СОДЕРЖАНИЕ

Номер 5, 2022

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

ПРОГРАММНАЯ ИНЖЕНЕРИЯ, ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ ПРОГРАММ

БАЗЫ ДАННЫХ, ХРАНИЛИЩА ДАННЫХ

CONTENTS

No. 5, 2022

COMPUTER GRAPHICS AND VISUALIZATION

SOFTWARE ENGINEERING, TESTING AND VERIFICATION

DATABASES, DATA WAREHOUSES

EDN: BSSDGX
ПРОГРАММИРОВАНИЕ, 2022, № 5, с. 3–14

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.942

РАСПОЗНАВАНИЕ И КООРДИНАТНАЯ ПРИВЯЗКА АВТОНОМНОГО НЕОБИТАЕМОГО ПОДВОДНОГО АППАРАТА К ОБЪЕКТАМ ПО ВИДЕОПОТОКУ

© 2022 г. В. А. Бобков[a,*] (http://orcid.org/0000-0001-9722-5158), А. П. Кудряшов[a,] (http://orcid.org/0000-0003-3595-3648), А. В. Инзарцев[b,***] (http://orcid.org/0000-0002-1842-9951)**

[a] Институт автоматики и процессов управления ДВО РАН
690041 Приморский край, г. Владивосток, ул. Радио, д. 5, Россия
[b] Институт проблем морских технологий ДВО РАН
690091 Приморский край, г. Владивосток, ул. Суханова д. 5а, Россия
**E-mail: bobkov@dvo.ru*
***E-mail: alkud1981@mail.ru*
****E-mail: inzar@marine.febras.ru*

Поступила в редакцию 16.05.2022 г.
После доработки 24.05.2022 г.
Принята к публикации 01.06.2022 г.

Статья посвящена актуальной проблеме инспекции объектов подводной промышленной инфраструктуры с использованием автономного необитаемого подводного аппарата (АНПА). Выполнение инспекционной миссии требует высокой точности координации АНПА относительно объектов, что не всегда обеспечивается штатными гидроакустическими средствами. Поэтому предлагается подход, основанный на обработке стереоизображений, который может обеспечить необходимую в этих условиях субметровую точность координации. В рамках единого подхода, основанного на использовании априорно формируемой точечной модели объекта, и применении критерия структурной когерентности, предлагается и анализируется ряд методов, которые решают задачу распознавания подводных объектов и координатной привязки к ним АНПА. В качестве геометрической модели объекта рассматривается его представление из характерных точек, определяющее пространственную структуру объекта. Наряду со стандартным способом формирования модели объекта по предварительным измерениям предлагается и новый способ, основанный на применении автоматизированной методики формирования модели по снимкам в режиме офлайн. Предлагаемые методы распознавания объектов базируются на сопоставлении формируемых 3D-облаков точек с моделями объектов с применением критерия структурной когерентности. 3D-облака точек формируются из фиксируемых камерой стереоизображений методом визуальной навигации. Генерация и сопоставление характерных точек-особенностей осуществляется с помощью детекторов SURF и Harris Corner Detector. Для повышения достоверности идентификации объектов предлагается использование совместной обработки исходных изображений с их векторизованной формой. Использование получаемых векторизацией отрезков существенно увеличивает число идентифицируемых характерных точек при сравнении 3D-облака точек с моделью. На основе идентифицированных точек объекта вычисляется матрица координатной привязки АНПА к объекту. Приведены оценки сравнительной эффективности рассматриваемых методов. В вычислительных экспериментах с модельными сценами использовалась виртуальная геометрическая модель распределенного подводного добычного комплекса. Эксперименты с реальными данными проводились в лабораторных условиях с использованием стереокамеры Karmin2 (Nerian's 3D Stereo Camera, baseline 25 cm).

Ключевые слова: автономный необитаемый подводный аппарат (АНПА), подводная промышленная инфраструктура, инспекция объектов, подводный добычный комплекс, стереоизображения, распознавание подводных объектов, сопоставление особенностей, координатная привязка, визуальная одометрия

DOI: 10.31857/S0132347422050028

1. ВВЕДЕНИЕ

В последнее время в связи с ростом подводной промышленной инфраструктуры (газовые и нефтяные трубопроводы, подводные добычные комплексы, кабели связи и передачи электроэнергии) возросла и актуальность проблемы автома-

тической инспекции подводных объектов с использованием автономных необитаемых подводных аппаратов-роботов (АНПА). Некоторый обзор по проблеме Subsea Infrastructure Inspection и обоснование необходимости разработки новых технологий на базе использования АНПА можно найти в [1–3]. Целесообразность применения АНПА определяется их более высокой эффективностью в сравнении с традиционно применяемыми привязными телеуправляемыми необитаемыми подводными аппаратами (ТНПА), особенно если речь идёт о работах, выполняемых на больших глубинах или подо льдом. Выполнение инспекционной миссии требует высокой точности координации АНПА относительно объектов при выполнении операций осмотра. Штатные средства бортовой навигации АНПА (доплеровские лаги, навигационно-пилотажные датчики, инерциальные и гидроакустические навигационные системы) для этих целей не подходят, поскольку координаты обследуемых объектов в общем случае известны с точностью, гораздо ниже требуемой для инспекции. Это оборудование может быть использовано для первоначального (грубого) сближения с объектом, но для координации АНПА, когда требуется высокоточное приближение подводного аппарата к объектам (включая посадку аппарата на объект), целесообразно использовать другие средства. В качестве таковых могут быть использованы звуковизоры или оптические изображения, получаемые видеокамерой АНПА.

Визуальная навигация потенциально позволяет получать необходимую в этих условиях субметровую и даже сантиметровую точность. Актуальность применения визуальной навигации для подводных сцен определяется также и тем, что во многих случаях карты морского дна отсутствуют. Кроме того, под водой недоступны широко применяемые навигационные системы GPS/ГЛОНАСС. Поэтому на разработку технологий автоматической инспекции подводных объектов на базе АНПА с обработкой видеоизображений и с интеграцией других сенсорных данных направлены усилия многих разработчиков. Ключевой задачей при этом в решении проблемы навигации АНПА в координатном пространстве инспектируемого подводного объекта является распознавание этого объекта. Поскольку расчет траектории движения АНПА делается методом визуальной навигации (МВН) [4], другой задачей является преодоление погрешности в точности навигации, свойственной визуальной одометрии при длительном перемещении АНПА. Также для реализуемой системы навигации на базе видеоинформации необходима высокая вычислительная производительность для обеспечения режима реального времени.

В проводимых в мире исследованиях по проблеме инспекции подводных структур затрагиваются разные аспекты, связанные с надежностью распознавания объектов и повышением точности навигации. В [5–9] представлены примеры разработки разных технологий навигации для АНПА, включая использование комбинаций разного типа сенсоров с различной прикладной направленностью. Однако в этих работах не решается задача расчета траектории АНПА непосредственно в координатном пространстве локального подводного объекта или комплекса объектов, и в неполной мере используется потенциал 3D-данных, получаемых обработкой видеоизображений.

В [10] предложен 3D-метод получения полной карты неизвестной среды с помощью АНПА с наполнением данных от разных датчиков, включая оптическое покрытие (optical coverage). Эффективность работы системы достигается за счет использования октодерева (octree).

В [11, 12] авторами настоящей статьи была предложена технология координатной привязки АНПА к подводному объекту, основанная на использовании предварительно заданной пространственной точечной модели объекта и применении критерия структурной когерентности при сравнении трехмерных точек объекта с моделью. А в другой работе, ими была рассмотрена интеграция двух сенсоров – стерео видеокамеры и линейного лазера для трекинга подводного трубопровода.

В [13] описана система с 3D-подводным лазерным сканером на АНПА для решения задачи simultaneous localization and mapping (SLAM).

В [14] представлен обзор и сравнение глобальных дескрипторов для целей распознавания 3D-объектов, когда 3D-модель объекта доступна априори.

В настоящей статье предлагаются и анализируются методы решения задачи распознавания и координатной привязки АНПА к инспектируемым подводным объектам по стерео видеопотоку. Методы разработаны в рамках подхода, основанного на использовании априорно формируемой точечной модели объекта и применении критерия структурной когерентности. Отличительной особенностью предлагаемых решений является совместная обработка исходных изображений с векторизованной формой и применение автоматизированной методики формирования геометрической модели объекта.

Последующий текст статьи организован следующим образом. Во втором разделе делается постановка задачи и описывается общий подход к ее решению. Третий раздел посвящен решению задачи распознавания и координатной привязки к объектам, включая методики, модели и алгоритмы. В том числе, описан точечный алгоритм распознавания с использованием модели объекта, построенной по снимкам, и показан расчет матрицы координатной привязки АНПА к объекту. В чет-

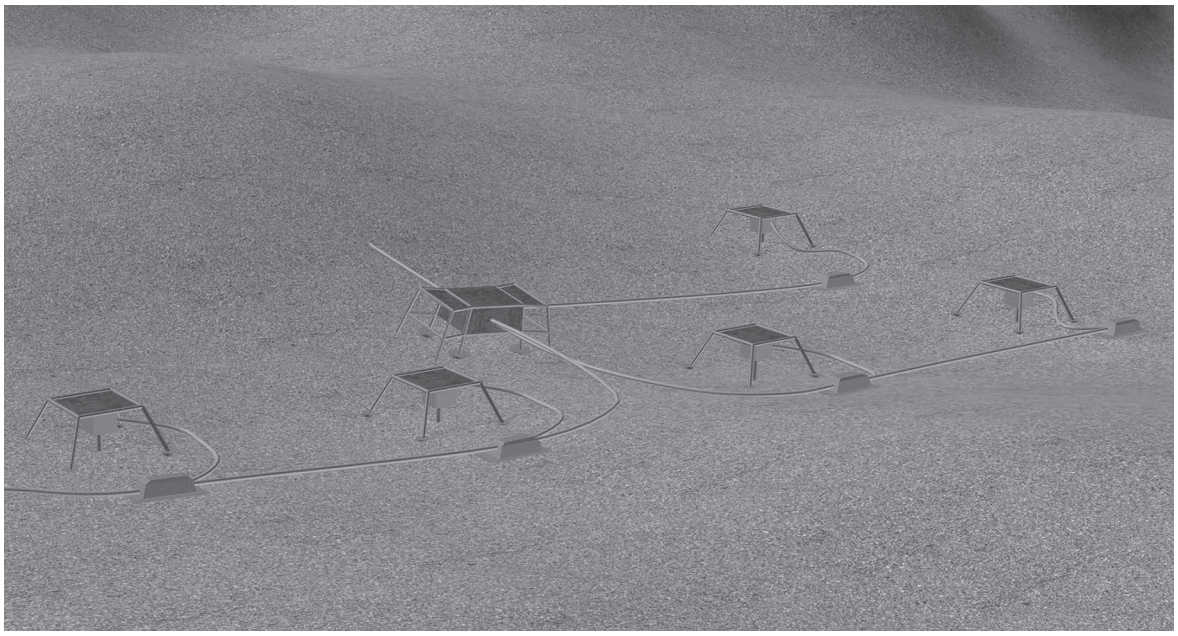

Рис. 1. Подводный добывающий комплекс (виртуальная сцена).

вертом разделе обсуждаются результаты вычислительных экспериментов. В заключение кратко сформулированы результаты выполненной работы и обозначен дальнейший план работы по усовершенствованию полученных результатов.

2. ПОСТАНОВКА ЗАДАЧИ И ОПИСАНИЕ ПОДХОДА

Предполагается, что инспекция подводного добычного комплекса (ПДК) выполняется с использованием АНПА. Аппарат оснащен штатными средствами навигации, включая гидроакустическую навигационную систему, доплеровский лаг и комплекс навигационно-пилотажных датчиков. АНПА также оборудован цифровой стерео видеокамерой высокого разрешения и мини-бортовым компьютером с соответствующим программным обеспечением. Стереокамера фиксирует видеопоток в процессе движения подводного аппарата по запланированной в инспекционной миссии траектории. Для выполнения заданий инспекционной миссии (операции фотографирования элементов объектов ПДК, возможная посадка АНПА на объекты и т.п.) требуется субметровая точность координации АНПА относительно объектов ПДК. Поскольку штатные средства навигации не всегда обеспечивают высокоточную навигацию в условиях небольших расстояний до объектов, целесообразно применять навигацию по видеоинформации, которая потенциально может обеспечить прецизионную координацию АНПА по отношению к подводным объектам. Штатные средства в этом случае используются только для организации выхода АНПА к объекту инспекции. Инспектируемый комплекс ПДК состоит из нескольких объектов, распределенных на ограниченной площади (схематичный пример ПДК (виртуальная сцена) показан на рис. 1).

Предполагается, что каждый объект может быть задан геометрической моделью, которая представляется 3D-характерными геометрическими элементами (ХЭ), определяющими пространственную структуру объекта. Координаты ХЭ заданы в системе координат (СК) модели объекта, а каждая СК объекта определена в СК модели ПДК. Чтобы обеспечить прецизионное движение АНПА в СК ПДК в режиме реального времени по видеоинформации, необходимо решать три взаимосвязанные задачи:

а) распознавание объекта при наличии предварительно заданной модели объекта (и модели ПДК в целом). Модель формируется по паспортным данным, определяющим ХЭ объектов, или по снимкам автоматизированным способом с участием оператора. Распознавание и идентификация объекта осуществляется с помощью алгоритмов, реализующих сопоставление с моделью объекта данных, получаемых из 2D-снимков. В свою очередь, сопоставленные с моделью 3D-элементы необходимы для вычисления матрицы преобразования координат, связывающей СК модели с СК АНПА в текущей позиции траектории АНПА;

б) задача координатной привязки состоит в непосредственном вычислении матрицы связи СК АНПА с СК модели ПДК. Наличие такой связи

позволяет планировать инспекционную траекторию АНПА в координатном пространстве ПДК;

в) траектория движения АНПА в пространстве ПДК вычисляется методом МВН с учетом вычисленной матрицы связи. При длительном перемещении АНПА, необходимы те или иные способы/методы уменьшения накапливаемой погрешности МВН.

Известны два основных подхода к описанию объекта – точечный подход, когда построение 3D точечной модели объекта осуществляется на основе сопоставления точечных особенностей на исходных 2D-снимках, и подход на основе сопоставления 2D-линий, принадлежащих объектам. В первом случае обрабатываются исходные снимки в растровом формате, во втором случае используется векторная форма, получаемая после предварительной фильтрации и векторизации исходных снимков. Первый подход носит универсальный характер, поскольку он не использует специфики геометрической формы объекта. Второй подход целесообразен применительно к объектам искусственного происхождения, для которых, как правило, характерно наличие прямолинейных отрезков. В частности, это относится к подводным сценам, когда речь идет об объектах подводной промышленной инфраструктуры.

В предлагаемом подходе в качестве ХЭ рассматриваются характерные точки (ХТ) объекта (в основном угловые точки) и отрезки линий, характерные для искусственных объектов. При обработке стереоснимков используется известный детектор SURF (Speeded Up Robust Features) и детектор Харриса (извлечение угловых точек). Для получения 3D-координат точек применяется метод триангуляции лучей. Для решения задачи распознавания применяются оригинальные алгоритмы, основанные на критерии структурной когерентности (одинаковое взаиморасположение точек модели объекта и точек, видимых камерой АНПА) и на сопоставлении отрезков. Расчет траектории осуществляется, как было отмечено выше, методом МВН в каждой расчетной позиции траектории. Для снижения погрешности, порождаемой методом МВН, может быть использована ранее описанная авторами в своих работах виртуальная сеть координатной привязки.

3. РАСПОЗНАВАНИЕ ОБЪЕКТОВ

Получаемые камерой АНПА стереоизображения обрабатываются, как на уровне 2D, так и на уровне 3D, соответствующими методами/алгоритмами с целью получить 3D-множество ХЭ, состоящее из точек – особенностей (применяются детектор Харриса и детектор SURF) и прямолинейных отрезков (на векторизованных снимках). Распознавание объектов ПДК основывается на сравнении полученного множества с предварительно (до выполнения инспекции) сформированной моделью ПДК. Модель объекта может формироваться из различных элементов и разными способами.

Рассматриваются два приведенных ниже априорных способа формирования модели объекта:

а) элементами модели являются видимые пространственные точки, отрезки объекта (или другие геометрические элементы), характеризующие геометрическую форму объекта. Их относительное пространственное расположение задается в системе координат, связанной с объектом. Для расчета координат используются данные, получаемые из существующего описания объекта, или непосредственным измерением. Следует отметить, что при достаточно простой геометрической модели недостатком этого способа является то, что процесс выбора (человеком) ХТ на этапе формирования модели ПДК и процесс выявления особенностей на снимках (с помощью детектора Харриса) являются относительно независимыми друг от друга. Это приводит, как показали эксперименты, к возможным ситуациям с небольшим количеством идентифицированных точек модели ПДК, что негативно сказывается на точности привязки;

б) другой способ формирования модели объекта – использовать изображения объекта ПДК, предварительно полученные фотосъемкой с АНПА. Предполагается, что после установки ПДК на морское дно АНПА может получить снимки объектов ПДК с разных ракурсов, проходя по обзорной траектории. В этом случае можно применить автоматизированную методику формирования модели с участием оператора. Оператор, просматривая изображения в режиме офлайн, непосредственно на изображении указывает, какие элементы будут принадлежать модели объекта. Преимущество этого способа в сравнении с вышеуказанным является то, что в модель включаются оператором те элементы, для которых вероятность сопоставления на изображениях рабочей траектории выше. Это способствует сопоставлению большего числа элементов.

Возможны различные варианты алгоритмов распознавания объектов в зависимости от выбора ХЭ модели объекта и способов формирования модели.

3.1. Алгоритм распознавания по характерным точкам. Модель построена по измерениям пространственных точек, определяющим структуру объекта

3.1.1. Алгоритм с применением критерия структурной когерентности. В ранее выполненной авторами работе [11] для описания модели использу-

ются характерные точки (ХТ) $P\{P_1, ..., P_N\}$, задающие пространственную структуру объекта, и множество измеренных расстояний $D\{d_{i,j}\}$ между ХТ, где d_{ij} – расстояние между ХТ P_i и P_j. ХТ задаются в СК, связанной с объектом ПДК. Алгоритм поиска видимых камерой 3D-точек, соответствующих ХТ-м модели, основан на применении критерия структурной когерентности. Для сравнения используются отдельные выборки из двух множеств – множества модели $P\{P_1, ..., P_N\}$ и множества $C^{\text{cloud}}(C_1, ..., C_M\}$, полученного из снимков. Поэтому во множестве модели проверяются все возможные выборки точек из N по n, а во множестве C^{cloud} – выборки из M по n. Число комбинаций определяется числом сочетаний $C_N^n = N!/n!\,(N-n)!$. Число возможных выборок на множестве C определяется, соответственно, числом сочетаний $C_N^n = M!/n!\,(M-n)!$.

С каждой выборкой $S_k^{model}\{P_1, ..., P_n\}$ однозначно связано множество расстояний $D_k^{model}\{d_{1,1}^{model}, ..., d_{n-1,n}^{model}\}$ между точками данной выборки. Здесь $d_{i,j}^{model}$ расстояние между точками P_i и P_j. Аналогичным образом, с каждой выборкой $S_l^{cloud}\{C_1, ..., C_n\}$ однозначно связано множество расстояний $D_l^{cloud}\{d_{1,1}^{cloud}, ..., d_{n-1,n}^{cloud}\}$ между точками данной выборки. Заметим, что $d_{i,j}^{model} = d_{j,i}^{model}$, аналогично для точек C_i и C_j $d_{i,j}^{cloud} = d_{j,i}^{cloud}$.

Число элементов в каждом из множеств D_k^{model}, D_l^{cloud} равно $n(n-1)/2$. Задача идентификации n ХТ ПДК была сведена к последовательному решению двух задач:

1. Нахождение такой выборки S_l^{cloud} из множества C, для которой существует выборка из модели S_k^{model}, удовлетворяющая условию: $D_l^{cloud}\{d_{1,1}^{cloud}, ..., d_{n-1,n}^{cloud}\} = D_k^{model}\{d_{1,1}^{model}, ..., d_{n-1,n}^{model}\}$. Равенство множеств здесь означает, что они состоят из одних и тех же элементов, но не обязательно в одном и том же порядке.

2. Идентификация ХТ в выборке S_l^{cloud} означает, что для каждой точки C_m из найденной выборки S_l^{cloud} требуется найти соответствующую ей точку в модели.

Первая задача решалась перебором выборок из множеств P и C с проверкой указанного условия $D_l^{cloud} = D_k^{model}$. Вторая задача решалась анализом индексов в формируемой таблице расстояний, куда отображаются все $d_{i,j}^{cloud}$ выборки S_l^{cloud}.

Соответствие анализируемой точки C_i некоторой ХТ модели P_j устанавливается сравнением двух пар расстояний, в которых анализируемая точка фигурирует:

Если $d_{i,k}^{cloud} = d_{l,j}^{model}$, а $d_{i,n}^{cloud} = d_{j,m}^{model}$, то соответствующая точка модели определяется как пересечение $\{P_l, P_j\} \cap \{P_j, P_m\} = P_j$.

Эксперименты подтвердили, что метод работоспособен, однако требует больших вычислительных затрат, связанных с комбинаторным характером используемых выборок. Другой недостаток метода – возможное небольшое количество сопоставлений при использовании детектора Харриса, что, в конечном счете, негативно сказывается на точности навигации.

3.1.2. Модификация метода за счет использования известных абсолютных координат нескольких ХТ объекта. Для уменьшения вычислительных затрат авторы модифицировали этот метод за счет существенного сокращения перебора выборок [12]. В этой работе исходили из предположения, что известны абсолютные координаты ограниченного числа ХТ модели (≥3) (на практике координаты в некоторых случаях могут быть зафиксированы при установке объекта на морское дно). В этом случае координаты ХТ модели могут быть пересчитаны в абсолютные координаты. Точки множества C тоже могут быть пересчитаны в абсолютные координаты с помощью МВН (с определенной, свойственной методу, погрешностью). А это, в свою очередь, позволяет рассматривать при сопоставлении для каждой ХТ модели только те выборки из множества C, которые содержат точки в окрестности анализируемой точки P_k модели. Это существенно сокращает перебор точек при формировании выборок. Эксперименты подтвердили вычислительную эффективность метода, позволяющую применять его в режиме реального времени.

3.1.3. Алгоритм распознавания по характерным точкам. Модель построена по снимкам. *Алгоритм распознавания с априорно заданной моделью объекта по изображениям.* При применении второго способа формирования модели объекта (указание оператором ХЭ объекта на изображениях предварительной/обзорной траектории АНПА) предлагается приведенный ниже универсальный алгоритм, предполагающий более высокую степень надежности распознавания в сравнении с рассмотренными выше.

1) *Формирование модели*

Предлагается автоматизированная методика формирования геометрической модели объекта. В режиме офлайн, основываясь на просмотре/анализе стерео видеопотока “предварительной/обзорной” траектории, оператор фиксирует один из видов. Для него сохраняется стереопара

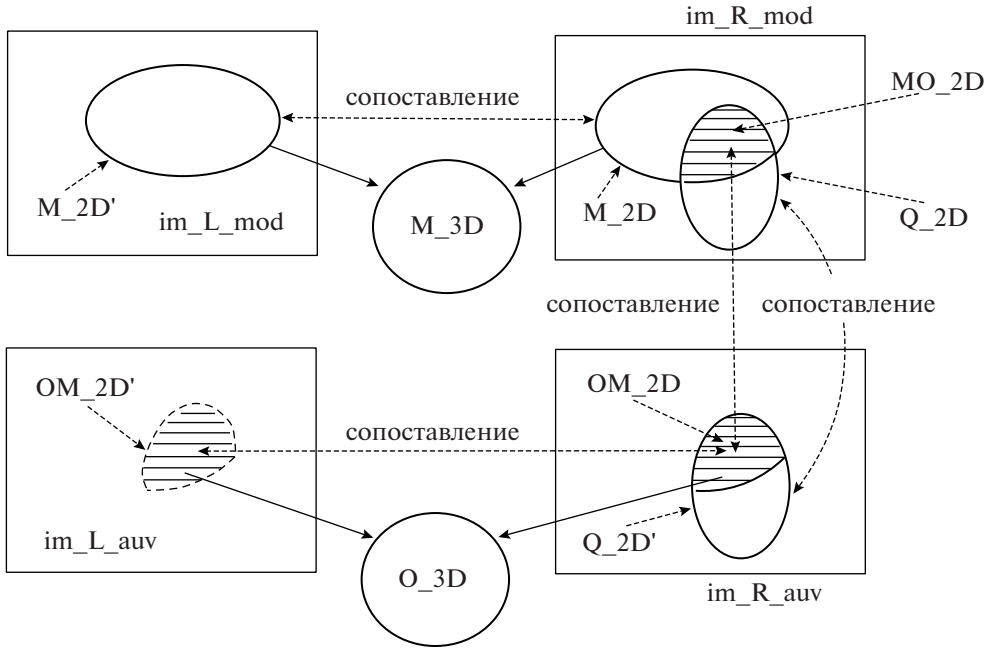

Рис. 2. Построение множества O_3D-пространственных точек, сопоставленных модели объекта. На стереопаре изображений im_L_mod и im_R_mod определена модель объекта. Стереопара изображений im_L_ auv и im_R_ auv получена из позиции рабочей траектории АНПА. Множества M_2D' и M_2D сопоставлены стереопаре модели. Множества Q_2D и Q_2D' сопоставлены на изображениях двух стереопар. Множество MO_2D сопоставлено множеству OM_2D. Множества OM_2D и OM_2D' сопоставлены на стереопаре позиции рабочей траектории АНПА.

снимков (для последующего сравнения со снимками, полученными при движении АНПА по рабочей инспекционной траектории). И для него генерируется и сопоставляется с помощью детектора SURF множество точек на левом и правом снимке.

Оператор указывает в этом множестве точки, принадлежащие объекту, которые будут представлять модель объекта. Обозначим это подмножество M_2D (рис. 2). Также оператор указывает точки инспекционного интереса с внесением семантической информации.

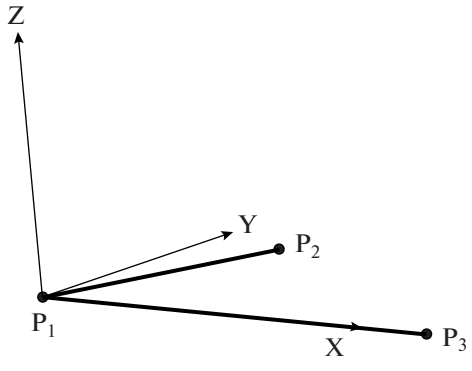

Рис. 3. Построение системы координат объекта по угловой точке P_1 и двум примыкающим к ней ребрам-отрезкам P_1P_3 и P_1P_2. Точка P_1 – начало СК, ось Z строится как нормаль к плоскости, образуемой ребрами-отрезками, (направлена вверх), а ось X берется по направлению отрезка P_1P_3 (по правилу правой СК). Ось Y берется как нормаль к плоскости ZX.

Для сопоставленных точек множества M_2D строится 3D представление (триангуляция лучей), т.е. множество M_3D (в СК камеры).

На указанной оператором тройке 3D-точек из множества M_3D строится $СК_{объект}$. В качестве начала $СК_{объект}$ фиксируется одна из угловых точек, а по двум примыкающим к ней ребрам строятся орты Z, X, Y (см. рис. 3).

В эту $СК_{объект}$ переводятся (из СК камеры) координаты всех точек M_3D. Таким образом, множество пространственных точек M_3D с зафиксированной, привязанной к объекту $СК_{объект}$ является моделью объекта, которая используется при распознавании объекта уже непосредственно при движении АНПА по рабочей траектории.

2) *Привязка АНПА к СК объекта из позиции рабочей траектории с использованием построенной модели*

Описываемый ниже алгоритм распознавания объектов на снимках позиции рабочей траектории основывается на сопоставлении с 2D моделью объекта точечных особенностей, извлекаемых из снимков позиции рабочей траектории. Заметим, что извлечение точечных особенностей на снимках выполняется с помощью детектора SURF, применяемого в методе визуальной навигации для расчета движения АНПА по рабочей траекто-

рии. Поэтому непосредственная работа алгоритма распознавания объекта заключается в использовании этих точечных особенностей для получения, в конечном счете, 3D-множества точек, сопоставленного с 3D-моделью объекта. Далее представлена пошаговая схема алгоритма с учетом вышесказанного.

• Генерация на снимках полученной в позиции рабочей траектории стереопары с помощью детектора SURF множества точечных особенностей (метод визуальной навигации).

• Сопоставление с помощью SURF правого (левого) изображения этой стереопары с правым (левым) изображением модельной стереопары (метод визуальной навигации) – результат множество Q_2D, сопоставленное с Q_2D' (см. рис. 2). Из полученного множества Q_2D (сопоставленного с Q_2D') выделяем подмножество MO_2D = = M_2D ∩ Q_2D, содержащие точки модели, которые сопоставлены точкам на снимке позиции рабочей траектории (подмножество OM_2D).

• Для этих точек строим 3D-представление по стереопаре и получаем множество O_3D, пространственных точек в СК камеры позиции рабочей траектории. Эти точки предположительно принадлежат объекту и соответствуют точкам модели объекта.

• Поскольку точки множества O_3D-координированы и в СК камеры позиции рабочей траектории и в $СК_{объект}$, то по ним вычисляется матрица преобразований координат, связывающая СК камеры и $СК_{объект}$.

В расширенном варианте алгоритма при формировании модели рассматриваются и обрабатываются несколько смежных видов – тогда множество O_3D будет объединять точки объекта с разных видов. Объединение видов с приведением к одной СК обеспечивается применением МВН. Наличие нескольких видов повышает вероятность распознавания точек объекта во время движения АНПА по рабочей траектории. Также отметим, что процедура координатной привязки АНПА к объекту активируется только при попадании в область (окрестность – параллелепипед) потенциальной видимости объекта камерой.

3.2. Модификация метода за счет выделения отрезков на изображениях стереопары

Как было выше отмечено, использование детектора Харриса может приводить к небольшому числу точек на изображении, сопоставленных с моделью объекта. Для преодоления этого недостатка предлагается другая методика обработки снимков стереопары на этапе получения 3D-облака точек для последующего сопоставления с точечной моделью объекта. Методика основана на предварительной векторизации снимков и выделении прямолинейных отрезков, концевые точки которых и будут добавлены в 3D-облако. После формирования, объединенного указанным образом 3D-облака, работает любой из выше рассмотренных методов распознавания объектов.

Предлагается два алгоритма 3D-реконструкции отрезков по снимкам. Первый из них имеет “универсальный” характер, но требует больше вычислительных затрат в сравнении со вторым алгоритмом. Второй алгоритм требует меньше вычислительных затрат за счет меньшей “универсальности”. Рассмотрим каждый из них. Первые два шага общие для обоих алгоритмов:

– векторизация изображений стереопары;

– выделение “длинных” отрезков на векторизованном изображении.

