, то $F_{дав} = -N = 0$, $F_{Тр.ск.x} = 0$, $F_{Тр.кач.y} = 0$, $F_y = 0$, $F_{макс.тр.ск.y} = 0$. Если при этом зажата тормозная муфта, то $\omega = 0$ и α не изменится. В противном случае колесо вращается с некоторой угловой скоростью ω и можно вычислить реальный момент силы $M_{вр}$ относительно его оси:

$$M_{вр} = \frac{\omega J}{dT},$$

где J – момент инерции колеса относительно оси x (задается как параметр колеса). Тормозящий момент $M_{Тор}$ не должен превышать максимального значения $M_{Тр}$ момента силы трения на оси колеса, которое задается как параметр колеса. Вычисление $M_{Тр}$ производится следующим образом:

Если $M_{вр} > M_{Тр}$, то

$M_{Тор} = M_{Тр}$

иначе

$M_{Тор} = M_{вр}$

Если колесо имеет двигатель, то этот двигатель создает дополнительный момент силы, величина которого вычисляется по формуле (3). Если двигателя нет, то $M = 0$. За счет трения колеса в оси дополнительный момент M уменьшается:

$$M = M - M_{Тор}$$

Уменьшается также угловая скорость и угол поворота колеса:

$$\omega = \omega + \frac{MJ}{dT}$$

$$\alpha = \alpha - \omega dT$$

2. Корректировка динамических характеристик основного объекта

Как отмечалось выше, основной объект – это платформа (со всеми необходимыми конструкциями на ней), к которой крепятся колеса. Поэтому движение основного объекта (рассчитанное в предыдущем кадре), естественно, передается колесам. Колеса испытывают трение о дорогу, которое изменяет движение основного объекта. Рассмотрим, как при этом изменяется линейная и угловая скорости основного объекта.

2.1. Корректировка линейной скорости основного объекта за счет трения на колесе

Текущие силы трения колеса о дорогу уменьшают линейную скорость движения основного объекта. Вычислим это уменьшение в системе координат E_0 основного объекта. Обозначим через V_0 линейную скорость основного объекта в системе E_0, а через $V_0^{'} = (V_{0,x}^{'}, V_{0,y}^{'}, 0)$ вектор уменьшения скорости V_0 за счет сил трения колеса о дорогу (последняя координата равна нулю, т.к. силы трения для основного объекта действуют только в плоскости xy системы координат E_0). Силы, которые надо приложить, чтобы за время dT уменьшить до нуля скорости вдоль осей x и y, равны:

$$Q_x = \frac{V_{o,x} m}{dT}$$

$$Q_y = \frac{V_{o,y} m}{dT}$$

Максимальная сила трения скольжения вдоль оси x равна $F_{Тр.ск.o.x} = F_{Тр.ск.x} Cos(Z_w, Z_m)$, где $Cos(Z_w, Z_m)$ – косинус угла между осями Z_w и Z_m мировой системы координат и системы координат колеса (совпадающей с E_0). Если сила Q_x не будет превышать максимальную силу $F_{Тр.ск.o.x}$ трения скольжения вдоль оси x, то скорость $V_{0,x}$ полностью погасится силой трения. В противном случае скорость уменьшится силой трения $F_{Тр.ск.o.x}$. Поэтому уменьшение скорости $V_{0,x}^{'}$ можно вычислить следующим образом:

Если $Q_x > F_{Тр.ск.о.x}$, то

$$Q_x = F_{Тр.ск.о.x}$$

Конец Если

$$V'_{0,x} = \frac{Q_x dT}{m}$$

Аналогично вычисляется уменьшение $V'_{0,y}$ скорости основного объекта вдоль y:

$$F_{Тр.кач.о.y} = F_{Тр.кач.y} Cos(Z_w, Z_m)$$

Если $Q_y > F_{Тр.кач.о.y}$, то

$$Q_y = F_{Тр.кач.о.y};$$

Конец Если

$$V'_{0,y} = \frac{Q_y dT}{m}$$

Переведем полученный вектор скорости V'_0 в мировую систему координат и обозначим результирующий вектор через $V'_{0,w}$. Тогда новая скорость $V_{0,w}$ основного объекта в мировой системе координат изменится следующим образом:

$$V_{0w} = V_{0w} - V'_{0w}$$

2.2. Корректировка угловой скорости основного объекта за счет трения на колесе

Обозначим через I_z инерцию основного объекта относительно оси z его системы координат, а через D_x – расстояние по оси x системы координат основного объекта от его центра масс до положения O начала системы координат колеса (O находится на пересечении внешней стороны колеса и его оси, см. рис. 1). Фактически это будет плечо силы, приложенной в точке P, относительно оси z основного объекта. Если I_z положительно и D_x не близко к нулю, то угловая скорость Ω_z основного объекта или полностью погасится или уменьшится за счет момента силы трения $F_{Тр.кач.y}$. Это можно записать следующим образом:

Если $I_z > 0$ и $D_x > 0.2$, то

$$W = \frac{\Omega_z I_z}{D_x dT}$$

Если $W > F_{Тр.кач.y}$, то

$$W = F_{Тр.кач.y}$$

Конец Если

$$\Omega_z = \Omega_z - \frac{W D_x dT}{I_z}$$

Конец Если

Здесь W обозначает силу, которую надо приложить для полного погашения угловой скорости Ω_z за время dT. Если $W > F_{Тр.кач.y}$, то Ω_z уменьшается моментом, созданным силой $F_{Тр.кач.y}$. Если же $W \le F_{Тр.кач.y}$, то Ω_z полностью гасится.

2.3. Корректировка линейной и угловой скорости основного объекта за счет силы тяги F_y на колесе.

Если колесо находится на земле, то передадим основному объекту остальные силы, действующие на колесо. К этим силам относятся сила F_y тяги, сила тяжести и сила давления от рессоры. Рассмотрим вектор $F = (0, F_y, 0)$ в локальной системе координат колеса и обозначим через φ угол поворота колеса от руля (если поворотный механизм у колеса отсутствует, то $\varphi = 0$). На данном этапе расчетов поворот колеса еще не осуществлен, поэтому F_y направлена вдоль оси транспортного средства. Повернем вектор F в плоскости xy на угол φ, переведем полученный вектор в мировую систему координат и обозначим его F_w. Пусть $\bar{n}$ – вектор нормали к земной поверхности в текущей точке положения транспортного средства. Если колесо находится на горизонтальной плоскости, то силы давления $F_{дав}$ и реакции опоры N равны и направлены в разные стороны оси z. Если же транспортное средство находится на наклонной поверхности (см. рис. 4), то сила mg тяжести направлена в отрицательном направлении оси z, а

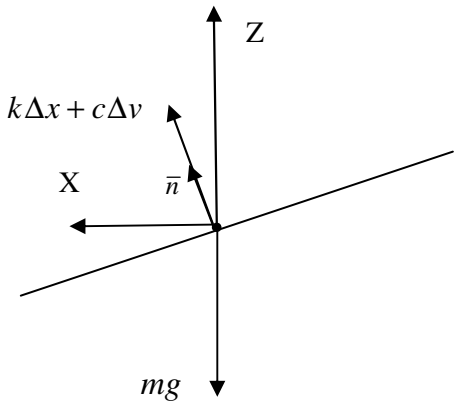


Рис. 4. ПТС на наклонной поверхности

сила $(k\Delta x + c\Delta v)$ давления от рессоры направлена вдоль нормали $\bar{n}$. Проецируя эти силы на оси координат, и добавляя их к силе F_w, получим:

$$F_{w,x} = F_{w,x} + (k\Delta x + c\Delta v) n_x$$

$$F_{w,y} = F_{w,y} + (k\Delta x + c\Delta v) n_y$$

$$F_{w,z} = F_{w,z} + (k\Delta x + c\Delta v) n_z - mg$$

Текущие сила $F_{об}$ и момент $M_{об}$, приложенные к основному объекту, корректируются с учетом силы F_w следующим образом:

$$F_{об} = F_{об} + F_w$$

$$M_{об} = M_{об} + [F_w, P]$$

При расчете динамики основного объекта эти величины будут участвовать в вычислении нового значения линейной и угловой скоростей основного объекта.

2.4. Поворот колеса от руля

Основному объекту надо также передать угол поворота, на который повернет транспортное средство за время dT, если колесо повернуто на угол φ. Обозначим радиус поворота транспортного средства через r. Приближенно можно считать, что расстояние s между передним и задним колесом равно длине дуги радиуса r между этими колесами (см. рис. 5). Тогда радиус поворота r можно вычислить по формуле:

$$r = \frac{s}{\varphi}$$

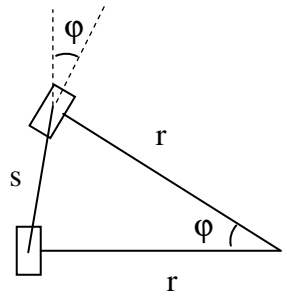


Рис. 5. Поворот ПТС

За время dT колеса пройдут расстояние:

$$L = vdT = \omega RdT,$$

следовательно, угол поворота транспортного средства за это время равен:

$$\psi = \frac{L}{r} = \frac{\omega RdT\varphi}{s}$$

Именно этот угол и надо передать основному объекту для вычисления его поворота с учетом рассчитанной угловой скорости Ω_z.

3. Вычисление высоты колеса относительно земли

Земная поверхность задана регулярной сеткой высот, на основе которой построена ее полигональная модель. Для определения наличия контакта между колесом и поверхностью на нижней части обода колеса вводится некоторый набор P из 12 точек (см. рис. 6).

Здесь R – радиус колеса, H – ширина колеса, $h = H/2$ – половина ширины колеса. Эти точки имеют следующие координаты:

$$P_{2i-2} = (h, R\sin(\alpha \cdot i), -R\cos(\alpha \cdot i)),\ i = 1 \div 5$$

$$P_{2i-1} = (h, -R\sin(\alpha \cdot i), -R\cos(\alpha \cdot i)),\ i = 1 \div 5$$

$$P_{10} = (0, 0, -R)$$

$$P_{11} = (H, 0, -R)$$

Для каждой из этих точек вычислим расстояние до поверхности земли и выберем минимальное из этих расстояний Δz. Для этого переведем координаты каждой i-ой точки из локальной системы координат колеса в мировую систему координат и полученные координаты обозначим через $(x_{P_i}, y_{P_i}, z_{P_i})$. Для каждой пары (x_{P_i}, y_{P_i}) вычисляется высота z_i земной поверхности в мировой системе координат (с помощью интерполяции в полигональной модели) и разность

$$\Delta z_i = z - z_{P_i}$$

Таким образом, Δz_i – расстояние от i-ой точки колеса до поверхности земли под ней, причем $\Delta z_i < 0$, если i-я точка находится над поверхностью земли, и $\Delta z_i > 0$, если i-я точка находится ниже поверхности земли. Тогда

$$\Delta z = \min(\Delta z_i), \quad i = 0,...,11$$

Эту разницу в высоте положения колеса и земной поверхностью должна компенсировать подвеска колеса. Подвеска имеет ограничения L_t и L_b по величинам своего хода вверх-вниз (см. п. 4). Для учета этих ограничений нужно скорректировать Δz следующим образом:

если $\Delta z > L_t$, то $\Delta z = L_t$,

если $\Delta z < L_b$, то $\Delta z = L_b$.

В первом случае точка на поверхности земли находится выше колеса, при этом подвеска срабатывает на максимально возможный уровень вверх, но точка на земле все равно остается выше нижней точки колеса. Тогда осуществляется переориентация с учетом этого оставшегося расстояния центра масс ПТС таким образом, что за счет общего наклона ПТС компенсируется это расстояние. Во втором случае подвеска срабатывает на максимальный уровень вниз, но контакта с землей все равно не происходит. В этом случае формируется флаг отсутствия соприкосновения с землей, и дальнейшие расчеты производятся как описано в пункте 1.2.

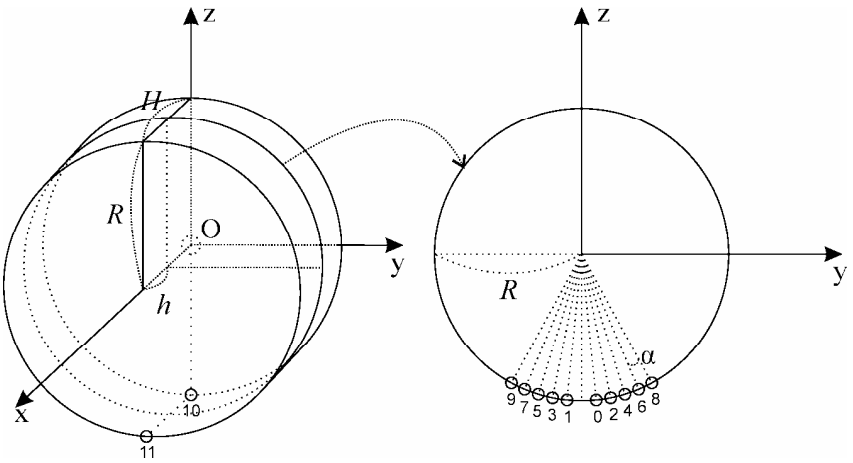


Рис. 6. Набор точек на нижней части обода колеса

4. Моделирование подвески колеса

Подвеска используется для соединения колеса с платформой ПТС и предназначена для обеспечения плавности движения платформы, преодоления неровностей поверхности, снижения динамических нагрузок, обеспечения точности управления и т.д. Это достигается применением в подвесках рессор, пружин, амортизаторов и т.д.

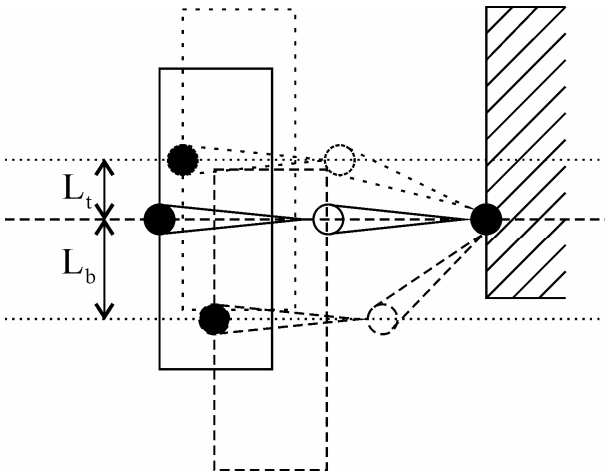


Рис. 7. Ограничения хода подвески

Подвеска с одной стороны жестко прикрепляется к колесу вдоль оси его вращения, а с другой стороны – к основному объекту (платформе) транспортного средства (см. рис. 7). Черные кружки обозначают места жесткого крепления. Основными характеристиками подвески являются ее конструкция (определяющая ход подвески, т.е. перемещение колеса относительно платформы ПТС), а также ограничения хода подвески вверх-вниз L_t и L_b. Исходной информацией для расчета динамики подвески в текущем кадре является расстояние Δz от нижней кромки колеса до подстилающей поверхности (см. рис. 8 и п. 3). За счет своего хода подвеска должна обеспечить поднятие (при $\Delta z > 0$) или опускание (при $\Delta z < 0$) колеса на нужную величину до касания с подстилающей поверхностью. Поэтому основной задачей является вычисление на основании Δz соответствующих углов поворота свободных конструкций подвески.

Рассмотрим задачу моделирования на примере трапециевидной поперечной подвески (см. рис. 9).

Для имитации подвески можно использовать средства скелетной анимации, а именно представить подвеску в виде иерархической модели, составленной из «костей». Для трапециевидной подвески модель показана на рис. 10. Из рисунка видно, что если повернуть кость b_3, то вместе с ней будут поворачиваться и ее потомки (в данном случае кости b_2, b_1, b_0, b_4 и К). Здесь В соответствует основному объекту (платформе), К – колесу, а b_0 – оси колеса. Заметим, что кость b_5 останется на месте, т.к. по иерархии она не привязана к кости b_3.

Пусть в предыдущем кадре подвеска находилась в некотором состоянии, характеризующемся величиной Δh ее отклонения от состояния покоя. Обозначим угол между b_3 и b_2 через β_0, а угол между b_4 и b_5 через γ_0. В текущем кадре при расчете динамики колеса была вычислена новая величина Δz отклонения.

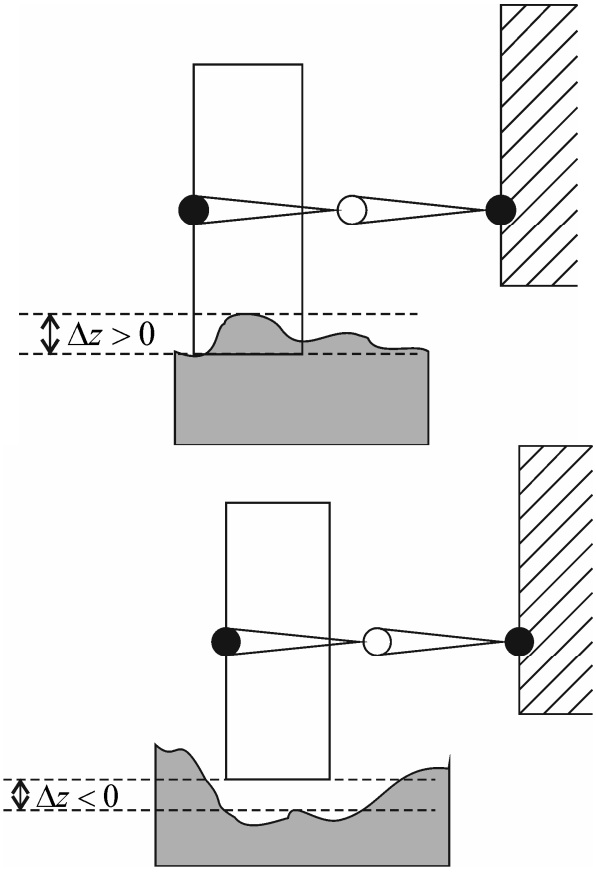


Рис. 8. Положение колеса относительно подстилающей поверхности

Задача состоит в вычислении углов поворота нужных костей, при которых колесо будет касаться подстилающей поверхности. Повернем кость b_3 на некоторый угол λ так, чтобы после поворота отклонение подвески стало равным Δz. Как отмечалось выше, вместе с b_3 повернутся все остальные кости, кроме b_5 и конфигурация костей будет такой, как изображено на рис. 11.

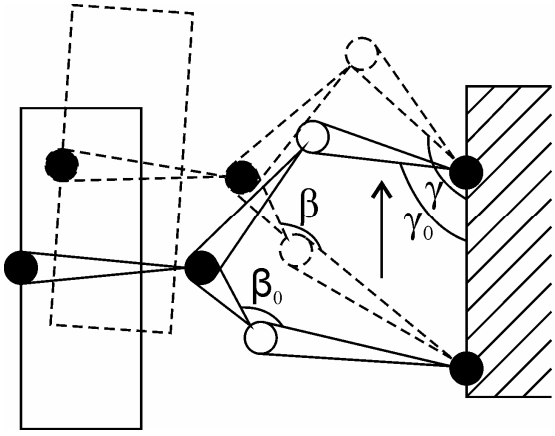


Рис. 9. Трапециевидная поперечная подвеска

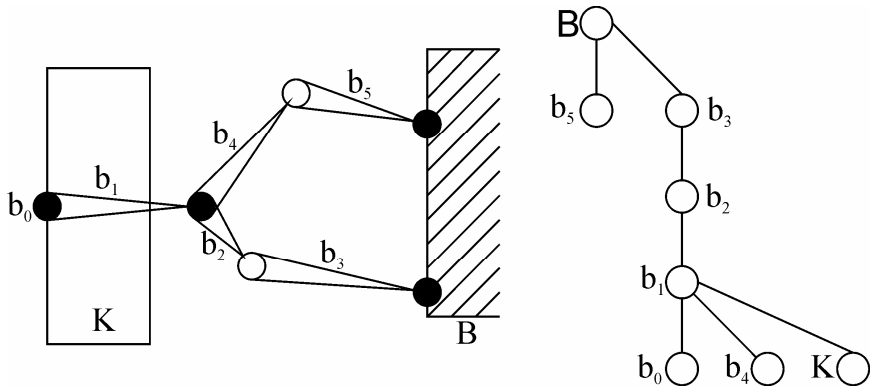


Рис. 10. Модель трапециевидной поперечной подвески

Для восстановления правильной конфигурации (см. рис. 12) достаточно повернуть кость b_2 (вместе с костью b_1) на некоторый угол $(-\beta')$ и кость b_5 – на угол γ'. Для вычисления угла λ заметим, что интервал времени между кадрами в реальном масштабе времени очень мал (около 40 мсек.) и за это время перемещение подвески также будет очень малым. Поэтому вместо угла λ можно вычислять значение синуса этого угла. Если обозначить через L длину кости b_3, то получим:

$$\lambda \simeq Sin\lambda = \frac{\Delta z - \Delta h}{L}$$

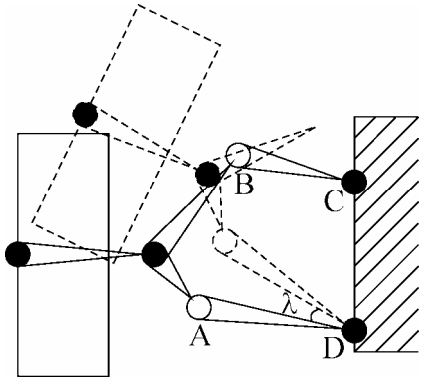


Рис. 11. Промежуточное состояние костей трапециевидной подвески

Для вычисления остальных углов обратимся к рис. 12. Обозначим длины костей b_3 и b_5 через a и c соответственно, а длины отрезков AB, AC и CD – через b, e и d. Из треугольников ABC и ACD по теореме косинусов можно вычислить углы

$$\beta = \angle BAC + \angle CAD$$

$$\gamma = \angle BCA + \angle ACD$$

Так как в предыдущем кадре между костями уже были некоторые углы β_0 и γ_0, то для получения углов β и γ необходимо повернуть кость b_2 (вместе с костью b_1) на угол $(-\beta') = -(\beta - \beta_0)$ и кость b_5 – на угол $\gamma' = (\gamma - \gamma_0)$. После таких поворотов колесо встанет в нужное место относительно земной поверхности.

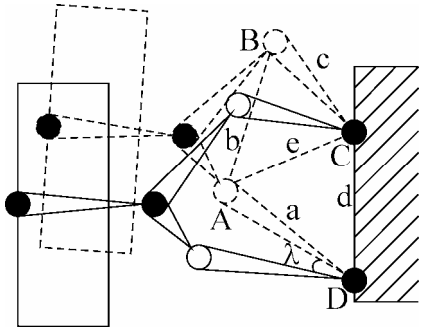


Рис. 12. Конечное состояние костей трапециевидной подвески

Литература

1. Михайлюк М.В., Трушин А.М. Моделирование динамики двигателей в виртуальных сценах. Сб. статей НИИСИ РАН «Моделирование и визуализация. Многопроцессорные системы. Инструментальные средства разработки ПО», М., 2010, стр. 48-55.

2. Сивухин Д.В. Общий курс физики. Учеб. пособие: Для вузов. В 5 т. Т.I. Механика. – 4-е издание., стереот. – М.: ФИЗМАТЛИТ; Изд-во МФТИ, 2005. – 560 с.

3. Damping. Wikipedia The Free Encyclopedia. Available from: http://en.wikipedia.org/wiki/Damping

Modeling the dynamics of the wheels in virtual scenes

M.V. Mikhaylyuk, A.M. Trushin

Abstract. The problem of modeling the dynamics of the wheels in 3D scenes for computer simulators is considered. The physical model is proposed to calculate the wheel's dynamics, taking into account the forces acting on the wheel (friction forces, force from the engine, gravity, etc.). The mechanism for determining the contact between wheels and underlying surface is offered. In the paper it's described the concept of a wheel's suspension bracket. Also it's given a description of one of the wheel's suspension bracket's types and the mechanism of its work.

Моделирование гусениц транспортных средств в системах визуализации

М.В.Михайлюк

доктор физико-математических наук, профессор

Рассматривается задача моделирования гусениц в процессе визуализации модели гусеничного транспортного средства как огибающей линии вокруг колес. Положение колес рассчитывается в подсистеме динамики. При этом рассматриваются как колеса прямого, так и колеса обратного хода. Предлагаемый подход позволяет осуществлять моделирование движения гусеничных транспортных средств в масштабе реального времени.

Введение

Многие транспортные средства (тракторы, танки, роботы и т.д.) передвигаются на гусеничном ходу. Поэтому моделирование таких средств в трехмерных виртуальных сценах представляет важную задачу. Такое моделирование можно осуществлять в подсистеме динамики, если гусеница участвует в динамике движения. Если же при расчете динамики характеристики гусеницы не учитываются, то ее моделирование целесообразно производить в подсистеме визуализации. В этом случае в динамике участвуют лишь колеса, поэтому, рассчитав положения колес в текущий момент времени, достаточно только «правильно натянуть» на них гусеницу.

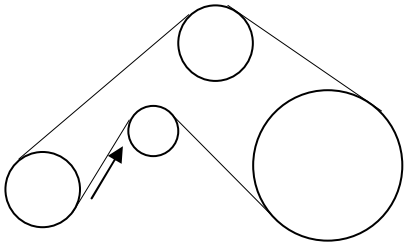


Рис. 1. Обход колес гусеницей

Исходные данные для моделирования гусеницы в подсистеме визуализации условно можно разделить на два класса: начальные исходные данные и динамические исходные данные. К первому классу относятся множество колес гусеницы (массив колес) и их начальные позиции и ориентации, а также объект – одно звено гусеницы (это звено будет повторяться вдоль линии прохождения гусеницы необходимое число раз). Так как в процессе движения колеса не всегда будут образовывать выпуклую оболочку, то мы будем считать, что в массиве колеса заданы в порядке их обхода гусеницей (т.е. в любой момент времени гусеница касается всех колес). Гусеница образует замкнутую линию, а колеса можно разделить на два типа: те, которые лежат внутри этой замкнутой линии – колеса прямого хода и те, которые лежат вне нее – колеса обратного хода (см. рис. 1.). Позиция и ориентация каждого колеса задается в виде модельно-видовой матрицы, тип колеса отмечается флагом, а звено гусеницы помещается в параллелепипед (box), параметрами которого являются длина, ширина и толщина звена. Ко второму (динамическому) классу исходных данных относятся текущие положения и ориентации всех колес (также в виде модельно-видовых матриц) и текущий угол поворота ведущего колеса.

1. Моделирование гусеницы с колесами прямого хода.

Рассмотрим сначала случай, когда все колеса гусеницы являются колесами прямого хода. Задача состоит в построении выпуклой замкнутой линии, состоящей из отрезков и дуг, при этом каждый отрезок касается двух колес, каждая дуга является частью окружности какого-либо колеса, и все колеса находятся внутри этой замкнутой линии (см. рис. 1, на котором огибающая показана тонкой линией). При этом может оказаться, что некоторые колеса не касаются огибающей. Выберем такое направление обхода огибающей, при котором при движении вдоль огибающей все колеса будут находиться слева (на рисунке это направление показано стрелкой).

Построение выпуклой огибающей для множества колес. Предположим, что уже построена часть огибающей от 0-го до i-го колеса и покажем, как построить участок огибающей до i+1 – го колеса. Обозначим вектор касательной между $i-1$–м и i–м колесом через D_i и рассмотрим все векторы касательных $D'_{i,j}$ между i-м колесом и остальными колесами (касательные правильные, т.е. такие, что колеса находятся слева от касательной). Первое колесо в массиве дублируется на последнем месте для построения касательной между ним и последним колесом. Вектор D_i будем называть локальным начальным направлением для i-го шага. Пусть $\alpha_{i,j}$ – угол между D_i и $D'_{i,j}$ ($0 \le \alpha_{i,j} \le 2\pi$) и пусть j_0 – номер колеса, для которого этот угол минимален. Тогда все колеса будут лежать слева от D'_{i,j_0}. Это легко понять, если мысленно «перекатывать» линию D_i по окружности i-го колеса до тех пор, пока она впервые не коснется какого-либо из ос-

 Статья поступила в редакцию 09.04.2012

тавшихся колес. Таким, образом, K_{j_0} – следующее колесо в списке выпуклой оболочки, $D_{i+1} = D'_{i,j_0}$ – следующий после D_i отрезок огибающей, а между ними – дуга колеса K_{j_0} от точки касания этого колеса отрезком D_i до точки касания отрезком D'_{i,j_0}. На рис. 2 видно, что угол между лучом AB и касательными между третьим колесом и колесами 1, 2 и 4 наименьший для колеса 4, т.е. для касательной CD. Если очередное колесо совпадает с первым, то процесс заканчивается. В этом случае необходимо подкорректировать дугу на первом колесе, задав ее от новой найденной точки касания до точки касания первой касательной (т.е. касательной между первым и вторым колесами). Для завершения алгоритма нам остается указать способ выбора первого колеса и первого локального направления. В качестве первого выбираем колесо с минимальным значением координаты x на окружности,

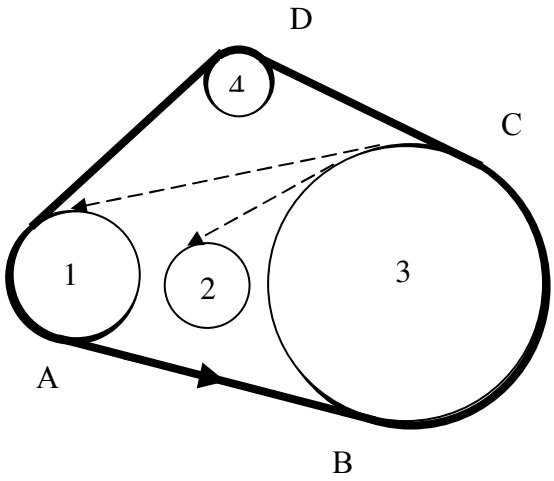


Рис. 2. Выбор следующего колеса

т.е. $\min_i(O_{x,i} - R_i)$, где $O_{x,i}$ - x-я координата центра i -го колеса, R_i – его радиус. Начальное направление задаем вертикальным вектором (0, -1).

Рассмотрим теперь, как построить направленную касательную к двум колесам K_1 и K_2, причем сначала предположим, что радиус R_1 первого колеса меньше радиуса R_2 второго. Эти два колеса можно соединить двумя различными касательными (AB и CD на рис. 3 а)). Так как колеса должны лежать слева от касательной (если двигаться по направлению этой касательной), то одна из них будет соответствовать движению от K_1 к K_2, а другая – движению от K_2 к K_1. Выберем направление движения от K_1 к K_2 (следовательно, будем строить касательную AB) и проведем отрезок O_1E, параллельный AB. Так как нам известны центры O_1 и O_2, а также радиусы R_1 и R_2 колес, то

$$\|AB\| = \|O_1E\| = \sqrt{\|O_1O_2\|^2 - (R_2 - R_1)^2}, \qquad (1)$$

$$\alpha = a\tan 2\left(R_2 - R_1, \|O_1E\|\right).$$

Здесь α - угол между O_1O_2 и AB. Функция $a\tan 2(y, x)$ вычисляет арктангенс величины y / x и принимает значения от -π до π. Обозначим через CDir единичный вектор вдоль оси O_1O_2, т.е. $CDir = \dfrac{O_1O_2}{\|O_1O_2\|}$. Тогда единичный вектор Dir направления касательной и точки A и B ее начала и конца можно вычислить по формулам:

$$\begin{aligned} Dir &= Rotate(CDir, -\alpha), \\ A &= O_1 + Rotate(CDir \cdot R_1, -\pi/2 - \alpha), \\ B &= A + Dir \cdot |AB|. \end{aligned} \qquad (2)$$

Процедура Rotate поворачивает вектор, заданный первым параметром на угол, заданный вторым параметром (за положительное направление вращения принимается вращение против часовой стрелки).

Пусть теперь радиус R_1 первого колеса больше радиуса R_2 второго колеса (см. рис. 3 б). Отрезок O_1E,

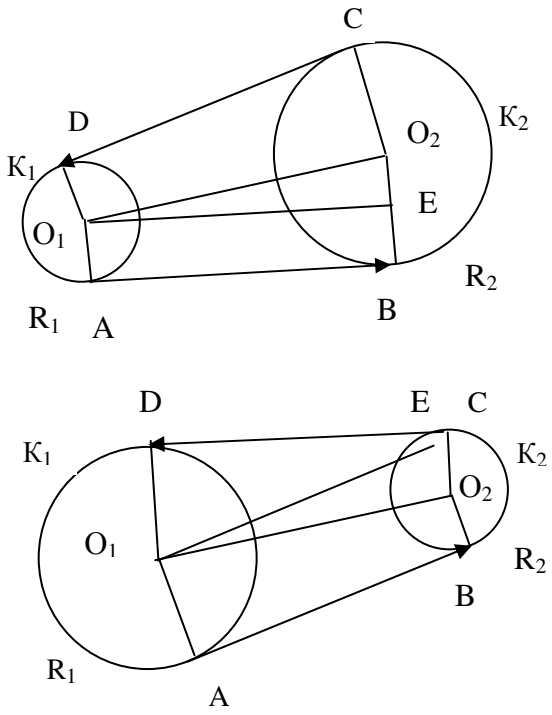


Рис. 3. Касательная к двум окружностям

параллельный касательной AB, получается из O_1O_2 поворотом на угол

$$\alpha_1 = a\tan 2\left(R_1 - R_2, \|O_1E\|\right), \quad \text{где}$$

$$\|AB\| = \|O_1E\| = \sqrt{\|O_1O_2\|^2 - (R_1 - R_2)^2}.$$

Обозначим через $\alpha = a\tan 2\left(R_2 - R_1, \|O_1E\|\right)$.

Тогда $\alpha_1 = -\alpha$,

$$\begin{aligned} Dir &= Rotate(CDir, \alpha_1) = Rotate(CDir, -\alpha), \\ A &= O_1 + Rotate(CDir \cdot R_1, -\pi/2 + \alpha_1) = \\ &= O_1 + Rotate(CDir \cdot R_1, -\pi/2 - \alpha), \end{aligned}$$

$$B = A + Dir \cdot \|AB\|.$$

Легко видеть, что формулы для вычисления касательной в этом случае полностью совпадают с формулами (1) и (2), то есть нет необходимости отдельно рассматривать случаи разного соотношения радиусов колес. Кроме того, точка A задает конец предыдущего дугового блока, а точка B – начало следующего дугового блока гусеницы. Таким образом, зная радиусы колес, можно вычислить длины всех дуговых блоков гусеницы. Рассмотрим этот вопрос более подробно.

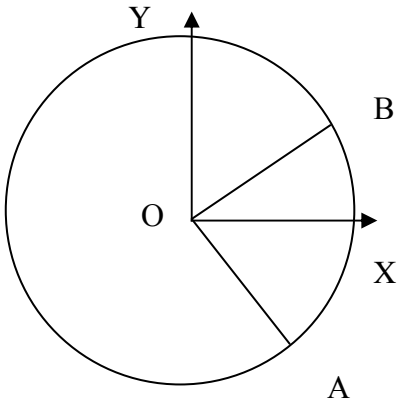


Рис. 4. Вычисление длины дуги AB

Пусть заданы точки A и B на ободе колеса радиуса R и требуется вычислить длину дуги AB (см. рис. 4.). Вычислим углы α и β, на которые надо повернуть ось OX для совмещения ее с отрезками OA и OB. Имеем:

$$\alpha = a\tan 2\left(A_y - O_y, A_x - O_x\right),$$
$$\beta = a\tan 2\left(B_y - O_y, B_x - O_x\right), \qquad (3)$$
$$\gamma = \beta - \alpha.$$

Угол γ может оказаться отрицательным – в этом случае заменяем его на $2\pi + \gamma$. Теперь длина дуги легко вычисляется по формуле $S = R\gamma$.

Таким образом можно построить выпуклую огибающую множества колес гусеницы. Эта огибающая будет представлять массив чередующихся линейных и дуговых блоков. Для линейного блока мы будем хранить координаты начала, вектор направления и длину, а для дугового блока – координаты начала и длину (зная радиус соответствующего колеса всегда можно вычислить остальные нужные параметры).

Вычисление длины гусеницы и числа звеньев в ней. Число звеньев гусеницы вычисляется на этапе предобработки или в первом кадре визуализации и не изменяется в остальных кадрах. Так как при движении длина гусеницы будет меняться (из-за изменения положения колес), то мы будем менять расстояние между звеньями гусеницы. Пример звена гусеницы показан на рис. 5. При изменении расстояния между звеньями будет меняться степень захода выпуклой части каждого звена в углубление следующего. В исходном состоянии степень захода равна половине. Обозначим через $L_з$ расстояние от левой границы звена до середины выпуклого участка ($L_з$ будет чуть меньше длины звена). Общая длина L гусеницы (в любом кадре) вычисляется как сумма длин блоков ее выпуклой огибающей. Тогда число N звеньев равно $N = \lfloor L / Lз \rceil$, где $\lfloor x \rceil$ обозначает ближайшее целое к x.

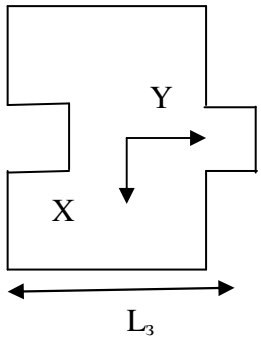


Рис. 5. Звено гусеницы

Расположение звеньев гусеницы вдоль выпуклой огибающей. В системе визуализации хранится только один объект - звено гусеницы. Для простоты будем считать, что его локальная система координат совпадает с глобальной, расположена на внутренней стороне звена, ось Y направлена вдоль, ось X – перпендикулярно направлению гусеничной ленты, а ось Z

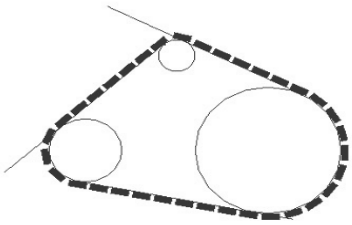


Рис. 6. Гусеница для выпуклой оболочки колес

направлена к внутренней стороне гусеничной ленты (см. рис. 5). Во время визуализации очередного кадра виртуальной сцены это звено помещается в нужное место гусеничной огибающей (путем формирования соответствующей модельной матрицы) и визуализируется. Будем считать, что при движении транспортного средства вперед у звеньев гусеницы, касающихся земли, ось Y должна быть направлена в сторону, противоположную движению транспортного средства. Рассмотрим некоторый блок огибающей гусеницы. Для расположения на нем i-го звена исходное звено надо перенести в некоторую точку P_i, расположенную на расстоянии S_i от начала блока и повернуть его на некоторый угол φ_i. Модельная матрица для i-го звена будет иметь вид:

$$M_i = \begin{pmatrix} Cos\phi_i & -Sin\phi_i & 0 & P_{i,x} \\ Sin\phi_i & Cos\phi_i & 0 & P_{i,y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (4)$$

На линейном блоке угол φ (один и тот же для всех звеньев) определяется из вектора направления этого блока: $\varphi = a\tan 2(Dir.y, Dir.x)$. Положение звена

вычисляется по формуле $P_i = P_0 + Dir \cdot S_i$, где P_0 – начало блока. Если S_0 – расстояние от начала для 0-го звена, то $S_i = S_0 + i \cdot L_з$. Если длина блока равна L и на блоке помещается k_i звеньев, то $L-(S_0 + k \cdot L_з)$ будет равно значению S_0 для первого звена следующего блока гусеничной огибающей (как вычислить значение S_0 для первого блока – будет рассмотрено ниже).

На дуговом блоке с центром в точке O и началом P_0 положение i-го звена вычисляется по формуле $P_i = O + Rotate(OP_0, \alpha_i)$, т.е. вектор OP_0 поворачивается на некоторый угол α_i и прибавляется к центру дуги. Угол α_i легко вычислить по известной длине дуги S_i для i –го звена и радиусу R дуги: $\alpha_i = S_i / R$. Угол φ_i в матрице (4) равен углу касательной к дуге в точке P_i. Так как вектор касательной получается из радиус-вектора $\overline{R}_i = OP_i$ поворотом на 90° против часовой стрелки, то его координаты равны $(-\overline{R}_{i,y}, \overline{R}_{i,x})$, а угол $\varphi_i = a\tan 2(\overline{R}_{i,x}, -\overline{R}_{i,y})$. Аналогично линейному блоку здесь также $S_i = S_0 + i \cdot L_з$, а $L-(S_0 + k \cdot L_з)$ будет равно значению S_0 для первого звена следующего блока гусеничной огибающей (если на дуговом блоке помещается ровно k звеньев).

Рассмотрим теперь вопрос о вычислении начального значения S_0 для первого линейного блока в каждом кадре (при этом первый блок в предыдущем кадре может уже не быть первым в следующем кадре) . Пусть в текущем j-м кадре это значение было S_0^j (в первом кадре это значение равно нулю) и пусть за промежуток времени между j-м и $j+1$-м кадрами ведущее колесо радиуса $R_в$ повернулось на угол ψ. За это время гусеница пройдет расстояние $U = R_в \cdot \psi$ и точка S_0^j перейдет в точку $S_0^j + U$. Первое звено на этом блоке будет располагаться в точке $S_0^{j+1} = \{(S_0^j + U) / L_з\}$, где фигурными скобками обозначена дробная часть числа. На рис. 6 показан пример гусеницы для выпуклой оболочки колес.

2. Моделирование гусениц с колесами обратного хода.