3.2.1. Первый алгоритм 3D-реконструкции отрезков. Соответствие двух отрезков на левом и правом изображении означает, что обе камеры (левая и правая) “видят” один и тот же 3D-отрезок, но с разных ракурсов. Обнаружение соответствия основывается на сопоставлении его концевых точек. Сопоставление точек, в свою очередь, можно выполнить с помощью сравнения их дескрипторов на левом и правом снимке. Однако сложность определения соответствия заключается в том, что, в общем случае, на левом и правом изображении 2D-образы пространственного отрезка, видимого левой и правой камерой различаются между собой в силу двух обстоятельств. Во-первых, левая и правая камеры разнесены в пространстве, и, во-вторых, процесс векторизации изображений может давать разные длины 2D-образов отрезка на изображениях стереопары. Различие в положении левой и правой камеры в пространстве может приводить к различным вариантам видимости 3D-отрезка стереокамерой в пространстве сцены: отрезок полностью видим левой и правой камерой; отрезок видим одной камерой, но не видим другой (или камеры видят не перекрывающиеся фрагменты отрезка); камеры видят разные фрагменты отрезка, но у них есть общая видимая часть. Рассмотрим подробнее последний случай, как более сложный в алгоритмическом плане. При наличии общей видимой части пространственного отрезка, концевые точки его 2D-образов на левом и правом снимке, в общем случае, не соответствуют друг другу. Факт обнаружения общей видимой части отрезка (путем сопоставления на левом и правом снимке), будет говорить о соответствии рассматриваемых 2D-отрезков. Поиск соответствия 2D-отрезков на изображениях осложняется также необходимостью трудоемкого перебора отрезков на правом снимке при анализе каждого отрезка на левом снимке. Однако задача упрощается за счет использования эпиполярных ограничений для концевых точек отрезков (поскольку известна калибровка стереокамеры). Если построить на правом снимке эпиполярные ли-

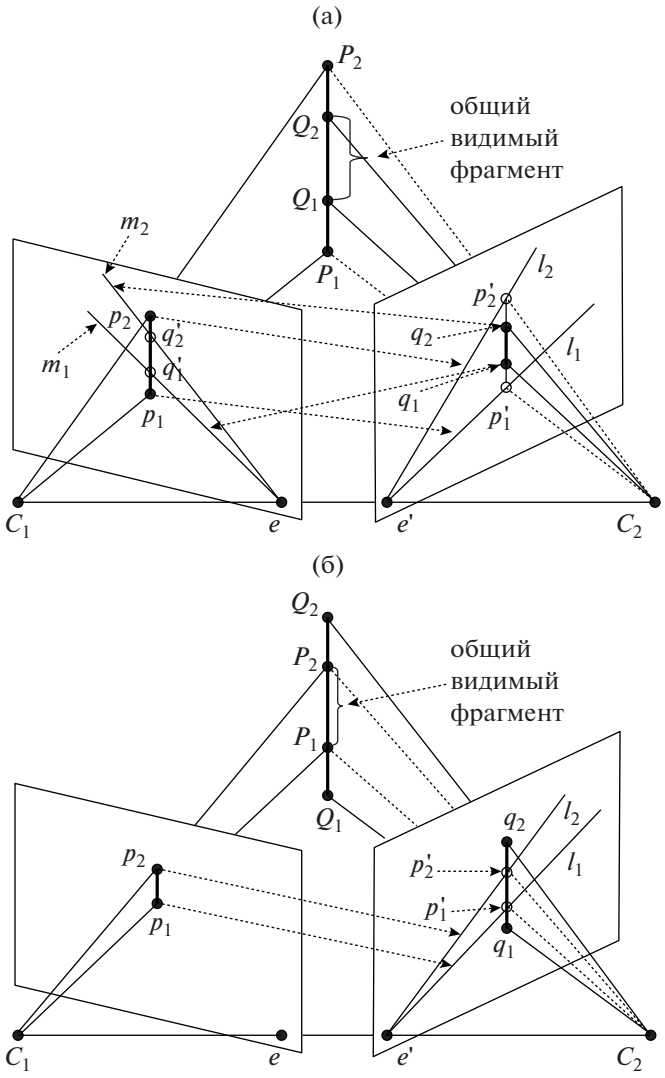

Рис. 4а. Сопоставление отрезков p_1p_2 и q_1q_2. Правый отрезок находится внутри сектора, образуемого эпиполярными линиями l_1и l_2. C_1,C_2 – центры проекций левой и правой камер; e и e' – эпиполюсы на левом и правом изображениях; P_1P_2 – 3D-отрезок, видимый из C_1; p_1p_2 – образ отрезка P_1P_2 на левом изображении; Q_1Q_2 – видимый из C_2 фрагмент отрезка P_1P_2; q_1q_2 – образ фрагмента Q_1Q_2 на правом изображении; l_1, l_2 – эпиполярные линии на правом изображении для точек p_1, p_2; p_1',p_2' – образы точек p_1, p_2 на правом изображении; m_1, m_2 – эпиполярные линии на левом изображении для точек q_1, q_2; $q_1'q_2'$ – образы точек q_1, q_2 на левом изображении. Отрезок q_1q_2 на правом изображении сопоставлен отрезку $q_1'q_2'$ на левом изображении.

Рис. 4б. Сопоставление отрезков p_1p_2 и q_1q_2. Средняя часть правого отрезка находится внутри, а его концевые точки вне сектора, образуемого эпиполярными линиями l_1и l_2. C_1,C_2 – центры проекций левой и правой камеры; e и e' – эпиполюсы на левом и правом изображении; P_1P_2 – 3D-отрезок, видимый из C_1; p_1p_2 – образ отрезка P_1P_2 на левом изображении; Q_1Q_2 – видимый из C_2 отрезок; q_1q_2 – образ отрезка Q_1Q_2 на правом изображении; l_1, l_2 – эпиполярные линии на правом изображении для точек p_1, p_2; p_1',p_2' – образы точек p_1, p_2 на правом изображении. Отрезок $p_1'p_2'$ на правом изображении сопоставлен отрезку p_1p_2 на левом изображении.

нии для концевых точек “левого” отрезка, то возможно: а) сократить перебор отрезков на правом снимке за счет фильтрации отрезков, не отвечающих ограничению; и б) определить соответствие для концевых точек общего видимого фрагмента отрезка.

Алгоритм сопоставления отрезков заключается в следующем. Выполняется векторизация изображения на левом и правом снимке. Для концевых точек каждого анализируемого на левом снимке отрезка строятся эпиполярные линии на

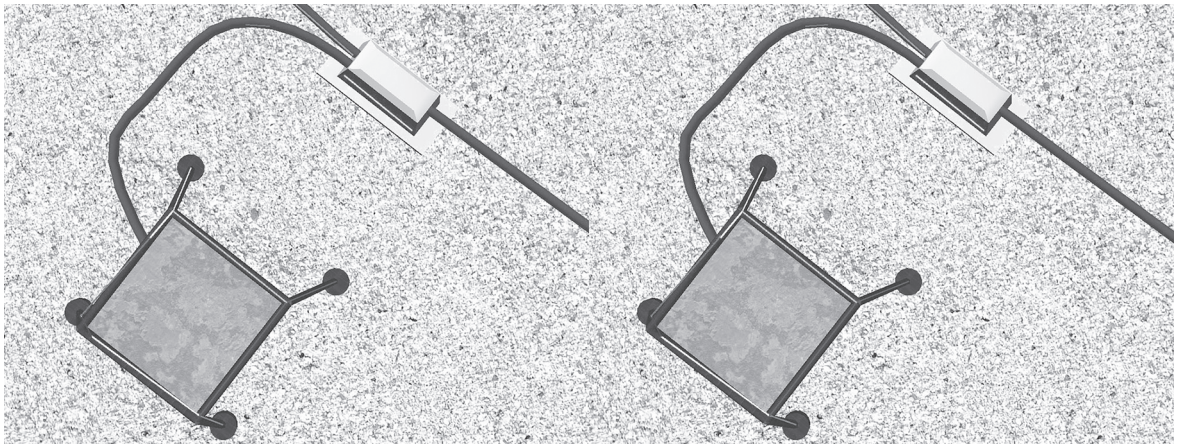

Рис. 5. Выделение отрезков на изображениях стереопары с последующим восстановлением 3D-координат их концевых точек методом триангуляции.

правом снимке. Они образуют сектор, внутри которого должен находиться сопоставленный отрезок. С учетом эпиполярного ограничения будем анализировать на правом снимке только те отрезки, которые полностью или частично попадают в сектор. Анализ соответствия пары {“левый” отрезок” ↔ “правый” отрезок}, основанный на использовании эпиполярных ограничений, как слева направо, так и справа налево, позволяет выделить общую видимую часть отрезка и установить соответствие концевых точек его левого и правого образа. Конечным критерием сопоставления является сравнение дескрипторов концевых точек. Если найдено соответствие для обеих концевых точек, то считаем, что отрезки сопоставлены.

На рис. 4 а, б иллюстрируются два варианта (из 4-х возможных) видимости пространственного отрезка на левом и правом изображениях, когда концевые точки образов отрезка на левом и правом изображениях не соответствуют друг другу.

Вариант 1. Отрезок на правом изображении лежит внутри сектора (обе концевые точки отрезка находятся внутри сектора) (рис. 4а):

– вычисляем точки пересечения p_1' и p_2' эпилиний l_1 и l_2 с линией отрезка q_1q_2;

– если концевые точки отрезка q_1q_2 находятся внутри сектора, то для концевых точек q_1 и q_2 отрезка на правом изображении строим эпиполярные линии m_1 и m_2 на левом изображении – вычисляем точки пересечения q_1' и q_2' эпилиний с отрезком p_1p_2;

– получаем дескрипторы для точек q_1, q_2, q_1', q_2';

– сопоставляем точки $q_1 \leftrightarrow q_1'$ и $q_2 \leftrightarrow q_2'$, сравнивая дескрипторы;

– если сопоставления нет, то переход к анализу следующего отрезка внутри сектора на правом изображении.

Вариант 2. Правый отрезок пересекается обеими эпилиниями (средняя часть правого отрезка находится внутри, а его концевые точки вне сектора) (рис. 4б):

– вычисляем точки пересечения p_1' и p_2' эпилиний l_1 и l_2 с линией отрезка q_1q_2;

– если концевые точки отрезка q_1q_2 находятся вне сектора, то получаем дескрипторы для точек p_1, p_2, p_1', p_2';

– сопоставляем точки $p_1 \leftrightarrow p_1'$ и $p_2 \leftrightarrow p_2'$, сравнивая дескрипторы;

– если сопоставления нет, то переход к анализу следующего отрезка внутри сектора на правом изображении.

Аналогичным образом осуществляется обработка всех других возможных ситуаций. В каждом из всех возможных случаев выделение образов “общего” фрагмента отрезка Q1Q2 (т.е. видимого на левом и правом изображениях) позволяет вычислить 3D-координаты этого фрагмента. В результате сопоставления всех отрезков на левом и правом снимках получаем два сопоставленных множества отрезков. По сопоставленным отрезкам строим множество E пространственных отрезков, видимых стереокамерой (используется стандартная процедура триангуляции лучей для построения 3D-концевых точек отрезков). На следующем этапе обработки данных это множество 3D-отрезков используется для построения видимых камерой 3D-угловых точек.

3.2.2. Второй алгоритм 3D-реконструкции отрезков. Пошаговая схема обработки данных:

– сопоставление отрезков на левом и правом изображении (используется оптимизированный

перебор анализируемых отрезков) с применением следующих критериев (рис. 5):

а) совпадение наклонов отрезков (с определенной погрешностью);

б) удовлетворение концевых точек отрезков эпиполярным ограничениям (поскольку известна калибровка камеры);

в) учет смещения образа отрезка на правом изображении по отношению к образу отрезка на левом изображении с учетом направления движения камеры.

Все удовлетворяющие этим критериям отрезки достраиваются до максимально видимой длины (с учетом эпиполярных линий);

г) оценка текстурной близости посредством вычисления коэффициента кросс-корреляции (полоса пикселей слева и справа от отрезка);

– восстановление 3D-координат концов полученных отрезков (методом триангуляции лучей).

Эксперименты подтвердили ожидаемое предположение, что при использовании концевых точек отрезков осуществляется более надежное распознавание объектов за счет того, что концевые точки отрезков хорошо сопоставляются с угловыми точками модели.

3.3. Координатная привязка АНПА к СК подводного объекта

После получения некоторого подмножества I идентифицированных во множестве C точек объекта решается задача вычисления матрицы преобразования координат из СК АНПА в СК объекта. В качестве СК объекта используется СК существующей модели объекта. Здесь следует заметить, что в подмножестве I может находиться та угловая точка, на которой построена СК модели, но может и не находиться. Поэтому рассмотрим два соответствующих варианта вычисления искомой матрицы преобразования координат из СК АНПА в СК объекта.

1. В первом случае задача решается просто. В подмножестве I ищем точку, сопоставленную точку модели, на которой построена СК модели объекта. На этой найденной точке строим $СК^{объект}$ по обозначенному ранее правилу. Таким образом, задача заключается в вычислении матрицы преобразования координат, связывающей $СК^{АНПА}$ и $СК^{объект}$. Она решается следующим образом:

Пусть e1, e2, e3 – орты $СК^{объект}$, а **r** – вектор начала $СК^{объект}$, заданные в $СК^{АНПА}$. Тогда:

$$H_{СК^{объект},К^{АНПА}} = \begin{vmatrix} e1x & e1y & e1z & 0 \\ e2x & e2y & e2z & 0 \\ e3x & e3y & e3z & 0 \\ rx & ry & rz & 1 \end{vmatrix}$$

2. Во втором варианте задача усложняется тем, что в подмножестве I не нашлась угловая точка, на которой построена СК модели. Поэтому, сначала строится промежуточная система координат $СК^{С\_промежут}$ на одной из точек подмножества I, и $СК^{модель\_промежут}$ на сопоставленной ей точке модели. Систему координат $СК^{С\_промежут}$ можно связать с $СК^{АНПА}$ аналогичным образом:

$$H_{СК^{С\_промежут},СК^{АНПА}} = \begin{vmatrix} e1x & e1y & e1z & 0 \\ e2x & e2y & e2z & 0 \\ e3x & e3y & e3z & 0 \\ rx & ry & rz & 1 \end{vmatrix},$$

где e1, e2, e3 – орты $СК^{С\_промежут}$, а **r** – вектор начала $СК^{С\_промежут}$, заданные в $СК^{АНПА}$.

Аналогичным образом свяжем $СК^{модель\_промежут}$ с системой координат модели объекта $СК^{модель\_объекта}$.

Получим матрицу

$$H_{СК^{модель\_промежут},СК^{модель\_объекта}} = \begin{vmatrix} e1x & e1y & e1z & 0 \\ e2x & e2y & e2z & 0 \\ e3x & e3y & e3z & 0 \\ rx & ry & rz & 1 \end{vmatrix},$$

где e1, e2, e3 – орты $СК^{модель\_промежут}$, а **r** – вектор начала $СК^{модель\_промежут}$, заданные в $СК^{модель\_объекта}$.

Тогда искомое преобразование $H_{СК\ АНПА,\ СКмодель\_объекта}$ будет:

$$H_{СК\ АНПА,\ СКмодель\_объекта} =$$
$$= (H_{СК^{С\_промежут},СК^{АНПА}})^{-1} \cdot H_{СК^{модель\_промежут},СК^{модель\_объекта}}$$

Полученная матрица геометрического преобразования координат позволяет теперь вычислять траекторию АНПА в координатном пространстве объекта.

4. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ

Для оценки эффективности рассматриваемых в статье методов был проведен ряд экспериментов с виртуальными сценами и с реальными данными. Эксперименты проводились на ПК: AMD Ryzen 7 3700X 8-Core 417 Processor 3.60 GHz // 32 Gb // AMD Radeon 5600XT. Реальные данные были получены с помощью стереокамеры Karmin2 (Nerian's 3D-Stereo Camera, baseline 25 cm) в лабораторных условиях. Оценивалась точность и надежность координатной привязки АНПА к объектам сцены, а также вычислительная трудоемкость применяемых алгоритмов. Результаты сравнительной эффективности методов приведены в табл. 1. Из анализа полученных результатов можно сделать следующие выводы:

Табл. 1. Сравнение методов

	Погрешность координации АНПА	Надежность привязки (число выполненных привязок АНПА к объекту по отношению к числу возможных привязок)	Вычислительные затраты в среднем на одну координатную привязку
Метод 1 * Вирт сцена (модель из 50 ХТ), расстояние до объектов – 6–8 м	От 9 см до 40 см	6 из 54	0.2 сек на кадр
Метод 2 ** Вирт сцена (модель из 50 ХТ), расстояние до объектов – < 5 м	до 6 см	4 из 18	0.02 сек на кадр
Метод 2 Реальная сцена (модель из 8 ХТ) (съемка камерой Nerian Karmin2, расстояние до объектов –1–1.5 м)	до 3 см	2 из 15	0.03 сек на кадр
Метод 3 *** Вирт сцена (модель из 50 ХТ), расстояние до объектов – < 5 м	от 8 до 20 см	12 из 18	0.07 сек на кадр
Метод 4 **** Вирт сцена (модель из 50 ХТ), расстояние до объектов – < 5 м	От 7 до 12 см	14 из 18	0.1 сек на кадр

* – метод с применением критерия структурной когерентности. Используется модель объекта, построенная по ХТ (см. 3.1.1).

** – метод с использованием абсолютных координат нескольких ХТ. Используется модель объекта, построенная по ХТ (см. 3.1.2).

*** – использование концевых точек отрезков в качестве дополнения к множеству точек Харриса. Используется модель объекта, построенная по ХТ (см. 3.2).

**** – Используется модель объекта, построенная по ХТ, указанным оператором на снимках (см. 3.1.3).

1. Метод с использованием в модели ограниченного числа ХТ с известными абсолютными координатами позволяет принципиальным образом сократить вычислительные затраты за счет ограниченного перебора комбинаций при формировании проверяемых выборок.

2. Использование концевых точек отрезков, извлекаемых из векторной формы снимков, повышает вероятность распознавания объектов за счет увеличения количества сопоставляемых точек 3D-облака с моделью. Это приводит к повышению надежности привязки (в сравнении с методом, использующим только угловые точки Харриса).

3. Применение методики автоматизированного формирования модели по снимкам имеет два преимущества: а) не требуются априорные измерения координат ХТ на реальном объекте при формировании модели (или/и использование документации);

б) модель формируется из таких точек на снимках, для которых вероятность сопоставления точек 3D-облака с моделью выше, чем при использовании только точек Харриса. Поэтому для метода 4 надежность распознавания объектов и, соответственно, надежность координатной привязки АНПА к объекту оказалось даже выше, чем для метода 3. Однако вычислительные затраты несколько выше даже при условии, что не учитывались затраты на работу детектора SURF.

5. ЗАКЛЮЧЕНИЕ

В работе выполнен сравнительный анализ ряда методов/алгоритмов, предложенных авторами в рамках единого подхода, основанного на использовании априорно формируемой точечной модели объекта и применении критерия структурной когерентности. В том числе, предложены алгоритмы с использованием векторизованной формы исходных стереоизображений, и автоматизированная методика формирования геометрической модели объекта по снимкам. Проведенные вычислительные эксперименты с виртуальными сценами, и частично с реальными данными, подтвердили правомерность подхода. В дальнейшем планируется разработка методов распознавания с непосредственным использованием векторизованной формы изображений при формировании модели объекта. Также будут проведены эксперименты с реальными подводными сценами для по-

лучения более полной оценки эффективности предлагаемых методов.

ФИНАНСИРОВАНИЕ

Исследование выполнено за счет Гранта Российского научного фонда № 22-11-00032, https://rscf.ru/project/22-11-00032/.

СПИСОК ЛИТЕРАТУРЫ

1. *Mai C., Pedersen S., Hansen L., Jepsen K., Yang Zh.* Subsea Infrastructure Inspection: A Review Study. 6th International Conference on Underwater System Technology: Theory and Applications. 2016. https://doi.org/10.1109/USYS.2016.7893928.
2. *Manley J.E., Halpin S., Radford N., Ondler M.* Aquanaut: A New Tool for Subsea Inspection and Intervention. Proceedings of OCEANS 2018 MTS/IEEE Conference. 22-25 Oct. 2018. Charleston. USA. https://doi.org/10.1109/OCEANS.2018.8604508.
3. *Zhang Yu., Zheng M., An Ch., Seo J.K.* A review of the integrity management of subsea production systems: inspection and monitoring methods. Ships and Offshore Structures. 2019. V. 14. Issue 8. P. 1–15. https://doi.org/10.1080/17445302.2019.1565071
4. *Nister D., Naroditsky O., Bergen J.* Visual odometry. in: Proc. Int. Conf. Computer Vision and Pattern Recognition. 2004. P. 652–659. https://doi.org/10.1109/CVPR.2004.1315094
5. *Jacobi M.* Autonomous inspection of underwater structures. Robotics and Autonomous Systems. 2015. V. 67. Issue C. P. 80–86. https://doi.org/10.1134/S2075108718010042
6. *Hou Y.Z., Pan Sh., Dai Ch.* A Novel Underwater Simultaneous Localization and Mapping Online Algorithm Based on Neural Network. International Journal of Geo-Information. 2019. V. 9. № 1. P. 5. https://doi.org/10.3390/ijgi9010005
7. *Santos M., Guilherme Z., Otávio R.P., Drews-Jr P., Botelho S.* Underwater place recognition using forward-looking sonar images: A topological approach. Journal of Field Robotics. 2019. P. 355–369. https://doi.org/10.1002/rob.21822
8. *Jongdae Jung, Ji-Hong Li, Hyun-Taek Choi, Hyun Myung.* Localization of AUVs using visual information of underwater structures and artificial landmarks. Intel Serv Robotics. 2017. V. 10. P. 67–76. https://doi.org/10.1007/s11370-016-0210-9
9. *Bao J., Li D., Qiao X., Rauschenbach T.* Integrated navigation for autonomous underwater vehicles in aquaculture: A review. Information Processing in Agriculture. 2020. V. 7. Issue 1. P. 139–151. https://doi.org/10.1016/j.inpa.2019.04.003
10. *Vidal E., Palomeras N., Istenič K., Gracias N., Carreras M.* Multisensor online 3D-view planning for autonomous underwater exploration. J Field Robotics. 2020. P. 1123–1147. https://doi.org/10.1002/rob.21951
11. *Bobkov V.A., Kudryashov A.P., Inzartsev A.V.* Technology of AUV High-Precision Referencing to Inspected Object. Gyroscopy and Navigation. 2019. V 10. № 4. P. 322–329. https://doi.org/10.1134/S2075108719040060
12. *Bobkov V., Kudryashov A., Inzartsev A.* Method for the Coordination of Referencing of Autonomous Underwater Vehicles to Man-Made Objects Using Stereo Images. J. Mar. Sci. Eng. 2021. V. 9. P. 1038. https://doi.org/10.3390/jmse9091038
13. *Palomer A., Ridao P., Ribas D.* Inspection of an underwater structure using point-cloud SLAM with an AUV and a laser scanner. J Field Robotics. 2019. P.1333–1344. https://doi.org/10.1002/rob.21907
14. *Himri K., Ridao P., Gracias N.* 3D-Object Recognition Based on Point Clouds in Underwater Environment with Global De-594 scriptors: A Survey. Sensors. 2019. V. 19. 4451. https://doi.org/10.3390/s19204451

EDN: CDRPIH
ПРОГРАММИРОВАНИЕ, 2022, № 5, с. 15–26

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.932

ВЫБОР ФУНКЦИИ АКТИВАЦИИ В СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЯХ ПРИ ПОВТОРНОЙ ИДЕНТИФИКАЦИИ ЛЮДЕЙ В СИСТЕМАХ ВИДЕОНАБЛЮДЕНИЯ

© 2022 г. Х. Чен[a,b,*] (http://orcid.org/0000-0003-4229-4505),
С. Игнатьева[c] (http://orcid.org/0000-0002-9780-5731),
Р. Богуш[c,] (http://orcid.org/0000-0002-6609-5810),**
С. Абламейко[d,e] (http://orcid.org/0000-0001-9404-1206)

[a] *Zhejiang Shuren University Shuren Str., 8, Hangzhou, China*
[b] *International Science and Technology Cooperation Base of Zhejiang Province: Remote Sensing Image Processing and Application Hangzhou, 310000, China*
[c] *Полоцкий государственный университет ул. Блохина, д. 29, г. Новополоцк, 211446, Республика Беларусь*
[d] *Белорусский государственный университет п-т Независимости, д. 4, г. Минск, 220030, Республика Беларусь*
[e] *Объединенный институт проблем информатики НАН Беларуси ул. Сурганова, 6, г. Минск, 220012, Республика Беларусь*
**E-mail: eric.hf.chen@hotmail.com*
***E-mail: bogushr@mail.ru*

Поступила в редакцию 25.02.2022 г.
После доработки 18.04.2022 г.
Принята к публикации 17.05.2022 г.

Статья посвящена исследованию влияния функции активации в сверточных нейронных сетях на точность реидентификации людей на изображениях, полученных с различных видеокамер распределенных систем видеонаблюдения. Проведены исследования для наиболее применяемых функций активации при обнаружении объектов на изображениях, таких как ReLU, Leaky-ReLU, PReLU, RReLU, ELU, SELU, GELU, Swish, Mish по критериям: точность реидентификации человека с использованием метрик Rank1, Rank5, Rank10, mAP и время, затраченное на обучение модели. В качестве экстрактора признаков использовались архитектуры ResNet-50, DenseNet-121 и DarkNet-53. Экспериментальные исследования выполнены на открытых наборах данных Market1501 и PolReID. Оценка точности реидентификации выполняется после трехкратного повторения обучения и тестирования при использовании разных функций активации, архитектур нейронных сетей и наборов данных с использованием усреднения полученных значений метрик.

DOI: 10.31857/S013234742205003X

1. ВВЕДЕНИЕ

Повторная идентификация (реидентификация) людей является важной задачей компьютерного зрения и позволяет обнаруживать одного и того же человека в разных местах и в разное время на изображениях, полученных с видеокамер распределенных систем видеонаблюдения. Однако реидентификация требует учета многих мешающих факторов включая окклюзии, разный уровень освещенности, различное разрешение видеокамер, отличающийся внешний вид одного и того же человека из-за различных ракурсов его сьемки, схожесть внешних признаков между разными людьми. Повторная идентификация схожа с классификацией, однако существенное отличие заключается в отсутствии определенного количества классов объектов, так как количество разных людей может быть различным.

В настоящее время наиболее эффективными инструментами для решения данной задачи являются сверточные нейронные сети (СНС), которые используются для извлечения признаков на изображении каждого человека. В [1] для алгоритма реидентификации небольшого количества людей в помещениях используют модифицированную быстродействующую СНС на основе ResNet-34. В [2] для реидентификации применяются СНС ResNet-34 и ResNet-50 и показано, что Res-

Net-50 требует больших временных затрат, но обеспечивает увеличение точности.

СНС DenseNet-121 [3] характеризуется наличием соединений между слоями, при которых карты признаков всех предыдущих слоев используются в качестве входных для всех последующих в блоке. Кроме этого, карты признаков не суммируются от слоя к слою, что характерно для ResNet, а конкатенируются. Такие особенности данной СНС позволили повысить эффективность реидентификации по сравнению с ResNet-50 [4, 5].

Архитектура DarkNet-53 была предложена в качестве базовой СНС для извлечения признаков в алгоритме обнаружения объектов YOLOv3. При оценке в метриках top-1 и top-5 на базе данных ImageNet в [6] показано, что данная СНС по точности соизмерима с ResNet-152, а по времени обработки быстрее более чем в два раза. Таким образом, DarkNet-53 является перспективной для повторной идентификации людей.

Известно, что эффективность дескриптора объекта определяется архитектурой СНС и выборкой данных, на которой она была обучена. Увеличение глубины нейронных сетей позволяет повысить точность работы, однако приводит к тому, что на этапе обучения при обратном распространении ошибки могут происходить такие явления, как взрывные или исчезающие градиенты. Взрывные градиенты приводят к проблеме, возникающей при накоплении больших градиентов ошибок, за счет чего веса сети обновляются слишком быстро. Это делает модель нестабильной и неспособной к обучению на тренировочных данных. Исчезающие градиенты представляют обратную проблему, которая так же приводит к невозможности эффективного обучения модели. Обучение СНС методом обратного распространения ошибки предполагает корректировку весов в зависимости от функции потерь и скорости обучения. При этом веса умножаются на производную от функции потерь, и чем глубже сеть, тем меньше эта величина. При значении градиента близком к нулю веса СНС вообще перестанут обновляться. Существуют разные способы решения этих проблем, одним из которых является поиск функции активации (ФА) для решения прикладной задачи. Целью данной статьи является исследование и выбор наиболее эффективной ФА для использования в сверточных нейронных сетях при повторной идентификации людей в системах видеонаблюдения.

2. ФУНКЦИИ АКТИВАЦИИ В СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЯХ

В настоящее время существуют различные ФА, и каждая характеризуется своими недостатками и преимуществами, которые влияют на эффективность их применения для решения прикладной задачи. Однако нет универсальной ФА, подходящей для множества разных задач компьютерного зрения.

С учетом схожести классификации и повторной идентификации объектов в системах видеонаблюдения для ФА СНС при реидентификации можно выделить основные требования:

1. Нелинейность. ФА должна быть нелинейной функцией, т.к. для линейной функции производная является константой и при обучении СНС изменения весов не будет зависеть от входных значений. Кроме этого, комбинация линейных функций так же является линейной, за счет чего утрачиваются преимущества использования многослойных СНС.

2. Схожесть ФА на положительном участке области определения с функцией идентичности. Такой подход позволяет ускорить обучение при небольших значениях градиентов.

3. Нулевой участок на отрицательной части области определения. Такие ФА обладают эффектом “прореживания” нейронов, что позволяет облегчить сеть и ускорить обучение. Ненулевой участок на отрицательной части области определения позволяет снизить количество потерянной информации, при этом сохраняет эффект “прореживания”.

4. Значение производной на положительной части области определения должно быть близко к единице. Производная ФА на всей или на большей части области определения для положительных входных значений нейронов должна принимать значения равные или больше 1, чтобы при обратном распространении ошибки по сети не происходило затухание градиентов из-за постоянного умножения на величину, близкую к 0.

Среди указанных обязательным требованием является нелинейность ФА. Схожесть ФА с функцией идентичности обусловлена ее наклоном в 45° к оси x, что позволяет передавать значения активированных нейронов без значительных изменений. Использование кусочно-заданных функций, где на положительной области определения используется функция идентичности, позволяет снизить вычислительные затраты, и, как следствие, ускорить обучение. Третье требование так же направлено на ускорение обучения за счет добавления эффекта “прореживания”. Решить проблему затухающих градиентов может позволить четвертое требование. Полное соответствие ФА всем требованиям позволит повысить эффективность работы СНС.

Выбор ФА для конкретной прикладной задачи предполагает проведение экспериментальных исследований, которые позволят определить наиболее эффективную по точности и временным затратам.

Одной из наиболее распространенных в настоящее время ФА является ReLU (выпрямленной линейной единицы), графически представленная на рис. 1а [7]. ReLU представляет собой кусочно-заданную функцию:

$$\varphi(x) = \begin{cases} x, & x > 0 \\ 0, & x \le 0 \end{cases}, \qquad (2.1)$$

где x – входное значение нейрона.

Основное преимущество заключается в низкой вычислительной сложности, которая обусловлена тем, что при прямом проходе по сети положительные значения сохраняются, а отрицательные приравниваются к нулю. При обратном проходе вычислительные затраты так же минимальны, и заключаются только в сравнении значений нейронов с нулем: при отрицательных значениях производная равна нулю, при положительных равна единице. Несмотря на сходство с линейной функцией в области положительных значений, ReLU нелинейная, а значит перспективна для реидентификации людей. Однако значения производной на положительной части области определения могут привести к такой проблеме, как взрывные градиенты. Отрицательная часть области определения функции приводит к существованию эффекта, называемого “прореживание нейронов”. Суть его заключается в том, что нейроны, которые не были активированы изначально никогда не смогут быть активированными, и к ним будут добавляться те нейроны, на входы которых поступали отрицательные значения. Алгоритм градиентного спуска не сможет настраивать веса таких нейронов.

Решением проблемы “прореженных” нейронов может быть ФА Leaky-ReLU [8], или ReLU с утечкой:

$$\varphi(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \le 0 \end{cases}, \qquad (2.2)$$

где α – угловой коэффициент, принимающий небольшие значения, традиционно $\alpha = 0.01$.

ФА Leaky-ReLU (рис. 1б) сохраняет для нейронов небольшие отрицательные значения, позволяя им быть активированными. Это достигается за счет коэффициента α, принимающего небольшие значения, обычно 0.01, на который умножается функция идентичности. Использование коэффициента α при обратном проходе позволяет весам обновляться для отрицательных входных значений, из-за отсутствия нулевого градиента, характерного в подобной ситуации для ReLU. Коэффициент α является гиперпараметром, требующим настройки, что можно отнести к недостаткам использования данной ФА, так как подбор коэффициента требует дополнительных исследований. Кроме этого, проблема взрывных градиентов остается открытой для этой ФА.

В [9] представлены результаты эмпирического исследования, в котором определяется влияние угла наклона отрицательной части функции на задаче классификации изображений при использовании ФА ReLU и Leaky-ReLU, а также их модификаций: параметрической выпрямленной линейной единицы (PReLU) и рандомизированной выпрямленной линейной единицы с утечкой (RReLU). Тестирование выполнено на наборах данных CIFAR-10 и CIFAR-100. В качестве СНС, формирующих признаки людей, используются архитектуры NiN и NDSB Network. В PReLU оптимальное значение коэффициента α подбирается в процессе обучения нейронной сети, а в RReLU этот коэффициент задается случайным образом для каждого слоя. Проведенные исследования показали, что лучшие результаты были получены при использовании PReLU. Однако в этом случае высока вероятность переобучения СНС при использовании небольшого набора данных, поэтому RReLU оказывается более эффективной на практике.

Кроме указанных модификаций, небольшой наклон в отрицательной части области определения функции имеют ФА ELU, SeLU, GeLU, что позволяет предположить эффективность их использования для повторной идентификации людей.

ФА ELU (Exponential Linear Unit) [10] определяется выражением:

$$\varphi(x) = \begin{cases} x, & x \ge 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}, \qquad (2.3)$$

где $\alpha > 0$ – коэффициент, ограничивающий величину выходных значений на отрицательном участке области определения функции.

ФА ELU имеет ненулевые значения в отрицательной области и описывается логарифмической кривой (рис. 1 в). Такой подход должен позволить достигнуть насыщения нейронов для отрицательных входных значений, и тем самым уменьшить вариативность данных, распространяемых дальше по сети. Форма производной повторяет форму кривой для отрицательных значений нейронов, и имеет нулевые значения только для насыщенных нейронов. Ненулевые значения производной позволяют решить проблему “прореженных” нейронов, характерную для ReLU. К недостаткам можно отнести необходимость подбора коэффициента α и большую вычислительную сложность, чем для предыдущих модификаций функции ReLU.