Колеса обратного хода – это колеса, которые при движении гусеничного транспортного средства касаются внешней стороны гусеницы (см. рис. 1). Поскольку колеса обратного хода чаще всего находятся внутри выпуклой огибающей, то в этом случае необходимо заранее задавать порядок обхода колес гусеницей. Поэтому в дальнейшем мы будем считать, что в массиве колес гусеницы колеса размещены в порядке их обхода. Далее, особенность моделирования состоит в том, что между колесом прямого и обратного ходов необходимо строить внутреннюю касательную (AB на рис. 7). Пусть первое колесо имеет прямой, а второе колесо - обратный ход (рис. 7). Аналогично случаю колес прямого хода имеем:

$$\begin{aligned} &\|AB\| = \|O_1E\| = \sqrt{\|O_1O_2\|^2 - (R_2 + R_1)^2}, \\ &\alpha_1 = a\tan 2\left(R_2 + R_1, \|O_1E\|\right), \\ &Dir = Rotate(CDir, \alpha_1), \\ &A = O_1 + Rotate(CDir \cdot R_1, -\pi/2 + \alpha_1), \\ &B = A + Dir \cdot \|AB\|. \end{aligned} \qquad (5)$$

Эти формулы несколько отличаются от формул (1)

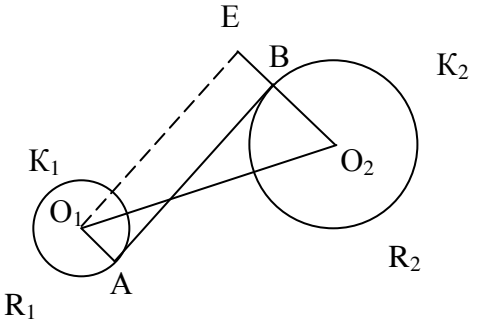


Рис. 7. Внутренняя касательная к колесам

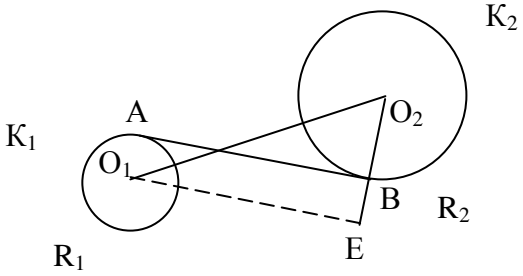


Рис. 8. Касательная к колесам обратного хода

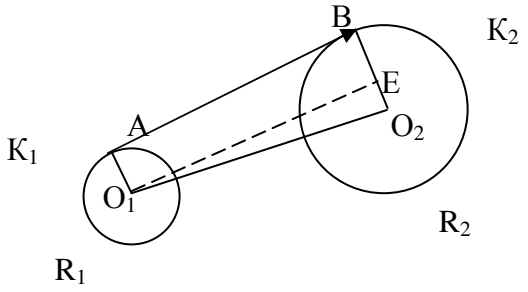
и (2). Однако, если считать, что колесо обратного хода имеет отрицательный радиус и подставить это значение в (1) и (2), то формулы совпадут. Действительно, заменяя в (1) и (2) R_2 на $(-R_2)$ и α_1 на $(-\alpha)$, получим:

$$\begin{aligned} &\|AB\| = \|O_1E\| = \sqrt{\|O_1O_2\|^2 - (-R_2 - R_1)^2}, \\ &\alpha = a\tan 2\left(-R_2 - R_1, \|O_1E\|\right), \\ &\alpha_1 = a\tan 2\left(R_2 + R_1, \|O_1E\|\right), \\ &Dir = Rotate(CDir, \alpha_1), \\ &A = O_1 + Rotate(CDir \cdot R_1, -\pi/2 + \alpha_1), \\ &B = A + Dir \cdot \|AB\|. \end{aligned}$$

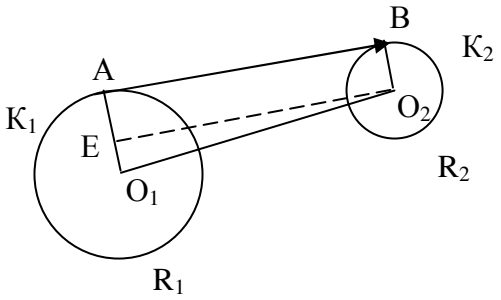
Эти формулы совпадают с (5).

В случае, если первое колесо имеет обратный ход, а второе – прямой (см. рис. 8) мы получим следующие формулы:

$$\|AB\| = \|O_1E\| = \sqrt{\|O_1O_2\|^2 - (R_2 + R_1)^2},$$
$$\alpha_1 = a\tan 2\left(R_2 + R_1, \|O_1E\|\right),$$
$$Dir = Rotate(CDir, -\alpha_1), \qquad (6)$$



а)



б)

Рис. 9. Колеса обратного хода

$$A = O_1 + Rotate(CDir \cdot R_1, \pi/2 + \alpha_1),$$
$$B = A + Dir \cdot \|AB\|.$$

При замене в (1) и (2) R_1 на $(-R_1)$ получаем формулы

$$\|AB\| = \|O_1E\| = \sqrt{\|O_1O_2\|^2 - (R_2 + R_1)^2},$$
$$\alpha = a\tan 2\left(R_2 + R_1, \|O_1E\|\right),$$
$$Dir = Rotate(CDir, \alpha),$$
$$A = O_1 + Rotate(CDir \cdot (-R_1), -\pi/2 - \alpha) =$$
$$= O_1 + Rotate(CDir \cdot R_1, \pi + (-\pi/2 - \alpha)) =$$
$$= O_1 + Rotate(CDir \cdot R_1, \pi/2 - \alpha),$$
$$B = A + Dir \cdot \|AB\|.$$

полностью совпадающие с (6).

Рассмотрим, наконец, случай, когда оба колеса имеют обратный ход (рис. 9 а и б). В обоих случаях имеем:

$$\|AB\| = \|O_1E\| = \sqrt{\|O_1O_2\|^2 - (R_2 - R_1)^2},$$
$$\alpha_1 = a\tan 2\left(R_2 - R_1, \|O_1E\|\right),$$
$$Dir = Rotate(CDir, \alpha_1), \qquad (7)$$
$$A = O_1 + Rotate(CDir \cdot R_1, \pi/2 + \alpha_1),$$
$$B = A + Dir \cdot \|AB\|.$$

Заменяя в (1) и (2) R_1 на $(-R_1)$, а R_2 на $(-R_2)$, получим:

$$\|AB\| = \|O_1E\| = \sqrt{\|O_1O_2\|^2 - (-R_2 + R_1)^2},$$
$$\alpha = a\tan 2\left(-R_2 + R_1, \|O_1E\|\right), \alpha_1 = -\alpha,$$
$$Dir = Rotate(CDir, \alpha_1),$$
$$A = O_1 + Rotate(CDir \cdot (-R_1), -\pi/2 + \alpha_1) =$$
$$= O_1 + Rotate(CDir \cdot R_1, \pi + (-\pi/2 + \alpha_1)) =$$
$$= O_1 + Rotate(CDir \cdot R_1, \pi/2 + \alpha_1),$$
$$B = A + Dir \cdot \|AB\|.$$

Эти формулы совпадают с (7).

Таким образом, по формулам (1) и (2) можно вычислять касательные к колесам прямого и обратного ходов независимо от соотношения их радиусов. Следует лишь для колес обратного хода указывать отрицательные радиусы. Однако есть еще одна тонкость. Так как колесо обратного хода касается внешней стороны гусеницы, а мы строим огибающую для внутренней стороны, то необходимо учесть толщину звена гусеницы. То есть для каждого колеса обратного хода радиуса $(-R)$ для вышеприведенных расчетов брать его радиус равным $(-R-d)$, где d – толщина звена гусеницы.

Для вычисления длины дуговой огибающей колеса обратного хода можно также использовать формулы (3). Для колеса обратного хода угол α точки начала дуги будет больше, чем угол β конца дуги, поэтому $\gamma = \beta - \alpha$ будет отрицательным. Однако, при умножении γ на отрицательный радиус R колеса получится положительная длина дуги $S = R \cdot \gamma$. Таким образом, при вычислении по формулам (3) угол γ надо заменить на $2\pi + \gamma$ только если $\gamma < 0$ и $R > 0$.

Расположение звеньев гусеницы на линейном блоке в случае колеса обратного хода осуществляется по тому же алгоритму, что и для колеса прямого хода. Для дугового блока звено необходимо повернуть на угол $\alpha_i = S_i / R_i$, где S_i – длина дуги для i – го звена. Так как R отрицателен, то α_i тоже будет отрицательно, поэтому поворот на угол α_i против часовой стрелки будет соответствовать повороту по часовой стрелке на угол $(-\alpha_i)$, что и требуется в данной ситуации. Кроме того, здесь вектор касательной получается из радиуса-вектора $\overline{R}_i = OP_i$ поворотом на 90° по часовой стрелке, т.е. его координаты равны $(\overline{R}_{i,y}, -\overline{R}_{i,x})$, а угол $\varphi_i = a\tan 2(-\overline{R}_{i,x}, \overline{R}_{i,y})$. Так как гусеница обходит колесо обратного хода так, что колесо находится справа от нее, то поворачивать надо на угол φ_i по часовой стрелке, что соответствует повороту на угол $(-\varphi_i)$ по часовой стрелке. Последний угол равен

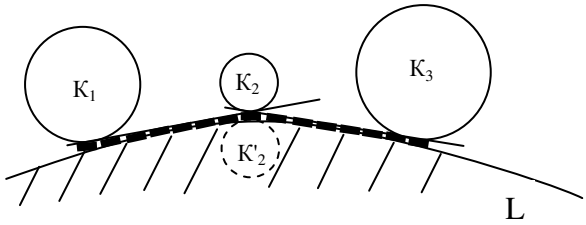


Рис. 10. Фиктивное колесо

$(-\varphi_i) = -a\tan 2(-\bar{R}_{i,x}, \bar{R}_{i,y}) = a\tan 2(\bar{R}_{i,x}, -\bar{R}_{i,y})$,

что совпадает с аналогичной формулой для колеса прямого хода.

3. Моделирование гусеницы в 3D пространстве.

До сих пор мы рассматривали моделирование гусеницы на плоскости, теперь рассмотрим случай трехмерного пространства. Теоретически при движении транспортного средства центры всех колес должны лежать в одной плоскости. Однако при расчетах за счет погрешностей(и) вычислений центры не будут лежать в одной плоскости. Для решения этой проблемы выберем какое-либо колесо и возьмем плоскость П, проходящую через его центр и перпендикулярную его оси. Предположим, что система координат E_K этого колеса имеет начало О в центре колеса, а ось X направлена вдоль оси вращения. Тогда точка О и оси Y и Z задают систему координат $E_П$ в П. Обозначим через Q матрицу перехода из мировой системы координат в систему координат E_K. Матрица Q невырожденная, т.е. существует обратная матрица Q^{-1}. В плоскости YZ системы координат $E_П$ мы можем вычислить все точки расположения и ориентации звеньев гусеницы. Модельная матрица M_i (см. равенство (4)) переводит исходное звено гусеницы из его локальной системы координат в систему координат $E_П$ на i - е место в гусенице. Тогда матрица $W_i = Q^{-1}M_i$ размещает i-е звено в нужном месте в мировой системе координат.

Как отмечалось выше, при движении виртуального транспортного средства сначала подсистема моделирования рассчитывает положения всех его колес, а затем подсистема визуализации строит и визуализирует гусеницу. При наезде колесом на рельефную возвышенность оно будет уходить внутрь выпуклой оболочки, охватывающей колеса. Гусеница в этом случае должна огибать эту возвышенность (см. рис. 10). Для реализации этого необходимо проверять, что колесо, касающееся земли вошло внутрь выпуклой оболочки. На рис. 10 колесо K_2 лежит слева от касательной между колесами K_1 и K_3, т.е. строго внутри выпуклой оболочки. В этом случае можно создать фиктивное колесо (на рисунке это K_2'), симметричное внутреннему колесу относительно горизонтальной оси. Если внутреннее колесо было прямого хода, то соответствующее ему фиктивное будет обратного хода (и наоборот). Затем следует построить гусеницу с учетом всех фиктивных колес, как описано выше. На рис. 11 показаны примеры моделей гусеничных роботов МФ-4 и РС-2.

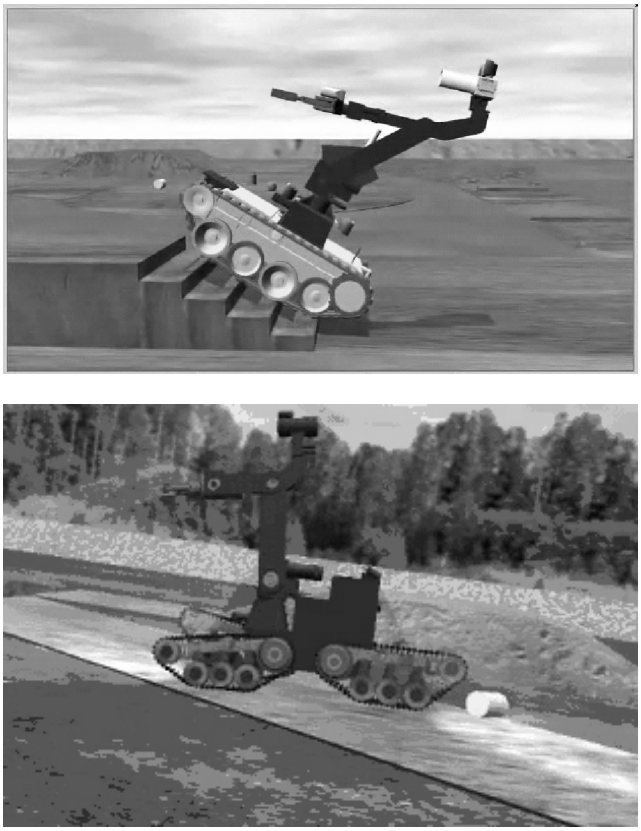


Рис. 11. Роботы МФ-4 и РС-2

Работа выполняется при поддержке РФФИ, грант №10-07-00404.

Modeling of the truck vehicles in visualization systems

Mikhaylyuk M.V.

Abstract. Caterpillar modeling during the visualization for caterpillar vehicle is considered. Wheels positions are computed in dynamic subsystem and the caterpillar is modeling as envelope around these wheels. We consider forward runs caterpillars as well as backward runs caterpillars. The suggested approach provides the modeling of movement of the caterpillar vehicles in real time.

Планирование вычислений в многопроцессорных системах с дополнительным ресурсом

М.Г. Фуругян

кандидат физико-математических наук, доцент

Рассматривается задача построения допустимых расписаний в системах реального времени при заданных директивных сроках выполнения работ. Дополнительно учитываются ограничения на память процессоров. Разработан метод, основанный на сведении исходной задачи к поиску многопродуктового потока в сети специального вида. Кроме того, рассматривается задача составления допустимого расписания с прерываниями в многопроцессорной системе в случае, когда заданы директивные интервалы, а длительности выполнения работ линейно зависят от количества выделенного им дополнительного ресурса.

1. Построение расписаний при ограничениях на память процессоров

1.1. Постановка задачи

Рассматривается вычислительная система, состоящая из m процессоров, работа которых представляется как набор последовательных тактов. Такты всех процессоров синхронизованы во времени, т.е. их начала и концы для всех процессоров совпадают. Процессор j (j=1,..., m) характеризуется производительностью s_j и объемом памяти V_j . Имеется n независимых друг от друга работ, каждая работа i (i=1,...,n) характеризуется объемом p_i, директивным интервалом исполнения [r_i, d_i] и объемом памяти v_i, необходимой для выполнения работы.

Без ограничения общности можно считать, что все r_i, d_i – натуральные числа (т.е. начала и концы всех директивных интервалов задают номера соответствующих тактов работы процессоров) и $\min\limits_i r_i = 0$, $\max\limits_i d_i = T$ (т.е. в задаче рассматривается T тактов работы процессоров). Таким образом, s_j – это объем работы, выполняемой процессором j за один такт, т.е. для того, чтобы полностью выполнить работу i, процессору j необходимо затратить $\lceil p_i/s_j \rceil$ тактов. В фиксированный момент времени каждая работа может выполняться не более чем одним процессором, и каждый процессор может выполнять не более одной работы. Предполагается, что прерывание выполнения работы на процессоре j_1 с целью ее переключения на одном из последующих тактов на процессор j_2 или для возобновления на этом же процессоре ($j_2 = j_1$) требует дополнительной работы процессоров j_1 и j_2 в суммарном объеме $\tau_{j_1 j_2}$.

Связи между процессорами могут меняться во времени, что определяется массивом I размером $m \times T \times m \times T$. При этом $I(j_1,k_1,j_2,k_2)=1$, если при выполнении работы на процессоре j_1 в конце такта k_1 ее можно прервать и возобновить на процессоре j_2 в начале такта k_2, и $I(j_1,k_1,j_2,k_2)=0$, если такое переключение невозможно $(1 \le j_1, j_2 \le m; 1 \le k_1 < k_2 \le T)$.

Необходимо построить такое расписание, при котором все работы выполняются на процессорах в своих директивных интервалах, или показать, что такого расписания не существует. В отличие от [1, 4, 5] дополнительно учитываются связи между процессорами и ограничения на память процессоров.

1.2. Построение сети

Для решения этой задачи построим многопродуктовую потоковую сеть $N=(V, E)$, где V – вершины, E – дуги. Множество V состоит из следующих вершин: u_1, $u_2,\ldots,u_n$ – источники; w_1, $w_2,\ldots,w_n$ – стоки; x_j^k, y_j^k ($j = 1,...,m; k = 1,...,T$) - внутренние вершины (рис. 1). Множество E состоит из следующих дуг: (u_i, x_j^k) (i = 1, …, n; j = 1,…, m; $k \in [r_i, d_i]$); (x_j^k, y_j^k) ($j = 1,...,m; k = 1,...,T$) ; $(y_{j_1}^{k_1}, y_{j_2}^{k_2})$ ($j_1, j_2 = 1,...,m$) и k_1, k_2 таких, что $k_1 < k_2$, $I(j_1,k_1,j_2,k_2)=1$; (y_j^k, w_i) (i = 1, …, n; j = 1,…, m; $k \in [r_i, d_i]$). Для сети, изображенной на рис. 1, наличие дуг (u_i, x_j^k) и (u_i, x_j^{k+1}) означает, что работу i можно начать выполнять на процессоре j как на k-м, так и на (k+1)-м такте, а наличие дуг (y_j^k, w_i) и (y_j^{k+1}, w_i) означает, что завершить работу i можно как в конце k-го, так и в конце (k+1)-го такта. Наличие дуги $\left(y_j^k, x_r^{k+1}\right)$ означает, что выполнение работы на процессоре j в конце k-го такта можно прервать и возобновить на процессоре r в начале (k+1)-го такта. Несложно заметить, что $|V| = 2n+2mT$, $|E| \le mT(m(T-1)/2+2n+1)$. Каждая дуга сети N имеет два параметра: пропускную способность и стоимость единицы потока по дуге. Пропускные способности всех дуг полагаем равными 1. Стоимость прохождения единицы потока

 Статья поступила в редакцию 09.04.2012

по дуге (x_j^k, y_j^k) полагаем равной s_j, по дуге $(y_{j_1}^{k_1}, x_{j_2}^{k_2})$ – равной $-\tau_{j_1 j_2}$, а по дугам (u_i, x_j^k) и (y_j^k, w_i) – равной 0.

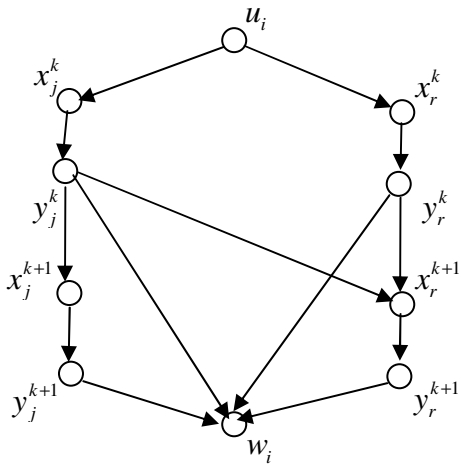


Рис. 1. Фрагмент сети *N*

Рассмотрим в этой сети многопродуктовый поток, состоящий из *n* типов потоков, причем поток *i*-го продукта исходит из источника u_i и входит в сток w_i (*i=1*,..., *n*). Эти потоки описывают выполнение работ многопроцессорной системой. Будем интерпретировать поток *i*-го продукта по дугам сети *N* следующим образом: по дуге (u_i, x_j^k) - начало выполнения работы i (на такте *k*); по дуге $\left(x_j^k, y_j^k\right)$ – выполнение работы *i* процессором *j* на такте *k*; по дуге $(y_{j_1}^{k_1}, x_{j_2}^{k_2})$ – прерывание выполнения работы *i* процессором j_1 на такте k_1 и ее переключение на процессор j_2 на такте k_2; по дуге (y_j^k, w_i) – конец выполнения работы *i* (на такте *k*).

При прохождении потока i-го продукта по дуге $\left(x_j^k, y_j^k\right)$ к его стоимости прибавляется величина, равная s_j (объем выполненной работы на *j*-м процессоре за один такт). При прохождении потока по дуге $(y_{j_1}^{k_1}, x_{j_2}^{k_2})$, которая соответствует прерыванию или переключению работы, из его стоимости вычитается величина $\tau_{j_1 j_2}$. При прохождении потока по дугам (u_i, x_j^k) и (y_j^k, w_i) его стоимость остается без изменений. Будем обозначать поток *i*-го продукта по дуге (*a,b*) как $f^i(a,b)$. Заметим, что в нашей модели $f^i(a,b) \in \{0,1\}$ при всех *a*, *b* и *i*. Например, если для сети, изображенной на рис. 1, $f^i\left(x_{j_1}^k, y_{j_1}^k\right) = f^i\left(y_{j_1}^k, x_{j_2}^{k+1}\right) = f^i\left(x_{j_2}^{k+1}, y_{j_2}^{k+1}\right) = 1$, то на *k*-м и (*k+1*)-м тактах суммарный объем работы процессоров j_1 и j_2 по выполнению работы *i* составляет $s_{j_1} + s_{j_2} - \tau_{j_1 j_2}$.

1.3. Необходимые и достаточные условия существования допустимого расписания

Л е м м а 1. Допустимое расписание существует тогда и только тогда, когда в сети *N* существует *n*-продуктовый поток, обладающий следующими свойствами:

$$\sum_{j,k} f^i\left(x_j^k, y_j^k\right) s_j \; - \sum_{j_1, j_2, k_1, k_2 : k_1 < k_2} f^i\left(y_{j_1}^{k_1}, x_{j_2}^{k_2}\right) \tau_{j_1 j_2} \ge p_i \qquad (1)$$

при всех $i = 1,...,n$, т.е. суммарная стоимость потока *i*-го продукта составляет не менее p_i;

$$\sum_{i, r_1, r_2 : r_1 < k < r_2} f^i\left(y_j^{r_1}, x_j^{r_2}\right) v_i + \sum_{i=1}^{n} f^i\left(x_j^k, y_j^k\right) v_i \le V_j \qquad (2)$$

при всех *j = 1, …, m; k = 1, …, T*, т.е. на каждом такте выполнено ограничение по объему памяти каждого процессора.

Д о к а з а т е л ь с т в о. 1) Предположим, что допустимое расписание существует. Будем задавать его матрицей *A* размера $m \times T$, каждый элемент a_{jk} которой может принимать целые значения в диапазоне от 0 до *n*, причем если $a_{jk} = i$ $(1 \le i \le n)$, то это означает, что на *k*-м такте работа *i* выполняется процессором *j*, а если $a_{jk} = 0$, то на *k*-м такте процессор *j* простаивает. Отметим, что в каждом столбце матрицы *A* значение только одного элемента может быть равным *i*, поскольку каждая работа может одновременно выполняться только на одном процессоре. Для каждой работы *i* выполним следующие действия. Пусть $k_0 < k_1 < ... < k_l$ – номера всех столбцов матрицы *A*, в каждом из которых есть элемент, равный *i*, и пусть $a_{j_0 k_0} = a_{j_1 k_1} = ... = a_{j_l k_l} = i$. Тогда полагаем значения потоков $f^i\left(u_i, x_{j_0}^{k_0}\right)$, $f^i\left(x_{j_0}^{k_0}, y_{j_0}^{k_0}\right)$, $f^i\left(y_{j_0}^{k_0}, x_{j_1}^{k_1}\right)$, $f^i\left(x_{j_1}^{k_1}, y_{j_1}^{k_1}\right)$, …, $f^i\left(y_{j_{l-1}}^{k_{l-1}}, x_{j_l}^{k_l}\right)$, $f^i\left(x_{j_l}^{k_l}, y_{j_l}^{k_l}\right)$, $f^i\left(y_{j_l}^{k_l}, w_i\right)$ равными 1. Потоки *i*-го продукта по всем остальным дугам полагаем равными 0.

Поскольку согласно допустимому расписанию каждая работа должна быть выполнена полностью, то неравенство (1) выполнено. Выполнение неравенства (2) следует из того, что допустимое расписание должно также удовлетворять ограничениям по памяти процессоров. Действительно, первое слагаемое в (2) задает поток по всем дугам, которые имеют своим началом вершины, относящиеся к процессору *j* на всех тактах, предшествующих текущему такту *k,* и окончание в вершинах, относящихся к этому же процессору *j* и какому-либо последующему такту. Ненулевой поток ка-

кого-либо *i*-го продукта по одному из подобных ребер означает, что задача *i* выполнялась на этом процессоре, затем была отложена и позднее возобновлена на нем же без переключения на другие процессоры, т.е. во время рассматриваемого такта она хотя и не выполняется, но находится в памяти *j*-го процессора и занимает объем v_i. Второе же слагаемое соответствует собственно выполнению некоторой работы процессором j на k-м такте. Поэтому сумма указанных двух величин не превышает V_j.

2) Предположим теперь, что в сети *N* существует многопродуктовый поток, удовлетворяющий условиям (1) и (2). Построим по этому потоку допустимое расписание путем заполнения матрицы *A*. Заметим, что каждому элементу a_{jk} матрицы A соответствует ребро $\left(x_j^k, y_j^k\right)$ сети *N*. Поэтому матрица *A* может быть заполнена по следующему простому правилу: $a_{jk} = i$, если $f^i\left(x_j^k, y_j^k\right) = 1$, и $a_{jk} = 0$, если $f^i\left(x_j^k, y_j^k\right) = 0$. Действительно, именно положительный поток *i*-го продукта по ребру $\left(x_j^k, y_j^k\right)$ соответствует выполнению работы *i* процессором *j* на *k*-м такте, и никаких дополнительных данных для задания расписания не требуется. В силу (1) все работы полностью выполняются, а структура сети *N* гарантирует, что каждая работа выполняется строго в своем директивном интервале. В силу условия (2) объем памяти каждого процессора на каждом такте при выполнении работ не будет превышен. Следовательно, построенное расписание является допустимым. Лемма доказана.

Рассмотрим пример, в котором $T = 5$; $n = 3$; $m = 2$; $s_1 = s_2 = 1$; $p_1 = p_2 = p_3 = 3$; $r_1 = r_2 = r_3 = 0$; $d_1 = d_2 = d_3 = 5$; $\tau_{12} = \tau_{21} = 1$; $I(j_1, k_1, j_2, k_2) = 1$ при всех $1 \le j_1, j_2 \le 2$; $1 \le k_1 < k_2 \le 5$; $v_1 = v_2 = v_3 = 1$; $V_1 = V_2 = 1$. На рис. 2 изображен фрагмент сети *N* для данного примера, включающий только те дуги, поток по которым равен 1. Поток первого продукта проходит по пути $u_1 - x_1^1 - y_1^1 - x_1^2 - y_1^2 - x_1^3 - y_1^3 - w_1$, и его стоимость равна 3, т.е. первая работа выполняется первым процессором на первых трех тактах и получает три единицы процессорного времени. Поток второго продукта проходит по пути $u_2 - x_2^1 - y_2^1 - x_2^2 - y_2^2 - x_1^4 - y_1^4 - x_1^5 - y_1^5 - w_2$, и его стоимость равна $2 - 1 + 2 = 3$, т.е. вторая работа выполняется первые два такта вторым процессором, затем переключается на первый процессор, на котором выполняется четвертый и пятый такты.

Суммарный объем работы рассматриваемой двухпроцессорной системы по выполнению второй работы равен 4 единицам процессорного времени, однако одна из этих единиц расходуется на переключение и на выполнение работы остаются необходимые 3 единицы.

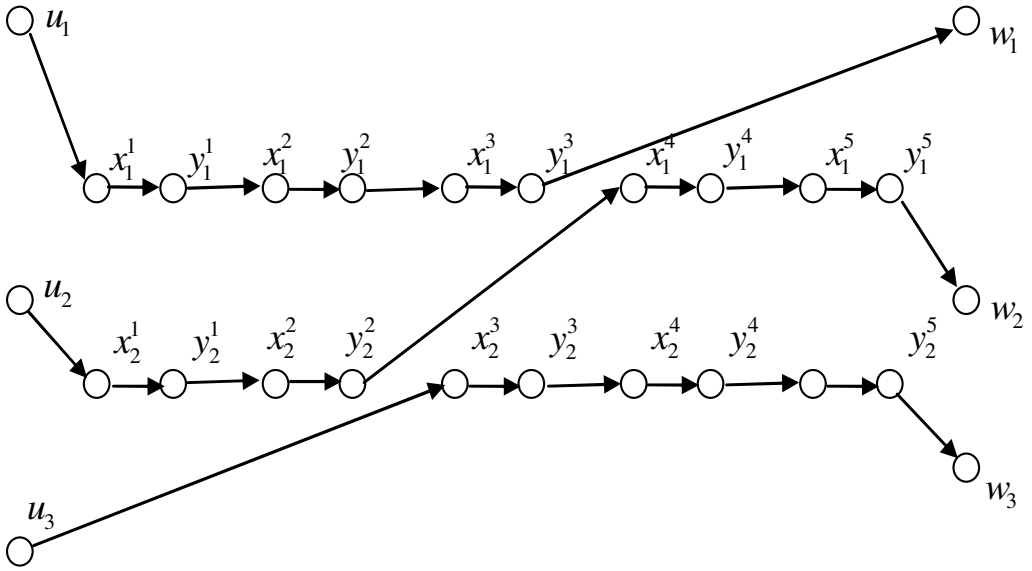


Рис. 2. Потоки в сети *N*, отображающие допустимое расписание для рассматриваемого примера

Поток третьего продукта проходит по пути $u_3 - x_2^3 - y_2^3 - x_2^4 - y_2^4 - x_2^5 - y_2^5 - w_3$, и его стоимость равна 3, т.е. третья работа выполняется вторым процессором на третьем, четвертом и пятом тактах и также получает три единицы процессорного времени. Таким образом, условие 1 леммы выполнено. Нетрудно заметить, что первое слагаемое в левой части условия 2 леммы всегда равно 0, а второе равно 1, поэтому условие 2 также выполняется. Таким образом, допустимое расписание существует и определяется матрицей

$$A = \begin{pmatrix} 11122 \\ 22333 \end{pmatrix}.$$

Итак, показано, что предлагаемое сведение задачи о поиске многопроцессорного расписания к задаче о поиске многопродуктового потока в сети *N* корректно, и однозначно интерпретирована одна задача в терминах другой. Поскольку поток по каждой дуге равен 0 или 1, то для нахождения искомого потока можно воспользоваться методами булевого линейного программирования [3, 7].

2. Алгоритмы распределения ресурсов в многопроцессорных системах с нефиксированными параметрами

2.1. Постановка задачи

Рассматривается вычислительная система, состоящая из m идентичных процессоров, и множество работ $N=\{1,2,...,n\}$. Предполагается, что в каждый момент времени каждый процессор может выполнять не более одной работы, а каждая работа выполняется не более чем одним процессором. При выполнении работ допускаются прерывания и переключения с одного процессора на другой. Предполагается, что прерывания и переключения не сопряжены с временными затратами. Для работы $i \in N$ установлен директивный интервал $(b_i, f_i]$ (т.е. работа i может выполняться только в этом интервале). Помимо процессоров в системе имеется дополнительный ресурс невозобновляемого типа. Суммарное количество этого ресурса составляет R. Если работе i выделено r_i единиц дополнительного ресурса, то ее длительность составляет $t_i = d_i - a_i r_i$, где

$$r_i \in [0, \bar{r}_i], i = 1,...,n, \quad (3)$$

$$\sum_{i \in N} r_i \le R, \quad (4)$$

$a_i, d_i, \bar{r}_i$ – заданные величины, $a_i > 0$, $d_i > 0$, $0 \le \bar{r}_i < d_i / a_i$. Таким образом, $t_i \in [d_i - a_i \bar{r}_i, d_i]$ причем $d_i - a_i \bar{r}_i > 0$. Требуется найти такое распределение ресурсов $(r_1^0, r_2^0, ..., r_n^0)$, при котором существует допустимое расписание (каждая работа выполняется в своем директивном интервале). При этом распределение ресурсов $(r_1^0, r_2^0, ..., r_n^0)$ должно удовлетворять ограничениям (3), (4). В случае, когда указанного распределения ресурсов не существует, требуется определить директивные интервалы $(b_i, f_i']$, при которых указанное распределение ресурсов существует и величина $\max_{i \in N}(f_i' - f_i)$ принимает минимальное значение.

Подобная задача для случая, когда дополнительный ресурс отсутствует, рассматривалась в [4] и была сведена к задаче о максимальном потоке в сети специального вида. Задача с произвольными процессорами и произвольными директивными интервалами рассматривалась в [5], а задача с произвольными процессорами и одинаковыми директивными интервалами – в [6]. В обеих этих работах дополнительный ресурс не рассматривался. Задача минимизации времени выполнения сетевого комплекса работ, когда длительности выполнения работ являются функциями от вектора распределения ресурсов, а число процессоров не ограничено, рассматривалась в [2] и была сведена к задаче нелинейного программирования.

2.2. Сведение исходной задачи к задаче о потоке минимальной стоимости

В общем случае, когда директивные интервалы произвольные, по аналогии с тем, как это сделано в [4], построим потоковую сеть $G = (V, A)$ (рис. 3) с источником s и стоком t (V – множество узлов сети, A – множество дуг). Пусть $y_0 < y_1 < ... < y_p$ $(p < 2n)$ – все различные по величине значения $b_i, f_i (i \in N)$. Определим $V = \{s, t, I_1, I_2, ..., I_p, w_1, ..., w_n\}$, где узел I_j соответствует интервалу $(y_{j-1}, y_j] (j = 1,2,...,p)$, а узел w_i – работе $i \in N$. Множество дуг A сети G определим следующим образом:
$(s, I_j)(j = 1,2,...,p); (w_i, t)(i \in N); (I_j, w_i)$ в случае, если $(y_{j-1}, y_j] \subseteq (b_i, f_i]$ (отметим, что для любых j ($1 \le j \le p$) и $i \in N$ интервал $(y_{j-1}, y_j]$ либо не пересекается с интервалом $(b_i, f_i]$, либо целиком лежит в нем); определим также возвратную дугу (t, s). Пусть $\Delta_j = y_j - y_{j-1}$ – длина интервала I_j. Для каждой дуги определим три параметра: L, U, C, где L – нижняя граница потока по дуге, U – верхняя граница потока

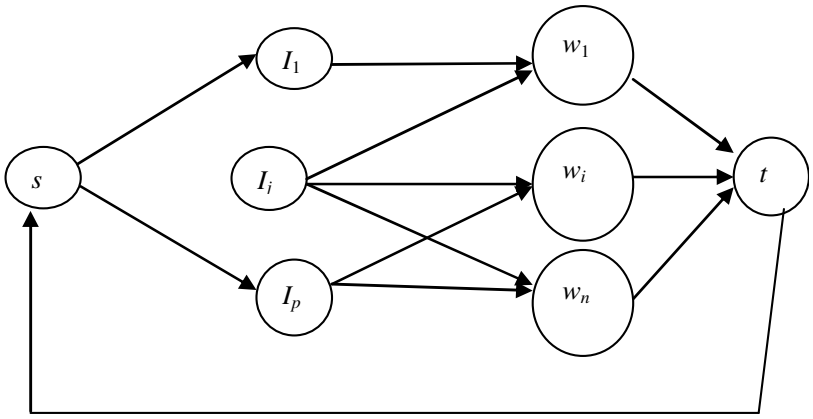


Рис.3. Потоковая сеть G

Таблица 1. Параметры дуг сети G

Дуга	***L***	***U***	***C***
(s, I_j)	0	$m\Delta_j$	0
(I_j, w_i)	0	Δ_j	0
(w_i, t)	$d_i - a_i \bar{r}_i$	d_i	$-1/a_i$
(t, s)	0	$\sum_{i \in N} d_i$	0

по дуге, C – стоимость единицы потока по дуге. Параметры дуг сети G указаны в табл. 1.

Л е м м а 2. Для существования допустимого расписания необходимо и достаточно существование в сети G циркуляции g, стоимость которой

$$c(g) \le R - \sum_{i \in N} d_i / a_i . \qquad (5)$$

Д о к а з а т е л ь с т в о. 1) Пусть в рассматриваемой задаче существует допустимое расписание и пусть r_i – величина дополнительного ресурса, выделенного работе i, а $t_i = d_i - a_i r_i$ – длительность работы i. Тогда $r_i = (d - t_i) / a_i$, $i \in N$, а суммарная величина дополнительного ресурса, необходимая для выполнения всего комплекса работ N, равна $\sum_{i \in N} (d_i - t_i)/a_i \le R$, откуда следует неравенство

$$\sum_{i \in N} (-t_i / a_i) \le R - \sum_{i \in N} d_i / a_i . \qquad (6)$$

По аналогии с тем, как это сделано в [4], можно показать, что в сети G существует такая циркуляция g (т.е. поток, для которого условие сохранения выполнено в каждом узле сети G, включая источник s и сток t), что

$$g_{it} = t_i \qquad (7)$$

(g_{it} – величина потока по дуге (w_i, t)). Стоимость $c(g)$ этой циркуляции равна $c(g) = \sum_{i \in N} (-g_{it} / a_i)$. Из (6), (7) следует, что $c(g) = \sum_{i \in N} (-t_i / a_i) \le R - \sum_{i \in N} d_i / a_i$, т.е. выполнено неравенство (5).

2) Пусть в сети G существует циркуляция g, для которой выполнено неравенство (5). Тогда по аналогии с тем, как это сделано в [4], можно показать, что существует допустимое расписание, в котором длительность t_i каждой работы $i \in N$ удовлетворяет равенству (7). Параметры дуг (w_i, t) определены таким образом, что $t_i \in [d_i - a_i \bar{r}_i, d_i]$, а стоимость циркуляции g равна $c(g) = \sum_{i \in N} (-g_{it} / a_i)$. Из (5) следует, что $\sum_{i \in N} (-t_i / a_i) \le R - \sum_{i \in N} d_i / a_i$ и $\sum_{i \in N} (d_i - t_i)/a_i \le R$. Поскольку $(d_i - t_i) / a_i$ – это величина дополнительного ресурса, выделяемого работе i, то из последнего неравенства следует, что суммарное количество дополнительного ресурса, требуемое для реализации допустимого расписания, не превосходит R. Лемма доказана.

Опишем *алгоритм* решения поставленной задачи, основанный на доказанной теореме.

Шаг 1. Построить потоковую сеть G.

Шаг 2. Найти циркуляцию g минимальной стоимости в сети G (для этого можно применить, например, алгоритм дефекта). Пусть $c(g)$ – ее стоимость, а g_{it} – величина потока по дуге (w_i, t).

Шаг 3. Если $c(g) > R - \sum_{i \in N} d_i / a_i$, то допустимого расписания не существует. Если $c(g) \le R - \sum_{i \in N} d_i / a_i$, то допустимое расписание существует. При этом величина g_{it} потока по дуге (I_j, w_i) равна величине процессорного времени, выделяемого работе i в интервале I_j, $i \in N$; $j = 1, 2, \ldots, p$). Для построения расписания в интервале I_j следует применить алгоритм упаковки, а для построения искомого расписания следует совместить расписания, построенные для всех интервалов I_j, $j = 1, 2, \ldots, p$.

Определим вычислительную сложность отдельных этапов и всего алгоритма. Сложность построения потоковой сети G составляет $O(n^2)$, нахождения циркуляции минимальной стоимости с помощью алгоритма дефекта $O(n^4 \sum_{i \in N} d_i)$. Сложность построения расписания в интервале I_j, $1 \le j \le 2n - 1$, равна $O(n)$. Таким образом, вычислительная сложность алгоритма есть $O(n^4 \sum_{i \in N} d_i)$.

Предположим, что $c(g) > R - \sum_{i \in N} d_i / a_i$. Поскольку общее количество ресурса фиксировано, то добиться выполнения неравенства (5) за счет увеличения величины R нельзя.

В дальнейшем будем предполагать, что директивные интервалы имеют вид $(0, f_i]$, $i \in N$. Без ограничения общности можно считать, что $f_i < f_{i+1}$ при всех $i \in N$, т.е. работы упорядочены по возрастанию правых границ их директивных интервалов. Построим для этого случая потоковую сеть и будем обозначать ее G^* (рис. 4). Заметим, что $I_1 = (0, f_1], I_2 = (f_1, f_2], \ldots, I_n = (f_{n-1}, f_n]$.

Таким образом, особенностью данной сети является то, что из узла I_j исходят дуги, ведущие в узлы $\{w_j, w_{j+1}, ..., w_n\}$. Пусть g – циркуляция минимальной стоимости в сети G^*, а $c(g)$ – ее стоимость. Обозначим потоки по дугам $\langle s, I_j \rangle$, $\langle I_j, w_i \rangle$ и $\langle w_i, t \rangle$ через $g_{\langle s, I_j \rangle}$, $g_{\langle I_j, w_i \rangle}$ и $g_{\langle w_i, t \rangle}$ соответственно.