ФА SELU (Scaled Exponential Linear Unit) (рис. 1г) является масштабированным вариантом ELU и описывается выражением:

$$\varphi(x) = \lambda \begin{cases} x, & x \ge 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}, \qquad (2.4)$$

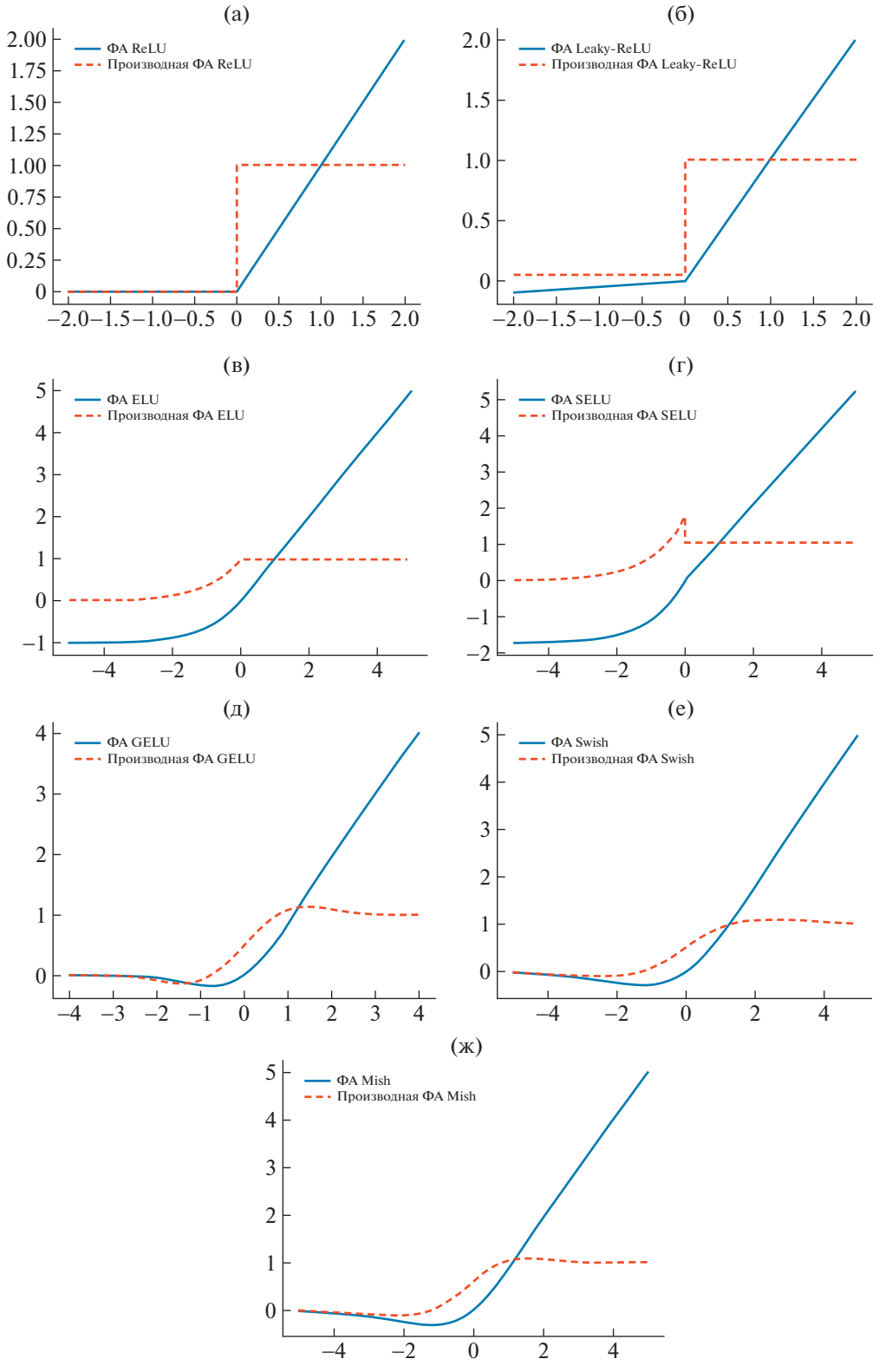

Рис. 1. Графики функций активации и их производной: а – ReLU; б – Leaky-RelU, где $\alpha = 0.05$; в – ELU, где $\alpha = 1$; г – SELU, где $\alpha = 1.67326$, $\lambda = 1.0507$; д – GELU; у – Swish, где $\beta = 1$; ж – Mish.

В исследовании, представленном в [11], определяются значения для коэффициентов $\alpha = 1.67326$, $\lambda = 1.0507$. Функция SELU обладает эффектом самонормализации, для которого характерно, что среднее значение выходов на каждом слое равно нулю, а стандартное отклонение равно единице. Это позволяет сети иметь более быструю сходимость. Применение SELU предполагает начальную инициализацию весов в соответствии с нормальным распределением для обеспечения самонормализующих свойств.

ФА GELU (Gaussian Error Linear Units) [12] определяется выражением:

$$\varphi(x) = x \cdot \frac{1}{2}[1 + \operatorname{erf}(x/\sqrt{2})] \approx \approx 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3))) \quad (2.5)$$

или:

$$\varphi(x) = x\sigma(1.702x), \quad (2.6)$$

где $\sigma = \frac{1}{1+e^{-x}}$ – функция активации сигмоиды.

Характерным отличием GELU от рассмотренных выше ФА является то, что GELU это невыпуклая немонотонная функция, нелинейная на всей области определения. Это позволяет GELU легче аппроксимировать сложные функции. В [12] представлено сравнение эффективности ФА ReLU, ELU и GELU для классификации на наборах данных MNIST и CIFAR10/100, и для распознавания речи на наборе данных TIMIT. Для данных задач показано улучшение эффективности их решения при использовании ФА GELU.

В [13] для поиска наиболее эффективной ФА используется подход автоматической генерации, основанный на последовательном переборе унарных и бинарных функций, которые поочередно объединяются, а результат оценивается эмпирически. Тестирование осуществлялось на наборах данных CIFAR-10 и CIFAR-100 с использованием CHC ResNet-164, WRN (Wide ResNet 28-10) и DenseNet 100-12, и на наборе ImageNet с использованием Mobile NASNet-A, Inception-ResNet-v2, Inception-v3 и Inception-v4 для классификации изображений, а также для решения задачи машинного перевода English-German на WMT2014. Наилучший результат при тестировании на разных наборах данных и архитектур нейронных сетей показала функция, получившая название Swish, которая определяется выражением:

$$\varphi(x) = x\sigma(\beta x), \quad (2.7)$$

где β – коэффициент, регулирующий степень кривизны функции, σ – функция сигмоиды.

Swish отличается от GELU коэффициентом β, который можно подбирать для улучшения обобщающей способности сети. Уменьшение β приближает эту функцию к линейной, а увеличение β – к ReLU. Swish не ограничена сверху, что ускоряет обучение при почти нулевых градиентах. Ограничение снизу для данной ФА приводит к эффекту регуляризации и позволяет отбросить большие отрицательные входные значения, что важно на первых шагах обучения, когда возникают большие отрицательные сигналы.

В [14] рассмотрена ФА, получившая название Mish, которая определяется выражением:

$$\varphi(x) = x\tanh(\mathit{softplus}(x)) = x\tanh(\ln(1 + e^x)). \quad (2.8)$$

Mish визуально схожа с ФА Swish и GELU (рис. 1д, е, ж), обладает всеми теми же свойствами. В большинстве экспериментов приведенных в [13], Mish показала лучшие результаты, чем другие рассмотренные ФА для классификации объектов на наборах данных CIFAR-10 и MNIST для CHC ResNet-20, WRN-10-2, SimpleNet, Xception Net, Capsule Net, Inception ResNet v2, DenseNet-121, MobileNet-v2, ShuffleNet-v1, Inception v3, EfficientNet B0 на наборе данных ImageNet-1k с CHC ResNet-18, ResNet-50, SpineNet-49, PeleeNet, CSP-ResNet-50, CSP-DarkNet-53, CSP-ResNext-50. Для задачи обнаружения людей на наборе данных MS COCO с CHC CSP-DarkNet-53 и CSP-DarkNet-53+PANet+SPP и в составе архитектуры CHC для обнаружения объектов YOLOv4 в различных модификациях Mish также позволяет улучшить результативность.

3. НАБОРЫ ДАННЫХ ДЛЯ РЕИДЕНТИФИКАЦИИ

Для алгоритмов реидентификации одним из крупных и распространенных наборов данных, с изображениями людей с нескольких камер видеонаблюдения является Market1501 [15]. Этот набор данных содержит 32668 различных изображений для 1501 человека. Для формирования набора данных использовались кадры, полученные с шести камер видеонаблюдения, установленных возле супермаркета в университете Циньхуа. Набор данных разделен на обучающую и тестовую выборки, и для обучения используются 12936 изображений для 751 человека, а в галерею включены и используются для тестирования 19732 изображения для 750 человек. Набор данных так же содержит 2793 ограничивающих прямоугольника, содержащих неверные, но достаточно правдоподобные ответы, включенные в тестовую выборку.

Для повышения объективности оценки точности реидентификации с применением разных ФА предложен и используется другой набор данных PolReID [16], сформированный при участии сотрудников и студентов Полоцкого государственного университета, которые подтвердили свое согласие на применение их изображений для проведения экспериментов. Таким образом, PolReID

Рис. 2. Примеры изображений набора данных PolReID.

является открытым набором данных и может использоваться другими исследователями [17]. Необходимость тестирования на нескольких наборах данных обусловлена проблемой смещения доменов при реидентификации. При обучении СНС на одном наборе данных, а тестировании на данных, полученных в других условиях, точность реидентификации значительно снижается, но является более объективной на практике, чем при использовании выборки, принадлежащей одному домену. В [16] показано, что объединение нескольких наборов данных позволяет повысить точность реидентификации в целом.

Текущий набор PolReID включает 31 919 изображений для 271 человека, при этом обучающая выборка представлена 16 770 изображениями для 145 человек, а тестовая – 15 149 изображений для 126 человек. Изображения людей из кадров получены с помощью СНС YOLOv4 [18] и рассортированы оператором. Для формирования комплекта изображений для каждого человека использовалось от двух до десяти камер и от одной до девяти видеопоследовательности с каждой камеры. Таким образом, всего использовалось более чем 200 локаций сьемки, среди них в уличных условиях 159 человек, 40 из которых также зафиксированы камерами внутреннего наблюдения в помещениях с естественным и искусственным освещением различной интенсивности. Изображения получены при разных погодных условиях и временах года: летом для 47 человек, осенью для 176 человек, зимой для 48 человек. PolReID включает изображения 194 мужчин и 77 женщин разной возрастной категории: 230 человек в возрасте до 30 лет, 41 человек в возрасте от 30 до 60 лет. Причем 134 человека на лице имеют маску, 9 из них на некоторых кадрах присутствуют без маски. Изображения людей представлены с нескольких ракурсов с различными предметами, такими как сумка, пакет, телефон, рюкзак, очки. Несколько примеров изображений из PolReID показаны на рис. 2.

При объединении наборов данных Market1501 и PolReID для корректной работы алгоритма реидентификации все имена изображений были приведены к единому формату записи: XXXXX_cYYYsZZ_AAAAAA_BB.jpg, где XXXXX – идентификатор человека, YYY – номер камеры, ZZ – номер видеопоследовательности, полученный с этой камеры, AAAAAA – номер кадра в видеопоследовательности, BB – количество человек на текущем кадре.

Таким образом, объединенный набор данных всего содержит 68 587 ограничительных рамок для 1772 человек. Для обучения использовалось 29 707 изображений для 896 человек, а для тестирования 38 881 изображение для 876 человек.

Таблица 1. Результаты экспериментов по определению наиболее эффективных функций активации для реидентификации человека на наборе данных Market1501

Функция активации	ResNet-50			DenseNet-121			DarkNet-53		
	Время обучения (м:с)	Rank1	mAP	Время обучения (м:с)	Rank1	mAP	Время обучения (м:с)	Rank1	mAP
ReLU	86:55	81.24*	57.31*	92:06	79.63*	56.62*	98:08	80.08	57.12
LeakyReLU	86:50	**81.50**	**57.44**	95:52	**79.84**	**56.68**	99:31	80.97*	57.16*
PReLU	102:49	79.57	55.54	138:25	78.12	53.82	112:09	76.93	52.87
RReLU	90:51	81.00	**58.24**	101:57	79.36	**56.68**	110:00	79.96	56.73
ELU	85:26	79.51	56.17	99:54	73.52	49.09	111:16	72.98	46.72
SELU	86:21	77.46	51.91	95:12	66.57	40.73	97:35	66.09	40.31
GELU	88:08	**81.59**	**57.92**	95:13	**79.69**	**56.98**	107:53	**80.98**	**57.26**
Swish	104:42	**81.38**	**57.68**	128:57	**80.08**	**56.87**	116:10	**81.09**	**58.65**
Mish	104:52	**82.16**	**58.95**	130:13	**80.76**	**57.10**	117:48	**81.44**	**57.97**

* – отмечены значения Rank1 и mAP, полученные для базовой функции активации для каждой из рассматриваемых СНС

4. ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ

4.1. Метрики для оценки точности реидентификации

Для оценки точности работы алгоритма реидентификации использовались метрики Rank1, Rank5, Rank10 и mAP.

RankN является метрикой качества ранжирования и показывает процент числа запросов, для которых верный выданный результат был среди первых N полученных результатов. Соответственно, метрика Rank1 показывает процент запросов, для которых идентификатор первого изображения-кандидата совпадает с идентификатором запроса. Rank5 показывает процент запросов, для которых среди первых пяти выданных изображений-кандидатов было верное решение, Rank 10 – среди первых 10. Для вычисления RankN определяется отношение суммы запросов, для которых верное решение было найдено среди первых выданных результатов, к общему числу запросов Q:

$$RankN = \frac{\sum K_{i,N}}{Q}, \tag{2.9}$$

где: i – номер запроса; $K_{i,N}$ – i-тый запрос, для которого верное решение было найдено среди первых N выданных результатов.

Метрика mAP является оценкой точности алгоритма повторной идентификации, отражающей среднее значение средней точности для всех запросов и рассчитывается по формуле:

$$mAP = \frac{1}{Q}\sum_{i=1}^{Q} AP_i, \tag{2.10}$$

где Q – общее число запросов, $AP = \frac{\sum precision}{\mathrm{I}}$ – средняя точность для каждого i-того запроса, I – число изображений в тестовой выборке, $precision = \frac{TP}{TP + FN}$ – точность запроса , TP – количество верно положительных предсказаний запроса, FN – ложно-отрицательных предсказаний запроса.

4.2. Результаты экспериментов

Первый этап экспериментов направлен на выявление наиболее эффективных ФА для повторной идентификации с использованием алгоритма [19], реализованном на фреймворке pyTorch. Обучение модели выполнено для ResNet-50, DenseNet-121 и DarkNet-53 без применения предварительного обучения, в течение 60 эпох со скоростью 0.03, уменьшенной после 40-й эпохи на 0.01, с размером пакета 16, на наборе данных Market1501 с использованием персонального компьютера с основными характеристиками: Intel Core i5 3.11 GHz, 16 Gb RAM, Nvidia GeForce RTX-3060 6 Gb.

В табл. 1 представлены результаты тестирования алгоритма реидентификации человека с различными ФА в СНС и отмечены значения метрик точности реидентификации, которые обеспечивают повышение значений по сравнению с базовой ФА: СНС ResNet-50 и DenseNet-121 реализации используют ФА ReLU, в DarkNet-53 используется Leaky-ReLU.

Из табл. 1 очевидно, что ФА PReLU, RReLU, ELU и SELU для задачи реидентификации наименее эффективны, точность реидентификации

Таблица 2. Влияние функции активации в СНС ResNet-50 на точность реидентификации людей при обучении на объединенном наборе данных Market1501 и PolReID

Функция активации	Market1501		PolReID		Market1501 и PolReID	
	Rank1	mAP	Rank1	mAP	Rank1	mAP
ReLU*	83.82	62.35	86.60	64.11	84.23	62.20
	84.00	62.24	88.55	63.23	84.62	62.00
	83.49	62.15	87.95	63.09	84.10	61.92
	83.77	62.25	87.70	63.48	84.32	62.04
Leaky-ReLU	83.25	62.50	86.90	63.85	83.76	62.30
	84.00	62.56	87.20	63.06	84.50	62.28
	84.12	63.21	88.40	63.10	84.62	62.78
	83.79	**62.76**	87.50	63.34	84.29	**62.45**
GELU	84.53	62.71	86.90	63.99	84.85	62.61
	83.76	62.50	89.76	64.28	84.72	62.46
	83.82	62.43	87.05	63.39	84.15	62.21
	84.04	62.55	87.90	63.89	**84.57**	62.43
Swish	83.37	61.08	88.86	63.96	84.20	61.25
	83.31	61.52	91.11	65.33	84.47	61.81
	83.91	61.51	89.46	65.33	84.72	61.79
	83.53	61.37	**89.81**	64.87	84.46	61.62
Mish	83.94	62.37	88.55	64.75	84.62	62.42
	82.81	62.10	88.55	65.09	83.58	62.24
	83.52	62.32	88.10	65.70	84.23	62.53
	83.42	62.26	88.40	**65.18**	84.14	62.40

* – отмечена базовая функция активации для СНС ResNet-50

при их использовании значительно меньше, чем при использовании базовой функции активации СНС. При применении ФА PReLU наблюдается снижение метрики Rank1 для всех рассмотренных. Максимальное уменьшение mAP характерно для СНС DarkNet-53 и составляет 4.29%. При использовании RReLU следует отметить снижение точности Rank1 по сравнению с базовой ФА. Применение ФА ELU и SELU приводит к значительному уменьшению значений Rank1 и mAP для СНС DenseNet-121 и DarkNet-53. Для всех используемых СНС наилучшие значения Rank1 и mAP характерны для ФА GELU, Swish и Mish, значит требуются дальнейшие исследования с расширением используемой выборки данных.

С учетом полученных результатов, на втором этапе исключены наименее эффективные ФА.

Для оценки воспроизводимости результатов точности повторной идентификации людей каждая СНС с разными ФА была повторно обучена трижды, каждый раз с одинаковыми гиперпараметрами на объединенном наборе данных Market1501 и PolReID.

Тестирование выполнялось на тестовых выборках отдельно для Market1501, PolReID и на объединенном наборе для каждой обученной СНС с последующим определением среднего арифметического значений метрик. Набор данных Market1501 разбит на тестовую и обучающую выборки согласно протоколам исходных документов, PolReID разделен случайным образом на обучающую (53%) и тестовую выборку (47%), которые не пересекаются. В табл. 2–4 представлены результаты тестирования реидентификации с использованием СНС ResNet-50, DenseNet-121 и DarkNet-53.

Строки в табл. 2–4, выделенные серым цветом содержат средние значения для каждой СНС. Числовые значения метрик, отмеченные полужирным шрифтом, являются лучшими среди

Таблица 3. Влияние функции активации в СНС DenseNet-121 на точность реидентификации людей при обучении на объединенном наборе данных Market1501 и PolReID

Функция активации	Market1501		PolReID		Market1501 и PolReID	
	Rank1	mAP	Rank1	mAP	Rank1	mAP
ReLU*	83.19	62.27	88.40	63.46	83.83	62.11
	83.76	62.22	88.70	63.35	84.57	62.09
	82.57	62.08	87.50	63.37	83.31	61.95
	83.17	**62.19**	88.20	63.39	**83.90**	**62.05**
Leaky-ReLU	82.99	62.14	87.65	63.22	83.73	62.00
	83.94	61.91	88.10	64.79	84.55	62.03
	83.34	62.17	87.80	63.72	83.98	62.06
	83.42	62.07	87.85	63.91	84.09	62.03
GELU	82.81	61.09	89.76	64.60	83.85	61.30
	82.63	60.95	89.16	64.93	83.48	61.23
	82.63	60.62	90.21	63.70	83.76	60.76
	82.69	60.89	**89.71**	**64.41**	83.70	61.10
Swish	81.65	58.72	88.70	63.48	82.76	59.14
	81.56	58.35	88.86	63.12	82.64	58.72
	80.76	58.22	88.40	62.08	81.97	58.52
	81.32	58.43	88.65	62.89	82.46	58.79
Mish	82.04	59.14	88.40	62.35	83.08	59.33
	80.76	59.07	87.95	62.04	81.87	59.20
	82.54	59.46	87.85	63.22	83.36	59.71
	81.78	59.22	88.07	62.54	82.77	59.41

* – отмечена базовая функция активации для СНС DenseNet-121

средних для каждой тестовой выборки. Анализ табл. 2–4 показывает, что при повторении экспериментов для СНС, не изменяя ее архитектуру гиперпараметры, обучающую и тестовую выборки возможен существенный разброс значений метрик точности реидентификации. Так, для СНС DarkNet-53 с ФА Swish на тестовой выборке Market1501 разница между лучшим и худшим результатом mAP составляет 3.26 (см. табл. 4). Поэтому оценка средних значений метрик является предпочтительнее. Анализ средних значений метрик моделей СНС показывает, что при разных тесто-

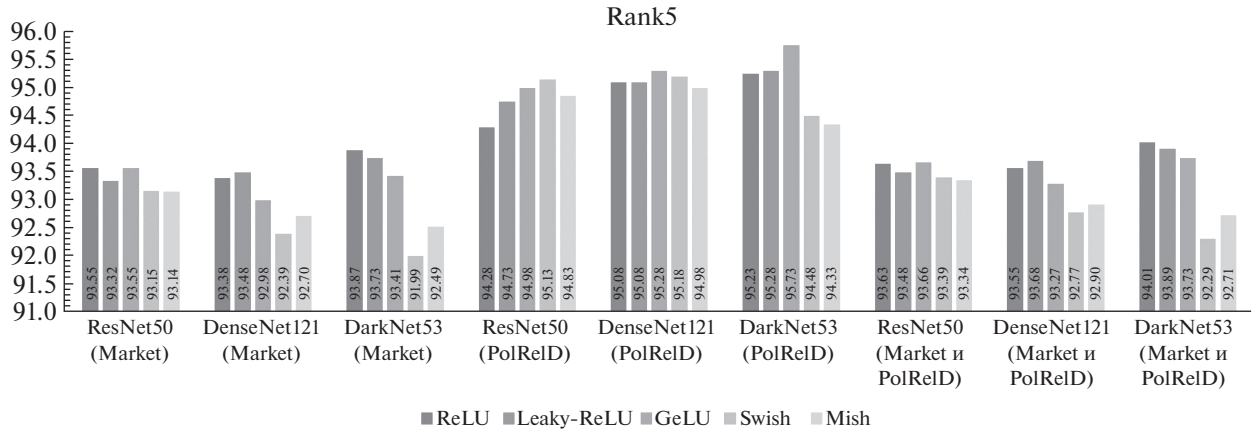

Рис. 3. Значения метрики Rank5 для различных СНС, ФА и наборов данных.

Таблица 4. Влияние функции активации в СНС DarkNet-53 на точность реидентификации людей при обучении на объединенном наборе данных Market1501 и PolReID

Функция активации	Market1501		PolReID		Market1501 и PolReID	
	Rank1	mAP	Rank1	mAP	Rank1	mAP
ReLU	83.91	63.61	88.40	64.80	84.55	63.47
	84.32	63.02	87.20	64.28	84.70	62.84
	84.59	63.63	89.16	65.27	85.27	63.50
	84.27	**63.42**	88.25	64.78	**84.84**	**63.27**
Leaky-ReLU*	83.08	62.43	87.95	64.45	83.68	62.40
	83.61	63.00	89.16	64.94	84.45	62.97
	84.95	64.03	89.46	65.39	85.59	63.89
	83.88	63.16	88.86	**64.93**	84.57	63.09
GELU	82.22	60.11	87.50	62.93	82.86	60.13
	84.00	62.28	89.91	64.84	84.90	62.34
	83.79	62.25	89.46	65.08	84.67	62.37
	83.34	61.55	**88.96**	64.28	84.14	61.61
Swish	80.82	55.56	87.95	59.80	81.87	55.75
	81.41	58.82	88.70	62.55	82.34	58.98
	80.43	55.90	87.35	60.63	81.47	56.22
	80.89	56.76	88.00	60.99	82.61	56.98
Mish	81.80	57.85	87.50	61.56	82.61	58.02
	81.89	58.98	88.86	62.35	82.99	59.11
	80.43	56.96	86.30	60.74	81.13	57.11
	81.37	57.93	87.55	61.55	82.24	58.08

* – отмечена базовая функция активации для СНС DarkNet-53

вых выборках выделить одну ФА, наиболее эффективную для всех СНС не представляется возможным, так как лучшие показатели имеют разные ФА. Из табл. 2–4 очевидно, что в большинстве случаев наиболее эффективна функция ReLU, для которой средние значения Rank1 более значимы в трех экспериментах, mAP наилучшие в четырех экспериментах. Функция GeLU обеспе-

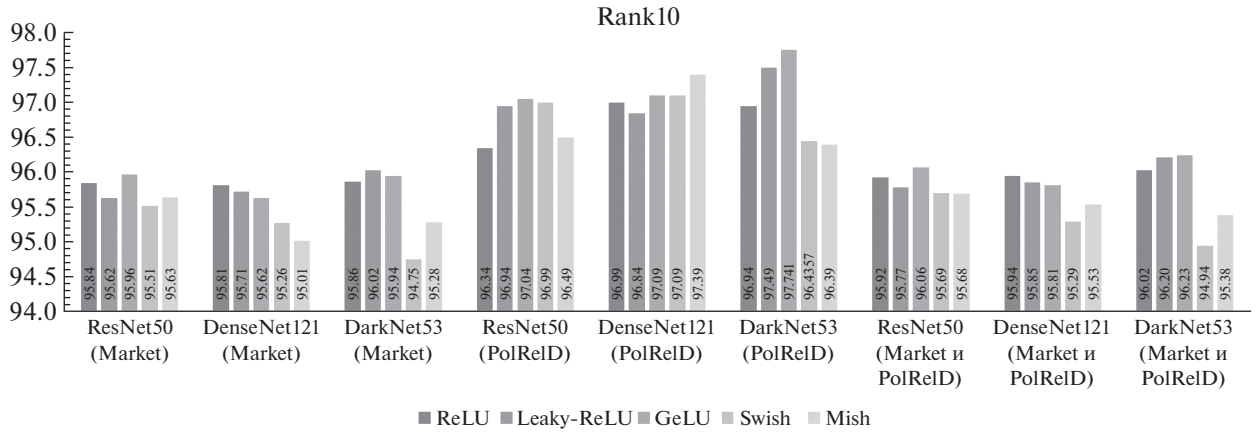

Рис. 4. Значения метрики Rank10 для различных СНС, ФА и наборов данных.

чивает максимизацию показателей для Rank1 в четырех случаях, а для mAP в одном.

На рис. 3 и рис. 4 показано сравнение значений метрик Rank5 и Rank10, соответственно, при обучении на объединенном наборе данных и тестировании на Market1501, PolReID и объединенном наборе данных Market1501 и PolReID для различных ФА.

Рис. 3 свидетельствует о том, что при оценке алгоритма реидентификации в метрике Rank5 в трех случаях из девяти наилучшие показатели характерны для ФА GeLU, в двух случаях для ReLU. Следует отметить, что применение ФА ReLU и GeLU в СНС ResNet-50 при тестировании на наборе данных Market1501 приводит к одинаковым средним значениям Rank5.

Анализ рис. 4, отражающего точность реидентификации в метрике Rank10 показал, что ФА GeLU обеспечивает более высокие показатели для большинства рассмотренных случаев.

5. ОБСУЖДЕНИЕ И РЕКОМЕНДАЦИИ

Анализ проведенных исследований показывает, что при оценке точности реидентификации с использованием метрик Rank5 и Rank10 наиболее эффективны ФА GeLU и ReLU. Причем GeLU показала лучший результат в трех случаях из девяти, а ReLU имела наибольшие результаты в двух экспериментах для Rank5. Сравнение точности по метрике Rank10 свидетельствует о том, что ФА GeLU показала наилучшие результаты в пяти случаях из девяти.

Оценка по метрикам Rank1 и mAP позволяет выделить наиболее эффективную ФА ReLU.

Разброс значений метрик точности реидентификации при повторном обучении СНС без изменения ее архитектуры гиперпараметров, ФА, обучающей и тестовой выборок варьируется для разных СНС и достигает максимального для DarkNet-53 с ФА Swish. Наименьший разброс значений зафиксирован для СНС ResNet-50 с ФА Leaky-ReLU. Однозначно сложно утверждать, что какая-то из рассмотренных ФА обладает большей стабильностью, так как пока неизвестна взаимосвязь между архитектурой СНС, ФА, набором данных и величиной разброса полученных значений точности для Rank1 и mAP. Однако анализ результатов свидетельствует о том, что для разных тестовых выборок и СНС лучшие средние показатели в большинстве случаев имеет ФА ReLU, что говорит о большей воспроизводимости результатов, по сравнению с другими ФА.

Усреднение показателей точности реидентификации для разных наборов данных и архитектур СНС позволило установить, что наиболее целесообразным является использование ФА ReLU для повторной идентификации, она показывает наилучшее соотношение точность реидентификации–скорость обучения–воспроизводимость результатов для разных моделей СНС.

6. ЗАКЛЮЧЕНИЕ

В работе рассмотрены и проанализированы наиболее распространенные функции активации, используемые в сверточных нейронных сетях, в приложении к задаче повторной идентификации человека в распределенных системах видеонаблюдения. При этом использованы три разные архитектуры СНС, ФА ReLU, Leaky-ReLU, PReLU, RReLU, ELU, SELU, GELU, Swish, Mish. Предложен набор данных для тестирования алгоритмов повторной идентификации PolReID, который в настоящее время включает 31 919 изображений для 271 человека. В результате анализа полученных результатов, было установлено, что для задачи реидентификации наиболее перспективными ФА являются ReLU и GeLU, при этом скорость работы и воспроизводимость результатов для ФА ReLU выше, чем при использовании GeLU.

БЛАГОДАРНОСТЬ

Эта работа поддержана Национальной программой высококвалифицированных иностранных экспертов (Гранты G2021016028L и G2021016001L) и Специальным фондом фундаментальных научных исследований Чжэцзянского Шурен Университета.

СПИСОК ЛИТЕРАТУРЫ

1. *Ye S., Bohush R.P., Chen H. et al.* Person Tracking and Reidentification for Multicamera Indoor Video Surveillance Systems // Pattern Recognit. Image Anal. 2020. V. 30. P. 827–837. https://doi.org/10.1134/S1054661820040136
2. *Porrello A., Bergamini L., Calderara S.* Robust Re-Identification by Multiple Views Knowledge Distillation. // ArXiv, abs/2007.04174. 2020.
3. *Huang G., Liu Z., Weinberger K.Q.* Densely Connected Convolutional Networks. // 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017. P. 2261–2269.
4. *Wang G., Lai J., Huang P., Xie X.* // Spatial-Temporal Person Re-identification. // ArXiv, abs/1812.03282. 2019.
5. *Mao S., Zhang S., Yang M.* // Resolution-invariant Person Re-Identification. // ArXiv, abs/1906.09748. 2019.
6. *Redmon J., Farhadi A.* // YOLOv3: An Incremental Improvement. *ArXiv* abs/1804.02767. 2018
7. *Nair, Vinod, Geoffrey E. Hinton* Rectified linear units improve restricted Boltzmann machines. // In ICML. 2010. P. 807–814.

8. *Maas, L. Andrew*. Rectifier non linearities improve neural network acoustic models. In ICML. 2013. V. 30.
9. *Xu B., Wang N., Chen T., Li M*. Empirical Evaluation of Rectified Activations in Convolutional Network. // ArXiv, abs/1505.00853. 2015.
10. *Clevert D., Unterthiner T., Hochreiter S.* Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs) // arXiv: abs/1511.07289v5. 2016.
11. *Klambauer G., Unterthiner T., Mayr A., Hochreiter S.* Self-Normalizing Neural Networks. // ArXiv, abs/1706.02515. 2017.
12. *Hendrycks D., Gimpel K.* Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. // ArXiv, abs/1606.08415. 2016.
13. *Ramachandran P., Zoph B., Le Q.V.* Swish: a Self-Gated Activation Function. // arXiv: abs/1710.05941v2. 2017.
14. *Misra D.* Mish: A Self Regularized Non-Monotonic Neural Activation Function. // ArXiv, abs/1908.08681. 2019.
15. *Zheng L., Shen L., Tian L., Wang Sh., Wang J., Tian Q.* Scalable Person Re-Identification: A Benchmark // IEEE International Conference on Computer Vision (ICCV2015). 2015. P. 1116–1124.
16. *Ihnatsyeva S., Bohush R., Ablameyko S.* Joint Dataset for CNN-based Person Re-identification // Pattern Recognition and Information Processing (PRIP'2021): Proceedings of the 15th International Conference, 21–24 Sept. 2021, Minsk, Belarus. Minsk: UIIP NASB. 2021. P. 33–37.
17. *Ihnatsyeva S., Bohush R.* PolReID. https://github.com/SvetlanaIgn/PolReID. 2021.
18. *Bochkovskiy A., Wang Ch.-Y., Liao H.-Y.M.* YOLOv4: Optimal Speed and Accuracy of Object Detection // ArXiv, abs/2004.10934. 2020.
19. Person reID baseline pytorch. https://github.com/layumi/Person_reID_baseline_pytorch.