Увеличим правые границы директивных интервалов всех работ на величину $\varepsilon > 0$: $f_i' = f_i + \varepsilon$. Из постановки задачи следует, что правые границы всех директивных интервалов можно увеличивать на одну и ту же величину. Получим, что $\Delta_1^* = \Delta_1 + \varepsilon$, $\Delta_j^* = \Delta_j$, $j \in N \setminus \{1\}$. Это означает, что после такого увеличения всех директивных интервалов изменились пропускные способности дуг $\langle s, I_1 \rangle$ и $\langle I_1, w_i \rangle$, $i \in N$. Увеличение верхней границы пропускной способности дуг $\langle I_1, w_i \rangle$, $i \in N$ означает возможность увеличения потока по дугам $\langle w_i, t \rangle$, а поскольку стоимость потока по дугам $\langle w_i, t \rangle$ – величина отрицательная, то увеличение потока по ним снижает его стоимость. Пусть $\delta = c(g) - \left[R - \sum_{i \in N} \frac{d_i}{a_i} \right]$. В дальнейшем будем рассматривать сеть G^*. Обозначим через $g^*_{\langle I_1, w_i \rangle}$ модифицированный поток по дугам $\langle I_1, w_i \rangle$. По дугам вида $\langle I_j, w_i \rangle$ ($j \geq 2$) поток изменяться не будет. Имеем

$$\begin{aligned} g^*_{\langle I_1, w_i \rangle} &= g_{\langle I_1, w_i \rangle} + x_i, \\ g^*_{\langle I_j, w_i \rangle} &= g_{\langle I_j, w_i \rangle}, \; j \in N \setminus \{1\}. \end{aligned} \tag{8}$$

Из неравенств $g_{\langle I_1, w_i \rangle} \leq \Delta_1$ и $g^*_{\langle I_1, w_i \rangle} \leq \Delta_1^* = \Delta_1 + \varepsilon$ получаем первое ограничение на величины x_i:

$$x_i \leq \Delta_1 + \varepsilon - g_{\langle I_1, w_i \rangle}. \tag{9}$$

Из условия сохранения потока в каждом узле сети следует, что величина потока, входящего в вершину I_1, изменилась на величину $\sum_{i \in N} x_i$, откуда следует второе ограничение:

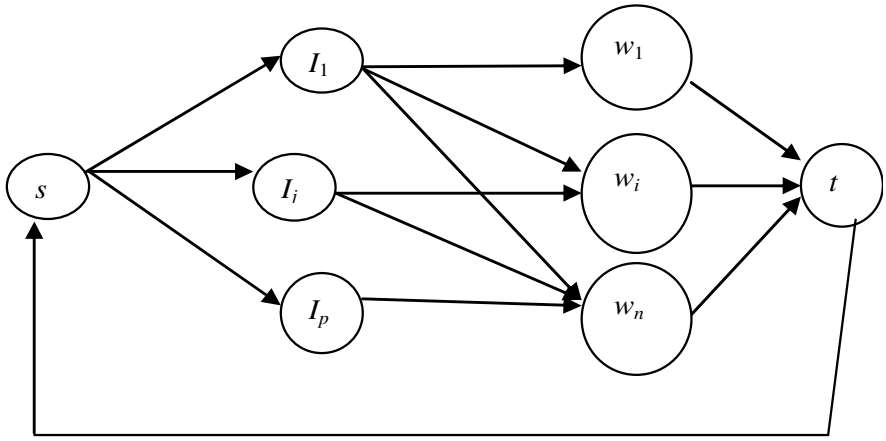


Рис.4 Потоковая сеть G*

$$\sum_{i \in N} x_i \leq m\varepsilon. \tag{10}$$

Поскольку ненулевую стоимость потока имеют только дуги $\langle w_i, t \rangle$, то общая стоимость потока изменится на величину $\sum_{i \in N} x_i \cdot \left(-\frac{1}{a_i} \right)$, откуда следует третье ограничение на x_i:

$$\sum_{i \in N} x_i \cdot \left(-\frac{1}{a_i} \right) + \delta \leq 0. \tag{11}$$

Минимизируя ε при указанных ограничениях (8) – (11), получаем следующую задачу линейного программирования:

$$\begin{aligned} &\varepsilon \to \min, \\ &x_i \leq \Delta_1 + \varepsilon - g_{\langle I_1, w_i \rangle}, \quad i \in N, \\ &\sum_{i \in N} x_i \leq m\varepsilon, \end{aligned}$$

$$\sum_{i\in N} x_i \cdot \left(-\frac{1}{a_i}\right) + \delta \le 0,$$
$$\varepsilon > 0, x_i \ge 0, i \in N.$$

2.3. Необходимое и достаточное условие существования допустимого расписания

Для произвольного подмножества $\overline{N} \subseteq N$ определим множество номеров l интервалов I_l

$$I(\overline{N}) = \left\{ l : 1 \le l \le p, \textit{существует } i \in \overline{N}, I_l \in (b_i, f_i] \right\}$$

и величину $\bar{n}(l) = \left| \left\{ i \in \overline{N} : I_l \in (b_i, f_i] \right\} \right|$. Как следует из [4], допустимое расписание существует тогда и только тогда, когда для любого подмножества $\overline{N} \subseteq N$ выполняется неравенство

$$\sum_{i\in\overline{N}} t_i \le \sum_{l\in I(\overline{N})} \Delta_l \min\left(m, \bar{n}(l)\right). \qquad (12)$$

При этом достаточно рассмотреть только такие подмножества $\overline{N}$, для которых $I(\overline{N}) = \{l_1, l_1+1, ..., l_2\}$ при некоторых l_1, l_2. Поскольку директивные интервалы имеют вид $(0, f_i]$, $i \in N$ и $f_i < f_{i+1}$ при всех $i \in N$, достаточно рассматривать только подмножества $\overline{N}$ следующего вида: $\{1\}$, $\{1,2\}$, $\{1,2,3\}$, ..., $\{1,2,...,n\}$. Для подмножества $\{1\}$ условие (8) имеет вид $t_1 \le \Delta_1$; для подмножества $\{1,2\}$ – $t_1 + t_2 \le \Delta_1 \cdot 2 + \Delta_2$. В общем случае для подмножества $\{1,2,...,k\}$ имеем $\sum_{i=1}^{k} t_i \le \sum_{i=1}^{k} \Delta_i \cdot \min(m, k-i+1)$. Увеличим правые границы директивных интервалов всех работ на ε. Получим, что $\Delta_1^* = \Delta_1 + \varepsilon$ и $\Delta_j^* = \Delta_j$, $j \in N \setminus \{1\}$. Таким образом, для некоторого фиксированного распределения ресурсов, которое можно найти, например, с помощью алгоритма, описанного в разд. 2, получаем следующую задачу линейного программирования:

$$\varepsilon \to \min,$$

$$\sum_{i\in N} r_i \le R,$$

$$\sum_{i=1}^{k} t_i \le \sum_{i=1}^{k} \Delta_i^* \cdot \min(m, k-i+1), k = \overline{1,n}.$$

Литература

1. *А.Б. Барский* Параллельные процессы в вычислительных системах. Планирование и организация. – М.: Радио и связь, 1990.

2. *Э.Г. Давыдов* Исследование операций. – М.: Высшая школа, 1990.

3. *М. Мину* Математическое программирование. Теория и алгоритмы. – М.: Наука, 1990.

4. *В.С. Танаев, В.С Гордон., Я.М. Шафранский* Теория расписаний. Одностадийные системы. – М.: Наука, 1984.

5. *A. Federgruen, H. Groenevelt.* "Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Technique". //Management Sci. – 1986, V. 32, No.3, March . PP. 812 – 829.

6. *T. Gonzales., S. Sahni* Preemptive scheduling of uniform processor systems//Journal of the Association for Computing Machinery. – 1978, 25, 1. Pp. 92 – 101.

7. *P.M. Vaidya.* Speeding up linear programming using fast matrix multiplication. // "Proc. 30th Annual Symposiuon Foundations Comp. Sci." – 1989, PP. 332 – 337.

Scheduling algorithms in multiprocessor systems with additional resource

M.G.Furugian

Abstract: The problem of deadline scheduling in multiprocessor real-time-system is studied. A restriction on memory is accounted. Method which based on reducing this problem to multiproduct network flow problem was elaborated. Moreover the problem of deadline scheduling in multiprocessor system in the case of execution times of works depend on additional resource is studied.

Хранение истории изменения информации в реляционной базе данных

А.Г. Прилипко

В статье рассматриваются основные подходы для хранения истории изменений в реляционной базе данных. Анализируются области их применения, особенности их реализации, достоинства и недостатки. Разработаны способ для хранения истории изменения данных в компактном виде и способ для протоколирования действий пользователя.

1. Введение

В автоматизированных информационных системах [5], применяемых для управления предприятием или организацией, часто требуется не только хранить данные, но и отслеживать историю их изменения. В частности, бывает необходимо сохранять предыдущие состояния данных, а также информацию о том, кем и когда были внесены изменения. Это позволяет получать данные из информационной системы не только на текущий, но и на любой из предшествующих моментов времени. Также хранение неактуальных на текущий момент времени данных может помочь при разборе спорных ситуаций и при необходимости откатить внесенные изменения.

Информация об истории изменения данных может использоваться и при инкрементальном переносе данных [4] для определения того, вносились ли изменения после предыдущего переноса данных, и какие именно.

2. Сравнение способов хранения истории изменений в реляционной базе данных

Поскольку в подавляющем большинстве современных корпоративных информационных систем для хранения данных применяются реляционные базы данных, то в дальнейшем при рассмотрении задачи хранения истории изменения данных мы будем считать, что данные располагаются в таблицах реляционной базы данных.

Рассмотрим задачу хранения истории изменений информации в реляционной базе данных. Здесь можно предложить несколько решений, их подробное описание будет приведено ниже:

1. добавление служебных полей в таблицу, которые позволят определить актуальную версию данных, а также кем и когда были внесены изменения;
2. создание аналогичной таблицы, в которой будут храниться как измененные, так и устаревшие данные;
3. создание общей таблицы, в которой будут храниться только измененные данные.

Каждое из этих решений имеет свои особенности реализации, связанные с выполнением дополнительных действий при выборке, модификации, вставке или удалении строки таблицы. Рассмотрим эти особенности, используя в качестве примера таблицу с именем Object, для которой организуется хранение истории изменения данных.

Способ, при котором история изменений хранится в той же таблице *Object*, проще всего реализовать при разработке базы данных «с нуля». В таблице *Object* уникальность записи определяется с помощью полей *Object_id* (ссылка на таблицу *Object_id*) и *Time_begin* (момент времени, когда запись стала актуальной). Специальная дополнительная таблица *Object_id* используется для хранения идентификаторов объектов. Использование этой таблицы позволяет автоматически гарантировать их уникальность при одновременной работе нескольких пользователей:

```
CREATE TABLE Object_id (
  Id INT UNSIGNED PRIMARY KEY
AUTO_INCREMENT
);

CREATE TABLE Object (
  Object_id INT INSIGNED NOT NULL,
  Name VARCHAR(100) NOT NULL,
  ...
  Time_begin DATETIME NOT NULL DEFAULT
getdate(),
  Time_end DATETIME NOT NULL DEFAULT
('3000-01-01'),
  Author VARCHAR(100) NOT NULL,
  Explanation TEXT,
  FOREIGH KEY (Object_id) REFERENCES
Object_id (Id),
  UNIQUE KEY (Object, Time_begin)
);
```

Рассмотрим описание полей таблицы *Object*:

- до поля *Time_begin* находятся информационные поля;
- *Time_begin* – момент времени, когда данные были изменены пользователем;
- *Time_end* – момент времени, когда данные перестали быть актуальными;
- *Author* – пользователь, который изменил запись в базе данных;
- *Explanation* – пояснение, почему было сделано изменение записи.

При добавлении нового объекта сначала создается запись в таблице *Object_id* для получения идентификатора объекта, после чего создается запись в таблице *Object* и во все поля этой записи заносится необходимая информация.

При изменении одного или нескольких полей в уже существующей записи в таблице *Object* создается новая запись с тем же идентификатором *Object_id*, но уже с другим значением поля *Time_begin*. В записи с тем же идентификатором *Object_id* необходимо изменить поле *Time_end* на время, когда было сделано изменение.

При удалении записи необходимо изменить поле *Time_end*, записав в него информацию о времени удаления. Хотя информация остается в базе данных, но благодаря условиям видимости, накладываемым на поля *Time_begin* и *Time_end* в запросах, конечному пользователю она не видна.

Преимуществами этого способа хранения истории изменения данных являются простота реализации и единообразие хранения актуального состояния и истории. Основным недостатком этого способа является большая избыточность хранимой информации. Хотя в современных условиях это и не является критичным при работе со сравнительно небольшими объемами данных, но требует добавления дополнительных полей во все таблицы, для которых необходимо хранить историю, а также дополнительного учета в операциях над данными наличия нескольких вариантов одного и того же объекта. Кроме того, при создании уникальных индексов для таблицы в список полей надо включать еще и поля с информацией о времени создания и удаления объекта.

Если необходимо хранить историю изменения данных для уже существующих таблиц базы данных или не требуется постоянный учет наличия в таблице нескольких вариантов объектов, то более оптимальным вариантом хранения истории изменения данных будет создание дополнительной таблицы *Object_history*, в которой хранится история изменений и актуальные данные. В таблице *Object* в этом случае хранятся только актуальные данные.

```
CREATE TABLE Object (
  Id INT UNSIGNED PRIMARY KEY
AUTO_INCREMENT,
  Name VARCHAR(100) NOT NULL,
  ...
);

CREATE TABLE Object_history (
  Object_id INT INSIGNED NOT NULL,
  Name VARCHAR(100) NOT NULL,
  ...
  Time_begin DATETIME NOT NULL DEFAULT
getdate(),
  Time_end DATETIME NOT NULL DEFAULT
('3000-01-01'),
  Author VARCHAR(100) NOT NULL,
  Explanation TEXT,
  FOREIGH KEY (Object_id) REFERENCES
Object_id(Id),
  UNIQUE KEY (Object, Time_begin)
);
```

Назначение полей в таблицах такое же, как и в первом случае. Первичный ключ в таблице *Object_history* состоит из двух полей: *Object_id* (идентификатор объекта из таблицы) и *Time_begin* (момент времени сохранения истории объекта в этой таблице).

При добавлении нового объекта создается по одной записи как в таблице *Object*, так и в таблице *Object_history*.

При изменении уже существующей записи в таблицу *Object* заносятся обновленные данные, а в таблице *Object_history* для предыдущего состояния записи с идентификатором *Object_id* в поле *Time_end* заносится время, когда было выполнено изменение, и создается запись с новым состоянием записи.

При выполнении операции удаления запись в таблице *Object* удаляется, а в таблице *Object_history* в поле *Time_end* устанавливается время удаления записи.

Этот метод наследует все преимущества и недостатки метода, рассмотренного ранее. Дополнительным преимуществом этого метода является наличие таблицы, в которой хранится актуальная информация, и доступ к последнему состоянию объектов осуществляется простым обращением к идентификатору необходимой записи. Кроме того, не требуется изменять уже существующие программы и отчеты для учета того факта, что в базе данных хранится история изменений. Недостатком же является дублирование актуальной информации.

Третьим вариантом хранения истории изменения данных является создание дополнительной таблицы, в которой хранятся только те данные, которые перестали быть актуальными. Такую таблицу *Object_old* можно сделать как для всей базы данных, так и для каждой таблицы отдельно.

```
CREATE TABLE Object(
  Id INT UNSIGNED PRIMARY KEY
AUTO_INCREMENT,
  Name VARCHAR(100) NOT NULL,
  ...
);

CREATE TABLE Object_old (
  Table VARCHAR(100) NOT NULL,
  Stroka VARCHAR(100) NOT NULL,
  Field VARCHAR(100) NOT NULL,
  Type_value VARCHAR(100) NOT NULL,
  Old_value VARCHAR(100) NOT NULL,
  Type_action INT,
  Time_begin DATETIME NOT NULL DEFAULT
getdate(),
  Time_end DATETIME NOT NULL DEFAULT
('3000-01-01'),
  Author VARCHAR(100) NOT NULL,
  Explanation TEXT,
  UNIQUE KEY (Table, Stroka, Field, Time_begin)
);
```

Опишем назначение полей таблицы *Object_old*:

- *Table* - указатель на таблицу, в которой произошло изменение;
- *Stroka* – указатель на строку таблицы, в которой произошло изменение;
- *Field* – название поля таблицы, в которой произошло изменение;
- *Type_value* – тип поля (целое, вещественное, строка и т.д.);
- *Old_value* – старое значение поля;
- *Type_action* – вид действия (изменение, удаление);
- *Time_begin* – дата и время, когда пользователь изменил данные;
- *Time_end* – дата и время, до которой использовалось значение поля указанное в *Old_value*;
- *Author* – идентификатор пользователя, который изменил состояние базы;
- *Explanation* – объяснение, почему было введено изменение.

При добавлении нового объекта вносится запись только в таблицу *Object*, при этом никаких дополнительных действий осуществлять не надо.

При внесении изменений в строку для каждого измененного поля его значение переносится в поле *Old_value* таблицы *Object_old* и заполняются соответствующими значениями остальные поля этой таблицы. Поле *Type_action* помечается признаком изменения строки. Устаревшие значения полей таблицы *Object* заменяются актуальными значениями.

При удалении какой-то строки в таблице *Object* создается количество записей в таблице *Object_old*, равное количеству информационных полей в таблице *Object*. При этом в поле *Type_action* заносится признак удаления строки. Строка из таблицы *Object* удаляется.

Для внесения изменений в строку и удаления записи из таблицы *Object* необходимо установить обработчик событий (триггер), который будет реагировать на соответствующие действия и автоматически заполнять таблицу *Object_old*. Опишем действия, которые должны произойти, и параметры, необходимые для обработки событий onUpdate и onDelete.

Обработчику событий onUpdate необходимо передать следующие параметры: старое и новое значения поля, а также комментарий, почему произошло изменение. В таблице *Object_old* создается новая запись, в поля *Table*, *Stroka*, *Field* и *Type_value* которой, заносится информация, указывающая на изменяемое в данный момент поле. Его старое значение, которое передается обработчику события onUpdate, заносится в поле *Old_value*. В поле *Type_action* устанавливается признак изменения строки. Исходя из полученной информации, обработчик событий проверяет таблицу *Object_old* с тем, чтобы понять, проводились ли изменения этого поля ранее. Если они уже были раньше, то полю *Time_begin* текущей записи присваивается значение поля *Time_end* предыдущей записи, а полю *Time_end* текущей записи – значение текущего момента времени. В поле *Author* заносится идентификатор пользователя, который обновил поле. Комментарии, которые передаются в обработчик событий onUpdate, заносятся в поле *Explanation*. На этом заполнение таблицы *Object_old* заканчивается. Кроме внесения изменений в таблицу *Object_old* необходимо также записать новое значение поля, переданного в качестве параметра триггера, в строку таблицы *Object*.

Обработчику событий onDelete значения полей передавать не нужно. При удалении строки достаточно выполнить ряд изменений в таблице *Object_old*. Для этого в цикле для каждого поля удаляемой строки выполняется следующая последовательность действий. В поле *Type_action* заносится признак удаления строки. Поля *Time_begin*, *Time_end*, *Author* и *Explanation* заполняются так же, как и при обработке события onUpdate. После этого запись из таблицы *Object* удаляется.

Преимуществом этого метода является меньшая избыточность хранимой информации по сравнению с предыдущими способами хранения истории при внесении изменений в одно или небольшое число полей любой таблицы, но из-за этого усложняется получение неактуальных данных. К недостатку этого метода можно отнести большое количество информации, вводимой при выполнении операции удаления или изменения большого числа полей.

Еще раз отметим, что этот подход можно применять как для одной таблицы, так и для всей базы данных. При его использовании для одной таблицы поле *Table* не требуется. Особенностью этого метода является возможность полной унификации хранения истории изменения данных с использованием одних и тех же дополнительных таблиц.

3. Хранение истории изменения данных в компактном виде

На основе изучения и анализа описанных методов в системе АСУ НИИСИ РАН [2,3] был разработан и внедрен способ хранения истории изменений данных с использованием специальных дополнительных таблиц, единых для всей базы данных. Отметим, что интерфейс для доступа пользователя к этой системе через web-интерфейс автоматически формируется с помощью специальных комментариев, задаваемых при описании таблиц [3].

Первая дополнительная таблица служит для хранения информации о логических транзакциях (добавление, изменение и удаление записей или полей) пользователей или программ. Во второй таблице содержатся предыдущие значения полей таблиц и информация об удаленных записях. При этом в ней хранятся только измененные значения полей в специальном компактном формате.

```
CREATE TABLE Object (
  id INT UNSIGNED PRIMARY KEY
AUTO_INCREMENT,
  name VARCHAR(100) NOT NULL,
  ...
);

CREATE TABLE Transaction (
```

```
    id INT UNSIGNED PRIMARY KEY
AUTO_INCREMENT,
    name VARCHAR(100) COMMENT “Описание”,
    auth UNSIGNED INT COMMENT “Автор”,
    dat DATATIME NOT NULL DEFAULT getdate()
COMMENT “Дата”,
    comm VARCHAR(512) COMMENT “Примечание”,
    FOREIGN KEY (auth) REFERENCES user(id)
  ) COMMENT “Транзакции”;

  CREATE TABLE Tizm (
    id INT UNSIGNED primary key
AUTO_INCREMENT,
    tr UNSIGNED INT COMMENT “Транзакция”,
    ntable VARCHAR(80) COMMENT “Исходная
таблица”,
    ids  UNSIGNED INT COMMENT “Номер строки”,
    nfield VACHAR(80) COMMENT “Название
столбца”,
    prev VARCHAR(32768) “Предыдущее значение”
  ) COMMENT “Изменения”;
```

В описании таблицы *Object*, для которой необходимо сохранять информацию о внесенных изменениях, в комментарии к полю *id* указывается признак, указывающий, что при выполнении операций пользователя необходимо дополнительно выполнить следующие дополнительные действия:

1. открыть транзакцию пользователя;
2. вызвать обработчик событий для каждой измененной, созданной или удаленной записи в таблице;
3. выполнить закрытие транзакции пользователя.

При открытии транзакции пользователя автоматически создается новая запись в таблице *Transaction*, в которую записывается информация о том, кто, когда и зачем ее открыл (поля *auth*, *dat*, *name* соответственно). Транзакции могут открываться не только при внесении данных пользователем, но и при выполнении программ, которые загружают данные**.** При использовании вложенных транзакций вносимые изменения будут рассматриваться как единое целое.

При обработке событий вызывается специальная функция SaveHistory для сохранения истории изменений. Ей передаются три параметра: имя таблицы *Object*, в которой произошло изменение, значение поля *id* записи из таблицы *Object* и дополнительный параметр *ido*, который указывает на запись, содержащую предыдущее состояние строки.

При добавлении новой строки в таблицу Object значение параметра *ido* для функции указывается равным нулю. В этом случае в таблице *Tizm* в поле *nfield* просто отмечается факт ее появления. При удалении строки аргументу *id* присваивается значение “-1”, а параметру *ido* — значение поля *id* удаляемой записи. При этом необходимо сохранить запись целиком. В поле *nfield* записи в таблице изменений *Tizm* указывается, что запись удалена, а ее содержимое в специальном компактном формате сохраняется в текстовом виде в поле *prev*, причем пустые поля сохранять не нужно. Если в поле *prev* не хватает места для хранения всех полей, то создаются дополнительные записи в таблице изменений.

При внесении изменений в строку перед вызовом функции средствами инструментальной системы в таблице *Object* временно создается строка с идентификатором *ido*, в которую копируется ее предыдущее состояние. Если в таблице *Object* есть уникальные индексы, то соответствующие им поля специальным образом изменяются. После выполнения функции строка с идентификатором *ido* автоматически удаляется.

Если изменений в строках мало, то их можно сохранить в виде нескольких строк в таблице *Tizm* с указанием поля и предыдущего значения в текстовом виде. В противном случае в поле *prev* сохраняется информация только об измененных полях в компактном формате с указанием названий полей и их предыдущих, в том числе и пустых, значений.

В системе управления базы данных MySQL предоставляются средства для получения информации о структуре таблицы, что позволяет организовать перебор полей и сравнение их значений в рамках вызываемой функции SaveHistory, а хранение нескольких полей с их предыдущими значениями представить в специальном компактном текстовом формате:

имя поля | текстовое значение | … | имя поля | текстовое значение,

где указываются имена полей таблицы и их предыдущие значения в текстовом виде, разделенные символом “|”. Символы “|” и “\”, встречающиеся внутри значения текстового поля, дополнительно экранируются с помощью символа “\”.

Такой способ хранения позволяет восстанавливать значения полей и при изменении набора и/или последовательности полей в таблице. Кроме того, транзакция может, вообще говоря, затрагивать не одну, а несколько таблиц.

Важным моментом является также то, что для хранения истории изменений не требуется внесения изменений в структуру исходных таблиц. А учитывая, что изменения чаще всего заключаются в добавлении строк и изменении полей пользователем, то такое хранение оказывается достаточно компактным.

При закрытии транзакции устанавливается время окончания транзакции в том случае, если изменения были подтверждены, и удаляется строка из таблицы transaction, если выполняется откат транзакции.

4. Протоколирование действий пользователя

Опишем другой способ хранения истории изменений, который реализован и успешно используется в системе АСУ НИИСИ РАН. В первую очередь он применяется для протоколирования действий пользователей, работающих в системе, при этом он позволяет хранить информацию за несколько лет.

Для реализации этого способа был создан ряд дополнительных таблиц. Основой данного механизма служит таблица *user_action*, в которой запоминается информация о входе пользователей в систему, в

частности, его имя, IP адрес и название подсистемы (домена), о выходе из системы, а также о просмотре, изменении данных, осуществлении операций поиска, отбора информации, формировании отчетов. Приведем описание таблицы *user_action* в формате MySQL:

```
CREATE TABLE user_action (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  user INT UNSIGNED DEFAULT NULL COMMENT "Пользователь",
  address VARCHAR(64) COMMENT "IP адрес",
  domain INT UNSIGNED DEFAULT NULL COMMENT "Домен",
  action_type INT UNSIGNED DEFAULT NULL COMMENT "Действие",
  action_comment VARCHAR(255) COMMENT "Дополнение",
  tstamp TIMESTAMP NOT NULL CURRENT_TIMESTAMP COMMENT "Время действия",
  obj_type INT UNSIGNED DEFAULT NULL COMMENT "Таблица",
  obj_id INT UNSIGNED DEFAULT NULL COMMENT "ID",
  obj_path INT UNSIGNED DEFAULT NULL COMMENT "Имя объекта",
  obj_state0 INT UNSIGNED DEFAULT NULL COMMENT "Состояние до изменений",
  obj_state1 INT UNSIGNED DEFAULT NULL COMMENT "Состояние после изменений",
  ...
  FOREIGN KEY (user) REFERENCES user(id),
  FOREIGN KEY (domain) REFERENCES domain(id),
  FOREIGN KEY (action_type) REFERENCES user_action_type(id),
  FOREIGN KEY (obj_type) REFERENCES table_list(id),
  FOREIGN KEY (obj_path) REFERENCES obj_path(id),
  FOREIGN KEY (obj_state0) REFERENCES obj_state(id),
  FOREIGN KEY (obj_state1) REFERENCES obj_state(id)
);
```

Поля *obj_type*, *obj_id*, *obj_path* однозначно идентифицируют объект, над которым осуществляется действие. Кроме того, наличие полей *obj_state0*, *obj_state1* позволяет получить информацию о старом и новом состояниях объекта при его создании, изменении или удалении. В этих полях хранятся ссылки на строки специальной таблицы *obj_state*, поле *string* в которой содержит названия полей и их значения в формате YAML [1]:

```
CREATE TABLE obj_state (
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  string TEXT COMMENT "Значение полей"
);
```

Наличие этой информация позволяет при необходимости восстановить случайно удаленные или измененные данные, а также определить, кто именно и когда производил изменения через web-интерфейс.

Отметим, что на основе этого механизма могут быть созданы строки и в других таблицах, в которых отражается история изменений. Например, необходимо хранить или восстановить историю перемещений материальных ценностей, а в таблице материальных ценностей есть поле с информацией об их местонахождении. Тогда на основе изменений этого поля можно получить сведения об их перемещениях. Правда, после переноса этой информации необходимо ее дополнительно проанализировать и удалить изменения, случайно сделанные пользователями.

С помощью механизма протоколирования действий пользователей можно, помимо всего прочего, определить, кто и когда что сделал или делает в системе. В частности, при определении того, работает ли кто-нибудь в данный момент в системе, может понадобиться информация и о последних действиях пользователей, поскольку при взаимодействии через web-интерфейс довольно часто пользователь забывает корректно выйти из системы и требуется понять, делает ли он что-нибудь в системе в данный момент времени. Для этого создано специальное представление (view) user_action_view, с помощью которого можно получить необходимую для этого информацию:

```
CREATE VIEW user_action_view AS
SELECT d.domain AS domain, u.name AS user, uat.name AS action, t.table_name AS table, ua.obj_id AS object, ua.tstamp AS time, ua.address AS address
FROM  ( ( user_action ua JOIN user u )
JOIN user_action_type uat )
LEFT JOIN domain d ON (ua.domain = d.id)
LEFT JOIN table_list t ON ( ua.obj_type = t.id )
WHERE ua.user = u.id AND ua.action_type = uat.id
ORDER BY ua.stamp DESC;
```

В результате можно написать макрос на языке shell, который позволяет из командной строки получить информацию о последних действиях (их количество указывается первым аргументом) пользователей:

```
echo SELECT \* FROM user_action_view LIMIT $1 | mysql -B asu -ureport_user -p<пароль>
```

5. Заключение

В статье проведен сравнительный анализ трех наиболее распространенных подходов хранения истории изменений в реляционной базе данных. На их основе были разработаны два способа хранения истории изменения данных с минимальными дополнительными накладными расходами на обработку и хранение данных. Один из них используется для хранения истории изменения данных в компактном виде и не требует изменений в уже существующих алгоритмах доступа к данным. Такой

подход к хранению истории изменения данных можно использовать как для одной таблицы, так и для всей базы данных. Другой способ используется для протоколирования действий пользователя, что позволяет администратору системы знать, кто и когда внес изменения в информационную систему. Разработанные способы внедрены в эксплуатацию и используются в системе АСУ НИИСИ РАН.

Литература

1. YAML официальный сайт. URL: http://www.yaml.org/ (дата обращения: 03.05.2012).

2. И. Б. Егорычев Инструментарий для построения автоматизированных учетных систем с web-интерфейсом // *Программные продукты и системы.* – № 4, 2008. С. 49-52.

3. И.Б. Егорычев Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей. Дисс. ... канд. ф.-м. н. Москва, 2010.

4. А.Г. Прилипко Инкрементальное тиражирование данных при взаимодействии информационных систем // *Программные продукты и системы.* – № 4, 2011. С. 10-13.

5. А.П. Реутов, М.В. Черняков, С.Н. Замуруев Автоматизированные информационные системы. Методы построения и исследования. – М.: Радиотехника, 2010.

Storage the history of data changes in a relational database

A. G. Prilipko

Abstract: In the present paper the main approaches for storage the history of changes in a relational database are considered. Areas of their application, features of their realization, advantages and disadvantages are analyzed. A new approach for the history of data changes in a compact view and approach for recording user's actions are developed.

ДССП для троичной виртуальной машины

А.А. Бурцев, к. ф.-м. н., доц.; М.А. Бурцев.

В НИЛ ЭВМ МГУ разработан программный комплекс ТВМ (Троичная Виртуальная Машина), имитирующий функционирование современного варианта троичного процессора двухстековой архитектуры. В статье представляются два варианта ДССП (Диалоговой Системы Структурированного Программирования) для ТВМ. ДССП-ТВМ - кросс-система, позволяющая создавать программы для ТВМ с помощью кросс-компилятора. ДССП/ТВМ - интерпретатор языка ДССП, функционирующий на ТВМ как резидентное ПО. Даётся характеристика их входного языка ДССП-Т как троичной версии языка ДССП.

1. Введение

В настоящее время, когда традиционная микроэлектроника, построенная на двоичной системе счисления, подходит к пределу своих технологических возможностей, в специальных изданиях и Интернете отмечается заметное повышение научной и инженерной активности в области исследований принципиально иных путей построения вычислительных средств, основанных на других системах счисления.

Как одна из наиболее перспективных, многих исследователей давно привлекает троичная симметричная система счисления (с цифрами -1,0,+1), которую когда-то Д.Кнут назвал “самой изящной”, и которая была полвека назад успешно воплощена в троичных цифровых машинах (ЦМ) “Сетунь” и “Сетунь-70” [1]. Использование симметричного троичного кода и трёхзначной логики даёт ряд неоспоримых преимуществ (перед двоичными системами), которые были подтверждены практической эксплуатацией этих машин.

Среди множества достоинств рассматриваемой троичной симметричной системы счисления можно выделить следующие:

- тем же количеством разрядов можно кодировать больший диапазон чисел;
- единообразное представление отрицательных и положительных чисел;
- упрощённая реализация ряда арифметических операций (сдвига, сравнения чисел, смены знака); трёхзначность функции “знак числа”;
- оптимальное округление чисел простым отсечением младших разрядов и взаимокомпенсируемость погрешностей округления в процессе вычисления.

В научно-исследовательской лаборатории (НИЛ) троичной информатики (ранее НИЛ ЭВМ) на факультете ВМК МГУ создан программный комплекс ТВМ (Троичная Виртуальная Машина) [2], имитирующий функционирование современного варианта троичного процессора двухстековой архитектуры с поддержкой структурированного программирования на уровне машинных команд [3], аналогичной той, что была обеспечена в троичной ЦМ “Сетунь-70”.

Предполагалось, что основным языком разработки программ для ТВМ станет ДССП-Т – троичный вариант языка ДССП [4-6]. Требовалось создать систему разработки программ для ТВМ на языке ДССП-Т. Эта работа была выполнена в два этапа.

Сначала (на 1-ом этапе) была построена кросс-система ДССП-ТВМ [7], позволяющая создавать программы для ТВМ на языке ДССП-Т с помощью кросс-компилятора. На 2-ом этапе, используя уже саму ДССП-ТВМ как среду разработки, был создан диалоговый интерпретатор ДССП/ТВМ, способный функционировать на троичной машине (ТВМ) в качестве её резидентной системы программирования.

2. Общая характеристика ДССП

Диалоговая система структурированного программирования (ДССП) была создана в начале 80-х годов XX века в проблемной лаборатории ЭВМ на факультете ВМК МГУ под руководством Брусенцова Н.П. ДССП была призвана существенно облегчить разработку ПО для широкого класса малых цифровых машин – мини- и микрокомпьютеров.

При создании ДССП преследовалось сразу несколько целей. Во-первых, ДССП создавалась в качестве программного имитатора (эмулятора) новой двоичной цифровой минимашины, в которой предполагалось воплотить ценные идеи двухстековой архитектуры и структурированных команд управления, унаследованные от модернизированной троичной ЦМ “Сетунь-70”. В роли такого имитатора или образно говоря виртуального процессора по сути и функционирует внутренний интерпретатор ДССП, являющийся сердцевиной (ядром) системы.

Во-вторых, ДССП создавалась как интегрированная среда разработки программ, легко доступная для освоения широкому классу пользователей микрокомпьютерной техники и персональных компьютеров. Поэтому при создании ДССП была заимствована идея словарной организации системы ФОРТ (Forth) [8] и обеспечен диалоговый характер взаимодействия с пользователем на всех стадиях разработки программы: начиная от её редактирования и вплоть до её окончательной отладки. Такой диалог обеспечивается внешним интерпретатором ДССП, функционирующим в режиме исполнения потока слов-команд.

Программа, составленная на входном языке ДССП, сначала преобразовывается во внутреннее представление, для характеристики которого применяется специальный термин – сшитый (threaded) код. Совокупность программных компонент, выполняющих такое преобразование, принято называть компилирующими средствами или просто компилятором системы. Внутрен-

ний и внешний интерпретатор, компилятор и словарь ядра, а также встроенные редактор текстов и отладчик вместе составляют базовый комплект ПО системы. На его основе путём наращивания словаря каждый пользователь может создать свой вариант расширения ДССП, отвечающим нуждам его прикладной области.

Программирование в ДССП заключается в наполнении её словаря новыми словами. Всякая операция (процедура), поддерживаемая системой, обозначается отдельным словом (слово – это любая последовательность печатных символов, отличных от пробела). Совокупность слов, понимаемых системой, и составляет ее язык (словарь). Система расширяется путем определения новых слов, которые сразу же после добавления в словарь могут быть использованы в системе.

Любое действие, совершаемое системой, вызывается словом. Предусмотренные в системе слова исполняют самые разнообразные функции. Они выполняют вычисления или обработку данных в стеке, доступ к памяти или к переменным, служат для организации ветвлений и циклов, обеспечивают построение тел для новых слов, вносимых в словарь, управляют состоянием словаря и даже самим процессом выполнения задания в системе.

ДССП характеризуется двухстековой архитектурой. Стек операндов, над которым действуют все операции по обработке данных, приводит к компактной польской инверсной записи арифметических выражений, а управляющий стек служит для сохранения адресов возвратов из процедур.

Следующие две характерные черты ДССП выгодно отличают её от системы ФОРТ: 1) специфическая процедурная версия структурированных команд управления; 2) и поддержка пошаговой (step-by-step) нисходящей (top-down) разработки программы.

В ДССП каждая из команд ветвлений вместо перехода по телу сшитого кода (как это обычно делается в системе ФОРТ) по сути осуществляет вызов процедуры слова, выбранного по условию, а каждая команда цикла организует многократный вызов процедуры слова, пока такой вызов допускается условием цикла. Таким образом, в ДССП поддержка структурированного программирования осуществляется непосредственно на уровне сшитого кода аналогично тому, как такая поддержка была обеспечена в ЦМ “Сетунь-70” на уровне машинных команд.

Определение нового слова в ДССП и в языке ФОРТ синтаксически выглядят одинаково:

: P P1 P2 **...** Pn **;** [определение процедуры с именем P]

Однако ФОРТ допускает подобное определение, только если все слова P1 P2 ... Pn уже были определены и стали известны системе. А ДССП допускает, что в этой последовательности слов P1 P2 **...** Pn могут встречаться в том числе и такие слова, которые еще не были определены и предполагается, что будут определены впоследствии. Именно благодаря такой особенности ДССП и позволяет разрабатывать программу методом нисходящего анализа: сначала определить главное слово, затем слова следующего уровня, из которых оно определяется, и так постепенно спуститься до детального определения самых простых слов.

Первая версия ДССП (ДССП-НЦ [4]) была создана для микрокомпьютеров “Электроника НЦ-03Д”. Уже с этой версии в ДССП нашли отражение характерные черты языка ассемблера ЦМ “Сетунь-70”: его структурированные команды управления для организации ветвления и цикла по условию: **BRS DW0N** .

На следующем этапе своего развития [9] ДССП создавалась уже для микрокомпьютеров так называемой унифицированной архитектуры (PDP-11). Причём наряду с основной версией ДССП-80 [5] разрабатывались также и экспериментальные версии, представляющие собой варианты развития ДССП, ориентированные на построение программ управления периферией (ДССП-ПМ [10]) и операционных систем реального времени (ДССП-РВ [11]).

В ДССП-80 были предложены новые команды **RP DO** для организации циклов с выходами по условию **EX- EX0 EX+** , а также типизированный набор префиксных операций: **! ’ ...** для единообразной работы с переменными и массивами различных типов. Была усовершенствована организация словаря (он стал перемещаемым, в нём появились подсловари и секции), позволившая упростить разработку сложных многомодульных программ. Для автоматизации сборки как самой ДССП, так и законченной прикладной программы, были предложены специальные инструментальные средства: компоновщик и целевой компилятор

Хотя экспериментальные версии ДССП-ПМ и ДССП-РВ не получили самостоятельного развития, предложенные в них новые программные механизмы были частично реализованы в последующих версиях ДССП [12,13]. Это механизм исключительных ситуаций, сопрограммный механизм, механизм прерываний на уровне сшитого кода, а также разнообразные средства взаимодействия параллельных процессов.

С начала 90-х годов ДССП стала активно использоваться в НИИСИ РАН в качестве инструментальной среды программирования для разработки внутреннего программного оснащения ряда компьютерных модулей, создаваемых на основе разных микропроцессоров (MC68020, R3000, Intel80386) [14].

В дальнейшем над развитием ДССП и распространением её на другие платформы трудились как сотрудники НИЛ ЭВМ МГУ вместе со студентами и аспирантами, так и сотрудники НИИСИ РАН [15]. В результате на протяжении ряда лет ДССП была реализована для компьютеров самой разнообразной архитектуры и в различных операционных средах. К середине 90-х годов ДССП функционировала почти на всех компьютерных платформах (PDP-11, Intel8080, Intel80x86, Motorola 68020, VAX, R3000) и в операционных средах (RT-11, RSX-11, UNIX, CP/M, MSDOS,Windows), получивших распространение в нашей стране.

Отдельные версии ДССП для разных платформ были плохо совместимы друг с другом. Их общий недостаток заключался в том, что ядро каждой из них создавалось на языке ассемблера базовой машины, что затрудняло перенос созданного ПО на другие компьютерные платформы. В 1998 г. в НИИСИ РАН была создана мобильная версия ДССП (ДССП/С), ядро которой было написано на языке Си [12]. При разработке ДССП/С была проведена такая модернизация работы с данными, благодаря которой в ДССП впоследствии были обеспечены возможности построения новых типов [16], аналогичные средствам объектно-

ориентированного программирования.

На протяжении почти 20-летнего периода своего развития [9] ДССП постоянно совершенствовалась, приобретая новые возможности, присущие современным языкам и системам программирования. В результате ДССП превратилась в среду программирования [13], позволяющую применять широкий спектр современных методов структурированной разработки сложных программных систем, сосредоточив в себе средства для структурированного, модульного, объектно-ориентированного и параллельного программирования, а также средства для структурированной обработки исключительных ситуаций.

3. Троичная виртуальная машина

Троичная виртуальная машина [2] (ТВМ, TVM - Ternary Virtual Machine) представляет собой программный эмулятор архитектуры троичного компьютера, основанного на троичной симметричной системе счисления и наследующего некоторые принципы троичных ЭВМ «Сетунь» и «Сетунь-70».

ТВМ включает в себя эмулятор троичного процессора, транслятор с языка ассемблера и интерактивный монитор для диалогового взаимодействия с пользователем, которые в совокупности образуют среду разработки и отладки программ.

Программный эмулятор работает на обычных двоичных компьютерах, полностью воссоздавая троичную среду на уровне машинных команд, регистрового файла, памяти и адресного пространства. Так что программист ТВМ имеет дело только с троичными объектами и не беспокоится об их реальном представлении.

В ТВМ используется троичная симметричная система счисления. Минимальной единицей информации в троичной машине является трит. Трит принимает одно из трёх значений: +1, 0, -1. Трайт в ТВМ состоит из 9 тритов. Основной единицей данных в ТВМ является слово, состоящее из 3 трайтов (27 тритов). ТВМ имеет двухстековую архитектуру. Для обработки данных используется стек данных. Арифметические и логические команды берут свои аргументы из стека данных и засылают в него обработанные результаты. Второй стек, называемый управляющим стеком или стеком возвратов, используется для сохранения адреса возврата из подпрограммы, сохранения контекста. Элементом стека возвратов является троичное слово.