EDN: HVHYLA
ПРОГРАММИРОВАНИЕ, 2022, № 5, с. 27–36

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.925.3

ВЫСОКОРЕАЛИСТИЧНАЯ ВИЗУАЛИЗАЦИЯ КАУСТИК И ШЕРОХОВАТЫХ ПОВЕРХНОСТЕЙ

© 2022 г. С. И. Вяткин[a],* (http://orcid.org/0000-0002-1591-3588),
Б. С. Долговесов[a], (http://orcid.org/0000-0002-6255-9315)**

[a] Институт автоматики и электрометрии СО РАН
630090 Новосибирск, просп. Академика Коптюга, д. 1, Россия
**E-mail: sivser@mail.ru*
***E-mail: bsd@iae.nsk.su*

Поступила в редакцию 22.03.2022 г.
После доработки 25.04.2022 г.
Принята к публикации 17.05.2022 г.

При рассеянии света от зеркальных поверхностей создаются сложные оптические эффекты, которые встречаются в реалистичных сценах. К ним относятся каустики, создаваемые из-за сфокусированного отражения, многократного преломления и высокочастотных бликов от зеркальной микроструктуры. Однако большой проблемой для визуализации являются сцены, содержащие зеркальные пути, включающие цепочки взаимодействий с гладкими металлическими и преломляющими поверхностями, поскольку затрудняется поиск световых путей. То есть снижается вероятность выборки допустимой конфигурации, соединяющей камеру и источник света. Каустики и шероховатые поверхности со случайными узорами бликов из-за зеркальной микрогеометрии, являются визуально яркими примерами таких зеркальных путей света. Существуют много специализированных методов, каждый из которых способен отображать определенные типы зеркальных путей со своим собственным набором ограничений. Однако не известны методы для общего случая. Например, фотонные карты являются хорошим решением для определенных видов путей, однако они могут привести к нежелательному размытию и не обрабатывают важные случаи, включая каустики на не рассеивающих поверхностях или бликах.

В данной работе представлен комбинированный метод визуализации отражающей и преломляющей каустики и шероховатых поверхностей. Используется стохастическая инициализация и оценка веса выборки при нахождении решения для сложной геометрии. Применяется двухпроходная стохастическая инициализация для шероховатых поверхностей с отображением карт нормалей. На первом проходе игнорируется карта нормалей и находятся зеркальные пути на исходной гладкой поверхности. Нормальное возмущение выбирается случайным образом из распределения нормалей, присутствующих на карте нормалей. Используется гауссова аппроксимация всей карты нормалей, полученной из самого низкого уровня MIP карты. Второй проход, начатый с шероховатой поверхности, приводит к корректному решению для сложной геометрии. Таким образом предлагаемый метод распространяется на цепочки с шероховатыми поверхностями и множественными взаимодействиями.

В результате визуализированы как прямые, так и косвенные каустики, создаваемые отражением от зеркальных поверхностей. Визуализирована также более сложная сцена с каустикой, в которой использована не ламбертовская модель отражения, а также комбинированная сцена каустики и шероховатой поверхности. При визуализации каустики узоры на металлической шероховатой поверхности не размываются. Метод обеспечивает очень хорошее представление для глянцевых, шероховатых и зеркальных материалов. В работе показаны результаты визуализации высокореалистичных сцен.

Ключевые слова: функционально заданная сцена, функции возмущения, световой путь, зеркально-диффузные отражения, каустика, шероховатая поверхность, блик

DOI: 10.31857/S0132347422050065

1. ВВЕДЕНИЕ

Распространение света в трехмерной сцене, в которой учитываются множественные отражения и рассеяния света, а также различные виды взаимодействия его с объектами сцены, является сложной проблемой. При синтезе графических сцен необходимо решать двуединую задачу: обеспечение высокой реалистичности воспроизведения графических объектов и достижение приемлемого для конкретной задачи времени формирования изображения. В работе [1] дан обзор методов расчета глобальной освещенности в за-

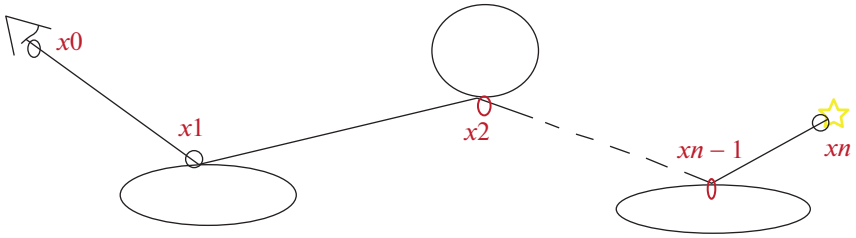

Рис. 1. Зеркальный перенос света.

дачах реалистичной компьютерной графики. Существуют методы как в нереальном времени с высоким реализмом сцен, так и алгоритмы, работающие в режиме реального времени, реализованные на графических процессорах. Например, эффект каустики, реализован в модуле визуализатора Vray для 3DS MAX. Визуализация с помощью Vray может занимать от нескольких минут до часа и больше в зависимости от сложности и качества изображения. Визуализатор Vray базируется на методе [2]. Визуализатор тоже нереального времени Corona основан на методе, описанном в работе [3]. С другой стороны, известны методы, работающие в интерактивном режиме с использованием графических процессоров, например, объемные каустики в однократно рассеивающих средах [4]. В статье [5] предлагается метод рендеринга изображений с глобальным освещением также реализованный на графических процессорах. В методе применяется двунаправленная трассировка пути.

По данному в работе [1] определению в нашей работе рассматриваются каустики второго типа. Как отмечено в [1] такие каустики не могут быть эффективно вычислены с помощью двунаправленной трассировки путей. Из данного обзора, на наш взгляд, следует такой вывод, что создан большой набор специализированных методов, каждый из которых способен отображать определенные типы зеркальных путей. Однако ни один из существующих методов не подходит для общего случая. Каждый подход имеет свой собственный набор ограничений. Например, двунаправленная трассировка путей [5], как уже отмечалось имеет проблемы с бликами и не эффективна для каустик второго типа. Одним из особенно проблемных случаев являются сцены, содержащие каустики или блестящие поверхности, которые имеют случайные узоры бликов из-за зеркальной микрогеометрии. Методы, работающие в терминах светового потока, используют фотонные карты [6] и хорошо решают определенный класс задач, однако они могут привести к нежелательному размытию и не обрабатывают важные случаи, включая каустики на нерассеянных поверхностях и бликах. Так в работе [1] справедливо замечено, что фотонные карты должны применяться с осторожностью на поверхностях с микрорельефом, поскольку могут искажать вид микрорельефа. С другой стороны, методы визуализации точно отображающие шероховатые поверхности с бликами, имеют проблемы с другими типами зеркальных путей [7, 8]. Таким образом, известные методы имеют ограничения, например, в случае бликов и переходов от зеркального к диффузному и обратно к зеркальному пути прохождения света, а также если геометрия сцены имеет высокочастотную детализацию.

В данной работе предлагается общий подход к визуализации каустик второго типа (зеркально-диффузно-зеркальные) [1] и шероховатых поверхностей с бликами. Его суть заключается в следующем: на перовом этапе работы алгоритма не рассматривается карта нормалей, а вычисляются зеркальные пути на исходной гладкой поверхности. Далее возмущение нормалей выбирается случайным образом из распределения нормалей, присутствующих на карте нормалей. Используется гауссова аппроксимация всей карты нормалей, полученная из самого низкого уровня многоуровневой карты [9].

В работе представлен метод визуализации рассеяния света от зеркальных поверхностей, и сложных каустик, а также многократного преломления света и бликов на зеркальных поверхностях. Разработан алгоритм отбора проб зеркальных путей, включая блики и переходы от зеркального к диффузному и обратно к зеркальному пути прохождения света. Метод позволяет обрабатывать высокочастотную геометрию с отображением карт нормалей, карт смещения и зеркально-диффузные отражения. Метод позволяет визуализировать отражающие и преломляющие каустики. При отображении шероховатых поверхностей обеспечивается лучшее приближение для большинства материалов в сравнении с известными подходами [7, 8]. Реализован алгоритм выборки для поверхностей с отображением карт нормалей. Метод отличается уменьшенной дисперсией, повышенной надежностью и сходимостью. Работа базируется на общей выборке путей в исходной задаче, стохастически выбираются отдельные решения. Добавленная дисперсия, вы-

Рис. 2. Множественные решения ограничения зеркального пути образуют суперпозицию каустик.

Рис. 3. Объемная каустика.

званная случайным решением, уменьшается вместе с дополнительными источниками дисперсии. К ним относятся отображение окружающей среды, косвенное освещение и т.д. Предлагаемый метод использует стохастическую инициализацию и эффективную оценку веса выборки. В результате можно находить решения для сложной геометрии. Показано, что стохастическая инициализация для поверхностей с отображением карт нормалей, переходы от зеркального к диффузному и обратно к зеркальному пути прохождения света, напрямую связаны с рендерингом бликов.

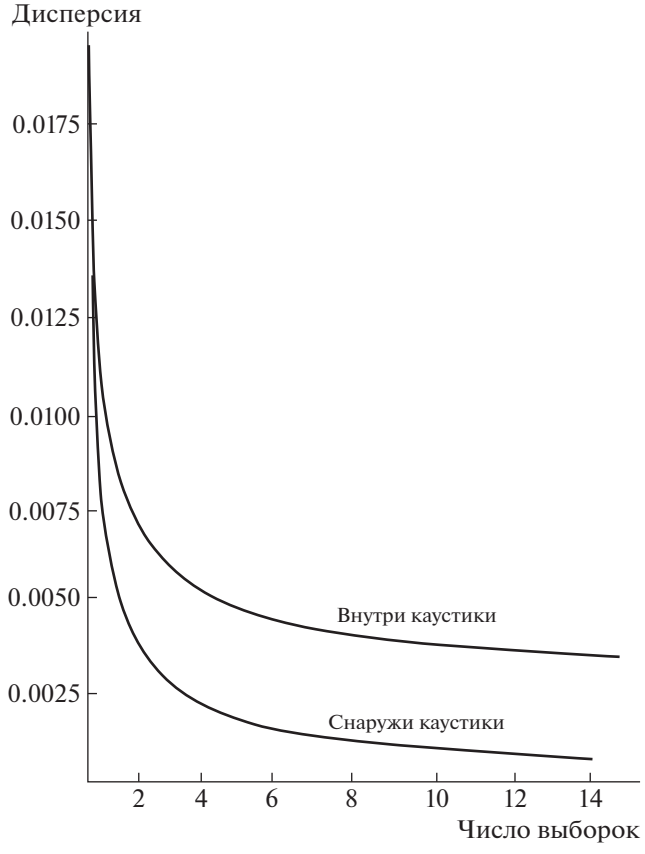

Рис. 4. Зависимости дисперсии от числа выборок.

Использование стохастической выборки решений для глянцевых поверхностей в методе также эффективно, поскольку одновременно уменьшается несколько дополнительных источников дисперсии.

2. ОПИСАНИЕ МЕТОДА

2.1. Функционально заданные сцены

Для описания геометрических объектов в функционально заданных сценах используются функции возмущения [9]:

$$F'(x,y,z) = F(x,y,z) + \sum_{i=1}^{N} f_i R_i(x,y,z), \qquad (1)$$

где $F'(x, y, z)$ – свободная форма; $F(x, y, z)$ – базовая квадрика; f_i – форм-фактор; $i = 1...N$ – количество функций возмущения; $R_i(x, y, z)$ – возмущение,

$$R_i(x,y,z) = \begin{cases} Q_i^3(x,y,z), & \text{if} \quad Q_i(x,y,z) \geq 0 \\ 0, & \text{if} \quad Q_i(x,y,z) < 0 \end{cases}, \qquad (2)$$

где $Q(x, y, z)$ – возмущающая квадрика.

Непрерывность функции $F(x, y, z)$ и ее производной $F'(x, y, z)$ гарантируется степенью возмущающей квадрики, которая должна быть больше двух (2).

2.2. Зеркальный перенос света

Рассмотрим путь переноса света, содержащий ряд зеркальных точек $\vec{x}_2, ..., \vec{x}_{n-1}$ между двумя не зеркальными конечными точками $\vec{x}_1$ и $\vec{x}_n$. Где $\vec{x}_1$ – точка затенения на поверхности, а $\vec{x}_n$ – положение источника света (рис. 1).

Заметим, что каждая зеркальная точка накладывает физические ограничения, например, когда в точке происходит преломление, которое описывается законом Снелла. Ограничения описываются с помощью функции $\vec{b}_i$, связанной с каждой точкой $\vec{x}_i$. Полувектор проецируется на локальное касательное пространство [12]:

$$\vec{b}_i(\vec{x}_{i-1}, \vec{x}_i, \vec{x}_{i+1}) = M_t(\vec{x}_i)^T \vec{v}_h(\vec{x}_i, \vec{x}_i\vec{x}_{i-1}, \vec{x}_i\vec{x}_{i+1}), \qquad (3)$$

где $M_t(\vec{x}_i)$ – матрица касательных векторов 3×2.

Объединенная функция зеркального пути:

$$\vec{b}_c(\overline{x}_i) = [\vec{b}_2, ..., \vec{b}_{n-1}]^T. \qquad (4)$$

Функция параметризуется с помощью UV-координат зеркальных точек. Решение уравнения $\vec{b}_c(\overline{uv}) = 0$ находится с использованием итерации Ньютона:

$$\overline{uv}_{i+1} = (\nabla \vec{b}_c(\overline{uv}_i))^{-1} \cdot \vec{b}_c(\overline{uv}_i), \qquad (5)$$

где $\nabla \vec{b}_c$ – Якобиан–матрица, состоящая из блоков 2×2.

Шаг Ньютона (5) создает точки, после каждой итерации требуется дополнительный шаг перепроекции. В предлагаемом методе случайным образом выбирается начальное предположение из распределения вероятностей $\rho(\overline{x}_0)$. Все решения находятся с ненулевой вероятностью, поскольку метод Ньютона имеет квадратичную сходимость, и начальная точка близка к корню. Однако вероятность успешной сходимости неизвестна. Выбираем два равномерно распределенных случайных числа в виде входных данных и преобразуем их в желаемое распределение. Цель состоит в генерации начальной выборки близкой к решению для сходимости итераций Ньютона (5).

Дискретная вероятность представляет собой площадь области конвергенции, так как выборки равномерно распределены:

$$\rho_n = \int_U S(\zeta) d\zeta, \qquad (6)$$

где S – форма, зависящая от положения конечных точек пути, ζ – два равномерно распределенных случайных числа, U – первичная область выборки.

$$\zeta = (\zeta_1, \zeta_2) \in [0,1)^2. \tag{7}$$

Вычисление интеграла (6) явно неосуществимо. Применим приближенную оценку:

$$\langle \rho_n \rangle = \frac{1}{K} \sum_{i=1}^{K} S(\zeta_i), \tag{8}$$

где $\zeta_i \in U$ – последовательность однородных переменных.

Вычисление $\langle 1/\rho_n \rangle$ осуществляется с использованием итеративного подхода.

$$\frac{1}{\rho_n} = \frac{1}{\int\limits_U S(\zeta) d\zeta} = \frac{1}{1-p} = \sum_{i=0}^{\infty} p^i, \tag{9}$$

где $p = 1 - \int_U S(\zeta) d\zeta$.

Интегрирование функции по единичному квадрату выполняется явно, если $|p| < 1$.

$$\langle 1/\rho_n \rangle = 1 + \sum_{i=1}^{\infty} \prod_{j=1}^{i} \langle p \rangle_j, \tag{10}$$

где $\langle p \rangle_j$ – начальные точки.

Во втором варианте подхода используется компромисс между дисперсией выборки и потерей энергии. Применяется параметр размера пробного набора. Определяется сколько вычислений тратится на оценку излучения от каустик, и контролируется, сколько каустик найдено. Вместо неограниченного количества пробных итераций использует фиксированное число выборок N. Выборки группируются в набор решений. Оценка обратной связи определяется по относительному числу вхождений каждого решения. Оценка общей пропускной способности в точке затенения $\vec{x}_1$:

$$\frac{1}{N} \sum_{m=1}^{M} n_m \frac{f(\vec{x}_2^{(m)})}{p(\vec{x}_2^{(m)})} \approx \frac{1}{N} \sum_{m=1}^{M} n_m f(\vec{x}_2^{m}) \frac{N}{n_m} = \sum_{m=1}^{M} f(\vec{x}_2^{m}), \tag{11}$$

где $x_2^{(m)}$ $(m = 1,\ldots,M)$ – набор решений.

По сравнению с первым подходом, этот вариант имеет фиксированное количество итераций. А также использует информацию, предоставленную всеми образцами.

Также предлагается функция ограничения, которая решает, как ограничение в уравнении (3) кодирует зеркальные конфигурации с помощью полувекторных проекций. Закон Снелла и закон отражения определяют направление рассеяния:

$$D_s(\vec{d}_i, \vec{n}, r) = \begin{cases} D_s^T(\vec{d}_i, \vec{n}), & r = 1, \\ D_s^T(\vec{d}_i, \vec{n}, r), & otherwise, \end{cases} \tag{12}$$

где $\vec{d}_i$ – направления падения света, $\vec{n}$ – нормали поверхности, r – относительный показатель преломления.

$$D_s^T(\vec{d}_i, \vec{n}) = 2\langle \vec{d}_i, \vec{n} \rangle \vec{n} - \vec{d}_i, \tag{13}$$

$$D_s^T(\vec{d}_i, \vec{n}, r) = \\ = -r(\vec{d}_i - \langle \vec{d}_i, \vec{n} \rangle \vec{n}) - \vec{n}\sqrt{1 - r^2(1 - \langle \vec{d}_i, \vec{n} \rangle^2)}. \tag{14}$$

Падающее и исходящее направления света взаимосвязаны следующим образом:

$$\vec{d}_0 = D(\vec{d}_i, \vec{n}, r). \tag{15}$$

Разница в сферических координатах векторов будет:

$$\vec{b}_i = \begin{pmatrix} \theta(D(\vec{x}_i\vec{x}_{i-1}, \vec{n}_i, r_i)) - \theta(\vec{x}_i\vec{x}_{i+1}) \\ \varphi(D(\vec{x}_i\vec{x}_{i-1}, \vec{n}_i, r_i)) - \varphi(\vec{x}_i\vec{x}_{i+1}) \end{pmatrix}, \tag{16}$$

$$\theta(\vec{d}_s) = \cos^{-1}(\vec{d}_{iz}), \tag{17}$$

$$\varphi(\vec{d}_s) = a\tan 2(\vec{d}_{iy}, \vec{d}_{ix}). \tag{18}$$

Для целенаправленного прогнозирования местоположения решений и определения отправных точек предлагается способ создания целевого начального предположения для особого случая отображенных по нормали поверхностей. Сначала карта нормалей не рассматривается, а находят-

Таблица 1. Методы, использованные при сравнении

Методы	База	Форма лепестка	Количество параметров
[7]	Модель освещения Кука–Торренса	Сдвинутое гамма-распределение	18
[8]	Дифракция	K-корреляционная модель	9
Наш метод	Гауссово поле высот	Гауссовский	10

Рис. 5. Зеркальная металлическая поверхность.

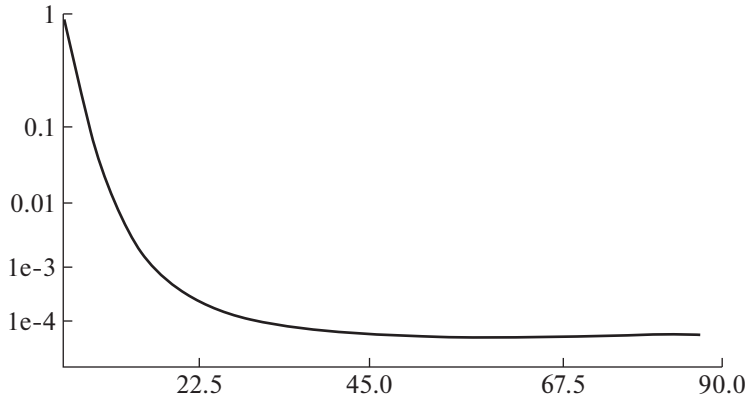

Рис. 6. Данные коэффициента отражения металлической поверхности в зависимости от угла между нормалью и полувектором модели BRDF.

ся зеркальные пути на исходной гладкой поверхности. Данный способ распространяется на карты смещения для представления деталей поверхности.

Метод также позволяет обрабатывать блики, которые представляют собой субпиксельные отражения света на зеркальной поверхности с микрогеометрией. Задача поиска бликов идентична случаю с каустикой. Отличием является то, что поиск ограничен небольшими областями поверхности. Для визуализации бликов используется двулучевая функция распределения света (BRDF) по площади пикселя. Генерируются случайные начальные точки внутри пикселя и происходит поиск решений, с помощью дискретного набора решений для вычисления пути $\overline{x} = \vec{x}_0, \vec{x}_1, \vec{x}_2$, соединяющего камеру с положением источника света с зеркальным отражением.

2.3. Алгоритм рендеринга

Визуализация бликов с использованием стандартных методов является чрезмерно дорогостоящей, поскольку для получения приемлемого результата могут потребоваться миллионы выборок на пиксель.

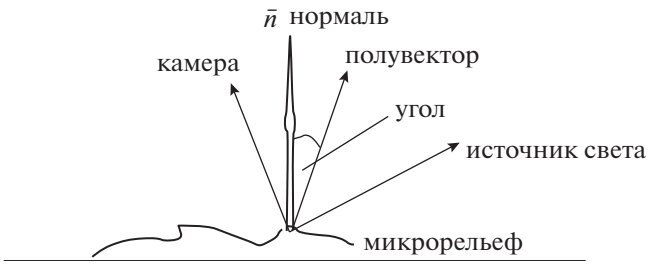

Рис. 7. Модель BRDF.

Таблица 2. Время визуализации

Изображение	Время визуализации
Рис. 2	1.8 мин
Рис. 3	4.5 мин
Рис. 5	34 сек
Рис. 9	5.2 мин

В нашем методе проблема рендеринга бликов, представляющих собой маленькие субпиксельные отражения источника света на высокочастотной зеркальной микрогеометрии, решается следующим образом. Генерируются случайные начальные точки внутри пикселя, а затем запускается алгоритм для поиска решений, показанный ниже.

Алгоритм рендеринга:

Вход: Точка затенения x_1 и положение излучателя x_3 с плотностью $\rho(x_3)$

1. $x_2 \leftarrow$ Выборка зеркальной вершины в качестве начальной позиции

2. $x_2'' \leftarrow$ Проход по многообразию (x_1, x_2, x_3)

3. $\langle 1/\rho_n \rangle \leftarrow 1$ Оценка обратной вероятности выборки x_2''

4. Цикл

5. $x_2 \leftarrow$ Выборка зеркальной вершины, как показано выше

6. $x_2' \leftarrow$ Проход по многообразию (x_1, x_2, x_3)

7. Если $\|x_2' - x_2''\| < \varepsilon$

8. Выход из цикла

9. $\langle 1/\rho_n \rangle \leftarrow \langle 1/\rho_n \rangle + 1$

10. Завершение выполнения функции

Результат: Оценка излучения, распространяющегося от x_3 до x_1.

Шаги 1–2 быстро сходятся к определенному решению, и требуется всего несколько итераций, чтобы найти то же самое решение еще раз в шагах 4–9. Двухэтапный подход к отбору проб выполняет два обхода. Первый этап игнорирует карту нормалей и находит зеркальные пути на исходной гладкой поверхности. Нормаль смещения выбирается перед каждым обходом и не меняется во время итерации, которая сходится быстро и с высокой вероятностью. Начальный случайный обход по многообразию приводит к близости различных решений. Оценка взаимной вероятности адаптированной выборки проста. Единственное требование состоит в том, чтобы двухэтапная выборка последовательно использовалась как в шагах 1–2, так и в шагах 4–9 алгоритма.

Если начальные точки $\langle p \rangle_j = 0$, (формула (10)), то обход по многообразию j сходится к корню ς^n, (формула (9)), если $\langle p \rangle_j = 1$, то процесс нашел другой корень или разошелся. То есть это процесс подсчета: выполняются повторные обходы многообразия до тех пор, пока не будет найдено ς^n. Количество испытаний дает объективную оценку $\langle 1/\rho_n \rangle$. Моделирование геометрического распределения обеспечивает оценку искомой обратной связи и составляет базовую компоненту алгоритма.

После отслеживания начального луча камеры проекционный размер пикселя аппроксимируется параллелограммом на основе дифференциалов лучей. Проход прекращается, если пути выходят за пределы параллелограмма. Конечные точки x0 и x2 находятся на расстоянии, и в этом случае изменения полувектора поперек параллелограмма минимальны. Таким образом, он устанавливается равным константе, что упрощает поиск решений. Так ограничение зеркального многообразия упрощается до функции. Используются карты нормалей высокого разрешения для кодирования деталей субпиксельной поверхности в реалистичных сценах.

Таким образом метод обобщается на зеркальные и шероховатые поверхности. Используется смещение распределения нормалей материала. Выполняется поиск зеркальных областей, включающих нормаль смещения вместо истинной нормали затенения. Метод обрабатывает шероховатость с использованием интегрирования по множеству зеркальных световых путей с нормалями смещения. Метод эффективно обрабатывает большой ряд зеркальных точек с множеством зеркальных взаимодействий.

3. РЕЗУЛЬТАТЫ

Тестирование производилось на компьютере с процессором Intel Core2 CPU E8400 3.0 GHz, и графическом процессоре GeForce GTX 470. Изображения были визуализированы с разрешением 1024 4 768 пикселей.

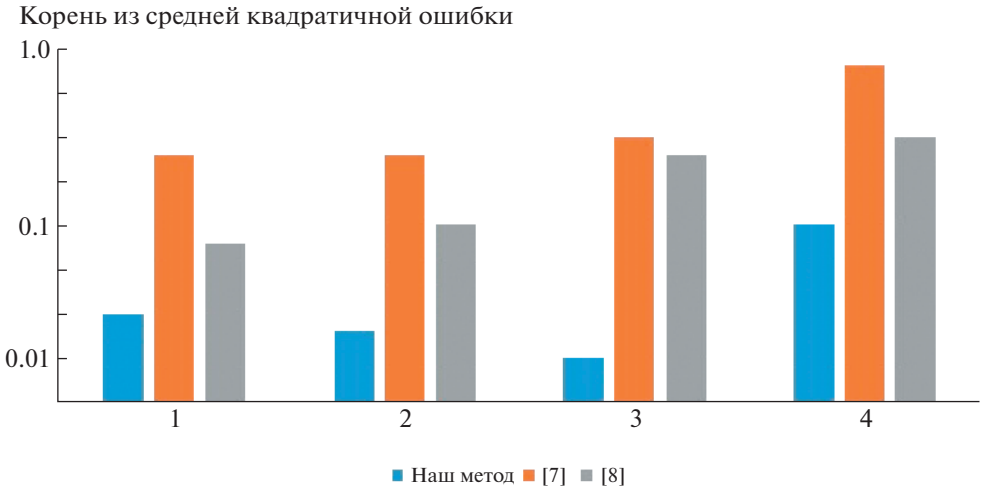

Рис. 8. Количественные показатели (корень из средней квадратичной ошибки) в BRDF для нескольких материалов.

Рис. 9. Комбинированная сцена каустики и шероховатой поверхности.

Сцена колец (рис. 2) показывает, как прямые, так и косвенные каустики, создаваемые отражением от кольца на зеркальной поверхности с диффузно-зеркальной составляющей $\alpha = 0.1$. Изображение демонстрирует каустики, созданные при отражении света внутри колец. Диффузно-зеркальная составляющая поверхности изменяется от 1 (диффузная) до 0 (зеркальная). Параметр α определяет, является ли поверхность диффузной или зеркальной. Такая модель была представлена Шликом в [10]. Эта модель проста и обладает полезным свойством, заключающимся в том, что она обеспечивает непрерывный переход от ламбертовского отражения к глянцевому зеркальному отражению.

Каустика формируется путем размещения источника света на краях колец, которые имеют отражающие внутренние стороны. Входящее направление света на краю каустики касательно к ней. Эта информация полезна, когда удаляем ламбертовское допущение с приемной поверхно-

сти. Это позволяет предсказать, как должна выглядеть каустика по мере того, как поверхность становится более глянцевой. Интенсивность каустики снижается в тех частях, где поступающее направление света отличается от направления входящего луча обзора. Время рендеринга увеличивается по мере того, как поверхность становится более глянцевой, так как алгоритм выборки изображения требует большего количества путей для рендеринга каустики на глянцевой поверхности. Использование фиксированного количества выборок на пиксель делает время рендеринга одинаковым для всех изображений.

Другой тестовый пример (рис. 3) представляет собой более сложную сцену, в которой использована не ламбертовская модель отражения. Изображен стеклянный сосуд с водой с каустикой. Образуется каустика, когда свет проходит через стекло и воду. На рис. 4 графики показывают дисперсию внутри каустики и снаружи как функцию от числа выборок. Диффузно-зеркальная составляющая – $\alpha = 0.1$. Выполняется оценка яркости в пикселях снаружи и внутри каустики. Дисперсия пикселей эквивалентна дисперсии оценки яркости. Графики показывают, что асимптотическое приближение хорошо предсказывает дисперсию.

Метод также эффективен при визуализации бликов со сложной микроструктурой, заданной с использованием карт нормалей (металлическая поверхность с очень сильной анизотропией (рис. 5)). Измеренные данные коэффициента отражения металлической поверхности в зависимости от угла между нормалью и полувектором, перенормированные по значению в начале координат, показаны на рисунке 6. Модель BRDF Кука–Торренса показана на рис. 7.

В работе рассмотрены количественные показатели (корень из средней квадратичной ошибки). Наша модель обеспечивает очень хорошее представление для глянцевых, шероховатых и зеркальных материалов. Модель имеет меньший показатель корня из средней квадратичной ошибки, чем другие тестируемые модели для нескольких материалов (см. рис. 8).

Мы сравнили нашу модель с двумя эталонными моделями: распределением гамма-излучения со смещением [7] и моделью плавного отражения [8] (табл. 1).

На рисунке 9 показана комбинированная сцена каустики и шероховатой поверхности, видно, что каустика от стеклянного сосуда не размывает шероховатый узор металлической поверхности (количество выборок на пиксель – 10).

Модели, основанные только на дифракции, обеспечивают хорошее описание диффузных и глянцевых материалов. Модели, основанные на модели освещения, как правило, лучше смотрятся на блестящих и зеркальных материалах.

Представленные в работе изображения визуализированы в нереальном времени с разрешением экрана 1024 × 768. Время визуализации зависит от многих факторов, главным образом от числа выборок и количества вершин зеркальных путей (рис. 1). В таблице 2 показано время визуализации для тестовых сцен.

4. ЗАКЛЮЧЕНИЕ

Представлен комбинированный метод визуализации каустик и шероховатых поверхностей. Рассмотрены рассеяние света на поверхностях, сложные каустики, многократное преломление света и блики на зеркальных поверхностях. Реализована эффективная техника выборки зеркального пути, которая сочетает в себе детерминированный поиск корней со стохастической выборкой. Базовый метод реализован в двух вариантах. Для первого варианта алгоритма представлено несколько дополнительных расширений и оптимизаций. Например, отбор проб путей улучшает сходимость. Введение ограничений многообразия увеличивают размер областей конвергенции в пространстве первичной выборки. Для генерации зеркальных бликов применяется простой итеративный алгоритм, для которого нет необходимости в использовании операций трассировки лучей.

ФИНАНСИРОВАНИЕ

Работа выполнена в рамках проекта № 121041800012-8 государственного задания.

СПИСОК ЛИТЕРАТУРЫ

1. *Фролов В.А., Волобой А.Г., Ершов С.В., Галактионов В.А.* Современное состояние методов расчета глобальной освещенности в задачах реалистичной компьютерной графики // Труды Института системного программирования РАН. 2021. Т. 33. № 2. С. 7–48. https://doi.org/10.15514/ISPRAS-2021-33(2)-1
2. *Hachisuka T., Ogaki S., Jensen H.W.* Progressive Photon Mapping // ACM Transactions on Graphics. 2008. V. 27. № 5. P. 1–8. https://doi.org/10.1145/1457515.1409083
3. *Sik M., Krivanek J.* Implementing One-Click Caustics in Corona Renderer // Eurographics Symposium on Rendering - Industry Track. T. Boubekeur and P. Sen (Editors). 2019. https://doi.org/10.2312/sr.20191221
4. *Hu W., Dong Z., Ihrke I., Grosch T., Yuan G., Seidel H.-P.* Interactive Volume Caustics in Single-Scattering Media // In Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. 2010. P. 109–117. https://doi.org/10.1145/1730804.1730822
5. *Celestino S., Romano D., Laccetti G., Lapegna M.* Bidirectional Path Tracing. A rendering method with Global Illumination on GPU // Applied Mathematical Sci-

ences. 2014. V. 8. № 133. P. 6783–6790. https://doi.org/10.12988/ams.2014.49694

6. *Jensen H.W.* Realistic image synthesis using photon mapping. Publisher: A. K. Peters, Ltd.63 South Avenue Natick, MA United States. 2001. 181 p. ISBN:978-1-56881-147-5
7. *Bagher M.M., Soler C., Holzschuch N.* Accurate fitting of measured reflectances using a Shifted Gamma micro-facet distribution // Computer Graphics Forum. 2012. V. 31. № 4. P. 1509–1518. https://doi.org/ https://doi.org/10.1111/j.1467-8659.2012.03147.x
8. *Löw J., Kronander J., Ynnerman A., Unger J.* BRDF models for accurate and efficient rendering of glossy surfaces // ACM Transactions on Graphics. 2012. V. 31. № 1. P. 1–14. https://doi.org/ https://doi.org/10.1145/2077341.2077350
9. *Вяткин С.И.* Моделирование сложных поверхностей с применением функций возмущения // Автометрия. 2007. Т. 43. № 3. С. 40–47. Vyatkin S.I. Complex Surface Modeling Using Perturbation Functions // Optoelectronics, Instrumentation and Data Processing. 2007. V. 43. № 3. P. 40–47.
10. *Schlick C.* A Customizable Reflectance Model for Everyday Rendering // In proceedings of four Eurographics Workshop on Rendering. 1993. P. 73–84. Corpus ID: 18967314

EDN: OCXQIY
ПРОГРАММИРОВАНИЕ, 2022, № 5, с. 37–53

ПРОГРАММНАЯ ИНЖЕНЕРИЯ, ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ ПРОГРАММ

УДК 004.052.42

Работа над статьей была завершена после кончины Валерия Александровича Непомнящего. Статья посвящается моему учителю, коллеге и соавтору Валерию Александровичу Непомнящему.