Система команд включает в себя команды пересылок данных, арифметические и логические операции, группу команд для работы со стеком. Для управления ходом исполнения программы ТВМ предлагает команды, соответствующие основным управляющим конструкциям структурированного программирования: команды вызова подпрограммы, ряд команд для условного вызова одной из процедур и команду многократного вызова процедуры по условию.

4. Кросс-система ДССП-ТВМ

Система программирования ДССП-ТВМ предоставляет возможность разработки программ для троичного процессора на языке ДССП-Т и их исполнения на имитационной модели троичного процессора – ТВМ.

В ДССП-ТВМ в качестве составляющих её компонент используются следующие программные средства:

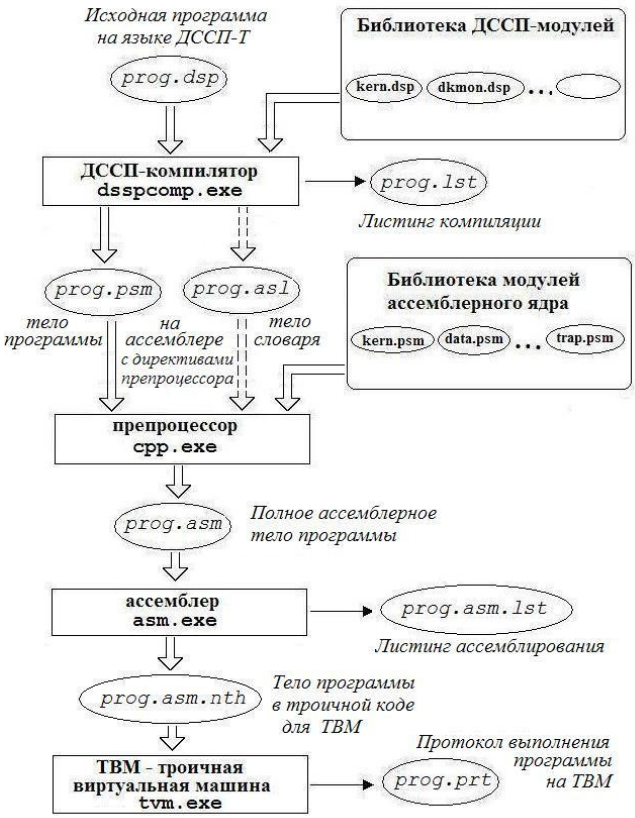


Рис.1. Основные компоненты ДССП-ТВМ и схема прогона ДССП-программы.

- кросс-компилятор (**dsspcomp.exe**);
- ДССП-библиотека (**kern.dsp,...**) – набор файлов на языке ДССП-Т (**dsp**-файлы), подключаемых в ДССП-программу командой LOAD в ходе её компиляции, которые содержат определения стандартного ассортимента наиболее употребительных слов;
- стандартный препроцессор (**cpp.exe**) языка Си ;
- ассемблерное ядро ДССП (**kern.psm,...**) – совокупность ассемблерных файлов (**psm**-файлов), подключаемых в ассемблерную программу на стадии препроцессорной обработки, которые содержат тела процедур, необходимые для исполнения ряда примитивов ядра ДССП;
- ассемблер (**asm.exe**) троичной машины;
- имитатор троичной машины (**tvm.exe**).

Программа, составленная на языке ДССП-Т, (файл **prog.dsp**) проходит несколько стадий обработки в ДССП-ТВМ (см. рис.1). Сначала она обрабатывается кросс-компилятором, который создаёт ассемблерную программу (файл **prog.psm**), содержащую также препроцессорные директивы. Компилятор формирует листинг сообщений о ходе компиляции (файл **prog.lst**), а также может создать ещё и ассемблерное тело словаря (файл **prog.asl**), сформированного программой.

Далее ассемблерная программа (файл **prog.psm**) вместе с файлами ассемблерного ядра ДССП обрабатывается препроцессором для получения единой ассемблерной программы (файл **prog.asm**), которая за-

тем обрабатывается ассемблером троичного процессора, в результате чего создаётся листинг ассемблирования программы (файл **prog.asm.lst**) и троичный код (файл **prog.asm.nth**), готовый для исполнения на троичном процессоре.

Наконец, имитатор троичного процессора (ТВМ) исполняет программу, представленную в троичном коде. Исполнение троичной программы на ТВМ можно отслеживать (на экране) не только в режиме диалога, но и получить протокол её работы (файл **prog.prt**).

Для автоматизации прохождения всех стадий обработки ДССП-программы (от её компиляции до исполнения на ТВМ), подготовлены соответствующие командные файлы. Так что полный прогон ДССП-программы можно вызвать всего одной командой: **dssp-tvm prog** .

5. Язык ДССП-Т

Язык ДССП-Т представляет собой ориентированный на троичную машину вариант языка ДССП, в котором удалось сохранить лучшие черты различных версий ДССП [4-6,10-15], разработанных для двоичных машин. Рассмотрим основные черты языка ДССП-Т, выделяя те его особенности, которые существенно отличают его от языка ДССП, описанного в [6].

Синтаксис языка ДССП-Т такой же простой, как и у обычной ДССП. Программа записывается как последовательность слов в свободном формате, разделителями слов являются пробелы (или приравненные к ним символы), а также комментарии. Словарь построен без разделения на подсловари и секции, но в нём можно осуществлять очистку, и обеспечена возможность помечать слова, которые всегда должны оставаться видимыми (неудаляемыми) в словаре.

В языке ДССП-Т допускаются разнообразные варианты комментаторных скобок так, чтобы никакие из символов: **([{}])** , употребляемых для комментариев, не запрещать программисту использовать в своих словах. Для настройки желаемого вида комментариев предусмотрены служебные слова: **() [] {}** . Допускаются также и комментарии, отделяемые двойными открывающимися скобками: **(([[{{** до конца текущей строки.

ДССП имеет двухстековую архитектуру. Все основные операции осуществляют действия над данными, помещёнными в стек операндов (арифметический стек). Одному элементу стека сопоставлено одно машинное слово из 27-тритов или 3-х трайтов. Стек операндов растёт в сторону увеличения адресов.

Управляющий стек (или стек возвратов) расположен в памяти так, чтобы расти навстречу стеку операндов (в сторону убывания адресов). Элементами стека являются 27-тритные слова, которые, как правило, трактуются как адреса возвратов из процедур. Предполагается, что помимо адресов возвратов в управляющий стек могут помещаться специального вида структуры – так называемые ловушки ситуаций.

Для объявления новых слов, представляющих те или иные сущности в ДССП-программе, используются специальные служебные слова, так называемые компилирующие слова или слова-компиляторы:

PROGRAM : VALUE VAR VCTR ARR TEXT STRUCT: ;STRUCT CLASS: SUBCLASS ;CLASS METHOD METHOD# METHOD= :M: :M= SITUATION ASM ;ASM

В языке ДССП-Т имя главного слова программы, с которого и предстоит начать исполнение всей программы, принято задавать в первой строке исходной программы после служебного слова **PROGRAM** :

PROGRAM DKINT

Слово-процедура в ДССП-программе определяется следующим образом:

: P P1 P2 ... PN **;**

При своём выполнении такая процедура P будет последовательно вызывать исполнение слов, из которых построено её тело: P1 ... PN. Причём в ДССП-программе не требуется, чтобы все используемые в таком определении слова должны уже быть определены заранее. В определяемом теле новой процедуры разрешается употреблять и те слова, которые могут быть определены позднее.

В общем виде объявление переменной в ДССП-программе выглядит так:

ИмяТипа **VAR** ИмяПерем {Пример:} TRYTE **VAR** X

Тип переменной зависит от того, какое слово с именем типа употреблялось перед её объявлением. Если тип не был задан, то он принимается по умолчанию (в настоящее время это тип TWORD , задающий троичное слово).

При объявлении вектора (одномерного массива):

{N} **VCTR** ИмяВектора {Пример:} 10 **VCTR** Y

в стеке задаётся его верхняя граница N (нижняя граница = 0). Так что в данном примере объявляется вектор Y[0:N] с N+1 элементом.

При объявлении многомерного массива вида:

{L1,..,Lm,m} **ARR** ИмяМас {Пример:} 10 20 2 **ARR** M

в стеке задаются его размерность m и m верхних границ по каждому измерению. Так что в примере объявляется массив M[0:10,0:20].

Тип элементов объявляемого массива определяется тем словом с именем типа, которое было употреблено непосредственно перед объявлением.

В текущей версии языка ДССП-Т (v.3a) предусмотрен лишь минимальный набор префиксных операций над переменными и массивами: операции взятия значения (по имени), адреса (**‘**) и присвоения значения (**!**). А в базовом словаре языка для объявления переменных и в качестве элементов массивов предусмотрены лишь слова-типы, представляющие возможность работать с трайтами, двухтрайтовыми значениями и троичными словами: **TRYTE DTRYTE TWORD .**

В языке ДССП-Т помимо десятеричных и 16-ных можно задавать также троичные и 9-ные числа, характеризующие значения в так называемой троичной симметричной системе счисления. Каждое из таких чисел опознаётся по специальным префиксам:

{девятеричные числа:} **#**8765 **0t**qwer **0T**QWER **#**1234
{троичные числа:} **.**++0--**. .**++0-- **0n**++0-- **0N**++0--
{16-ные числа:} **$**13579ace **0x**1290abcd **0X**345678EF
{10-ные числа:} 56789 +24680 -13579

Большие и малые латинские буквы в записи числа не различаются, а для представления отрицательных девятеричных цифр (-1,-2,-3,-4) можно использовать как цифры (8765), так и буквы (qwer или QWER).

Для представления символьных констант используется общепринятая форма их записи в апострофах ‘X’ (как в языках Си и Паскаль).

Хотя числовые константы-литеры автоматически распознаются (по своим специфическим изображениям) как особые слова, в языке ДССП-Т можно объявить слово, представляющее константу, придав ему нужное имя:

{N} **VALUE** ИмяКнст {Пример:} 10 **VALUE** Ten

При таком объявлении в стеке задаётся то значение константы N(10), которое будет помещаться в арифметический стек при вызове слова-константы (Ten).

В языке ДССП-Т можно определить и слово, задающее константу-строку:

TEXT ИмяКС Строка
{Пример:} **TEXT** Red “Красный”

При вызове имени такой константы-строки в арифметический стек будут помещаться её параметры: адрес начала строки и длина (количество символов).

В языке ДССП-Т сохранены привычные обозначения слов для операций целочисленной арифметики, но выполняются они над числами, представленными в троичной симметричной системе счисления, так что результаты некоторых операций (например, сдвигов и делений с остатком), могут оказаться иными, чем в двоичной машине:

{деление с остатком:} 47 / 3 { [16,-1] вместо [15,2] }
{сдвиг вправо= /3 :} 47 SHR { [16] вместо [23] }
{сдвиг влево = *3 :} 13 SHL { [39] вместо [26] }

В языке ДССП-Т нет слов (**& &0 ‘+’ INV**), выполнявших привычные действия над отдельными битами машинного слова. Вместо них присутствуют слова, позволяющие выполнять разнообразные действия над тритами машинного слова:

TMIN TMAX TADD TMUL NEG .

Для доступа к одиночным и двойным трайтам, а также к словам троичной памяти в ДССП-Т предусмотрены операции чтения: **@T @TT @W** и записи: **!T !TT !W** . Причём для продвижения по памяти к очередному троичному слову надо добавлять 3 к значению адреса.

В языке ДССП-Т представлен полный ассортимент ДССП-команд условного вызова процедур, предусмотренных для организации ветвлений в ДССП-программе: **IF- IF0 IF+ BR- BR0 BR+ BRS BR ELSE** .

Чтобы обеспечить возможности вычисления логических условий, эквивалентные средствам построения логических выражений языков высокого уровня, в ДССП-Т имеются операции отношений для сравнения двух чисел: **< <= = <> >= >** , а также операции логических связок: **NOT AND OR** , вырабатывающие значение 1(ДА) или 0 (НЕТ).

Кроме того, в языке ДССП-Т предусмотрены также слова-операции для сравнения двух чисел и оценки знака числа: **CMP SGN** , предусматривающие троичные результаты: -1 (меньше), 0 (равно), +1 (больше), которые далее могут использоваться для вычисления сложного троичного условия путём применения потритных операций: **TMIN TMAX TADD TMUL NEG** .

В ходе создания и развития ДССП применялись две различные концепции организации циклов. Первая основывается на том, что условие цикла должно задаваться в самой команде цикла. Вторая предполагает, что выход из цикла может осуществляться с любого места тела цикла по специальным командам, а сама команда цикла должна лишь обеспечивать повторный вызов процедуры её тела.

Первая концепция обозначилась командой **DWON** уже в первой версии ДССП-НЦ [4], которая по сути унаследовала команду цикла DW машины “Сетунь-70”. Эта команда многократно вызывала стоящую следом процедуру, пока вершина стека оставалась ненулевой. Затем было предложено усовершенствовать эту команду цикла и выделить вычисление условия в отдельную процедуру, а значение условия удалять из стека после его проверки. Так в ДССП [10] появилась конструкция: **WHILE условие DO тело**, которая впоследствии (в ДССП/С [13]) превратилась в команду: **условие DW тело**. Вместе с ней были предложены команда **DO-** для организации цикла со счётчиком, а также команда бесконечного цикла **LOOP**, которые исполняют эти виды циклов эффективней, чем универсальная команда **DW**.

Вторая концепция организации циклов была предложена в версии ДССП-80 [5]. В ней появились команды для повторного вызова процедуры тела цикла заданное или бесконечное число раз: **DO RP** . Были предусмотрены специальные команды для экстренного выхода из цикла по условию или безусловно: **EX- EX0 EX+ EX** . Эти команды могли осуществлять требуемый выход из цикла с любой точки как самой процедуры тела цикла, так и вызванных ею процедур.

Обе концепции организации циклов применялись в ДССП на протяжении достаточно длительного периода и потому стали привычными для многих разработчиков ДССП-программ. Чтобы предоставить каждому из них желаемый способ организации циклов, в языке ДССП-Т обеспечены команды циклов обеих этих концепций: **DW DO- LOOP RP DO EX EX+ EX- EX0** .

В языке ДССП-Т поддерживается полный вариант (как в ДССП/С [13]) механизма работы с исключительными ситуациями. Предусмотрены команды: **ON EON RON** для установки трёх типов реакций (Notify, Escape, Retry) на заданную (следом по телу) ситуацию; на ситуацию, переданную через стек: **ON_ _ESCAPE _NOTIFY _RETRY** ; на произвольную ситуацию **ANY** ; а также команда для повторного возбуждения ситуации **RERAISE .**

Объявлять ситуацию можно так:

SITUATION ИмяСит КнРеак

При таком объявлении помимо имени ситуации задаётся также имя слова-процедуры конечной реакции, которая будет вызвана в случае, когда возбуждённая ситуация не будет обработана в ДССП-программе.

Программа, разработанная в ДССП-ТВМ, должна уметь общаться с внешними устройствами, которые ей может предоставить имеющаяся в настоящее время среда исполнения. В качестве минимального набора устройств ввода и вывода, доступных исполняемой ДССП-программе, в настоящее время ей обеспечивается диалоговое общение через консоль терминала и операции доступа к файлам операционной среды для последовательного чтения и записи.

При построении тел некоторых новых слов ДССП-программы может возникнуть потребность запрограммировать такие действия, выразить которые возможно

удастся лишь командами базовой машины (т.е. командами ТВМ). Поэтому язык ДССП-Т предоставляет возможность определить новое ДССП-слово, задав процедуру его исполнения на языке ассемблера ТВМ. Такое определение задаётся между служебными словами **:ASM** и **;ASM** .

Объявление нового типа как структуры начинается словом **STRUCT:** и завершается словом **;STRUCT** , а располагаемые между ними определения объектов данных (с помощью слов **VAR VCTR ARR**) объявляют в этом случае не самостоятельные переменные и массивы, а поля определяемой структуры:

STRUCT: ИмяТипаСтрукт **VAR** ИмяПоля1 . . . **VAR** ИмяПоляK **;STRUCT**	**STRUCT:** TPoint TRYTE VAR .X TRYTE VAR .Y **;STRUCT**

После объявления нового типа данных можно использовать его имя для объявления объектов (переменных и массивов) такого типа:

```
TPoint VAR P      TPoint 100 VCTR V {вектор точек}
```

Имеющиеся в языке ДССП-Т префиксные операции над переменными и массивами можно теперь применять и для доступа как ко всему объекту типа структуры целиком, так и к отдельным его полям:

P {‘P} 3 ! V {V[3]:=P}	P .X {P.X} 55 P ! .Y {P.Y:=55}
5 V ! P {P:=V[5]}	P .X 6 V ! .X {V[6].X:=P.X}
i V j ! V {V[j]:=V[i]}	8 V .Y P ! .Y {P.Y:=V[8].Y}

Ещё одним средством построения новых типов данных в языке ДССП-Т являются классы. Объявление нового типа как класса начинается со слова **CLASS:** и заканчивается словом **;CLASS** , а внутри такого объявления можно задавать не только поля (как и при объявлении структуры), но и методы класса:

CLASS: ИмяТипаКласса **VAR** ИмяПоля1 **. . .** **VAR** ИмяПоляK **METHOD** ИмяМетодаI **. . .** **METHOD** ИмяМетодаJ **;CLASS**	**CLASS:** TPoint **VAR** .X **VAR** .Y **METHOD** Init **METHOD** Show **METHOD=** Show Print 4 **METHOD#** >> **;CLASS**

Словом **METHOD** объявляется новая операция, которую можно будет совершать над объектом этого типа-класса помимо тех методов (префиксных операций **!** и **‘**), которые над ним уже предусмотрены изначально (как и над структурами). Слово **METHOD=** позволяет назначить новое имя методу, используя его прежнее имя, а слово **METHOD#** – задав номер метода. Так в примере для метода Show было назначено два новых имени: Print и >> .

Далее необходимо с помощью компилирующих слов **:M:** и **:M=** определить процедуры, которые будут назначены для исполнения новых методов над объектами объявленного класса:

```
TPoint :M: Init {P} 0 C2 ! .X 0 E2 ! .Y { } ;
: .Show {P} ‘(’ TOB C .X . D ‘,’ TOB .Y . D ‘)’ TOB {};
TPoint :M= Show .Show
```

Имя типа класса следует задать перед употреблением слова **:M:** (или **:M=**). Слово **:M:** создаёт тело новой процедуры, компилируя его как и слово **:** (до **;**), а слово **:M=** принимает имя уже существующей процедуры (.Show), чтобы назначать её в качестве исполнителя метода.

Внутри объявления нового типа как класса с помощью слова **SUBCLASS** можно задать, какой уже известный тип использовать для наследования:

```
CLASS: T3DPoint SUBCLASS TPoint VAR .Z ;CLASS
T3DPoint :M: Init {P} C Init AS TPoint 0 E2 ! .Z ;
T3DPoint :M: Show {P} {реализуем по-новому:}
  ‘[’ TOB C .X . D ‘,’ C .Y . D ‘,’ .Z . D ‘]’ TOB ;
```

Такое наследование означает, что у нового объявляемого типа изначально считаются уже заданными все поля и методы, которыми обладает наследуемый тип, после чего можно добавлять к его определению новые поля, задавать новые методы, а также переопределять исполнителей прежних методов (Show).

6. ДССП-библиотека

Возможности языка ДССП-Т не ограничиваются только словами базового словаря. Словарь компонуемой программы можно дополнить определениями слов, заготовленных в файлах стандартной библиотеки (ДССП-библиотеки). Для этого достаточно загрузить их определения из требуемого файла командой:

LOAD Имя_Файла

В настоящее время ДССП-библиотека состоит из двух частей: ДССП-ядра (файл kern.dsp) и диалогового командного монитора (файл dkmon.dsp).

Ядро ДССП складывается как бы из двух частей: первая часть содержит тела процедур ядра, составленные на языке ассемблера ТВМ, а вторая - определения слов ядра на языке ДССП-Т. Ассортимент этих слов весьма разнообразен и достаточно обширен. В настоящее время он охватывает:

- определения ряда констант;
- операции вывода на терминал управляющих символов, строк, чисел ;
- операции ввода с терминала строк;
- операции преобразования строки в число заданного формата;
- операции преобразования числа в строку заданного формата;
- операции сравнения и копирования строк;
- операции проверки символов;
- операции над нумерованными тритами троичного слова;
- определения новых типов данных **ACT** и **TSET** с операциями над ними;
- операции с файлами операционной среды ТВМ;
- а также разнообразные вспомогательные операции.

Диалоговый Командный МОНитор (далее ДКМОН) предназначен для исполнения скомпонованной ДССП-программы в режиме привычного диалога. Такой диалог предполагает, что в ответ на стандартное ДССП-приглашение (в виде символа * с новой строки) можно последовательно ввести в командной строке и исполнить любое слово скомпонованной программы (или несколько слов), просмотреть содержимое стека операндов, стека возвратов, ячеек памяти.

ДКМОН не обеспечивает всех тех функций, которыми обычно должен обладать внешний интерпретатор ДССП. Кроме слов, приготовленных в компонуемой программе, он умеет распознавать только числа (в этом случае в качестве действия он посылает в стек

значение принятого числа). Ни комментарии, ни строки вида **“…”**, ни текст для печати вида **.“…”** он не воспринимает. На любое непонятное слово он печатает сообщение об ошибке.

ДКМОН принимает слова из командной строки по одному. Встретив известное слово, ДКМОН сначала проверяет, можно ли это отдельное слово исполнить. Если нельзя, сообщает об ошибке, что это неисполнимое слово.

ДКМОНу известны только те слова, которые остались видимыми в словаре после компоновки программы. Из них исполнимыми считаются только те слова, которые могут выполняться в одиночку. В частности, ДКМОН не позволит исполнить введённую в командной строке последовательность слов, содержащую какие-либо управляющие команды: **IF+ BR- BRS DW DO RP** или префиксные операции **` !** над переменными, либо какие-то другие слова, требующие доступа к последующим словам по телу сшитого кода.

Модуль ДКМОНа можно использовать и просто как ДССП-библиотеку, загрузив в программу содержащиеся в нём определения слов (командой `LOAD`) без обязательного последующего запуска монитора при старте программы.

7. Интерпретатор ДССП/ТВМ

При разработке интерпретатора ДССП/ТВМ предполагалось обеспечить совместимость его входного языка с той текущей версией (v.3a) языка ДССП-Т, которая в настоящее время поддерживается кросс-компилятором ДССП-ТВМ. Такая совместимость означает, что интерпретатор должен уметь воспринимать и обрабатывать любую ДССП-программу, ранее созданную с помощью кросс-компилятора. И обеспечивать при этом точно такой же эффект исполнения данной программы при её запуске в среде интерпретатора, который проявлялся ранее при её прогоне в кросс-системе ДССП-ТВМ, где она могла запускаться на ТВМ автономно или вызываться в диалоговом командном мониторе.

ДССП-программа подаётся на вход интерпретатору как последовательность слов, которую он должен суметь проанализировать за один проход. Основной цикл ДССП-интерпретатора заключается в приёме очередного слова из входного потока и исполнении действия, соответствующего этому слову.

Интерпретатор поддерживает словарь известных ему слов. Каждому такому слову словаря сопоставлена процедура, которую интерпретатору надлежит вызывать, когда он встретит такое слово во входном потоке.

Помимо слов, накопленных в его словаре, интерпретатор умеет также распознавать особые слова-литеры, представляющие изображения чисел, символов, строк текста. Интерпретатор заранее знает, какое следует исполнить действие, встречая слово-литеру. Для числовой константы требуется занести в стек изображаемое ею значение, для символьной – её код, для обычной строки – поместить в стек её адрес и длину, а для строки, изображающей печатный текст, требуется напечатать этот текст.

Все слова, вызывающие исполнение тех или иных функций интерпретатора, можно условно разделить на несколько групп.

7.1. Средства построения тела программы

Основная задача, которую должен уметь выполнять интерпретатор языка ДССП, заключается в том, чтобы переводить анализируемый текст задаваемой во входном потоке ДССП-программы во внутреннее представление (в виде так называемого сшитого кода) с тем, чтобы запускать потом исполнение этого сформированного кода на троичной машине (ТВМ) при поддержке процедур ассемблерного ядра ДССП.

Предполагается, что программа, подаваемая во входном потоке интерпретатору, построена с использованием того же ассортимента компилирующих слов, которые предусмотрены в языке ДССП-Т и которые уже умеет понимать кросс-компилятор системы ДССП-ТВМ. Такие слова-компиляторы предназначены для объявления (определения) новых сущностей в программе. Каждое из них даёт указание интерпретатору создать новое слово в словаре и сформировать соответствующее ему тело во внутреннем представлении:

- слово **:** определяет новую процедуру в ДССП-программе;
- слово **VAR** определяет новую переменную;
- слово **VCTR** определяет одномерный массив;
- слово **ARR** определяет многомерный массив;
- слово **VALUE** определяет числовую константу;
- слово **TEXT** определяет константу-строку;
- слово **SITUATION** определяет исключительную ситуацию.

Обрабатывая эти слова, кросс-компилятор формировал тело компонуемой программы в виде файла ассемблерного кода. А интерпретатор, обрабатывая такие слова, должен формировать тело принимаемой программы во внутреннем представлении в особой области памяти, называемой областью тел.

Комплект процедур, обеспечивающих реализацию действий слов-компиляторов, как раз и позволяет интерпретатору осуществлять свою основную задачу. Этот комплект и составляет, по сути, основное ядро, образно говоря, сердцевину интерпретатора.

Встречая во входном потоке компилирующее слово из перечисленного выше набора, интерпретатор должен выполнить ряд характерных действий: 1) создать в словаре заголовок для вновь определяемого слова; 2) сформировать для него тело во внутреннем представлении и сохранить адрес этого тела в заголовке слова; 3) распределить память для объекта (переменной или массива) в особой области данных.

7.2. Слова управления входным потоком

В качестве основного способа взаимодействия с пользователем ДССП-интерпретатор предлагает диалоговый режим работы. Это значит, что входной поток слов, предназначенных для исполнения, интерпретатору можно задавать вводом строки с терминала, а получать от него ответы – в виде сообщений на терминал. Первоначально после своего запуска и всякий раз, когда интерпретатор входит в диалоговый режим, он вы-

даёт приглашение (обычно в виде символа * с новой строки), означающее, что он ожидает от пользователя ввода очередных слов-команд с терминала.

Обычно с терминала удобно задавать короткие последовательности команд, чтобы тут же в ходе диалога наблюдать эффект их исполнения в среде интерпретатора. Но вводить с терминала весь тот огромный поток слов, с помощью которого строится достаточно большая ДССП-программа, не представляется разумным. Такую программу лучше предварительно подготовить в виде отдельного текстового файла, а потом подавать целиком на обработку интерпретатору.

Чтобы воспринимать входной поток с файла, в интерпретаторе предусмотрена операция:

LOAD ИмяФайла {Пример:} **LOAD** MyProg

Эта операция запоминает параметры, характеризующие текущее состояние входного потока, и переключает интерпретатор на чтение входного потока слов из заданного файла. Предполагается, что, завершив чтение слов из этого файла, интерпретатор вернётся к обработке прежнего входного потока.

Интерпретатор воспринимает поток слов из файла, пока не встретит в нём специальное завершающее слово (**$$**) или не достигнет конца файла. Если при обработке очередного файла снова встретится слово-операция **LOAD**, то интерпретатор переключится на чтение нового потока из другого файла, завершив который, вернётся к продолжению обработки слов предыдущего файла. Т.е. интерпретатор допускает возможность вложенных вызовов для операции **LOAD**.

С помощью специальных слов: {} [] () можно указать интерпретатору, какой вид комментариев учитывать далее при обработке входного потока. Текущий установленный вид комментариев запоминается интерпретатором при переключении входного потока на другой файл и восстанавливается при возвращении на обработку прежнего потока.

7.3. Слова управления состоянием словаря

Словарь ДССП-программы является одной из важнейших её компонент. Почти все компилирующие операции добавляют в него новые слова. Но в языке ДССП-Т предусмотрены также и особые слова-операции для управления словарём:

- операция :: помечает следующее определяемое слово так, чтобы оно всегда оставалось в словаре;
- операция **CLEAR** очищает словарь, оставляя в нём лишь заголовки тех слов, которые специально были помечены (операцией **::**) как неудаляемые; заметим, что удаляя из словаря заголовки остальных слов, интерпретатор оставляет всё же их тела;
- операция **UNDEF** позволяет узнать, остались ли на данный момент в словаре неопределённые слова (т.е. слова, которые использовались в описании других слов, но сами описаны не были).

Диалоговый интерпретатор дополняет язык ДССП-Т новыми операциями, которые позволяют запоминать состояние словаря, достигнутое в ходе диалога, так, чтобы впоследствии можно было бы восстановить нужное состояние словаря (запомненное ранее) перед тем, как продолжать дальнейшую работу. Необходимость в таком восстановлении требуемого состояния словаря возникает, например, когда требуется повторно загрузить из файла (командой LOAD) ту программу, которая уже когда-то ранее загружалась в ходе текущего диалогового сеанса работы с интерпретатором. (Предполагается, что эта программа содержала ошибки, которые удалось обнаружить и исправить).

В языке ФОРТ [8] и в версии ДССП-НЦ [4], которая имела такое же простое линейное строение словаря, для восстановления состояния словаря использовалась операция **FORGET** с указанием имени обычного слова, которая удаляла из словаря это слово вместе со всеми словами, созданными после него.

Интерпретатор ДССП/ТВМ обеспечивает аналогичную возможность, но разрешает применять операцию **FORGET** не с обычным, а со специальным словом (назовём его маркером), которое создаётся в словаре (операцией **GROW**) именно с целью фиксации в его теле параметров, характеризующих текущее состояние словаря. Восстанавливая значения этих параметров из тела указанного слова, операция **FORGET** сумеет вернуть словарь в то состояние, которое было зафиксировано на момент создания этого слова.

Чтобы повторно загружаемая (командой **LOAD**) ДССП-программа могла корректно обрабатываться интерпретатором, обычно в её начало помещают последовательность команд **FORGET** и **GROW** с одним и тем же словом: **FORGET** $PROG **GROW** $PROG .

Для краткой записи таких действий предлагается команда: **PROGRAM** $PROG . Это позволяет интерпретатору корректно обрабатывать любые программы, разработанные ранее в системе ДССП-ТВМ.

7.4. Операция сохранения тела и словаря

Диалоговый интерпретатор обладает ещё одной важнейшей операцией, которая совсем не предусматривалась во входном языке кросс-компилятора. Но такая операция является характерной особенностью диалоговой среды разработки интерпретируемых программ. Она имеется во всех интерпретируемых версиях языков ФОРТ и ДССП и обычно называется **SAVE** :

SAVE ИмяФайла {Пример:} **SAVE** mydssp

Эта операция позволяет сохранить результаты диалогового сеанса общения с интерпретатором так, чтобы начать с ним следующий диалоговый сеанс в том состоянии, в котором был завершён предыдущий. Для этого в указанном файле сохраняются все компоненты, характеризующие состояние диалоговой среды ДССП-интерпретатора: программный код самого интерпретатора, область его системных переменных, словарь и область сформированных тел ДССП-программы.

Существенно, что такой файл обычно формируется в формате, пригодном для последующей его загрузки в память с целью непосредственного исполнения на той же машине или в той же операционной среде, в которой и запускался ранее диалоговый интерпретатор. Поэтому можно сказать, что операцией **SAVE** на самом деле создаётся файл новой версии интерпретатора, в словарь которой каждый пользователь может включить свои новые слова, созданные им в результате проведённого сеанса работы.

Интерпретатор ДССП/ТВМ, выполняя операцию сохранения, создаёт файл в формате троичного кода (nth-файл), который впоследствии может быть загружен для исполнения троичной машиной (ТВМ).

7.5. Операции для создания новых слов-компиляторов

При разработке процедур исполнения компилирующих слов в интерпретаторе были предусмотрены вспомогательные слова-операции, которые могут оказаться полезными для разработчиков ДССП-программ, если им потребуется создавать свои собственные слова-компиляторы.

Так, например, для работы с формируемым телом уже предусмотрен следующий набор вспомогательных операций:

- операция **,** заносит в формируемое тело троичное слово, взятое из стека;
- операция **,T** заносит в формируемое тело трайт, взятый из вершины стека;
- операция **,Str** копирует в формируемое тело строку из трайтов, адрес и длина которой задаются в стеке;
- операция **,WBlk** копирует в формируемое тело блок троичных слов, адрес и длина (количество слов) которого задаются в стеке;
- операция BA засылает в стек адрес текущей позиции формируемого тела.

Другой набор вспомогательных операций позволяет прочитать слово из входного потока и выполнить с ним необходимые действия в словаре:

- операция **GETWORD** считывает следующее слово из входного потока;
- операция **_WORD** выдаёт в стек адрес и длину прочитанного слова;
- операция **FINDWORD** производит поиск в словаре введённого ранее слова;
- операция **WORD?** считывает слово и осуществляет его поиск по словарю.
- операция **NewWord** добавляет в словарь заголовок нового слова с заданной строкой его имени и заданной для него командой (адресом тела) в стеке;
- операция **HEAD** считывает слово из входного потока и подготавливает его заголовок в словаре для формирования тела этого слова;
- операция **WORDCODE** считывает слово из входного потока, находит его в словаре и помещает в стек сопоставленную ему команду (адрес тела).

В качестве вспомогательных в словаре интерпретатора предусмотрены даже такие операции, которые осуществляют компиляцию тела как для отдельного слова (**COMPILEWORD**), так и для последовательности слов (**COMPILEBODY**), заканчивающейся **;** . А также ряд операций, помогающих выделить память (**MUSE**) и назначить адрес (**LOCATE**) для объявляемого объекта данных.

7.6. Слова для определения новых типов

Интерпретатор обеспечивает средства построения новых типов данных как структур и классов, появившиеся в новой версии (3а) языка ДССП-Т.

Встречая компилирующее слово **STRUCT:**, он создаёт слово-имя для нового типа данных и переключается в особый режим обработки слов, объявляющих данные, пока не встретит слово **;STRUCT**, завершающее определение нового типа. В этом особом режиме он создаёт для объявляемых объектов данных такие тела, которые позволяют им функционировать не в качестве самостоятельных переменных, а выступать в роли полей объявляемой структуры. Завершая обработку структуры, интерпретатор формирует в области тел дескриптор для нового типа данных.

Создание нового типа данных как класса интерпретатор осуществляет, обрабатывая целый комплект компилирующих слов: **CLASS: SUBCLASS ;CLASS METHOD METHOD# METHOD= :M: :M=** .

Но в этом случае помимо слова-имени для нового типа и слов-имён полей он создаёт ещё и слова с именами новых методов, объявляемых в определении класса, а также тела процедур, назначаемых в качестве их исполнителей.

Формируя дескриптор для нового типа-класса, интерпретатор сначала заимствует дескриптор наследуемого типа-класса, а затем расширяет его, добавляя позиции для новых методов. Впоследствии интерпретатор заполняет эти позиции ссылками на тела процедур-исполнителей методов, когда обрабатывает слова-компиляторы **:M: :M=** .

7.7. Усовершенствованные возможности командного монитора

Созданный интерпретатор не только поддерживает все функции по прогону построенной ДССП-программы, который умел ранее выполнять диалоговый командный монитор ДКМОН, но и обеспечивает для этого ряд новых возможностей, которыми ДКМОН не обладал.

При разработке интерпретатора были преодолены отмеченные ранее недостатки монитора. Так что интерпретатор умеет корректно распознавать и комментарии, и любые слова-литералы языка ДССП-Т, в том числе: большие числа, строки, печатный текст.

Существенно, что интерпретатор позволяет исполнять в режиме монитора так называемые Р-слова, т.е. особые слова, помеченные Р-флагом, которые требуют для своего исполнения доступа к следующему слову по телу сформированного сшитого кода. Для исполнения таких Р-слов в режиме монитора интерпретатор создаёт на некоторое время тело невидимой процедуры (без имени), а затем запускает её исполнение.

Большинство Р-слов требуют доступа лишь к одному последующему слову. Поэтому, встретив в командной строке слово, помеченное Р-флагом, интерпретатор помещает в тело такой временно создаваемой процедуры результат компиляции самого этого слова вместе со словом, следующим за ним, а также команду возврата, после чего запускает исполнение этого тела.

Однако, такой алгоритм обработки Р-слов не позволяет интерпретатору, вообще говоря, корректно исполнять (в режиме монитора) слова, требующие доступа более чем к одному слову по телу сшитого кода. Например, было невозможно исполнять последова-

тельности слов, содержащие управляющие команды ветвления и цикла: **BR+ BRS BR DW** .

Чтобы обеспечить исполнение всевозможных Р-слов в режиме монитора, интерпретатор предоставляет дополнительно слово-операцию с именем **;:** , которая создаёт на время процедуру без имени из последовательности слов, заканчивающейся, как и обычная процедура, словом **;** , а затем исполняет её.

8. Заключение

В работе представлены ДССП-ТВМ - кросс-система разработки программ для троичной машины (ТВМ) и диалоговый интерпретатор ДССП/ТВМ, способный функционировать на троичной машине в качестве её резидентной системы программирования.

Хотя ДССП-ТВМ сейчас функционирует в среде ОС Windows, её можно легко перенести в другую операционную среду (Linux), т.к. кросс-компилятор, ассемблер и имитатор ТВМ написаны на языке Си.

Опыт создания интерпретатора ДССП/ТВМ в кросс-системе разработки ДССП-ТВМ подтверждает, что эта кросс-система (как и созданный интерпретатор) могут служить основой для построения полноценного первичного программного оснащения аппаратно реализованной троичной машины той же архитектуры.

Литература

1. Н.П. Брусенцов, Рамиль Альварес Х. Троичные ЭВМ “Сетунь” и “Сетунь 70”. // Первая Международная конференция "Развитие вычислительной техники в России и странах бывшего СССР: история и перспективы" SORUCOM-2006 (3-7 июля 2006 г., г. Петрозаводск, Россия): Труды конференции в 2-х ч. Петрозаводск: Изд-во ПетрГУ, 2006. ч.1, с. 45-51.

2. С.А. Сидоров, Ю.С. Владимирова Троичная виртуальная машина // Программные системы и инструменты. Тематический сборник №12, М.: Изд-во факультета ВМиК МГУ,2011. С.46-55.

3. Н.П. Брусенцов, Рамиль Альварес Х. Структурированное программирование на малой цифровой машине. // Вычислительная техника и вопросы кибернетики. Вып. 15. М.: Изд-во МГУ, 1978, с.3-8.

4. Н.П. Брусенцов, Г.В Златкус, И.А Руднев. ДССП - диалоговая система структурированного программирования. // Программное оснащение микрокомпьютеров. М.: Изд-во МГУ, 1982, с.11-40.

5. Н.П. Брусенцов, В.Б. Захаров, И.А. Руднев, С.А.Сидоров Диалоговая система структурированного программирования ДССП-80. // Диалоговые микрокомпьютерные системы. М.: Изд-во МГУ, 1986, с.3-21.

6. Н.П. Брусенцов, В.Б. Захаров, С.А. Руднев, С.А.Сидоров, Н.А. Чанышев Развиваемый адаптивный язык РАЯ диалоговой системы программирования ДССП. М.: Изд-во Моск. Ун-та, 1987 г. – 80 с.

7. А.А. Бурцев, Рамиль Альварес Х. Кросс-система разработки программ на языке ДССП для троичной виртуальной машины // Программные системы и инструменты. Тематический сборник №12, М.: Изд-во факультета ВМиК МГУ,2011. С.183-193.

8. С.Н. Баранов, Н.Р. Ноздрунов Язык Форт и его реализации. – Л.: Машиностроение, 1988. – 157 с.

9. А.А.Бурцев, С.А. Сидоров История создания и развития ДССП: от "Сетуни-70" до троичной виртуальной машины. // Вторая Международная конференция "Развитие вычислительной техники и её программного обеспечения в России и странах бывшего СССР" SORUCOM-2011 (12-16 сентября 2011 г., г. Великий Новгород, Россия): Труды конференции. В.Новгород: Изд-во НовГУ, 2011. с. 83-88.

10. А.А.Бурцев Периферийный монитор – развитие архитектуры ввода/вывода ДССП. // Диалоговые микрокомпьютерные системы. М.: Изд-во МГУ, 1986, с.42-51.

11. А.В. Борисов Диалоговая система структурированного программирования в реальном времени – ДССП-РВ. // Диалоговые микрокомпьютерные системы. М.: Изд-во МГУ, 1986, с.51-62.

12. А.А.Бурцев, Д.В. Франтов, М.Н. Шумаков Разработка интерпретатора сшитого кода на языке Си. // Вопросы кибернетики. Сб. статей под ред. В.Б.Бетелина. М., 1999. с.64-76.

13. А.А.Бурцев ДССП – среда структурированной разработки программ как сложных систем. // Вторая Международная конференция “Системный анализ и информационные технологии” САИТ-2007 (10-14 сентября 2007 г., Обнинск, Россия): Труды конференции. Изд-во ЛКИ, 2007. т. 2. с.190-194.

14.Е.Н. Лякина, К.А Пшеничный, С.А. Сидоров, М.Н. Шумаков Система внутреннего программного оснащения DPROM. // Вопросы кибернетики. Сб. статей под ред. В.Б.Бетелина. М., 1999. с.77-86.

15. С.А. Сидоров, М.Н. Шумаков ДССП как открытая система. // Дискретные модели. Анализ, синтез и оптимизация. Спб.: СпбГУ, 1998. с.191-201.

16. А.А.Бурцев, Рамиль Альварес Х. Средства объектно-ориентированного программирования в ДССП. // Программные системы и инструменты. Тематический сборник №4, М.: Изд-во факультета ВМК МГУ, 2003. с.166-175.

DSSP for ternary virtual machine

A.A. Burtsev, M.A. Burtsev

Abstract. Ternary virtual machine (TVM) is a simulator of ternary computer, which architecture has two stacks (data stack and control stack) and machine commands for structured programming like “Setun-70”. TVM has been constructed as computer program in the computer laboratory leaded by Brusentsov N.P. (at the Moscow State University). Two variants of Dialogue System of Structured Programming (DSSP) for TVM are considered. DSSP-TVM let it possible to create DSSP-program for TVM by means of cross-compiler. DSSP/TVM is interpreter, which can run on TVM as its resident software. Programming language DSSP-T used in both systems is described as ternary version of language of the DSSP.