АВТОМАТИЗАЦИЯ ДЕДУКТИВНОЙ ВЕРИФИКАЦИИ С-ПРОГРАММ БЕЗ ИСПОЛЬЗОВАНИЯ ИНВАРИАНТОВ ЦИКЛОВ

© 2022 г. Д. А. Кондратьев[a,*] (http://orcid.org/0000-0002-9387-6735),
В. А. Непомнящий[a] (http://orcid.org/0000-0003-1364-5281)

[a] *Институт систем информатики им. А.П. Ершова СО РАН*
630090 Новосибирск, пр. Ак. Лаврентьева, 6, Россия

*****E-mail: apple-66@mail.ru*

Поступила в редакцию 01.12.2021 г.
После доработки 22.03.2022 г.
Принята к публикации 28.03.2022 г.

Автоматизация верификации С-программ – актуальная проблема современного программирования. Для ее решения необходимо автоматизировать решение проблемы инвариантов циклов, доказательство условий корректности и локализацию ошибок в случае ложных условий корректности. В Институте систем информатики СО РАН разрабатывается система C-lightVer, использующая комплексный подход к автоматизированной дедуктивной верификации С-программ. Данный подход включает символический метод верификации финитных итераций для элиминации инвариантов циклов, стратегии доказательства условий корректности и метод локализации ошибок. Символический метод верификации финитных итераций основан на замене действий циклов определенного вида применением специальной рекурсивной функции rep. Метод локализации ошибок основан на сопоставлении условий корректности с исходным кодом и генерации текста о соответствии условий корректности и фрагментов программы. Естественно возникает задача автоматизации верификации С-программ с вложенными циклами. Применение символического метода верификации финитных итераций для таких программ приводит к композиции функций rep для внешнего и вложенного циклов. Новым результатом этой статьи является стратегия автоматизации доказательства таких условий корректности. Данная стратегия основана на индукции по номеру итерации внешнего цикла. Для доказательства шага индукции мы используем другой результат этой статьи, которым является стратегия для программ, спецификации которых содержат функции со свойством конкатенации. В данной статье также представлены стратегии локализации ошибок и модификация метода локализации ошибок для случая вложенных циклов. Эти стратегии используются для проверки выполнения свойств циклов, которые могут означать наличие ошибок. В качестве примера применения наших результатов рассматривается автоматическая верификация сортировки простыми вставками без инвариантов циклов.

DOI: 10.31857/S0132347422050053

1. ВВЕДЕНИЕ

Актуальной проблемой современного программирования является автоматизация дедуктивной верификации программ [1–3]. Для решения этой проблемы необходимо автоматизировать задание инвариантов циклов [4, 5], доказательство условий корректности (УК) [6] и локализацию ошибок [7] в случае ложных УК.

В Институте систем информатики СО РАН разрабатывается система C-lightVer [8–11], основанная на комплексном подходе к автоматизации дедуктивной верификации С-программ. Данный подход включает символический метод верификации финитных итераций для элиминации инвариантов циклов [12], стратегии доказательства [9, 10] для проверки УК на истинность и метод локализации ошибок [7, 11].

Метод локализации ошибок основан на добавлении в правила вывода УК семантической разметки для объяснения результатов применения правил. Генератор УК в процессе вывода добавляет к различным подформулам соответствующие метки, которые извлекаются из УК и переводятся в текст о соответствии УК и фрагментов программы.

Для автоматизации дедуктивной верификации в системе C-lightVer используется ориентация на класс программ, осуществляющих финитные итерации над последовательностями данных [12]. Тело цикла финитной итерации исполняется один раз для каждого элемента последовательности данных. В символическом методе верификации финитных итераций используется специальное правило вывода УК для таких итераций, которое основано на применении операции замены (функции *rep*), выражающей действие цикла в символической форме. Функция *rep* определяется рекурсивно по номеру итерации [9]. Таким образом, УК программ с финитными итерациями могут содержать *rep*.

Но системы доказательства не всегда справляются в автоматическом режиме с доказательством УК, содержащих *rep*. Поэтому, были разработаны стратегии автоматизации доказательства таких УК [9, 10], реализованные для системы ACL2 [13].

Естественно возникает задача автоматизации верификации С-программ с вложенными циклами, например, программ линейной алгебры, программ сортировки массивов и программ, реализующих интерфейс BLAS. Применение символического метода верификации финитных итераций для таких программ приводит к композиции функций *rep* для внешнего и вложенного циклов. В данной статье описана стратегия автоматизации доказательства таких УК, которая основана на индукции по номеру итерации внешнего цикла. Для доказательства шага индукции разработана стратегия для программ, спецификации которых содержат функции со свойством конкатенации. Эта стратегия описана в разделе 4.1.

Также в данной статье описаны стратегии локализации ошибок и модификация метода локализации ошибок для случая вложенных циклов. Эти стратегии используются для проверки выполнения свойств циклов, которые могут означать наличие ошибок. Наш подход к локализации ошибок позволяет генерировать объяснения для таких случаев.

В качестве примера применения наших результатов в данной статье была использована сортировка массивов простыми вставками.

Данная статья состоит из шести разделов. Комплексный подход к автоматизации дедуктивной верификации описан во втором разделе. В третьем разделе описан метод автоматизации локализации ошибок в программах с финитными итерациями. Метод автоматизации доказательства УК программ с финитными итерациями описан в четвертом разделе. В пятом разделе описаны эксперименты по автоматизированной локализации ошибок и автоматической верификации программы сортировки простыми вставками.

1.1. Обзор литературы

Рассмотрим работы в области автоматизации решения проблемы инвариантов циклов. Подход, похожий на символический метод верификации финитных итераций, был предложен в работе [14], где предлагается заменять действие цикла на результат применения рекурсивной функции. Но в этой работе рассматривается модельный язык, более простой, чем язык C-light. Подобный подход [15] также предложен для Scala-программ. Но в работе [15] предлагается использовать инварианты, которые транслируются в аннотации соответствующих циклам рекурсивных функций. В отличие от символического метода верификации финитных итераций, большинство работ в этой области основано на генерации инвариантов циклов. Отметим работу [16], в которой предложен метод генерации инвариантов для класса циклов, называемых P-разрешимыми. Но, в отличие от финитных итераций, правые части присваиваний в теле P-разрешимых циклов должны иметь вид полиномов. В работе [17] предлагается генерировать инварианты специального вида для циклов над массивами, но циклы с инструкцией `break` не рассматриваются. Рассмотрим, как решается проблема инвариантов для программ сортировки. В работе [18] предложен динамический метод, основанный на известной идее модификации постусловия, но свойство перестановочности результирующего массива по отношению к исходному массиву не было доказано. В работе [19] предложено использовать шаблоны инвариантов, задаваемые пользователями, однако, доказано более слабое свойство, чем свойство перестановочности. В работе [20] предложено задавать инварианты специального вида, которые могут быть проще полных инвариантов и позволяют автоматизировать доказательство некоторых свойств. Этот подход применен для автоматизации верификации программы, реализующей обратную перестановку. Но в работе [20] для этой программы доказано выполнение более слабого свойства, чем свойство перестановочности.

Рассмотрим работы в области автоматизации доказательства УК. Подход [21] состоит в генерации лемм, которые могут помочь доказать целевую теорему. Такой подход для системы доказательства ACL2 предложен в работе [22]. В отличие от предложенных нами стратегий доказательства для системы ACL2, стратегии [22] основаны на машинном обучении. Но машинное обучение не подходит для автоматизации доказательства УК, так как предметные области могут отличаться для разных программ. Рассмотрим проблему автоматизации доказательства УК для программ сортировки вставками. Работа [23] была первой работой по использованию систем автоматизированного доказательства для верификации сортировки

вставками, но не все УК были доказаны автоматически. В работе [24] были предложены стратегии-доказательства, которые позволили автоматизировать доказательство некоторых УК. В работе [25] были предложены стратегии-автоматизации верификации свойства перестановочности. Но решение проблемы инвариантов циклов не было автоматизировано в работах [23–25].

Рассмотрим работы со специализированными стратегиями, ориентированными на упрощение формальной верификации. В работе [26] предложено задавать для циклов предусловия и постусловия специального вида вместо инвариантов. В работе [27] рассматривается метод леммафункций, реализованный в системе AstraVer. Этот метод основан на использовании спецификаций специального вида вместо инвариантов циклов. В работе [28] описан похожий метод, реализованный в системе Frama-C [29]. Но эти методы [26–28] основаны на задании спецификаций пользователем. Рассмотрим задание специализированных стратегий, ориентированных на упрощение верификации программ сортировки. В работе [30] на базе модельного функционального языка предложена стратегия для автоматизации верификации сортировки с помощью дизъюнктов Хорна.

Рассмотрим работы об автоматизации локализации ошибок при дедуктивной верификации программ. В работе [31] описано использование контрпримера, сгенерированного SMT-решателем, для локализации ошибок. Но анализ контрпримера может оказаться достаточно сложным, что продемонстрировано в работе [32]. В работе [7] предложен способ установления соответствия между УК и исходным кодом. В работе [33] описан новый подход в системе Frama-C [29] к автоматизации локализации ошибок при дедуктивной верификации, основанный на изменении выражений программы. В работе [34] предложена логика, названная некорректной логикой разделения. Истинность специальных формул в этой логике означает наличие ошибок в программе. Но авторы [34] предложили сложную модель памяти, приводящую к генерации сложных формул в отличие от используемого нами метода смешанной аксиоматической семантики [8]. Отметим работу [35] в области автоматизации локализации ошибок при дедуктивной верификации программ сортировки. В этой работе описана локализация ошибки при верификации реализации сортировки в Java-машине OpenJDK. Ошибка была локализована с помощью доказательства невыполнения инварианта в определенных случаях. Но решение проблемы инвариантов циклов не было автоматизировано в работах [33–35].

Рассмотрим работы про обоснование актуальности автоматизации дедуктивной верификации программ. В работе [36] предложен единый подход к доказательству корректности программ и к доказательству наличия ошибок в программах. В данной статье мы также используем общий подход к этим проблемам. Но наш подход основан на доказательстве выполнения свойств о функциях замены для циклов и мы, в отличие от работы [36], рассматриваем программы с вложенными циклами. В работах [37, 38] описана актуальность автоматизации дедуктивной верификации и путь к этой цели при разработке систем верификации Dafny-программ и Ada-программ. Из работ [39, 40] о верификации ядра Linux следует актуальность автоматизации дедуктивной верификации для таких задач. В работах [41, 42] содержатся рекомендации по добавлению автоматизированной локализации ошибок в систему AutoProof для верификации Eiffel-программ. В работе [43] описана новая система дедуктивной верификации C-программ, в которой используется автоматизация верификации с помощью доказателя теорем Coq. Но в этой работе не рассматривается решение проблемы инвариантов циклов. Отметим работу [44], в которой содержится обзор, подтверждающий актуальность автоматической верификации программ сортировки простыми вставками.

2. КОМПЛЕКСНЫЙ ПОДХОД К АВТОМАТИЗАЦИИ ДЕДУКТИВНОЙ ВЕРИФИКАЦИИ ПРОГРАММ

В Институте систем информатики СО РАН разрабатывается комплексный подход к автоматизации формальной верификации C-программ. Данный подход реализован в системе C-lightVer.

2.1. Система C-lightVer

Входным языком системы C-lightVer является язык C-light [45]. Данный язык является представительным подмножеством языка C. Для языка C-light была разработана операционная семантика [45]. C-kernel [46] – промежуточный язык верификации в системе C-lightVer. Данный язык является ограниченным подмножеством языка C-light. Для языка C-kernel определена аксиоматическая семантика [46].

Первым этапом исполнения программной системы C-lightVer является трансляция из C-light в C-kernel. Эта трансляция основана на наборе правил трансляции из конструкций языка C-light в эквивалентные конструкции C-kernel. На втором этапе генерируются УК полученной C-kernel программы. Родственный подход применяется в системе верификации Reflex-программ, где в качестве промежуточного языка рассматривается расширенный язык C-kernel [47].

В работе [8] описан метод смешанной аксиоматической семантики, который позволяет использовать специальные версии правил вывода

Рис. 1. Система C-lightVer.

для частных версий программных конструкций. Отметим, что в C-программах есть переменные, к которым не применяются операции взятия адреса и разыменования указателя. Для таких переменных можно использовать более простую модель памяти, чем модель, основанную на конструкциях взятия адреса и разыменования указателей. Поэтому, для конструкций над такими переменными можно применять более простые правила вывода. Это позволяет упростить УК.

Система верификации C-lightVer базируется на классическом дедуктивном подходе Хоара [5]. Схема системы C-lightVer изображена на рис. 1. Система состоит из семи основных модулей:

1. Модуль трансляции из C-light в C-kernel принимает на вход аннотированную C-light программу и производит

• трансляцию в эквивалентную аннотированную C-kernel программу;

• добавление информации [8], позволяющей транслировать конструкции C-kernel программы в конструкции C-light программы.

Эта информация добавляется к конструкциям, измененным в результате трансляции. Это позволяет снабдить конструкции названиями примененных правил трансляции.

2. Модуль генерации УК порождает формулы, из истинности которых следует корректность программы. Если программа содержит финитные итерации [12], то данный модуль генерирует операции замены для них.

3. Модуль управления доказательством УК применяет стратегии доказательства УК и запускает генерацию лемм, соответствующих данным стратегиям.

4. Модуль генерации лемм принимает на вход либо стратегию доказательства УК, либо стратегию локализации ошибок. Данный модуль генерирует леммы, соответствующие входным стратегиям. Леммы передаются на вход модулю доказательства теорем.

5. Модуль доказательства теорем проверяет истинность УК и лемм. В качестве данного модуля мы используем систему ACL2 [13].

6. Модуль локализации ошибок запускает исполнение стратегий локализации ошибок в случае недоказанного УК. Данный модуль генерирует объяснение недоказанных УК, используя семантические метки с информацией о соответствии между подформулами УК и конструкциями программы.

7. Модуль трансляции конструкций C-kernel в конструкции C-light основан на использовании ин-

формации, добавленной транслятором из C-light в C-kernel. Эта информация позволяет применить специальные правила трансляции [8] из C-kernel в C-light. Отчет о результатах верификации, который генерирует модуль локализации ошибок, основан на информации, хранящейся в семантических метках. Но метки содержат информацию не о конструкциях C-light программы, а о конструкциях C-kernel программы. Поэтому, используется данный модуль.

Рассмотрим этапы исполнения программной системы C-lightVer:

1. Аннотированная C-light программа транслируется в аннотированную C-kernel программу.

2. Генерируются УК полученной C-kernel программы.

3. Система ACL2 проверяет истинность УК. Доказанные УК передаются пользователю. Если все УК доказаны, то C-lightVer завершает исполнение.

4. Модуль управления доказательством последовательно применяет все подходящие стратегии для доказательства УК. Их применение состоит в генерации и доказательстве лемм. Доказанные леммы добавляются в теорию предметной области, которая используется в ACL2. Если применение стратегий привело к доказательству всех УК, то C-lightVer завершает исполнение.

5. Модуль локализации ошибок последовательно применяет все подходящие стратегии локализации ошибок. Их применение состоит в генерации и доказательстве лемм. Данный модуль также генерирует отчет о локализации ошибок. Базовой частью отчета является текст о соответствии УК и фрагментов программы. Если ACL2 доказала лемму, порожденную стратегией локализации ошибок, то в отчет добавляется текст, представляющий эту стратегию.

6. Конструкции C-kernel программы заменяются на конструкции C-light программы в отчете о локализации ошибок. Также номера строк C-kernel программы заменяются на номера строк C-light программы. Полученный отчет о локализации ошибок в C-light программе передается пользователю.

2.2. Символический метод верификации финитных итераций

Рассмотрим следующий вид цикла: *for x in S do* $v := body(v, x)$ *end*, где S – последовательность данных, x – переменная типа “элемент S”, v – вектор переменных цикла, который не содержит x, и *body* представляет тело цикла, которое не изменяет x и которое завершается для каждого $x \in S$. Тело цикла может содержать только инструкции присваивания, инструкции *if* (возможно вложенные) и инструкции *break*. Такие циклы *for* называются финитными итерациями [12]. Пусть v_0 – вектор значений переменных v до исполнения цикла. Чтобы выразить эффект финитной итерации, определим операцию замены $rep(n, v, S, body)$, где $rep(0, v, S, body) = v_0$, $rep(i, v, S, body) = = body(rep(i-1, v, S, body), s_i)$ для каждого $i = 1, 2, \ldots, n$. Если оператор выхода из цикла сработал на итерации i $(1 \le i \le n)$, то финитная итерация продолжает свое исполнение, но вектор v не изменяется: $\forall j (i \le j \le n)\ rep(i, v, S, body) = rep(j, v, S, body)$.

2.3. Генерация операции замены

Рассмотрим генерацию операции замены на языке системы ACL2 [13] в случае отсутствия вложенных итераций.

Пусть S – одномерный массив из n элементов примитивных типов языка C-light и $S \in v$, где v – вектор изменяемых переменных финитной итерации. Рассмотрим специальный случай финитной итерации над одномерным массивом S: *for* $(i = 0; i < n; i++)$ $v := body(v, i)$, где тело цикла является допустимой конструкцией [9].

Допустимая конструкция – это один из следующих операторов языка C-kernel:

1. Пустой оператор, в том числе пустой блок.

2. Оператор выхода из цикла **break;**.

3. Оператор присваивания **a = b;**, где a – переменная простого типа, либо имеет вид $S[i]$, b – выражение языка C-kernel.

4. Условный оператор **if (a) b**, где a – выражение языка C-kernel, b – допустимая конструкция.

5. Условный оператор **if (a) b else c**, где a – выражение языка C-kernel, b и c – допустимые конструкции.

6. Блок $\{\mathbf{a_1\ a_2 \ldots a_{k-1}\ a_k}\}$, где a_r – допустимая конструкция для каждого $r: 1 \le r \le k$.

В вектор v входит S и переменные простого типа, которые могут изменяться в теле цикла.

Генерация операции замены основана на трансляции допустимых конструкций тела цикла в конструкции языка системы ACL2. Рассмотрим конструкцию (b^* (…(*var expr*)…) *result*), где конструкция вида (*var expr*) означает связывание переменной *var* со значением выражения *expr*. Выражение *expr* может зависеть от связанных ранее переменных. Значением b^* является значение *result*, которое может зависеть от связанных переменных. Значения переменных вектора изменяемых переменных v соответствуют значениям полей структуры *fr* типа *frame*. Поэтому для моделирования изменения значения переменной вектора

v мы связываем объект *fr* с новым объектом, который отличается от старого значением соответствующего поля. Для моделирования выхода из цикла мы используем булевское поле *loop–break* объекта *frame*. Это поле истинно только после срабатывания `break`. Для моделирования `break` мы используем связывание вида ((*when t*) *fr*). Так как в этом случае условием *when* является *t*, т.е. "истина", то такое связывание прекращает исполнение текущего блока *b** и возвращает *fr*.

Определим генератор операции замены как рекурсивную функцию *gen_rep* [11]:

- *gen_rep*(**empty statement**) = (*fr fr*)
- *gen_rep*(**break;**) = ((*when t*) *fr*)
- *gen_rep*(**c = b;**) = (*fr* (*change-frame fr* :*c b*))
- *gen_rep*(**a[i] = b;**) = (*fr* (*change-frame fr* :*a* (*update-nth i b fr.a*)))
- *gen_rep*(**if (c) b else d**) = (*fr* (*if c* (*b** (*gen_rep*(*b*)) *fr*) (*b** (*gen_rep*(*d*)) *fr*))) ((*when fr.loop-break*) *fr*)
- *gen_rep*($\{\mathbf{a}_1\ \mathbf{a}_2\ \ldots\ \mathbf{a}_{k-1}\ \mathbf{a}_k\}$) = (*fr* (*b** (*gen_rep*(a_1) … *gen_rep*(a_k)) *fr*)) ((*when fr.loop-break*) *fr*)

Результатом работы данного алгоритма для заданной финитной итерации является представление функции *rep* на языке системы ACL2, которая является операцией замены для этой финитной итерации.

2.4. Метод Денни и Фишера

Денни и Фишер предложили добавить в правила Хоара [6] семантическую разметку [7] для объяснения результата применения правила. Будем использовать обозначение $\lceil t \rceil_l$, означающее, что терму *t* сопоставляется метка *l*. Метки имеют вид *c*(*o*), где *c* – тип метки, *o* – диапазон строк. Денни и Фишер предложили несколько типов меток для разных видов подформул из спецификаций и программ. Каждому типу метки соответствует текстовый шаблон. Метод Денни и Фишера [7] состоит из следующих шагов:

1. Генерация УК с использованием правил вывода с семантической разметкой. При применении такого правила вывода в метках сохраняются номера строк кода исходной программы. В полученных УК подформулы помечены семантическими метками.

2. Задание порядка на семантических метках из УК с помощью создания специального списка меток. Это называется извлечением меток из УК. Денни и Фишер предложили извлекать метки из УК в порядке увеличения номеров соответствующих метке строк.

3. Генерация объяснения УК с помощью последовательного обхода полученного списка меток. Для каждой посещенной при обходе метки текст ее заполненного номерами строк шаблона добавляется к тексту, объясняющему УК.

2.5. Метод локализации ошибок

Рассмотрим работу модуля локализации ошибок системы C-lightVer. Данный модуль работает в том случае, если не удалось доказать истинность УК. В таком случае модуль локализации ошибок анализирует УК и применяет наш подход к локализации ошибок. Рассмотрим отличия нашего подхода [11] от подхода Денни и Фишера.

В нашем подходе используется не ограниченный, а произвольный набор концепций семантических меток. Эта возможность реализована как задание пользователем новых концепций семантических меток и правил вывода с этими метками. Для каждого типа метки пользователь системы верификации задает шаблон текста. Такие шаблоны подаются на вход генератору УК.

Для поддержки произвольных типов меток в системе C-lightVer язык описания правил вывода был расширен специальной конструкцией *label* [11], используемой для описания меток. Конструкция *label* имеет вид (*label t c*), где *t* – терм, снабженный меткой, а *c* – строка (тип метки).

В нашем подходе используется добавление семантических меток не только в подформулу УК, но и в определение *rep*. Во-первых, добавление семантических меток в тело функции *rep* позволяет автоматизировать верификацию циклов с помощью символического метода верификации финитных итераций [12]. Во-вторых, добавление семантических меток в тело функции *rep* позволяет сопоставлять финитные операции и определение операции замены для генерации подробных объяснений УК с применениями *rep*.

Так как определение операции замены представляет собой последовательность связываний в блоке *b** вектора изменяемых переменных с новыми значениями компонент, то для реализации семантической разметки была использована такая конструкция языка системы ACL2, как одновременное связывание. Эта конструкция позволяет связать с новым значением не только вектор изменяемых переменных, но и специальную переменную *label*, соответствующую семантической метке. Каждое такое связывание представляет собой присваивание новых значений метке

label и переменной *fr*, соответствующей вектору изменяемых переменных *v*. Метка *label* и структура *fr* связываются с конструкцией *cons*, которая создает пару значений. Первым компонентом этой пары является список, который содержит все атрибуты метки. Вторым компонентом этой пары являются новые значения компонент структуры *fr*. Это одновременное связывание позволяет снабдить семантической меткой присваивание новых значений вектору изменяемых переменных.

В нашем подходе используется извлечение семантических меток из УК не в порядке номеров строк, а порядке обхода дерева УК в глубину. Это позволяет улучшить генерацию текста в случае вложенных меток. Семантические метки из определения операции замены извлекаются с помощью обхода в глубину дерева кода функции *rep*. В случае применения функции *rep* в УК, дерево кода *rep* рассматривается как поддерево УК и включается в общий обход в глубину для извлечения меток.

В нашем подходе используются заданные пользователем шаблоны объяснений меток для генерации объяснения всего УК. При их задании можно использовать специальные символы для обозначения диапазона строк.

В нашем подходе используются стратегии локализации ошибок, которые проверяют, удовлетворяют ли конструкции программы определенным свойствам. Выполнение этих свойств может означать наличие ошибок в программе. Если применение стратегии приводит к доказательству выполнения какого-либо свойства, в объяснение УК добавляется текст о наличии соответствующей проблемы. Самым общим видом такой стратегии является проверка ложности УК. Ложность УК означает наличие ошибки в программе, либо в ее спецификациях. Но в системе ACL2 доказательство ложности формулы вызывает трудности. Так как переменные языка системы ACL2 находятся под квантором общности, то доказательство ложности УК приводит к появлению квантора существования. Поэтому, описанные стратегии не подходят для доказательства отрицания УК. Также необходимо, чтобы стратегия проверки ложности работала в случае цикла с инструкцией break. Поэтому, была разработана стратегия [11] проверки ложности недоказанного УК, содержащего операцию замены. Данная стратегия генерирует формулу, истинность которой означает ложность УК. Такая формула основана на импликации, где из посылок доказывается отрицание заключения. Актуальна задача создания набора автоматических стратегий проверки программ на наличие ошибок.

3. АВТОМАТИЗАЦИЯ ЛОКАЛИЗАЦИИ ОШИБОК ПРИ ДЕДУКТИВНОЙ ВЕРИФИКАЦИИ ПРОГРАММ

Для автоматизации локализации ошибок в программах с финитными итерациями мы расширили комплексный подход алгоритмом генерации операции замены для программ с вложенными циклами, стратегией поиска циклов с неиспользуемыми присваиваниями элементам массива и стратегией проверки исполнения инструкции `break` на первой итерации цикла.

3.1. Алгоритм генерации операции замены с семантическими метками для программ с вложенными циклами

Введем нумерацию финитных итераций в программе. Для первой финитной итерации генерируется функция rep_1, для второй генерируется rep_2 и т.д. Расположение финитных итераций в программе можно представить как последовательность деревьев, корнями которых являются финитные итерации на самом верхнем уровне вложенности программы. Потомками любой вершины этих деревьев являются циклы, вложенные в цикл, соответствующий вершине. Сначала нумеруются итерации из первого дерева вложенности, потом нумеруются итерации из второго дерева вложенности и т.д. Итерации внутри каждого дерева вложенности нумеруются путем обхода в ширину. При переходе к следующему дереву вложенности нумерация продолжается.

Определим трансляцию вложенных итераций на язык системы ACL2 с помощью функции *gen_rep*:

gen_rep(
for(i = 0; i < n; i + +) v := body(v, i) end)) =
((*cons* ?!*label fr*)
(*cons* (*list 'inner_iter begin end 'break_path*)
(rep_i *n* env_i fr_i))),

где

• '*inner_iter* – тип семантической метки, предложенный нами для вложенной финитной итерации;

• *begin*/(*end*) – специальная конструкция. На место этой конструкции алгоритм генерации операции замены вставляет номер первой (последней) строки вложенной итерации. Такая конструкция позволяет записывать начало (конец) диапазона строк итерации в список атрибутов метки. Хранение номеров диапазона строк итерации в семантической метке позволяет сопоставить применение функции rep_i и фрагмент программы;

• '*break_path* – опциональный атрибут метки. Алгоритм генерации операции замены добавляет

этот атрибут в список, когда вложенная итерация лежит на пути к одному из операторов выхода из цикла. Этот атрибут позволяет генерировать более подробные объяснения в случае, когда стратегия локализации ошибок позволяет доказать, что исполнение внешней итерации завершилось в результате исполнения `break`.

Конструкция, которую генерирует *gen_rep* для вложенной итерации, представляет собой одновременное связывание метки *label* со своим значением и вектора изменяемых переменных (структуры *fr*) со своим значением. Обе эти переменные связываются с парой, созданной конструкцией *cons*. Метка *label* связывается с первым элементом этой пары. Первым элементом этой пары является список атрибутов метки. Структура *fr* связывается со вторым элементом этой пары. Вторым элементом этой пары является применение функции rep_i к аргументам n, env_i, fr_i, где

- n – обозначение количества итераций,
- env_i – обозначение вектора неизменяемых переменных итерации i,
- fr_i – обозначение вектора изменяемых переменных итерации i.

Это одновременное связывание позволяет снабдить применение операции замены для вложенной итерации семантической меткой с соответствующими атрибутами.

Отметим, что определение операции замены для внешней итерации содержит применение функции *rep* для внутренней итерации. Эта зависимость усложняет автоматизацию доказательства и приводит к разработке новых стратегий доказательства УК, содержащих применение вложенных операций замены.

3.2. Стратегия поиска циклов с неиспользуемыми присваиваниями элементам массива

Пусть в цикле, реализующем финитную итерацию над массивом, содержатся присваивания элементам этого массива и значения элементов массива после исполнения цикла равны значениям элементов массива до исполнения цикла. Значит, эти присваивания могут быть неиспользуемыми операциями. Такая ситуация может означать наличие ошибки. Поэтому, можно предупредить пользователей системы верификации о таких присваиваниях и о содержащем такие присваивания цикле.

Стратегия поиска таких циклов проверяет каждый цикл над массивом с присваиваниями элементам массива. Пусть такой цикл встречается i-тым в коде программы. Стратегия основана на генерации и проверке истинности леммы $P \to (a = rep_i(a, args).a)$, где

- P – предусловие,
- a – массив, над которым исполняется финитная итерация,
- rep_i – операция замены для финитной итерации,
- $args$ – аргументы rep_i, вычисленные с помощью обратного прослеживания,
- $rep_i(a, args).a$ – массив a после исполнения цикла.

Если такую лемму удалось доказать, то с помощью метода локализации ошибок генерируется текст с соответствующим предупреждением.

3.3. Стратегия проверки исполнения инструкции `break` *на первой итерации цикла*

Пусть в цикле, реализующем финитную итерацию над массивом, присутствует операция `break` и эта операция всегда исполняется на первой итерации цикла. Значит, вместо этого цикла можно задать последовательность операций, соответствующих инициализирующему выражению и телу цикла. Такая ситуация может означает наличие ошибки. Поэтому, можно предупредить пользователей системы верификации о такой операции `break`, об условии, при котором исполняется такая операция `break`, и о содержащем такую операцию `break` цикле.

Стратегия проверяет каждый цикл над массивом с инструкцией `break`. Пусть такой цикл встречается i-тым в коде программы. Стратегия основана на генерации и проверке истинности следующей леммы:

$$P \to ((j_0 = rep_i(a, args).j) \wedge \\ \wedge (rep_i(a, args).loop\text{-}break)),$$

где

- P – предусловие,
- a – массив, над которым исполняется финитная итерация,
- j – счетчик цикла *for*, реализующего финитную итерацию,
- $args$ – аргументы rep_i, вычисленные с помощью обратного прослеживания. Также вычисляется j_0 – значение счетчика j до исполнения цикла,
- $rep_i(a, args).j$ – значение j после исполнения цикла,
- $loop\text{-}break$ – специальное поле в возвращаемой структуре данных. Его значение истинно тогда и только тогда, когда при исполнении цикла исполнилась инструкция `break`.

Таким образом, первый конъюнкт заключения леммы является утверждением, что исполнение цикла не изменило значения счетчика цикла. Второй конъюнкт из заключения является утвер-

ждением, что при исполнении цикла исполнилась инструкция `break`.

Если лемму удалось доказать, то вычислим условие исполнения `break` с помощью обратного прослеживания. Истинность леммы гарантирует, что итерации, следующие за первой, не исполняются. Отметим, что невозможно вычислить символически условие исполнения `break` на произвольной итерации цикла, так как неизвестно, сколько итераций исполнилось до этой итерации. Но, в случае истинности леммы, возможно символически вычислить условие исполнения break с помощью подстановки значения переменных цикла до итерации. Если лемма доказана, то с помощью метода локализации ошибок генерируется текст с предупреждением о такой инструкции `break`, об условии исполнения этой инструкции `break`, и о цикле с этой инструкцией `break`.

3.4. Автоматизация локализации ошибок для программ с вложенными циклами

Расширение комплексного подхода для локализации ошибок в случае вложенных циклов привело к модификации системы C-lightVer. Новые алгоритмы и стратегии были реализованы в соответствующих модулях данной системы.

Алгоритм генерации операции замены для программ с вложенными циклами был реализован в модуле генерации УК. Отметим, что этот алгоритм позволяет снабдить вложенные итерации специальными семантическими метками. Для такого типа меток мы предложили специальный шаблон текста с описанием структуры вложенной итерации. В случае программ с вложенными финитными итерациями данный текст добавляется к объяснению УК.

Стратегии локализации ошибок реализованы в модуле локализации ошибок и в генераторе лемм. Ранее модуль локализации ошибок только генерировал объяснения недоказанных УК [11]. Модифицированный модуль также запускает последовательное исполнение всех стратегий локализации ошибок в случае недоказанного УК. Если УК не удалось доказать, то к циклам, соответствующим операциям замены из этого УК, последовательно применяются стратегии:

• Стратегия поиска циклов с неиспользуемыми присваиваниями элементам массива;

• Стратегия проверки исполнения инструкции `break` на первой итерации цикла.

Если применение стратегии локализации ошибок позволяет доказать выполнение соответствующего свойства для цикла, то, можно предположить, что программа содержит ошибку. В таком случае модуль локализации ошибок запускает генерацию текста с предупреждением о возможной ошибке, используя шаблон для стратегии. Номера строк и выражения из программы подставляются в специальные места шаблона. Полученный текст добавляется к объяснению УК. Шаблоны для обеих рассмотренных стратегий – это вторая и третья часть отчета из раздела 5.2 без номеров строк и выражений программы.

4. АВТОМАТИЗАЦИЯ ДЕДУКТИВНОЙ ВЕРИФИКАЦИИ ПРОГРАММ БЕЗ ИНВАРИАНТОВ ЦИКЛОВ

Для автоматизации доказательства УК программ с финитными итерациями мы расширили комплексный подход стратегией для программ, спецификации которых содержат функции со свойством конкатенации, стратегией для программ с финитными итерациями над массивами и стратегией для программ с вложенными циклами.

4.1. Стратегия для программ, спецификации которых содержат функции со свойством конкатенации

Важным этапом автоматизации верификации является этап автоматизации доказательства УК. Доказательство УК, содержащих операцию замены для финитной итерации, основано на индукции по длине массива, над которым осуществляется данная итерация. При этом, использования индукции по длине массива не достаточно для доказательства УК в системе ACL2 [13], если итерация изменяет элементы массива или содержит инструкции `break`. Рассмотрим разработанную для таких случаев стратегию автоматизации доказательства.

Определим такое свойство предикатов, как свойство конкатенации. Будем говорить, что предикат R обладает свойством конкатенации, если для R выполнено свойство вида

$$R(i,k,u_1,\ldots,u_n) \wedge R(k+1,j,u_1,\ldots,u_n) \rightarrow \\ \rightarrow R(i,j,u_1,\ldots,u_n)$$

Будем говорить, что предикат R обладает свойством конкатенации со склейкой на границе по предикату f, если для R выполнено свойство следующего вида:

$$(R(i,k,u_1,\ldots,u_n) \wedge R(k+1,j,u_1,\ldots,u_n) \wedge \\ \wedge f(u_1[k],u_1[k+1]) \wedge \ldots f(u_m[k],u_m[k+1])) \rightarrow \\ \rightarrow R(i,j,u_1,\ldots,u_n)$$

Для каждого $m(1 \le m \le n)$ будем называть выражение $f(u_m[k],u_m[k+1])$ условием склейки на границе для свойства конкатенации.

Пусть A – исходный массив, B – массив, полученный в результате применения функции *rep* к массиву A. Рассматриваемая стратегия генериру-

ет утверждения о равенстве подмассивов массива A и подмассивов массива B. Эти подмассивы выбираются с помощью следующих эвристик для цикла *for*:

• если счетчик цикла с инструкцией `break` возрастает (убывает), то генерируется утверждение, что равны подмассивы массивов B и A, начинающиеся с итогового значения счетчика (с нуля) до длины массива минус 1 (до итогового значения счетчика);

• если i – счетчик цикла с выражениями вида $A[i + expr] = A[i]$, то генерируется утверждение, что подмассив массива B является результатом сдвига подмассива массива A;

• если счетчик цикла возрастает (убывает) на произвольной итерации цикла, то генерируется утверждение, что равны подмассивы массивов B и A, начинающиеся с текущего значения счетчика (с нуля) до длины массива минус 1 (до текущего значения счетчика).

Утверждения о равенстве этих подмассивов преобразуются в леммы, которые поступают на вход системе доказательства. В результате образуется множество D пар подмассивов, равенство которых удалось доказать. Первым элементом каждой пары является подмассив массива A, а вторым элементом является подмассив массива B, для которого удалось доказать равенство первому элементу пары.

Каждый предикат из постусловия проверяется на выполнение свойства конкатенации или свойства конкатенации со склейкой на границе по предикату, найденному с помощью синтаксического анализа теории предметной области. Для каждого предиката S, удовлетворяющего любому из этих свойств, выполним следующие действия. Рассматриваемая стратегия генерирует леммы о выполнении свойства S для вторых элементов пар D, исходя из выполнения свойства S для первых элементов пар D. Для предиката со свойством конкатенации со склейкой на границе стратегия генерирует леммы о выполнении условия склейки на границах вторых элементов пар D. Такие леммы помогают системе ACL2 доказать выполнение свойства S для подмассивов, полученных в результате объединения вторых элементов пар D. Следовательно, такие леммы заданы для автоматизации доказательства выполнения свойства конкатенации для результирующего массива.

4.2. Стратегия для программ с финитными итерациями над массивами

Пусть $a[i : j]$ – обозначение подмассива a от элемента с индексом i до элемента с индексом j (включая элементы $a[i]$ и $a[j]$). Пусть n – длина массива a. Пусть j-я итерация цикла является финитной итерацией над $a[0 : j - 1]$ и $j + 1$-я итерация является финитной итерацией над $a[0 : j]$. Такая ситуация может означать, что подмассив $a[j + 1 : n - 1]$ после исполнения j-й итерации равен подмассиву $a[j + 1 : n - 1]$ после исполнения $j + 1$-й итерации. В этом случае будем называть подмассив $a[j + 1 : n - 1]$ необработанной частью массива $a[0 \ldots n - 1]$. Аналогично будем называть подмассив $a[0 :]$ обработанной частью массива $a[0 : n - 1]$. Проверим истинность леммы о равенстве необработанной части массива после двух смежных итераций. В случае успеха добавим эту лемму в теорию предметной области.

Стратегия генерирует лемму для каждой финитной итерации. Пусть такой цикл встречается i-тым в коде программы. Сгенерируем лемму:

$$(P \wedge j \in N \wedge 0 < j < length(a)) \rightarrow$$
$$\rightarrow (rep_i(j, a, args).a)[j + 1 : length(a) - 1] =$$
$$= (rep_i(j + 1, a, args).a)[j + 1 : length(a) - 1],$$

где

• P – предусловие,

• a – массив, над которым исполняется финитная итерация,

• j – счетчик цикла *for*, реализующего финитную итерацию,

• N – множество натуральных чисел,

• $length$ – функция, вычисляющая длину массива,

• $args$ – аргументы rep_i, вычисленные с обратного прослеживания,

• $rep_i(j, a, args).a$ – массив a после j-той итерации,

• $rep_i(j + 1, a, args).a$ – массив a после $j + 1$-й итерации.

Такая лемма может быть полезна если предикаты из спецификации программы удовлетворяют свойству конкатенации. Так как свойство конкатенации определено с помощью разбиения массива на части, то доказательство равенства необработанных частей массива может облегчить доказательство выполнения свойства конкатенации.

4.3. Стратегия автоматизации доказательства условий корректности для программ с вложенными циклами

Рассмотрим случай, когда возрастание количества итераций внешнего цикла приводит к возрастанию части последовательности, обрабатываемой внутренним циклом. Тогда попытаемся доказать УК, используя индукцию по количеству итераций внешнего цикла.

Пусть УК имеет вид $P \rightarrow Q$, где

• Q – заключение УК, зависящее от rep_i,

• P – посылка УК,

• a – массив, над которым исполняется финитная итерация,

• n – длина массива,

• $args$ – аргументы rep_i, вычисленные с помощью обратного прослеживания,

• rep_i – функция замены для i-того цикла, тело которой содержит применение rep_{i+1}.

Стратегия заключается в доказательстве такого УК индукцией по n. Такая стратегия может быть полезна, если внутренний цикл определяет ту часть последовательности, над которой исполняется каждая итерация внешнего цикла. Тогда подстановка вместо операции замены для внешнего цикла ее определения приводит к формуле, где доказываемое свойство сформулировано относительно внутреннего цикла. Доказывать это свойство позволяет индукционная гипотеза о части последовательности, обрабатываемой внутренним циклом.

5. ЭКСПЕРИМЕНТЫ ПО АВТОМАТИЧЕСКОЙ ВЕРИФИКАЦИИ ПРОГРАММЫ СОРТИРОВКИ

Опишем эксперимент по автоматизированной локализации ошибки в сортировке простыми вставками и эксперимент по автоматической верификации данной сортировки.

5.1. Спецификации программы сортировки простыми вставками

Рассмотрим программу на языке C-light без инвариантов циклов, реализующую сортировку простыми вставками:

```
1. /* P */
2. void insertion_sort(int a[], int n){
3.     int k, i, j;
4.     for (i = 1; i < n; i++){
5.       k = a[i];
6.       for (j = i - 1; j >= 0; j-){
7.          if (a[j] <= k) break;
8.          a[j + 1] = a[j];}
9.       a[j + 1] = k;}}
10. /* Q */,
```

где спецификации записаны в конструкциях, задающих на языке C комментарии. Задание спецификаций в комментариях позволяет избежать влияния спецификаций на семантику исходной программы. P и Q обозначают предусловие и постусловие соответственно.

Отметим, что данная программа не содержит явных операций взятия адреса и разыменования указателей. Поэтому, метод смешанной аксиоматической семантики [8] позволяет нам использовать в данном случае простую модель памяти без конструкций взятия адреса и разыменования указателей. Это приводит к упрощению УК.

Также отметим, что данная программа относится к классу программ с вложенными циклами. Мы не задаем инварианты циклов для этой программы.

Спецификации и формулы в системе C-light-Ver задаются на языке системы ACL2, но для записи формул в этой статье мы используем классическую инфиксную нотацию.

Предусловие P является следующей формулой: $0 < n \le length(a_0) \wedge a = a_0$, где

• $length$ – функция, принимающая на вход массив и вычисляющая его длину,

• a_0 – вспомогательный массив, который хранит исходное значение массива a.

Вспомогательный массив используется для задания в постусловии исходного состояния массива a.

Постусловие Q является следующей формулой: $perm(0, n-1, a_0, a) \wedge ord(0, n-1, a)$, где

• $perm$ – предикат, задающий свойство перестановочности,

• ord – предикат, задающий свойство упорядоченности.

Конъюнкция свойств перестановочности и упорядоченности является свойством отсортированности массива.

Первым аргументом предиката *perm* является нижняя граница диапазона индексов. Вторым аргументом *perm* является верхняя граница диапазона индексов. Третьим и четвертым аргументами предиката *perm* являются массивы. Предикат *perm* проверяет, являются ли эти массивы перестановкой друг друга в диапазоне от нижней границы (включая нижнюю границу) до верхней границы (включая верхнюю границу).

Первым аргументом предиката *ord* является нижняя граница диапазона индексов. Вторым аргументом предиката *ord* является верхняя граница диапазона индексов. Третьим аргументом предиката *ord* является массив. Предикат *ord* проверяет, является ли этот массив упорядоченным по возрастанию в диапазоне от нижней границы (включая нижнюю границу) до верхней границы (включая верхнюю границу).

Предикаты *perm* и *ord* являются предикатами предметной области. Мы задали модуль для системы ACL2 с определением предиката *perm* и с теоремами о предикате *perm*. Этот модуль задан в файле “permut.lisp”. Данный файл доступен в репозитории [48]. Также мы задали модуль для системы ACL2 с определением предиката *ord* и с теоремами о предикате *ord*. Этот модуль задан в

файле “ordered.lisp”. Данный файл также доступен в репозитории [48].

5.2. Автоматизированная локализация ошибки в программе сортировки

Рассмотрим программу на языке C-light без инвариантов циклов, реализующую сортировку простыми вставками с внесенной ошибкой:

```
1. /* P */
2. void insertion_sort(int a[], int n){
3.     int k, i, j;
4.     for (i = 1; i < n; i++){
5.        k = a[i];
6.        for(j=i;j>=0;j-){
7.           if (a[j] <= k) break;
8.           a[j + 1] = a[j];}
9.        a[j + 1] = k;}}
10. /* Q */
```

Ошибка состоит в том, что инициализирующей инструкцией вложенного цикла `for` является инструкция `j=i` вместо инструкции `j=i-1`.

Следующее УК генерируется с помощью алгоритма генерации операции замены для программ с вложенными циклами:

$$0 < n \le length(a_0) \wedge a = a_0 \rightarrow$$
$$\rightarrow perm(0, n-1, a_0, rep_1(n, a, 1).a) \wedge$$
$$\wedge ord(0, n-1, rep_1(n, a, 1).a),$$

где

- rep_1 – операция замены для внешнего цикла,
- 1 – начальное значение счетчика внешнего цикла,
- $rep_1(n, a, 1).a$ – массив a после исполнения внешнего цикла.

Определения функций rep_1 и rep_2 снабжены семантическими метками в результате применения алгоритма генерации операции замены для программ с вложенными циклами. Определение функции rep_1 основано на применении операции замены rep_2 для внутреннего цикла. Ошибка в программе приводит к тому, что функция rep_2 применяется в теле функции rep_1 с неправильными аргументами. Вместо начального значения счетчика цикла $i-1$ в качестве аргумента функции rep_2 используется i.

Рассмотрим влияние ошибки в программе на исполнение i-той итерации внешнего цикла. Ошибка приводит к тому, что внутренний цикл всегда завершает исполнение на первой итерации из-за истинности условия инструкции *if*, содержащей `break` в положительной ветви. После завершения внутреннего цикла происходит присваивание элементу массива с индексом $i+1$ элемента массива с индексом i. В итоге, данное УК является истинным, когда элемент массива с индексом 0 имеет произвольное значение, а все остальные элементы совпадают и их значение не меньше значения нулевого элемента. Элемент с индексом 0 может отличаться от всех остальных, так как начальным значением счетчика внешнего цикла является 1, а не 0. Во всех остальных случаях это УК является ложным.

Система ACL2 не доказала истинность данного УК. Также система ACL2 не доказала ложность этого УК во всех случаях, используя проверку на истинность отрицания данного УК. Эта стратегия локализации ошибок не приводит к успеху из-за истинности УК в некоторых случаях. Модуль локализации ошибок в системе C-lightVer последовательно применяет стратегию поиска циклов с неиспользуемыми присваиваниями элементам массива и стратегию проверки исполнения инструкции `break` на первой итерации цикла.

Ошибка в программе приводит к тому, что вложенный цикл не изменяет массив a. Отметим, что вложенный цикл содержит присваивание элементам массива. Применение стратегии поиска циклов с неиспользуемыми присваиваниями элементам массива приводит к успешному доказательству соответствующей леммы. В репозитории [48] в файле “warnings.lisp” эта лемма задана как `warnings-lemma-11`.

Также ошибка в программе приводит к тому, что исполнение вложенного цикла всегда завершается на первой итерации. Инструкция `break` вложенного цикла всегда исполняется на первой итерации и вложенный цикл не изменяет значение счетчика цикла (переменной j). Поэтому, применение стратегии проверки исполнения инструкции `break` на первой итерации цикла приводит к успешному доказательству соответствующей леммы. В репозитории [48] в файле “warnings.lisp” эта лемма задана как `warnings-lemma-9`. Условие `a[i] <= a[i]` было символически вычислено, используя подстановку i вместо j и постановку $a[i]$ вместо k в условие инструкции *if*. Это условие исполнения инструкции `break` во вложенном цикле. Это условие вычислено для отчета о локализации ошибки. Отметим, что данное условие всегда истинно и является условием инструкции `if`. Эта информация является еще одним признаком наличия ошибки в программе.

В итоге, алгоритм генерации объяснений недоказанных УК для программ с вложенными циклами генерирует отчет, состоящий из трех частей. Рассмотрим первую часть отчета, которая является результатом применения метода локализации ошибок:

```
This formula corresponds to lines 1–10 in
function “insertion_sort”. This formula has
not been proven by C-ligthVer system using
ACL2 prover. Hence,
– the following explanation of this formula
    precondition from line 1
       implies
    postcondition goal from line 10
      with substitution loop effect
      from lines 4–9 by rep1
       that corresponds to the loop with
       initialization expression “i = 1”
       from line 4
       condition expression “i < n”
       from line 4
       iteration expression “i++”
       from line 4
       and the following loop body
       sequence of the following statements
       from line 5 to line 9
         assignment statement “k = a[i]”
         from line 5
         substitution of inner loop effect
         from lines 6–8 by rep2
         that corresponds to the inner loop
         with
           initialization expression “j=i”
           from line 6
           condition expression “j>=0”
           from line 6
           iteration expression “j--”
           from line 6 and
           the following loop body sequence of the following statements
           from line 7 to line 8
             if statement from line 7
               with condition “a[j] <= k”
               from line 7
               and with
               the following positive branch
               sequence of
               the following statements
               from line 7 to line 7
                 break statement
                 from line 7
           array update “a[j + 1] = a[j]”
           from line 8
       array update “a[j + 1] = k”
       from line 9
has been generated
```

Эта часть отчета позволяет сопоставить подформулы УК и исходную программу, а также сопоставить определение операции замены для внешнего цикла и исходную программу. Снабжение семантическими метками определения операции замены для внутренних циклов позволило сопоставить определение операции замены для внутреннего цикла и исходную программу.

Рассмотрим вторую часть отчета, которая является предупреждением о возможной ошибке, генерируемым стратегией поиска циклов с неиспользуемыми присваиваниями элементам массива:

```
- the following warning about inner loop
  assumption that precondition
  from line 1 holds,
  ensures that
  array update "a[j + 1] = a[j]"
  from line 8
  is always unused
has been generated
```

Эта часть отчета позволяет обратить внимание пользователя на вложенный цикл.

Рассмотрим третью часть отчета, которая состоит из двух предупреждений о возможной ошибке, генерируемых стратегией проверки исполнения инструкции `break` на первой итерации цикла:

```
- the following warning about inner loop
    assumption that precondition
    from line 1 holds,
    ensures that
    break statement
    from line 7 always executes
    at first iteration
  has been generated
- the following warning about inner loop
    assumption that precondition
    from line 1 holds,
    ensures that
    break condition "a[j] <= k"
    from line 7
    is always true at first iteration as
    "a[i] <= a[i]"
  has been generated
```

Эта часть отчета также позволяет пользователю обратить внимание на вложенный цикл. Кроме того, эта часть отчета позволяет обратить внимание пользователя на условие исполнения `break`.

Данный отчет позволяет обратить внимание пользователя на вложенный цикл и локализовать ошибку. Отметим, что пользователю не нужно просматривать всю программу, чтобы локализовать ошибку. Вместо изучения сложной структуры УК пользователю предоставляется текст о сопоставлении подформул УК и исходной программы. Комплексный подход к автоматизации дедуктивной верификации С-программ, реализованный в системе C-lightVer, позволяет упростить и автоматизировать локализацию ошибки в данном эксперименте.

5.3. Автоматическая верификации программы сортировки

Рассмотрим программу на языке C-light из раздела 5.1, реализующую сортировку простыми вставками.

Следующее УК генерируется с помощью алгоритма генерации операции замены для программ с вложенными циклами:

$$0 < n \le length(a_0) \wedge a = a_0 \rightarrow$$
$$\rightarrow perm(0, n-1, a_0, rep_1(n, a, 1).a) \wedge$$
$$\wedge ord(0, n-1, rep_1(n, a, 1).a),$$

где

- rep_1 – операция замены для внешнего цикла,
- 1 – начальное значение счетчика внешнего цикла,
- $rep_1(n, a, 1).a$ – массив a после исполнения внешнего цикла.

В репозитории [48] в файле “vc-1.lisp” данное УК задано как `vc-1-lemma-103`. Определение rep_1 задано в репозитории [48] в файле “rep1.lisp” как `rep1`. Определение rep_2 задано в репозитории [48] в файле “rep2.lisp” как rep2. Определение функции rep_1 основано на определении функции rep_2. Следовательно, УК содержит неявную композицию функций rep_1 и rep_2.

Система ACL2 не доказала истинность данного УК без применения стратегий доказательства. Так как спецификации УК содержат предикаты, которые могут удовлетворять свойству конкатенации, то модуль управления доказательством применил стратегию для программ, спецификации которых содержат функции со свойством конкатенации. Так как данное УК содержит вложенные циклы, то модуль управления доказательством применил стратегию для программ с финитными итерациями над массивами и стратегию для программ с вложенными циклами.

Спецификации рассматриваемой программы содержат функции со свойством конкатенации, так как для предиката *perm* выполнено свойство конкатенации и для предиката *ord* выполнено свойство конкатенации со склейкой на границе.

Предикат *perm* удовлетворяет свойству конкатенации, так как в теории предметной области содержится следующая лемма о свойствах *perm*:

$$(i \in N \wedge j \in N \wedge k \in N \wedge i \le k \wedge k < j \wedge$$
$$\wedge j < length(u) \wedge j < length(v) \wedge$$
$$perm(i,k,u,v) \wedge perm(k+1,j,u,v)) \rightarrow$$
$$\rightarrow perm(i,j,u,v),$$

где N – множество натуральных чисел. Выполнение свойства конкатенации автоматически доказывается благодаря этой теореме. В репозитории [48] в файле “permut.lisp” эта теорема задана как `permutation-7`.

Предикат *ord* удовлетворяет свойству конкатенации со склейкой на границе, так как в теории предметной области содержится следующая лемма о свойствах *ord*:

$$(i \in N \wedge j \in N \wedge k \in N \wedge i \le k < j \wedge$$
$$\wedge j < length(u) \wedge ord(i,k,u) \wedge ord(k+1,j,u) \wedge$$
$$\wedge u[k] \le u[k+1]) \rightarrow ord(i,j,u),$$

где N – множество натуральных чисел, $u[k] \le u[k+1]$ – условие склейки на границе для свойства конкатенации.

Эвристический анализ обнаружил в теле циклов инструкции присваивания элементам массива элементов того же самого массива. В результате, были сгенерированы леммы о равенстве следующих пар подмассивов:

1. $(rep_2(i,a,args).a)[0:j]$ и $a[0:j]$
2. $(rep_2(i,a,args).a)[j+2:i]$ и $a[j+1:i-1]$,

где

- a – массив, над которым исполняется финитная итерация,
- i – количество итераций внутреннего цикла в случае, если не исполнится break,
- *args* – аргументы rep_2, вычисленные с помощью обратного прослеживания,
- $rep_2(i,a,args).a$ – массив a после i-той итерации.

Равенство подмассивов $(rep_2(i,a,args).a)[0:j]$ и $a[0:j]$ означает, что вложенный цикл не изменил начало массива до этой позиции j. В репозитории [48] в файле “rep2.lisp” эта теорема задана как `rep2-lemma-76`.

Равенство подмассивов $(rep_2(i,a,args).a)[j+2:i]$ и $a[j+1:i-1]$ означает, что вложенный цикл привел к сдвигу на единицу последовательности $a[j+1:i-1]$. В репозитории [48] в файле “rep2.lisp” эта теорема задана как `rep2-lemma-55`.

Данные леммы о равенстве подмассивов были автоматически доказаны и добавлены в теорию предметной области.

Сравним результаты двух смежных итераций внешнего цикла. Результат следующей итерации отличается от результата предыдущей возрастанием отсортированной части массива на один элемент и убыванием неотсортированной части массива на один элемент. Отметим, что на смежных итерациях необработанные части последовательностей за вставляемым элементом равны. В данном случае необработанная часть последовательности внешнего цикла – это необработанная часть последовательности внутреннего цикла. Поэтому, применение стратегии для программ с финитными итерациями над массивами приводит к успешному доказательству соответствующей леммы и добавлению этой леммы в теорию предметной области. В репозитории [48] в файле “rep2.lisp” эта лемма задана как `rep2-lemma-53`.

Рассмотрим результат i-той итерации внешнего цикла. В результате такой итерации массив делится на две части: отсортированная часть до элемента с индексом i (включающая элемент с индексом i), и неотсортированная часть (не включающая элемент с индексом i). Значение i является границей между отсортированной частью массива и неотсортированной частью массива. Поэтому, в данном случае, стратегия автоматизации доказательства УК для программ с вложенными циклами является индукцией по границе между отсортированной и неотсортированной частью массива. Система ACL2 автоматически доказывает базу индукции.

Индукционная гипотеза является утверждением о выполнении свойств перестановочности, и упорядоченности от начала массива до i-го элемента (не включая i-тый) элемент. Леммы, порожденные примененными стратегиями, являются утверждениями о структуре массива после исполнения i-той итерации. Система ACL2 автоматически доказывает индукционный переход, используя данные леммы. Автоматическое доказательство базы индукции и индукционного перехода приводит к автоматическому доказательству данного УК.

Символический метод верификации финитных итераций позволяет избежать задания инвариантов для сортировки простыми вставками. Применение описанных стратегий доказательства позволяет автоматически доказать полученное УК. Итого, комплексный подход к автоматизации дедуктивной верификации С-программ, реализованный в системе C-lightVer, позволяет автоматически верифицировать программу сортировки простыми вставками на языке С.

6. ЗАКЛЮЧЕНИЕ

В данной статье представлены следующие основные результаты:

1. Описаны стратегии автоматизации доказательства УК:

• для программ с вложенными циклами;

• для программ, спецификации которых содержат функции со свойством конкатенации;

• для программ с финитными итерациями над массивами.

2. Описаны стратегии автоматизации локализации ошибок при дедуктивной верификации:

• поиска циклов с неиспользуемыми присваиваниями элементам массива;

• проверки исполнения инструкции `break` на первой итерации цикла.

Также представлены следующие алгоритмы:

1. Генерации операции замены для программ с вложенными циклами;

2. Генерации объяснений недоказанных УК программ с вложенными циклами для локализации ошибок.

Впервые проведен успешный эксперимент по автоматической верификации программы сортировки простыми вставками без инвариантов циклов. Также проведен эксперимент по автоматизированной локализации ошибки в этой программе.

Представленные результаты описывают расширение комплексного подхода к автоматизации дедуктивной верификации С-программ, реализованного в системе C-lightVer.

В будущем планируется расширить этот подход новыми стратегиями автоматизации доказательства УК и новыми стратегиями автоматизации локализации ошибок. Также планируется проведение экспериментов по автоматической верификации других классов программ с финитными итерациями и автоматизированной локализации ошибок в этих программах.

СПИСОК ЛИТЕРАТУРЫ

1. *Apt K.R., Olderog E.-R.* Assessing the Success and Impact of Hoare's Logic // Theories of Programming: The Life and Works of Tony Hoare. 2021. P. 41–76.
2. *Hähnle R., Huisman M.* Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools // Computing and Software Science. Lecture Notes in Computer Science. 2019. V. 10000. P. 345–373.
3. *Müller P., Shankar N.* The First Fifteen Years of the Verified Software Project // Theories of Programming: The Life and Works of Tony Hoare. 2021. P. 93–124.
4. *Furia C.A., Meyer B., Velder S.* Loop invariants: Analysis, classification, and examples // ACM Computing Surveys. 2014. V. 46. Is. 3. Article 34. P. 1–51.
5. *Apt K.R., Olderog E.-R.* Fifty years of Hoare's logic // Formal Aspects of Computing. 2019. V. 31. Is. 6. P. 751–807.
6. *Hoare C.A.R.* An axiomatic basis for computer programming // Communications of the ACM. 1969. V. 12. Is. 10. P. 576–580.
7. *Denney E., Fischer B.* Explaining verification conditions // Proc. AMAST 2008. Lecture Notes in Computer Science. 2008. V. 5140. P. 145–159.
8. *Maryasov I.V., Nepomniaschy V.A., Promsky A.V., Kondratyev D.A.* Automatic C Program Verification Based on Mixed Axiomatic Semantics // Automatic Control and Computer Sciences. 2014. V. 48. Is. 7. P. 407–414.
9. *Kondratyev D.A., Maryasov I.V., Nepomniaschy V.A.* The Automation of C Program Verification by the Symbolic Method of Loop Invariant Elimination // Automatic Control and Computer Sciences. 2019. V. 53. Is. 7. P. 653–662.
10. *Kondratyev D., Maryasov I., Nepomniaschy V.* Towards Automatic Deductive Verification of C Programs over Linear Arrays // Proc. PSI 2019. Lecture Notes in Computer Science. 2019. V. 11964. P. 232–242.
11. *Kondratyev D.A., Promsky A.V.* The Complex Approach of the C-lightVer System to the Automated Error Localization in C-Programs // Automatic Control and Computer Sciences. 2020. V. 54. Is. 7. P. 728–739.
12. *Nepomniaschy V.A.* Symbolic method of verification of definite iterations over altered data structures // Programming and Computing Software. 2005. V. 31. Is. 1. P. 1–9.
13. *Moore J.S.* Milestones from the pure lisp theorem prover to ACL2 // Formal Aspects of Computing. 2019. V. 31. Is. 6. P. 699–732.
14. *Myreen M.O., Gordon M.J.C.* Transforming programs into recursive functions // Electronic Notes in Theoretical Computer Science. 2009. V. 240. P. 185–200.
15. *Blanc R., Kuncak V., Kneuss E., Suter P.* An overview of the Leon verification system: verification by translation to recursive functions // Proc. 4th Workshop on Scala. 2013. Article 1. P. 1–10.
16. *Humenberger A., Jaroschek M., Kovás L.* Invariant Generation for Multi-Path Loops with Polynomial Assignments // Proc. VMCAI 2018. Lecture Notes in Computer Science. 2018. V. 10747. P. 226–246.
17. *Chakraborty S., Gupta A., Unadkat D.* Inductive Reasoning of Array Programs Using Difference Invariants // Proc. CAV 2021. Lecture Notes in Computer Science. 2021. V. 12760. P. 911–935.
18. *Galeotti J.P., Furia C.A., May E., Fraser G., Zeller A.* Inferring loop invariants by mutation, dynamic analysis, and static checking // IEEE Transactions on Software Engineering. 2015. V. 41. Is. 10. P. 1019–1037.
19. *Srivastava S., Gulwani S., Foster J.S.* Template-based program verification and program synthesis // International Journal on Software Tools for Technology Transfer. 2013. V. 15. Is. 5–6. P. 497–518.
20. *Filliâtre J.-C.* Simpler proofs with decentralized invariants // Journal of Logical and Algebraic Methods in Programming. 2021. V. 121. Article ID: 100645.
21. *Johansson M.* Lemma Discovery for Induction // Proc. CICM 2019. Lecture Notes in Computer Science. 2019. V. 11617. P. 125–139.
22. *Heras J., Komendantskaya E., Johansson M., Maclean E.* Proof-pattern recognition and lemma discovery in ACL2 // Proc. LPAR 2013. Lecture Notes in Computer Science. 2013. V. 8312. P. 389–406.
23. *Filliâtre J.-C., Magaud N.* Certification of Sorting Algorithms in the Coq System // Proc. conf. on "Theorem

Proving in Higher Order Logics: Emerging Trends". Nice. 1999.

24. *Imine A., Ranise S.* Building Satisfiability Procedures for Verification: The Case Study of Sorting Algorithms // Proc. LOPSTR'03. 2003.

25. *Safari M., Huisman M.* A Generic Approach to the Verification of the Permutation Property of Sequential and Parallel Swap-Based Sorting Algorithms // Proc. IFM 2020. Lecture Notes in Computer Science. 2020. V. 12546. P. 257–275.

26. *Tuerk T.* Local reasoning about while-loops // Proc. Theory Workshop at VSTTE 2010. 2010. P. 29–39.

27. *Volkov G., Mandrykin M., Efremov D.* Lemma functions for Frama-C: C programs as proofs // Proc. 2018 Ivannikov ISP RAS Open Conference. 2018. P. 31–38.

28. *Blanchard A., Loulergue F., Kosmatov N.* Towards full proof automation in Frama-C using auto-active verification // Proc. NFM 2019. Lecture Notes in Computer Science. 2019. V. 11460. P. 88–105.

29. *Baudin P., Bobot F., Bühler D., Correnson L., Kirchner F., Kosmatov N., Maroneze A., Perrelle V., Prevosto V., Signoles J., Williams N.* The dogged pursuit of bug-free C programs: the Frama-C software analysis platform // Communications of the ACM. 2021. V. 64. Is. 8. P. 56–68.

30. *de Angelis E., Fioravanti F., Pettorossi A., Proietti M.* Proving Properties of Sorting Programs: A Case Study in Horn Clause Verification // Proc. HCVS/PERR 2019. Electronic Proceedings in Theoretical Computer Science. 2019. V. 296. P. 48–75.

31. *Dailler S., Hauzar D., Marché C., Moy Y.* Instrumenting a weakest precondition calculus for counterexample generation // Journal of Logical and Algebraic Methods in Programming. 2018. V. 99. P. 97–113.

32. *Becker B., Lourenço C.B., Marché C.* Explaining Counterexamples with Giant-Step Assertion Checking // Proc. F-IDE 2021. Electronic Proceedings in Theoretical Computer Science. 2021. V. 338. P. 82–88.

33. *Könighofer R., Toegl R., Bloem R.* Automatic error localization for software using deductive verification // Proc. HVC 2014. Lecture Notes in Computer Science. 2014. V. 8855. P. 92–98.

34. *Raad A., Berdine J., Dang H.H., Dreyer D., O'Hearn P., Villard J.* Local Reasoning About the Presence of Bugs: Incorrectness Separation Logic // Proc. CAV 2020. Lecture Notes in Computer Science. 2020. V. 12225. P. 225–252.

35. *de Gouw S., de Boer F.S., Bubel R., Hähnle R., Rot J., Steinhöfel D.* Verifying OpenJDK's Sort Method for Generic Collections // Journal of Automated Reasoning. 2019. V. 62. Is. 1. P. 93–126.

36. *Möller B., O'Hearn P., Hoare T.* On Algebra of Program Correctness and Incorrectness // Proc. RAMiCS 2021. Lecture Notes in Computer Science. 2021. V. 13027. P. 325–343.

37. *Grebing S., Klamroth J., Ulbrich M.* Seamless Interactive Program Verification // Proc. VSTTE 2019. Lecture Notes in Computer Science. 2020. V. 12031. P. 68–86.

38. *Dailler S., Marché C., Moy Y.* Lightweight Interactive Proving inside an Automatic Program Verifier // Proc. F-IDE 2018. Electronic Proceedings in Theoretical Computer Science. 2018. V. 284. P. 1–15.

39. *Mandrykin M.U., Khoroshilov A.V.* Towards Deductive Verification of C Programs with Shared Data // Programming and Computing Software. 2016. V. 42. Is. 5. P. 324–332.

40. *Efremov D., Mandrykin M., Khoroshilov A.* Deductive verification of unmodified Linux kernel library functions // Proc. ISoLA 2018. Lecture Notes in Computer Science. 2018. V. 11245. P. 216–234.

41. *de Carvalho D., Hussain R., Khan A., Khazeev M., Lee JY., Masiagin S., Mazzara M., Mustafin R., Naumchev A., Rivera V.* Teaching programming and design-by-contract // Proc. ICL 2018. Advances in Intelligent Systems and Computing. 2020. V. 916. P. 68–76.

42. *Khazeev M., Mazzara M., Aslam H., de Carvalho D.* Towards a broader acceptance of formal verification tools // Proc. ICL 2019. Advances in Intelligent Systems and Computing. 2020. V. 1135. P. 188–200.

43. *Sammler M., Lepigre R., Krebbers R., Memarian K., Dreyer D., Garg D.* RefinedC: automating the foundational verification of C code with refined ownership types // Proc. 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. 2021. P. 158–174.

44. *Jiang D., Zhou M.* A comparative study of insertion sorting algorithm verification // Proc. 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference. 2017. P. 321–325.

45. *Nepomniaschy V.A., Anureev I.S., Mikhailov I.N., Promskii A.V.* Towards verification of C programs. C-light language and its formal semantics // Programming and Computing Software. 2002. V. 28. Is. 6. P. 314–323.

46. *Nepomniaschy V.A., Anureev I.S., Promskii A.V.* Towards Verification of C Programs: Axiomatic Semantics of the C-kernel Language // Programming and Computing Software. 2003. V. 29. Is. 6. P. 338–350.

47. *Anureev I.S., Garanina N.O., Lyakh T.V., Rozov A.S., Zyubin V.E., Gorlatch S.P.* Dedicative Verification of Reflex Programs // Programming and Computing Software. 2020. V. 46. Is. 4. P. 261–272.

48. *Kondratyev D.A.* Automatic verification of insertion sorting. https://bitbucket.org/Kondratyev/verify-loops (Accessed 11 Nov 2021)

EDN: TGIZYH
ПРОГРАММИРОВАНИЕ, 2022, № 5, с. 54–67

БАЗЫ ДАННЫХ, ХРАНИЛИЩА ДАННЫХ

УДК 004.65, 004.658, 519.688

РЕПОЗИТОРИЙ ДАННЫХ В КОНТЕКСТЕ ВЫЧИСЛИТЕЛЬНЫХ ЭКСПЕРИМЕНТОВ: МОДЕЛЬ, АРХИТЕКТУРА, ИНТЕРФЕЙСЫ, ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММНЫХ РЕАЛИЗАЦИЙ

© 2022 г. А. А. Иванков*, Г. А. Мануилов[a],**

[a]*ООО "ДСП Лабс" 194044 Санкт-Петербург, ул. Гельсингфорсская, д. 2, Россия*

**E-mail: a.vnkv1@gmail.com*

***E-mail: george.manuilov@gmail.com*

Поступила в редакцию 04.05.2022 г.
После доработки 11.05.2022 г.
Принята к публикации 18.05.2022 г.

DOI: 10.31857/S0132347422050041

1. ВВЕДЕНИЕ

Работа имеет отношение к такой области исследований, как архитектура и реализации программного обеспечения (ПО), которые пригодны для построения компонентно-ориентированных архитектур конвейерной обработки данных. Заметим, далее речь пойдет об архитектуре и реализациях такой обработки в контексте одного единственного процесса. Научные интересы одного из авторов сосредоточены на решении задач, имеющих отношение к статистике случайных процессов и полей, задач анализа различного рода статистических моделей. Поэтому в качестве контекста, в котором обсуждается предлагаемое нами решение, мы будем использовать формализм и понятия, имеющие отношение к статистике случайных процессов. Реализацию процедуры решения отдельной задачи мы будем называть вычислительным экспериментом (ВЭ). Более конкретное определение этого понятия будет предложено ниже.

2. МОТИВАЦИЯ

На протяжении более чем двух десятилетий в поле нашего зрения попадало различное прикладное ПО (ППО), позволяющее построить ВЭ как конвейерную обработку данных, оперируя программными компонентами посредством публичного API, декларируемого разработчиками этого ППО как стандарт де-факто.

Изо всего множества разнообразных библиотек и фрейворков, с которыми пришлось иметь дело, мы преимущественно выделяли свободно-распространяемое ППО, разработанное на языках программирования C/C++. Упомянем лишь небольшое подмножество в определенном смысле полярных решений: Boost.Accumulators [1], RaftLib [2], R-package [3] и TensorFlow [4].

Выбор субъективен, но он, на наш взгляд, позволяет пояснить те особенности архитектуры, интерфейсов существующих реализаций, которые мы расценивали как недостатки. Предложенное нами собственное решение – попытка устранить эти недостатки.

2.1. Boost.Accumulators

Фреймворк Boost.Accumulators изначально рассматривался как один из прототипов нашего будущего решения. Возможность автоматически построить орграф вычислений искомых оценок – наиболее привлекательный механизм, который реализован в Boost.Accumulators. Однако он не компенсирует следующие недостатки его высокоуровневой архитектуры: 1) построение публичного API на основе статического полиморфизма; 2) очень лаконичная, примитивная, на наш взгляд, система поддержки метаданных, которые ассоциированы как с наборами данных (в контексте Boost.Accumulators это сущность feature – оценка), так и с компонентами орграфа (в контексте Boost.Accumulators это сущность accumulator – аккумулятор). Статический полиморфизм не позволяет спроектировать универсальные, унифицированные публичные интерфейсы как для сущности "компонента", именуемой нами далее по тексту эстиматором, так и для сущности "набор данных", как элемента потока данных, автоматически порождаемого реализацией орграфа вычислений. Система поддержки метаданных, построенная на основе шаблона features⟨⟩ представляется нам весьма ограничительной. Определение синонимов отдельных оценок с помощью шаблонов feature_of⟨⟩, as_feature⟨⟩ – это очень

скромные средства реинтерпретации метаинформации, ассоциированной с конкретной оценкой. Очевидная потребность построения потока данных, содержание которых – многомерные наборы данных, порождает очень сложные задачи как раз в части реинтерпретации компонентами-потребителями отдельных подмножеств таких наборов. Определение функциональных зависимостей между оценками с помощью шаблона depends_on⟨⟩ не позволяет перейти к декларативному способу их определения, не сломав сам механизм автоматического построения орграфа вычислений.

2.2. RaftLib

Код этой библиотеки – обширная система шаблонов. Автоматическое распараллеливание высокоуровневых ETL-операций, которое реализуется кодом самой RaftLib, будет полезно, если только семантика этих операций в задачах обработки данных достаточно тривиальна.

Масштабируемость решений, построенных на основе библиотеки, представляется проблематичной, как только потребуется перейти к обработке многомерных наборов данных, подмножества которых неоднородны с точки зрения их семантики. Статический полиморфизм, положенный в основу архитектуры, существенно осложняет не только переход к проектированию интерфейсов на основе динамического полиморфизма, но и разработку кода компонент согласно PIMPL-принципу.

2.3. R-package

R-package – программное решение, за более чем два десятилетия своего существования ставшее авторитетной альтернативой коммерческим. Оно постоянно пополняется новыми пакетами для статистического анализа данных. С точки зрения конечных пользователей, т.е. с точки зрения функциональных возможностей, мы действительно наблюдаем эволюцию этого пакета. Однако на уровне программных интерфейсов низкого уровня кардинальных изменений нет: в основе интерфейсов кода, написанного на языке C, обширная система макросов, которая является типичным средством его параметризации, в том числе на уровне типов данных. Это одно из самых принципиальных ограничений, которые не позволяют трансформировать существующие интерфейсы так, чтобы переориентировать вектор их развития в направлении компонентно-ориентированных архитектур. Добавим, что и в этом пакете на уровне интерфейсов низкого уровня крайне сложно перейти к организации потока многомерных и неоднородных, с точки зрения семантики, наборов данных. Примечательно, что в документации, которая включена в дистрибутив R-package, мы уже много лет встречаем рассуждения его разработчиков, суть которых: “неплохо было бы попробовать построить код на основе компонентно-ориентированного подхода” (Перевод авт.) [http://cran.org/README, section 4. GOALS.]. Однако конкретных изменений в коде дистрибутива R-package пока не наблюдаем.

2.4. TensorFlow

TensorFlow – фреймворк, предназначенный для построения произвольных графов вычислений, путем их декларативного описания. Граф представляется набором операторов – отдельных программных компонентов, которые оперируют так называемыми тензорами – многомерными массивами данных. Фреймворк TensorFlow предназначен для решения задач машинного обучения, а конкретно – для обучения и инференса (inference) искусственных нейронных сетей (ИНС). Для этих целей TensorFlow успешно комбинирует две концепции: автоматическое дифференцирование и программирование потоков данных. Как и в рассмотренных ранее библиотеках, в TensorFlow передача данных производится напрямую от производителя к потребителю, а также граф вычислений автоматически распараллеливается. К существенным недостаткам TensorFlow в контексте рассматриваемой проблемы можно отнести: ориентированность сугубо на ИНС, сложность высокоуровневого Python API и еще большая сложность аналогичных операций через C++ API, сложность разработки собственных компонентов-операторов. Кроме того, как и все рассмотренные выше программные решения, TensorFlow обезличивает данные при их передаче от оператора к оператору, таким образом не позволяя производить непротиворечивую реинтерпретацию данных в соответствии с их семантикой.

Вышеуказанные особенности (прежде всего низкоуровневых интерфейсов) большинства рассмотренных нами пакетов мы трактуем как недостатки на уровне их архитектуры. Эти фундаментальные расхождения между существующим и проектируемым нами ПО – основные причины, мотивация работы, результаты которой изложены ниже.

3. НЕКОТОРЫЕ АСПЕКТЫ ПАРАДИГМЫ КОМПОНЕНТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Компонентно-ориентированное проектирование и разработка ПО (Component-Based Development), как философский и технологический подход, было предложено более полувека назад (большинство авторов упоминают в качестве пионерской работы проект конца 60-х годов [5]). В СССР начало развитию компонентно-ориентированного программирования было положено

еще в середине 70-х годов прошлого века. В отечественной литературе эти технологии именуют сборочным программированием, а одни из первых работ были выполнены специалистами научной школы академика В.М. Глушкова [6]. Монография [7], выдержавшая два издания, уже давно стала классическим университетским учебником. Стоит заметить, что один из авторов, Лаврищева Е.М., является сотрудником ИСП РАН, а с самыми последними ее работами читатель может ознакомиться на сайте этого института [8]. Принимая во внимание столь давнюю историю развития этого направления научных исследований, обширную литературу, посвященную компонентно-ориентированному/сборочному программированию, может сложиться впечатление, что большинство проблем решены и эти state-of-art-решения доступны любому разработчику ПО. На самом деле все почти с точностью до наоборот. Круг вопросов, которые приходится решать, намного обширнее подмножества решений, которые уже обрели статус стандартов де-факто и де-юре. Мы попытаемся затронуть лишь отдельные аспекты проектирования архитектуры отдельных компонент, когда весь вычислительный конвейер помещается в контекст одного процесса. Наш интерес к таким архитектурам объясняется достаточно просто – производительность вычислений повышается как минимум на порядок, а, исходя из собственного опыта, чаще всего на два-три порядка (если сравнить с производительностью массовых вычислений на основе механизмов межпроцессного взаимодействия). Как в русскоязычных, так и в англоязычных изданиях, посвященных вопросам компонентно-ориентированного/сборочного программирования, нам не удалось найти работы, результатами которых можно было бы воспользоваться буквально как лекалом: от формального определения семантики интерфейсов до публично доступных исполняемых модулей с реализациями предложенных авторами решений. Но это вовсе не означает, что авторы настоящей статьи претендуют на уникальность своего подхода и ПО. Компонентно-ориентированные архитектуры массовых вычислений в контексте одного-единственного процесса реализованы в огромном количестве программных продуктов (несколько обсудили выше), а их разработчики иногда даже не отдают себе отчет о том, какую же архитектуру они выбрали для проектирования своего ПО, ограничиваясь при подготовке документации описанием пользовательских интерфейсов. Комментарии к API – единственный источник информации для разработчиков. В тех редких случаях, когда авторы ПО все-таки находят силы и время для подготовки статей, в их содержании преобладает изложение вопросов вида “для решения каких задач проектировалось ПО?”, “какие конкретные результаты могут получить специалисты предметной области с помощью этого ПО?”. Это наблюдения в т.ч. и за собственной практикой. Далеко не первая наша реализация компонентно-ориентированной архитектуры в контексте одного процесса (язык разработки – СИ с классами) была описана [9, 10] в предположении, что интерес к ней проявят прежде всего пользователи этого ПО, нейрофизиологи и нейрохирурги.

Таким образом, за все время развития компонентно-ориентированного программирования, наряду с огромным количеством предложенных реализаций, растет и число вопросов, которые остаются открытыми. Мы полагаем, что CBD может играть доминирующую роль в IT-технологиях ближайшего будущего, если наметится существенный прогресс по меньшей мере в следующих трех направлениях: разработка системообразующих критериев и стандартов классификации компонент, стандартизация публичных интерфейсов компонент, разработка стандартов координирующих языков. Наша работа – попытка поделиться своими результатами как рабочим материалом для дискуссий уровня “proof of concept” (обоснование подходов к решению некоторого подмножества вышеперечисленных вопросов).

4. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Отдельная компонента в контексте ВЭ с компоненто-ориентированной архитектурой – программная реализация следующего отображения:

$$\begin{gathered} Estimator(Model, Parameters): \\ INPUT \to OUTPUT, \end{gathered} \tag{1}$$

где $INPUT$ – множество входных данных, элементы которого в нашей архитектуре – это многомерные наборы данных;

$OUTPUT$ – множество выходных данных этого экземпляра, элементы которого – тоже многомерные наборы данных;

Parameters – параметры этого экземпляра компоненты, предоставленные ей как множество многомерных наборов данных;

Model – экземпляр модели, которая параметризует отображение. В общем случае модели необходимы конкретному экземпляру компоненты как для интерпретации входных данных, $INPUT$, так и для интерпретации параметров алгоритма, $Parameters$.

Формальное описание одной транзакции с участием экземпляра компоненты:

$$\begin{gathered} \{pre(Model, Parameters, INPUT)\} \\ Estimator(Model, Parameters): \\ INPUT \to OUTPUT\{post(OUTPUT)\}, \end{gathered} \tag{2}$$

где $\{pre(Model, Parameters, INPUT)\}$ – предусловие, предикат, формальное определение условий, которым должны удовлетворять модель, параметры алгоритма и множество входных данных; $\{post(OUTPUT)\}$ – постусловие, предикат, формальное определение условий, которым должно удовлетворять множество выходных данных по окончании транзакции.

Отдельный ВЭ определяется как композиция отображений в смысле (1). Далее, в ходе изложения, мы будем пользоваться и другой моделью ВЭ, определяя последний как орграф, узлы которого – экземпляры конкретных компонент, а ребра – потоки данных.

Сущность "эстиматор" в контексте архитектуры наших ВЭ – синоним понятия "компонента". Мы определяем "эстиматор" как самостоятельную, независимую, параметризуемую реализацию алгоритма вычисления конкретного набора оценок. Свойство "независимость" следует понимать как возможность развернуть ее на рабочей станции независимо от других эстиматоров, включаемых в орграфы ВЭ. Свойство "параметризуемость" тоже следует понимать в широком смысле: конкретный экземпляр эстиматора в ходе ВЭ может быть параметризован как на уровне структур данных (классическая реализация параметризации алгоритмов), так и на уровне кода – один эстиматор может быть параметризован экземпляром другого эстиматора. Совокупность экземпляров эстиматоров/экземпляров программных компонент/модулей, их конфигурационных файлов, определяющих порядок решения этими эстиматорами/модулями конкретной задачи, а также совокупность исходных наборов данных, необходимых для решений этой задачи, будем называть конкретной реализацией ВЭ. Выполнение ВЭ – это, в самом простейшем случае, сессия приложения/процесс в конкретной операционной среде, в которой/котором эстиматорами выполняется обработка исходных наборов данных, промежуточных результатов ВЭ. Порядок обработки определен в конфигурационном файле ВЭ. Выполнение отдельного ВЭ завершается после выполнения вычислений всеми эстиматорами, задействованными для решения задачи. Количество высокоуровневых вычислительных операций (высокоуровневая операция задана отдельной директивой конфигурационного файла эксперимента и выполняется отдельным эстиматором) в ходе выполнения ВЭ может быть как детерминированным, так и недетерминированным. Детерминированное количество высокоуровневых операций соответствует ациклическому орграфу ВЭ (например, эстиматоры указаны в линейном порядке). Недетерминированное количество высокоуровневых операций соответствует орграфу ВЭ с циклами (т.е. в конфигурационном файле ВЭ определяется циклический порядок выполнения этих операций и инварианты цикла).

Конкретное количество эстиматоров в ВЭ и порядок вычислений (последовательно, параллельно или некоторая комбинация этих вариантов) пользователь определяет в конфигурационном файле этого ВЭ. Пример фрагмента конфигурационного файла ВЭ представлен в Приложении 1. Соответственно, в Приложении 2 – директивы конфигурационного файла отдельного экземпляра эстиматора. Таким образом, отдельный ВЭ, точнее, его орграф, определяется в конфигурационном файле этого ВЭ, где помимо топологии орграфа явно или косвенно указаны параметры эстиматоров (узлы орграфа ВЭ). Явное определение параметров эстиматора предполагает, что параметры непосредственно указаны в директивах конфигурационного файла. Неявное определение параметров эстиматора – в директивах конфигурационного файла указан запрос к репозиторию (хранилищу данных ВЭ), после успешного выполнения которого эстиматор получит из репозитория искомые параметры как элементы многомерного набора данных, предоставленного ему репозиторием в режиме доступа "только чтение". Входные данные эстиматора всегда определены в форме запроса к репозиторию (см. директиву REPO_ENTRIES в Приложении 2). Наиболее типичная форма запроса к репозиторию содержит определение суррогатного ключа запрашиваемого набора данных:

$$key : '[' \ PK \ '[' \ list\_of\_parent\_PKs \ ']' \ ']';$$

PK – это уникальный ключ запрашиваемого набора данных в репозитории, *list_of_parent_PKs* – список уникальных ключей наборов данных, содержимое которых было использовано при вычислении элементов набора с ключом *PK*. Оба эти элемента грамматики запросов к репозиторию определяются следующим образом:

$$list\_of\_parent\_PKs$$

$$: (PK \,|\, (PK(',' \, PK)^*));$$

$$PK : UINTEGER;$$

$$UINTEGER : [1-9][0-9]^*;$$

Поэтому еще раз подчеркиваем, в объем понятия "параметры эстиматора" включены в т.ч. и выражения на языке запросов к репозиторию, определяющие содержание запроса этого эстиматора к репозиторию как заявку на доступ к хранимым в репозитории данным.

Сущность "набор данных" (далее НД) в контексте нашей архитектуры – это контейнер, который инкапсулирует как сами данные, так и их метаданные. Собственно данные в наших реализациях сущности НД – это двухмерный массив, подмножества которого имеют различную мощ-

ность и разную семантику. Метаданные НД включают дескриптор эстиматора – источника этих оценок, декларативное определение модели, согласно которой был получен этот НД, дескриптор всей совокупности оценок, хранящихся в этом НД (аналог собирательного обобщения понятия feature в контексте фреймворка Boost.Accumulators), наконец, дескрипторы отдельных подмножеств, далее треков, этого многомерного НД как самостоятельных оценок (аналог понятия feature в контексте фреймворка Boost.Accumulators). Поскольку программные реализации эстиматоров могут иметь различное происхождение (программные продукты различных разработчиков), координирующая роль ПО нашего фреймворка сводится к управлению порядком работы эстиматоров (включая время их жизни) и управлению потоками данных между ними. Предлагаемое нами решение, в части организации потоков данных ВЭ, реализует эти потоки между эстиматорами с обязательным участием посредника – централизованного хранилища НД ВЭ (репозиторий).

Мотивация такой архитектуры достаточно очевидна. Централизованное хранилище данных, порождаемых в ходе ВЭ, как самостоятельная компонента, вполне согласуется с компонентно-ориентированной архитектурой ВЭ. Оно реализует основные функциональные требования, предъявляемые к контейнерам с аналогичной семантикой [11]:

– контроль времени жизни данных;

– контроль доступа к данным;

– реализация механизма транзакций с этими данными;

– эффективное использование двух основных ресурсов вычислительной системы: оперативной памяти и процессорного времени.

В той или иной форме подобные хранилища подразумевают авторы многих программных систем, спроектированных согласно компонентно-ориентированной архитектуре. Например, в [11] отмечают, что в системах распределенной обработки данных, построенных ими на основе компонентно-ориентированной архитектуры, конкретное содержание потоков данных между компонентами – дескрипторы информационных пакетов (информационный пакет, ИП, – это семантический аналог нашего набора данных). Компоненты получают доступ к данным, используя эти дескрипторы, но владельцами ИП сами компоненты не могут быть по целому ряду причин. О реализации репозитория ИП автор [12] упоминает неявно, когда поясняет суть этих причин: делегирование компонентам права владения ИП порождает ряд сложных задач по управлению потоками данных, а их решения будут неэффективными во всех смыслах. Становится ясно, что хранение в НД данных вместе с метаданными обусловлено отчуждением эстиматорами их результатов в репозиторий. С того момента, как НД попадает в репозиторий, хранимые им данные становятся обезличенными с точки зрения их семантики, если только в НД не хранится метаинформация. Поэтому совместное хранение в НД собственно самих оценок вместе с их метаданными – один из наиболее эффективных способов определения семантики данных на стороне их потенциальных потребителей – эстиматоров. Напомним, что транзакция эстиматора, в смысле (2), предваряется вычислением операндов предиката $\{pre(Model, Parameters, INPUT)\}$. В части, касающейся множества входных НД, $INPUT$, эти вычисления построены на операндах – элементах метаданных НД из $INPUT$.

5. МОДЕЛЬ РЕПОЗИТОРИЯ

Принимая во внимание две основные функции репозитория – эффективное хранение НД и предоставление доступа на чтение к хранимым НД, мы можем абстрактно описать работу такого хранилища моделью системы массового обслуживания (СМО). В каждый момент времени состояние такой СМО определяется множеством

$$E = E_{RAM} \cup E_{DISK}, \tag{3}$$

где

$$E_{RAM} = \cup_i DS_i \tag{4}$$

– множество НД, хранимых в этот момент в оперативной памяти (далее кэш репозитория),

$$E_{DISK} = \cup_j DS_j \tag{5}$$

– множество НД, хранимых на внешнем запоминающем устройстве (диске). При этом:

$$E_{RAM} \cap E_{DISK} = \varnothing$$

Эффективное хранение НД в репозитории – это оптимальное разделение множества (3) на два непересекающихся подмножества (4) и (5). Задачу оптимизации в общем виде сформулируем следующим образом. Пусть V_{RAM} – максимальный размер кэша, в байтах. V_{DISK} – объем доступного пространства на ВЗУ, в байтах. Выполняется неравенство:

$$V_{RAM} \ll V_{DISK}$$

Пусть отображение:

$$V : E \to v \in N$$

– размер, в байтах, участка оперативной памяти или дискового пространства, необходимого для хранения НД. Пусть функция времени доступа к НД (к хранимым в нем данным):

$$T : E \to at, \quad at \in R_1,$$

причем время доступа к НД в кэше на несколько порядков (согласно нашим результатам по мень-

шей мере на два порядка) меньше времени доступа к НД такого же размера на диске:

$$\forall DS_i \in E_{RAM}, \quad DS_j \in E_{DISK}:$$
$$V(DS_i) == V(DS_j) \Rightarrow T(i) \ll T(j)$$

Ограничение на размер кэша в общем случае не позволяет разместить в нем все НД ВЭ. Безусловное размещение НД на диске приведет к значительному увеличению времени доступа к данным. В ходе ВЭ эстиматоры, как клиенты СМО, обращаются к репозиторию с заявками двух типов: на размещение в репозитории нового НД (s-тип) и на получение доступа к хранимому в репозитории НД (l-тип). Каждому запросу l-типа соответствует некоторый запрашиваемый НД $DS_l \in E$.

Номер t элемента s_t или l_t в случайной последовательности заявок к репозиторию:

$$R = s_{t=1}^{Ns} \cup l_{t=1}^{Nl} \tag{6}$$

можно рассматривать как момент времени (номер шага), в который репозиторий получил заявку s_t или l_t.

Политика размещения НД – отображение:

$$h : (E_{RAM}, E_{DISK}, v(DS_t)) \to (E^*_{RAM}, E^*_{DISK}), \tag{7}$$

которое определяет новое состояние репозитория (E^*_{RAM}, E^*_{DISK}) на основе текущего состояния репозитория (E_{RAM}, E_{DISK}) и метрик сохраняемого/запрашиваемого НД (в данном случае – размера этого НД). Последовательность состояний репозитория – это множество пар подмножеств (E_{RAM}, E_{DISK}). При этом последовательность заявок s_t порождает пары (E^*_{RAM}, E^*_{DISK}):

$$((E_{RAM} \subset E^*_{RAM}) \; and \; (E_{DISK} == E^*_{DISK}))$$

$$or$$

$$((E_{RAM} == E^*_{RAM}) \; and \; (E_{DISK} \subset E^*_{DISK}))$$

Последовательность заявок l_t порождает пары:

$$((E^*_{RAM} == (E_{RAM} E^{(d)}_{RAM}) \cup E^{(l)}_{DISK})$$

$$and$$

$$(E_{DISK} \subseteq E^*_{DISK})),$$

где $E^{(d)}_{RAM})$ – подмножество E_{RAM}, состоящее из тех НД, которые вытеснены из кэша на диск, а $E^{(l)}_{DISK}$ – подмножество E_{DISK}, состоящее из тех НД, которые загружены в кэш с диска. Для описания поведения репозитория при получении заявок на чтение, заявок l-типа, определим следующую полную группу событий: кэш-попадание (НС) – запрошенный НД оказался в кэше, кэш-промах (МС) – запрошенный из кэша НД в этот момент в нем отсутствует. Оценка уровня попаданий OHR (object hit ratio) – отношение числа кэш-попаданий к общему числу запросов к кэшу. Пусть

$$OHR = ohr_{t=1}^{Nl} \tag{8}$$

статистика (случайная последовательность) уровня попаданий, которая собрана на некоторых конечных интервалах времени, то есть это статистика, которая получена на основе некоторого подмножества (6), составленного из l-запросов к кэшу. Тогда интегральная оценка:

$$\Sigma(OHR) = \sum_{t=1}^{Nl} (ohr_t) \tag{9}$$

одна из количественных характеристик эффективности хранения НД в репозитории.

Основные потери времени связаны с операциями перемещения/копирования НД между кэшем и диском в ходе обработки заявок l-типа (низкий уровень попаданий в последовательности (8)), и операциями доступа к данным тех НД, которые СМО не смогла разместить в кэше. В последнем случае, когда СМО не смогла разместить в кэше запрошенный НД, причины совершенно объективны – все НД, находящиеся в кэше, все еще используются эстиматорами, следовательно, не могут быть вытеснены на диск, а ограничение на размер кэша не позволяет увеличить его емкость. Другими словами, в подобном случае нет никаких предпосылок для того, чтобы улучшить обслуживание l-заявки. Оптимизация в смысле совокупного времени обслуживания l-заявок возможна путем построения политик размещения, которые минимизируют перемещения/копирование НД между кэшем и диском (и, как следствие, время на выполнение этих операций).

Можно дать альтернативное (7) определение политики размещения. Оно позволит сформулировать задачу оптимизации, оперируя понятиями последовательности заявок к репозиторию, R, и последовательности операций размещения/перемещения НД в репозитории, которые реализуют обработку заявок из R:

$$h : R \to m,$$

где R – множество последовательностей R заявок к репозиторию, m – множество последовательностей операций перемещения НД в репозитории.

Пусть задан функционал H, отображающий из декартова произведения множества h реализуемых политик размещения и множества R последовательностей заявок во множество интегральных оценок $\Sigma(OHR)$ из (9):

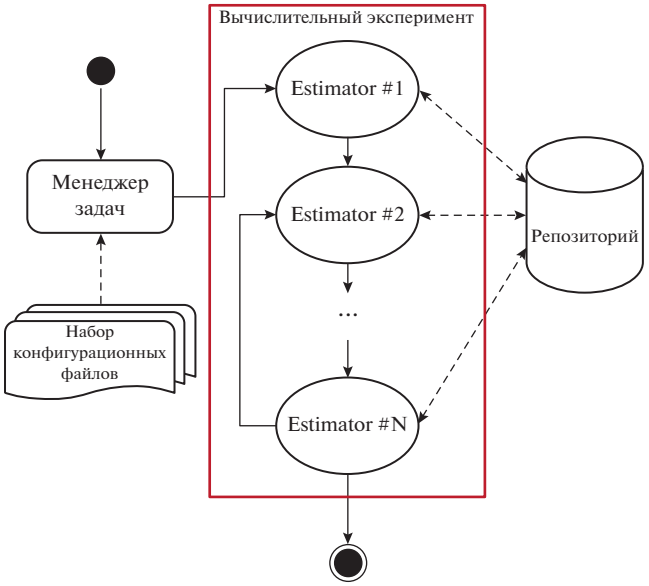

Рис. 1. Архитектура отдельного ВЭ.

$$H : h \times R \to \Sigma(OHR).$$

Тогда для детерминированной последовательности R разработка алгоритмов обслуживания репозиторием заявок в явном или неявном виде может быть сведена к решению следующей задачи максимизации функционала H:

$$h^* = \text{argmax}_h H(h, R),$$

где R – фиксированная последовательность запросов.

В общем случае мы имеем дело со стохастической постановкой задачи оптимизации, поскольку последовательности заявок к репозиторию, R, случайны по своей природе. В таких ситуациях для решения задачи потребуется, во-первых, определить набор инвариантов, которые сохраняются на множестве предполагаемых входных последовательностей R, во-вторых, уточнить определения оптимального решения. Другими словами, необходимо уточнить, в каком смысле мы понимаем оптимальное решение, например, как доставляющее нам максимум математического ожидания оценки $\Sigma(OHR)$ из (9), минимум дисперсии этой оценки, $D[\Sigma(OHR)]$.

5.1. Формальное определение семантики интерфейса репозитория

Определим две аксиомы, которые должны выполняться минимальным публичным API репозитория.

$$\{pre(DS)\}saveData : (E_{RAM}, E_{DISK}, DS) \to$$

$$\to (E^*_{RAM}, E^*_{DISK})\{post((E^*_{RAM}, E^*_{DISK}))\},$$

где $\{pre(DS)\}$ – предикат, формальное определение условия “DS – правильно сформированный экземпляр НД”;

$\{post((E^*_{RAM}, E^*_{DISK}))\}$ – постусловие в виде дизъюнкции: “репозиторий успешно перешел в новое состояние, $((E_{RAM} \subset E^*_{RAM})$ and $(E_{DISK} == E^*_{DISK}))$ *or* $((E_{RAM} == E^*_{RAM})$ *and* $(E_{DISK} \subset E^*_{DISK}))$ ” or “репозиторий остался в прежнем состоянии $(E_{RAM} == E^*_{RAM})$ and $(E_{DISK} == E^*_{DISK})$”. Ограничения на объем статьи не позволяют нам уточнить определения операндов каждого из предикатов. Для этого потребуется указать алфавит, который однозначно определяется структурой программных реализаций отдельных сущностей. Операция передачи репозиторию на хранение НД должна быть семантически эквивалентна операции копирования.

$$\{pre(DS\_def)\}$$

$$getData : (E_{RAM}, E_{DISK}, DS\_def) \to$$

$$\to (E^*_{RAM}, E^*_{DISK}, DS_sc)$$

$$\{post((E^*_{RAM}, E^*_{DISK}, DS_sc))\},$$

где $\{pre(DS)\}$ – предикат, формальное определение условия “DS_def – правильно сформированная строка, идентифицирующая экземпляр НД на основе его метаданных уровня репозитория или уровня самого НД”;

$\{post((E^*_{RAM}, E^*_{DISK}, DS_sc))\}$ – постусловие в виде дизъюнкции: “репозиторий успешно перешел в новое состояние, $((E^*_{RAM} == (E_{RAM}E^{(d)}_{RAM}) \cup E^{(l)}_{DISK})$ and $(E_{DISK} \subseteq E^*_{DISK}))$ или остался в прежнем, $((E_{RAM} == E^*_{RAM})$ and $(E_{DISK} == E^*_{DISK}))$, сформировав shallow copy затребованного НД”, or “репозиторий остался в прежнем состоянии $((E_{RAM} == E^*_{RAM}, E_{DISK} == E^*_{DISK}))$, затребованный НД в репозитории не обнаружен”.

Операция передачи из репозитория хранимого там НД должна быть семантически эквивалентна операции копирования shallow copy.

6. ДВЕ АРХИТЕКТУРЫ РЕПОЗИТОРИЯ ДАННЫХ ВЭ

Поскольку хранилище тоже было спроектировано, следуя концепции компонентно-ориентированного программирования, его архитектуру можно описать, оперируя четырьмя сущностями: справочник репозитория (далее RD), логика кэша (RCL),

кэш репозитория (RC), долговременное хранилище (RPS).

Обработка заявок, поступающих от клиентов – эстиматоров, – это транзакция, совместно осуществляемая экземплярами всех четырех сущностей. Код самого репозитория координирует порядок выполнения транзакций. Другими словами, нашу реализацию хранилища можно рассматривать как транзакционную систему управления многомерными наборами данных. Она (система) размещается, как синглтон, в контекст каждого процесса (см. рис. 1) для того, чтобы контролировать потоки данных между отдельными компонентами ВЭ.

Публичные интерфейсы каждой из четырех компонент составляют всего три операции: runTransaction, commitTransaction, rollbackTransaction. Семантика этих операций в RCL, RD, RC, RPS определяется семантикой конкретной компоненты. Полагаем, она очевидна читателям.

Алгоритм 1: Алгоритм извлечения НД из репозитория

Вход: Уникальный или суррогатный ключ DS в репозитории K

Выход: Shallow copy набора данных DS

```
1  TP := RepoCacheLogic::makeTransactionPlan(K)
2  RepoDirectory::notifyOnTransactionPlan(TP)
3  RepoCache::runTransaction(TP)
4  RepoPersistentStorage::runTransaction(TP)
5  RepoDirectory::commitTransaction(TP)
6  RepoCache::commitTransaction(TP)
7  RepoPersistentStorage::commitTransaction(TP)
8  RepoCacheLogic::commitTransaction(TP)
9  DS := RepoCache::retrieveDS_sc(K)
```

Алгоритм 2: Алгоритм размещения НД в репозитории

Вход: Набор данных DS

```
1  TP := RepoCacheLogic::makeTransactionPlan(DS)
2  RepoDirectory::notifyOnTransactionPlan(TP)
3  RepoCache::runTransaction(TP)
4  RepoPersistentStorage::runTransaction(TP)
5  RepoDirectory::commitTransaction(TP)
6  RepoCache::commitTransaction(TP)
7  RepoPersistentStorage::commitTransaction(TP)
8  RepoCacheLogic::commitTransaction(TP)
```

Алгоритмы представлены в сокращенном варианте, исключены операции восстановления состояния репозитория после сбоя любого из его компонентов.

7. ОСОБЕННОСТИ ПРОГРАММНЫХ РЕАЛИЗАЦИЙ

Нам не удалось найти свободно-распространяемое ПО, которое по своим функциональным возможностям было бы сопоставимо с нашим репозиторием. Чтобы провести сравнительный анализ нашей реализации (далее m-реализация), мы разработали еще и ее альтернативный вариант. Его, по нашему мнению, мог бы спроектировать среднестатистический программист, располагая формальной спецификацией публичного интерфейса (API) репозитория.

Альтернативная, наивная реализация (далее n-реализация) позволяет нам обратить внимание на существенные особенности нашего ПО. Во-первых, в n-реализации кода сущности НД сами данные почти наверняка будут храниться как одномерный массив. Но все метаданные НД: строковые литералы, идентифицирующие эстиматоры, модели, семантика НД и отдельных треков, массивы смещений отдельных треков в экземпляре НД, будут размещаться в нем, не заботясь о возникающей при этом фрагментации оперативной памяти. Наша m-реализация НД гарантирует размещение в памяти всех данных и метаданных НД без фрагментации. Экземпляры m-реализации НД копируемы (copyable) и перемещаемы (moveable) в том же смысле, в котором эти характеристики в стандарте языка C++ указывают для массивов POD-типов [14]. Во-вторых, возможная структура n-реализации кода репозитория почти наверняка, следуя принципу минимализма, – совокупность двух компонент: кэш и долговременное хранилище, где кэш инкапсулирует еще метаданные, код справочника (т.е. RD) и логики хранения (т.е. RCL). Копии экземпляров НД, принимаемых на хранение таким репозиторием, будут размещаться в памяти по наивному сценарию, т.е. не заботясь об устранении фрагментации памяти. Наша m-реализация кода репозитория гарантирует размещение копий НД в RC без фрагментации оперативной памяти. Сама возможность таким образом хранить НД обеспечена вышеуказанными характеристиками m-реализации сущности НД (копируемы и перемещаемы). В-третьих, в n-реализации долговременное хранилище тоже использует механизм файлов, отображаемых в память, но каждый НД сохраняется в отдельном файле. Такой вариант размещения НД на диске основан на наивном предположении о том, что он существенно сокращает время построения представлений на запись и чтение.

По нашим оценкам, только первые две, из этих трех наиболее существенных особенностей m-реализации, снижают совокупный расходуемый репозиторием объем оперативной памяти, по сравнению с n-реализацией, минимум в полтора раза.

8. МЕТОДИЧЕСКИЕ АСПЕКТЫ ОЦЕНИВАНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММНЫХ РЕШЕНИЙ

В течение всей сессии мы контролируем ту долю адресного пространства процесса, которую используют две основные компоненты в структуре хранилища, RC и RPS, точнее их контейнеры: кэш и представление, используемое для записи НД во внешний файл, проецируемый в память (mmap). Объем используемой ими оперативной памяти ограничен сверху. Во всех четырех компонентах хранилища, в RCL, RD, RC, RPS, имеются собственные локальные справочники. Они представляют собой динамические массивы. По мере накопления метаинформации растет объем используемой ими оперативной памяти, а описать такой процесс можно линейной функцией от количества хранимых в репозитории НД. Эти неизбежные накладные расходы гарантируют логарифмическую временную сложность операции поиска метаинформации в локальных справочниках.

Наибольший интерес, с точки зрения производительности программной реализации наших хранилищ, представляют следующие характеристики:

– средняя длительность транзакции на получение доступа к хранимому в репозитории НД (далее l-транзакция), т.е. среднее время получения shallow copy такого НД;

– средняя длительность транзакции (далее s-транзакция), в ходе которой сохраняем в репозитории новый НД.

Далее мы рассматриваем два типа l-транзакций, поскольку запрашиваемый НД может находиться либо в кэше (RC), либо в долговременном хранилище (RPS). В последнем случае l-транзакция включает в себя еще и операцию загрузки такого НД из RPS в RC.

Два замечания об оценках динамики средней длительности вышеуказанных трех типов транзакций:

1. Мы будем строить такие оценки, интерпретируя накопленный статистический материал как реализации нестационарных с.п. с дискретным аргументом. Для их описания нами были выбраны регрессионные модели, переменная-регрессор которых – порядковый номер транзакции соответствующего типа. В случае s-транзакций их порядковый номер однозначно связан с количеством НД, накопленных к этому моменту в репозитории. В случае l-транзакций такую связь можно трактовать в смысле оценок в среднем.

2. Реализации с.п., которые были получены в результате мониторинга транзакций, содержат достаточно большое количество выбросов. В подобной ситуации классические, неробастные оценки параметров регрессионных моделей в L_2-метрике неустойчивы. Поэтому мы использовали методы робастного регрессионного анализа.

Серии вычислительных экспериментов были спланированы так, чтобы в ходе сессии приложения было обработано не менее 10^4 s-транзакций и не менее 3×10^4 l-транзакций обоих типов. Последовательности таких s- и l-транзакций порождали последовательности s- и l-запросов, в смысле (6). Тип каждого элемента такой последовательности – случайная величина, которую генерировали с помощью биномиального распределения:

$$\forall r_t \in R, \quad P(r_t = s_t) = Bi(1, \theta)$$

Величина параметра θ выбиралась таким образом, чтобы обеспечить вышеуказанное отношение количества s- и l-транзакций. Подобная стохастическая модель потока – попытка построить последовательность случайных переходов между состояниями СМО в смысле (3), когда априорная информация об инвариантах последовательности (6) минимальна. Наборы данных, порождаемые в ходе таких ВЭ, содержали три трека. Это фиксированное число треков на самом деле оценка среднего количества треков в НД, которые порождаются в наших типичных сценариях ВЭ для решения прикладных задач статистического анализа. Размер каждого из треков – это случайная величина из равномерного распределения, а границы носителя меры, $support(track\_size)$, заданы как параметры ВЭ. Ниже представлены оценки, полученные в результате четырех серий ВЭ. В каждой серии носитель меры:

$$sup(track\_size) := U[1, max\_track\_size],$$

определяли, выбирая левую его границу, max_track_size, из множества $\{5,10,100,1000, 10000\}$. Другими словами, средний размер НД (в байтах) в каждой серии ВЭ был $\{250, 500, 10^3, 10^4, 10^5\}$ соответственно. Это множество было выбрано опять же исходя из нашего собственного опыта решения конкретных прикладных задач. Статистика была собрана на машинах с процессорами Intel, линейка i5 с тактовой частотой не ниже 3 ГГц, оперативная память – DDR2/DDR3 объемом от 2 до 8 Гб, внешние запоминающие устройства с SATA/PATA – интерфейсами, а операционные системы, выбранные для тестирования, – Windows XP/7, Linux. Одна из версий нашего приложения, позволяющего воспроизвести оценки, предложенные нами в этой работе, находится в свободном доступе [13].

9. ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММНЫХ РЕШЕНИЙ

Вначале представим оценки производительности l-транзакции. Для этого мы выбрали такую

Таблица 1. Оценки границ 95% толерантного интервала для средней длительности транзакции “извлечение shallow copy из репозитория по уникальному ключу НД, когда НД находится в кэше”, мкс

h *S*	250	500	10^3	10^4	10^5
LRU	7; 13	6; 13	6; 12	7; 16	7; 16
MRU	7; 12	7; 12	7; 13	7; 16	7; 16
GDSF	7; 16	7; 16	7; 16	7; 16	7; 17

Таблица 2. Оценки границ 95% толерантного интервала для средней длительности транзакции “извлечение shallow copy из репозитория по уникальному ключу НД, при условии, что этот НД был помещен в кэш в предыдущей транзакции”, мкс

h *S*	250	500	10^3	10^4	10^5
LRU	7; 20	7; 20	7; 20	7; 20	7; 20
MRU	7; 12	7; 13	7; 14	7; 17	7; 20
GDSF	10; 12	10; 15	10; 17	10; 25	10; 30

Таблица 3. Оценки границ 95% толерантного интервала для средней длительности транзакции “извлечение shallow copy из репозитория по суррогатному ключу НД, когда НД находится в кэше”, мкс

h *S*	250	500	10^3	10^4	10^5
LRU	6; 14	6; 12	6; 12	7; 17	6; 17
MRU	6; 9	6; 9	6; 16	6; 16	6; 17
GDSF	7; 16	7; 16	6; 16	7; 19	7; 20

Таблица 4. Оценки границ 95% толерантного интервала для средней длительности транзакции “извлечение shallow copy из репозитория по суррогатному ключу НД, при условии, что этот НД был помещен в кэш в предыдущей транзакции”, мкс

h *S*	250	500	10^3	10^4	10^5
LRU	6; 15	6; 12	6; 12	7; 13	6; 15
MRU	6; 7	6; 7	6; 7	6; 15	6; 16
GDSF	6; 7	6; 7	6; 7	6; 17	7; 20

Таблица 5. Оценки границ 95% толерантного интервала для средней длительности транзакции “извлечение shallow copy из репозитория по уникальному ключу НД, когда НД находится в кэше”, мкс

h *S*	250	500	10^3	10^4	10^5
LRU	4; 9	4; 9	4; 10	4; 10	4; 10
MRU	4; 8	4; 8	4; 9	4; 9	4; 9
GDSF	5; 12	5; 12	5; 12	5; 14	5; 16

меру, как 95% толерантные интервалы средней длительности транзакции соответствующего типа, и ограничились тремя хорошо известными реализациями алгоритмов политики размещения НД. В настоящее время наша m-реализация хранилища параметризуема более чем десятком различных алгоритмов политики размещения, большая часть из которых разработана нами. Но оценки производительности транзакций во всех случаях сопоставимы по порядку величин. В приведенных таблицах средний размер НД, в байтах, обозначен как *S*, а политика размещения – как *h*.

Нетрудно видеть, что средняя длительность l-транзакций незначительно увеличивается с ростом среднего размера НД. Более высокие оценки в случае GDSF-политики объясняются тем, что в ее алгоритме обработки l-транзакций выполняется еще и сбор статистики, регистрируется количество запросов на доступ к НД. На это дополнительно расходуется, по нашим оценкам, от 2 до 6 мкс.

Для сравнения приведем аналогичные оценки, полученные для альтернативной, наивной архитектуры хранилища и ее n-реализации.

В случае n-реализации границы 95% толерантных интервалов несколько ниже по сравнению с нашей m-реализацией. Это объясняется тем, что во втором случае в m-реализации хранилища дополнительное время (т.е. помимо копирования shared_ptr) расходуется на то, чтобы снабдить shallow-copy оригинального экземпляра НД, хранимого в RC, экземпляром сущности Control-Block. Этот тип данных спроектирован нами для контроля операций доступа shallow-copy к данным, хранящимся в оригинальном НД.

Мы трактуем приведенные выше оценки как обоснование следующего вывода: средняя длительность l-транзакций определяется константой, которая незначительно растет по мере увеличения размера НД (последнее – в логарифмической шкале).

Далее речь пойдет об оценках для l-транзакций, когда запрашиваемый НД находится в долговременном хранилище (RPS). В таблице 9 они приведены для m-реализации хранилища, в таблице 10 – для n-реализации.

В серии экспериментов, которые проводились для сбора этой статистики, в ряде случаев наблюдаем появление линейной зависимости средней длительности такой транзакции от количества НД в репозитории. Этот рост мы объясняем деградацией производительности системного механизма файлов, отображаемых в память, когда по мере увеличения объема файла увеличиваются смещения, по которым необходимо построить представления.

Таблица 6. Оценки границ 95% толерантного интервала для средней длительности транзакции “извлечение shallow copy из репозитория по уникальному ключу НД, при условии, что этот НД был помещен в кэш в предыдущей транзакции”, мкс

h *S*	250	500	10^3	10^4	10^5
LRU	4; 5	4; 5	4; 11	4; 11	4; 11
MRU	4; 5	4; 5	4; 12	4; 12	4; 12
GDSF	7; 8	6; 12	7; 19	7; 19	7; 20

Таблица 7. Оценки границ 95% толерантного интервала для средней длительности транзакции “извлечение shallow copy из репозитория по суррогатному ключу НД, когда НД находится в кэше”, мкс

h *S*	250	500	10^3	10^4	10^5
LRU	16; 35	15; 20	16; 45	16; 45	30; 70
MRU	15; 20	15; 20	15; 45	16; 50	20; 70
GDSF	16; 38	17; 25	17; 50	17; 50	25; 70

Таблица 8. Оценки границ 95% толерантного интервала для средней длительности транзакции “извлечение shallow copy из репозитория по суррогатному ключу НД, при условии, что этот НД был помещен в кэш в предыдущей транзакции”, мкс

h *S*	250	500	10^3	10^4	10^5
LRU	16; 30	16; 20	16; 50	16; 50	20; 60
MRU	15; 22	16; 20	16; 50	16; 50	20; 60
GDSF	17; 30	17; 20	17; 50	18; 50	20; 70

Как видим, в случае наивной реализации хранилища длительность l-транзакций этого типа значительно растет по мере увеличения размеров НД.

Оценки динамики s-транзакций демонстрируют линейные зависимости от количества хранимых в репозитории НД.

Оценка минимального времени s-транзакций, которая в большинстве серий экспериментов получена на участках, где количество хранимых в RPS НД не превышает 10^2, находится в пределах 200–300 мкс. Коэффициент, определяющий линейную зависимость среднего времени s-транзакций от числа хранимых НД, имеет порядок 2×10^{-2}–$3 \times \times 10^{-2}$. Это означает, что после накопления в файле RPS более 10^3 НД будет наблюдаться значимое (на 20–30 мкс) увеличение средней длительности s-транзакций.

Чтобы понять природу такой связи, необходимо принять во внимание семантику s-транзакции, реализуемой в коде RPS. Обработка s-транзакции в RPS начинается операцией поиска метаданных полученного НД в локальном (уровня RPS) справочнике. Если такой НД зарегистрирован в справочнике RPS, то s-транзакция на этом и заканчивается (т.к. НД уже хранится в RPS). Временная сложность такого поиска – логарифмическая, длительность этой операции в наших сериях ВЭ вырастает от нескольких микросекунд до двух-трех десятков микросекунд. Основные затраты, по времени, приходятся на непосредственно копирование (memcpy) НД во внешний файл, проецируемый в память. Такое копирование периодически предваряется операцией динамического изменения (увеличения) размера внешнего файла, а также последовательностью (атомарной транзакцией уровня RPS) из двух операций: закрытие текущего представления на запись и создание нового представления (по новому смещению). Динамическое расширение размера внешнего файла – это еще один механизм экономии вычислительных ресурсов. Пользователь, при желании, может указать в конфигурационных файлах ВЭ начальный размер файла RPS в несколько десятков гигабайтов, тем самым вовсе исключив операции динамического расширения этого файла. Перестройка представления на запись выполняется достаточно часто. Этот объект позволяет нам сократить до минимума время копирования НД в файл, но за счет потери той части адресного пространства, в которую представление отображается. Становится ясно, в структуру оценки средней длительности s-транзакции входят по меньшей мере три слагаемых: длительность самой операции копирования, длительность операции динамического увеличения файла RPS (редкое событие), длительность операций по перестройке представления для записи в файл (повторяются периодически). Согласно накопленной нами статистики, именно последнее слагаемое определяет линейную зависимость s-транзакций от количества хранимых в репозитории НД, точнее, от совокупного размера НД, хранимых в этом момент во внешнем файле.

Более того, перестройка представлений на запись (согласно статистике, полученной в результате мониторинга s-транзакций), довольно часто имеет характер переходного процесса в этой СМО. Представить адекватное статистическое описание таких участков без тщательного анализа кода, реализующего механизм файлов, отображаемых в память (уровень системных библиотек), не представляется возможным. Но такой анализ не входит в настоящее время в круг наших первоочередных задач.

10. ЗАКЛЮЧЕНИЕ

Предвосхищая возможные замечания, касающиеся приведенных в статье оценок производительности наших программных реализаций, мы согласимся, что они не покрывают варианты конфи-

Таблица 9. Оценки границ 95% толерантного интервала для средней длительности транзакции “извлечение shallow copy из репозитория по уникальному ключу НД, когда НД загружается в кэш в этой же транзакции”, мкс

h *S*	250	500	10^3	10^4	10^5
LRU	170; 400	110; 385	120; 500	125; 10^3	150; 1.2×10^3
MRU	180; 325	100; 380	100; 400	110; 950	130; 1.1×10^3
GDSF	175; 370	165; 310	90; 400	100; 2750	120; 2.3×10^3

Таблица 10. Оценки границ 95% толерантного интервала для средней длительности транзакции “извлечение shallow copy из репозитория по уникальному ключу НД, когда НД загружается в кэш в этой же транзакции”, мкс

h *S*	250	500	10^3	10^4	10^5
LRU	200; 1.6×10^3	180; 1.7×10^3	320; 6.5×10^4	500; 8.8×10^4	1460; 9×10^4
MRU	190; 10^3	180; 1.5×10^3	220; 4.9×10^4	500; 8×10^4	1490; 7×10^4
GDSF	200; 1.35×10^3	185; 1.6×10^3	300; 7.1×10^4	500; 8.6×10^4	1500; 9.2×10^4

гураций аппаратного обеспечения (АО), на котором возможно выполнение ВЭ. Однако заметим, что, во-первых, методика построения множества таких конфигураций определяется множеством доступного АО или тем его конкретным подмножеством, которое предполагают использовать для выполнения ВЭ. Во-вторых, мы приводим оценки производительности не в качестве конкретных ориентиров для конечного пользователя программной реализации хранилища, а как экспериментальное обоснование выводов, касающихся временной сложности транзакций всех типов. Если средняя длительность l-транзакции определяется константой, то на машине с другим аппаратным обеспечением мы можем иметь порядок, отличный от указанного в наших таблицах (даже если в последнем случае мы использовали те же параметры конфигурации экземпляра хранилища и выполняли ВЭ согласно тем же сценариям). Однако эта оценка средней длительности l-транзакции останется константой. Установив на машине SSD-накопители, пользователь может уменьшить порядок частного двух типов l-транзакций, но

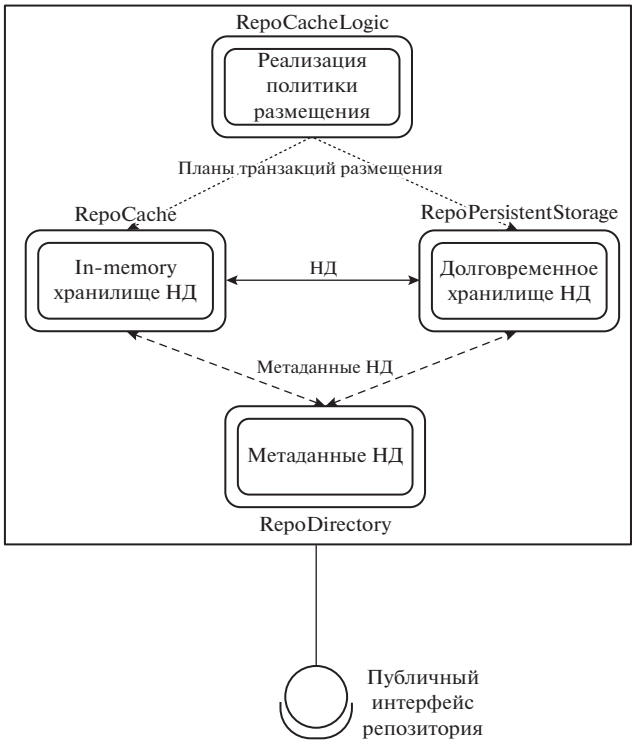

Рис. 2. Архитектура нашего репозитория.

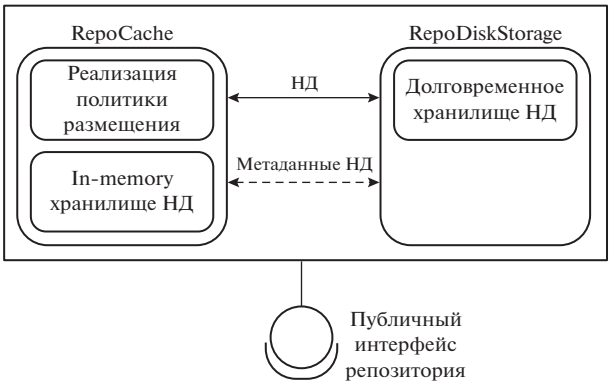

Рис. 3. Архитектура наивной реализации репозитория.

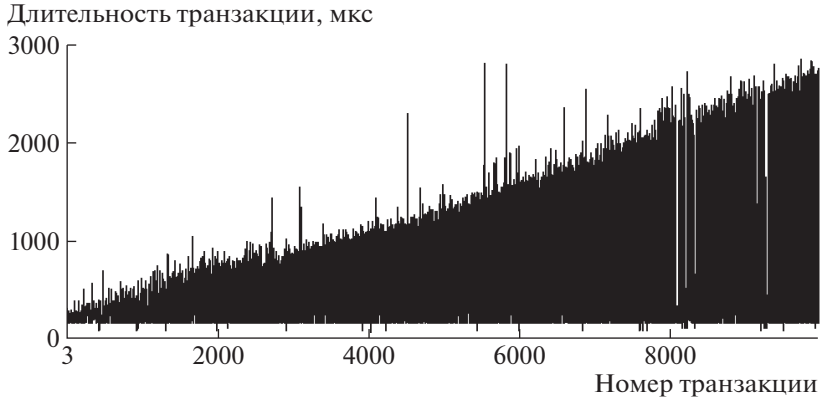

Рис. 4. Динамика длительности s-транзакций. Типичные оценки, полученные нами в одном из ВЭ.

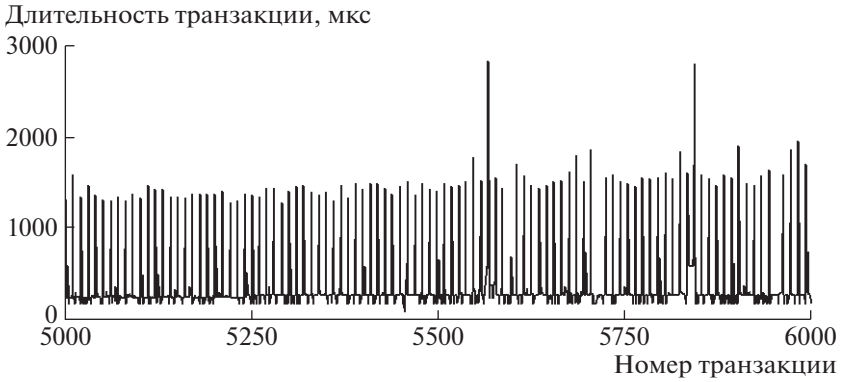

Рис. 5. Один из участков реализации, представленной на рис. 4, как иллюстрация структуры регистрируемого с.п.

сам характер этих зависимостей от количества хранимых в репозитории НД останется прежним. Прикладную значимость нашего программного решения нисколько не умаляет различие на два порядка средней длительности двух типов l-транзакций: в одном случае необходима предварительная загрузка затребованного НД из RPS в RC, а во втором случае в ней нет необходимости, т.к. затребованный НД уже размещен в RC. Напомним, что потери времени на загрузку НД из RPS в RC несравнимы с потерями времени на доступ к элементам этого НД, если его не удается разместить в кэше хранилища и доступ к данным осуществляется последовательными операциями чтения с внешнего запоминающего устройства. В заключение остается надеяться, что результаты нашей работы представляют интерес для достаточно широкого круга IT-специалистов, чья профессиональная деятельность включает в себя решение подобных задач, а неизбежная критика наших результатов будет конструктивной.

11. БЛАГОДАРНОСТИ

Один из авторов хотел бы выразить благодарность своим ученикам и коллегам: Андрею Олеговичу Гаврилову и Артёму Сергеевичу Тетюхину, которые в разное время и в разной мере принимали участие в проектах, имеющих отношение к этой работе.

Работа выполнялась нами как инициативная без какой-либо поддержки (финансовой или в иной форме) от каких-либо юридических или физических лиц. Для реализации изложенных в этой статье алгоритмов нами использовалось только свободно распространяемое программное обеспечение с открытым исходным кодом (CMake, CodeBlocks, gcc).

Приложение 1.
Пример фрагмента конфигурационного файла ВЭ

```
# интервальная оценка параметра пуассоновской модели
STEP_2 = task_cfg/POISSONLAMBDA/taskStep2.cfg
```

Приложение 2.
Содержимое конфигурационного файла

task_cfg/POISSONLAMBDA/taskStep2.cfg.

```
# определение входных данных эстиматора на языке запросов к репозиторию
REPO_ENTRIES = [ 1 [ 0 ] ]; [2 [0]]

# определение типа эстиматора
ESTIMATOR_TYPE = INTERVAL_ESTIMATOR

# строковый литерал, определяющий конкретную программную реализацию эстиматора
ESTIMATOR_NAME = IntervalPoissonLambda

# имя файла, содержащего параметры эстиматора
ESTIMATOR_PARAMS_SRC = task_cfg/POISSONLAMBDA/IntervalPoissonLambda.cfg

# декларативное определение класса модели, которой параметризуем эстиматор
MODEL_INTERFACE_TYPE =ProbabilisticModelCompositionInterface

# декларативное определение модели, которой параметризуем эстиматор
MODEL_NAME = POISSON_MODEL
```

СПИСОК ЛИТЕРАТУРЫ

1. Boost. http://www.boost.org
2. RaftLib. http://raftlib.io
3. R-package. http://www.cran.org
4. *Dean J. et al.* TensorFlow: A system for large-scale machine learning. – Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation. 2016. P. 265–283.
5. *McIlroy M.D.* Mass-produced software components. Eds. Naur, Randall. Proceedings, NATO Conference on Software. 1969. P. 88–98.
6. *Лаврищева Е.М.* Развитие отечественной технологии программирования. Кибернетика и системный анализ. 2014. Т. 50. № 3. 16 с.
7. *Лаврищева Е.М., Грищенко В.Н.* Сборочное программирование. Основы индустрии программных продуктов: 2-изд. Дополненное и переработанное. Киев: Наук. думка, 2009. 372 с.
8. https://www.ispras.ru/lavrishcheva/
9. *Иванков А.А.* Программный комплекс для изучения механизмов авторегуляции транскраниального кровообращения в режиме реального времени. Научно-технические ведомости СПбГПУ. Физико-математические науки. 2014. № 3(201). С. 92–109.
10. *Ivankov A.A.* Software platform for real time investigation of cerebral hemodynamics. Humanities and Science University Journal. 2014. V. 10. P. 37–49.
11. *Szyperski Clemens, Gruntz Dominik, Murer Stephan* Component Software Beyond Object-Oriented Programming. Second Edition. Addison-Wesley, 2002. P. 589,
12. *Morrison J. Paul* Flow-Based Programming: A New Approach To Application Development. Second Edition, J.P. Morrison Enterprises, Ltd, 2011
13. https://github.com/flower-flavour/embedded-repo
14. https://en.cppreference.com/w/cpp/language/move_constructor, абзац “Trivial move constructor”

EDN: LEDBTZ
ПРОГРАММИРОВАНИЕ, 2022, № 5, с. 68

ПРИГЛАШЕНИЕ К УЧАСТИЮ В 32-Й МЕЖДУНАРОДНОЙ КОНФЕРЕНЦИИ ГрафиКон-2022

DOI: 10.31857/S0132347422050077

ГрафиКон – крупнейшая в России и странах СНГ международная конференция по компьютерной графике, обработке изображений и машинному зрению, системам визуализации и виртуального окружения. Конференция проводится ежегодно, начиная с 1991 года. В сообщество организаторов конференции ГрафиКон на протяжении многих лет входят такие крупные учебные и исследовательские организации, как Московский государственный университет имени М.В. Ломоносова, Институт прикладной математики имени М.В. Келдыша РАН и др.

Миссией международной конференции ГрафиКон является содействие развитию в России компьютерной графики и связанных с ней областей; популяризация этих областей, совершенствование системы подготовки специалистов в сфере компьютерной графики, обработки изображений и машинного зрения; привлечение талантливых студентов, аспирантов, ученых и специалистов; расширение связей между академической наукой и промышленностью.

Приглашаем Вас принять участие в ***32-й Международной конференции по компьютерной графике, обработке изображение и машинному зрению, системам визуализации и виртуального окружения ГрафиКон 2022***, которая состоится 19-22 сентября 2022 года на базе Рязанского государственного радиотехнического университета им. В.Ф. Уткина.

Научные направления конференции

- Интеллектуальные решения в компьютерной графике
- Компьютерное зрение
- Научная визуализация и визуальная аналитика
- Геометрическое моделирование. Компьютерная графика и образование
- Обработка и анализ биомедицинских изображений
- Цифровая Земля и Большие данные
- Компьютерная графика в материаловедении, светотехнике и дизайне
- Искусственный интеллект, когнитивные технологии и робототехника
- Графические информационные системы и иммерсивные технологии на стадиях жизненного цикла продукта

Программа конференции дополнена молодежной сессией, в рамках которой будут представлены доклады студентов, аспирантов и молодых ученых, а также проведена выставка-презентация молодежных научных и инновационных проектов.

Все участники конференции получат сертификаты о повышении квалификации. Лучшие доклады и презентации среди участников молодежной сессии будут отмечены дипломами.

Доклады будут опубликованы в сборнике трудов, индексируемом в РИНЦ. Лучшие доклады будут рекомендованы к публикации в виде полнотекстовых статей в журналах “Программирование”, “Светотехника”, “Научная визуализация”, “Вестник Рязанского государственного радиотехнического университета”, “Геоконтекст” и др.

Сайт конференции: https://graphicon.rsreu.ru.