ISSN 0132-3474

Номер 5 Сентябрь - Октябрь 2024

ПРОГРАММИРОВАНИЕ

НАУКА
1727

СОДЕРЖАНИЕ

Номер 5, 2024

ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

ТЕОРИЯ ПРОГРАММИРОВАНИЯ: ФОРМАЛЬНЫЕ МОДЕЛИ И СЕМАНТИКА

Contents

No. 5, 2024

PARALLEL AND DISTRIBUTED SOFTWARE

COMPUTER GRAFICS AND VISUALIZATION

THEORETICAL COMPUTER SCIENCE: FORMAL MODELS AND SEMANTICS

ПРОГРАММИРОВАНИЕ, 2024, № 5, с. 3–13

ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ

УДК 514.182.7:004.925.8

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ МАТЕМАТИЧЕСКОГО АППАРАТА РЕАЛИЗАЦИИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ В СИСТЕМАХ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ

© 2024 г. Е. Конопацкий[a, *]

[a] *Нижегородский государственный архитектурно-строительный университет*
603000 Нижний Новгород, Нижегородская обл., ул. Ильинская, 65, корп. 1, Россия
**E-mail: e. v.konopatskiy@mail.ru*

Поступила в редакцию: 18.03.2024 г.
После доработки: 25.04.2024 г.
Принята к публикации: 25.04.2024 г.

Цель работы – разработка математического аппарата и вычислительных алгоритмов реализации параллельных вычислений в системах геометрического моделирования и автоматизированного проектирования. Выполнен анализ существующих подходов к реализации параллельных вычислений в САПР. В результате установлено, что для большинства систем информационного моделирования и автоматизированного проектирования отсутствует поддержка параллельных вычислений на уровне геометрического ядра. Предложена концепция разработки геометрического ядра САПР, основанного на инвариантах параллельного проецирования геометрических объектов на оси глобальной системы координат, которая объединяет в себе потенциал конструктивных методов геометрического моделирования, способных обеспечить распараллеливание геометрических построений по задачам (message passing), и математического аппарата “Точечное исчисление”, способного реализовать распараллеливание по данным за счет покоординатного расчета (data parallel). Использование покоординатного расчета точечных уравнений позволяет не только распараллелить вычисления по координатным осям, но и обеспечить согласованность вычислительных операций по потокам, что значительно уменьшает простой вычислений и оптимизирует работу процессора для достижения максимального эффекта от использования параллельных вычислений.

Ключевые слова: САПР, математический аппарат, параллельные вычисления, точечное исчисление, покоординатный расчет, координатный вектор, инварианты параллельного проецирования, скрытый параллелизм

DOI: 10.31857/S0132347424050012, **EDN:** OMCLWB

1. ВВЕДЕНИЕ

В современном мире аппаратная часть современных персональных компьютеров постоянно совершенствуется в части распараллеливания вычислительных процессов. Например, Ampere Altra Q80–33 является 80-ядерным серверным процессором на базе ARM. Все большей популярностью для реализации параллельных вычислений пользуются графические процессоры, оснащенные уже не сотнями, а тысячами ядер (например, видеокарта Nvidia RTX 3090 имеет 10496 ядер). Это приводит к тому, что имея достаточное количество средств и времени можно мобилизовать огромные вычислительные ресурсы даже в домашних условиях. Не говоря уже о том, что производство суперкомпьютеров уже стало серийным. Все это ставит новые требования к программному обеспечению, краеугольным камнем которого должны быть параллельные вычисления. Современные же программные продукты не всегда реализованы с поддержкой параллельных вычислений. Это приводит к частому простою вычислительных потоков и сводит их огромный вычислительный потенциал до возможностей одноядерного процессора. На данный момент, подобная картина знакома уже многим пользователям домашних операционных систем, не говоря уже о суперкомпьютерах. Она же характерна для всех без исключения систем твердотельного моделирования и автоматизированного проектирования, у которых лишь немногие операции реализованы с помощью параллельных вычислений. Учитывая темпы цифровизации всех современных отраслей промышленности, включающие построение цифровых двойников изделий и их сопровождение на всех этапах жизненного цикла,

приводит к необходимости одномоментно оперировать огромными массивами информации. Уже не фантастикой, а реальностью становятся цифровые двойники целых заводов, содержащие до нескольких миллионов отдельных деталей. Таким образом, чтобы обеспечить потребности в цифровизации производственных циклов, необходима разработка новой концепции программного обеспечения, которая опиралась бы на математический аппарат, способный обеспечить реализацию параллельных вычислений на уровне геометрического ядра систем твердотельного моделирования и автоматизированного проектирования не только на этапе проектирования, но и на этапе сопровождения существования промышленных объектов на протяжении всего их жизненного цикла. А решение научной проблемы разработки математического аппарата параллельных вычислений является актуальной и весьма значимой научной задачей, которая создает предпосылки для формирования новых научных направлений в геометрическом и компьютерном моделировании объектов, процессов и явлений, а также создании высокопроизводительных программных продуктов на их основе.

2. СОВРЕМЕННОЕ СОСТОЯНИЕ ИССЛЕДОВАНИЙ ПО ДАННОЙ ПРОБЛЕМЕ

За последние десятилетия развития компьютерной техники активное распространение получили многоядерные процессоры, которые могут параллельно выполнять несколько вычислительных операций. Это обеспечивает современным компьютерным системам значительный прирост производительности и в значительной мере экономит энергопотребление центрального процессора, что привело к их массовому внедрению для эффективного решения многих научно-технических задач в различных отраслях человеческой деятельности [1–6]. Вместе с тем возникает проблема разработки программного обеспечения, способного в полной мере реализовать потенциал многоядерных процессоров за счет согласованного выполнения параллельных вычислений.

На текущий момент существует два основных подхода к реализации параллельных вычислений: распараллеливание по задачам (message passing) и распараллеливание по данным (data parallel). Распараллеливание по задачам подразумевает фрагментацию решаемой задачи на независимые подзадачи, каждую из которых можно решать отдельно. Одним из недостатков такого подхода является уникальность фрагментации каждой отдельной задачи. Кроме того, далеко не каждая задача может быть подвергнута фрагментации. Распараллеливание по данным является более универсальным подходом, не привязанным к решению конкретной задачи. Такой подход часто используется при решении задач численного моделирования, связанных с формирование многомерной сети точек, формирующих простые геометрические объекты. В математической физике такой подход к реализации параллельных вычислений получил название геометрический параллелизм [7–9].

Несмотря на большое количество исследований [10–15], одним из проблемных секторов внедрения многоядерных компьютерных систем остается компьютерная графика. К сожалению, существующие системы автоматизированного проектирования (САПР) и твердотельного моделирования не используют в полной мере многоядерные возможности современных процессоров. Например, на официальном сайте одного из флагманов САПР компании Autodesk имеется следующая информация: “Программы AutoCAD и AutoCAD for Mac поддерживают технологию многоядерных процессоров только в отдельных областях применения (2D-регенерация). Чтобы воспользоваться всеми преимуществами многоядерных процессоров, необходимо использовать многопоточное программное обеспечение. AutoCAD представляет собой приложение с однопотоковой обработкой”. В отечественных САПР многопоточность также реализована далеко не для всех операций геометрического моделирования. Например, основные многопоточные операции одного из лидеров среди отечественных производителей – геометрического ядра C3D включают: построение плоских проекций, расчет полигональных сеток, расчет массо-центровочных характеристик, операции конвертеров и некоторые другие. Для реализации многопоточности в ядре C3D используется технология OpenMP, которая реализует параллельные вычисления путем выделения ведущего (master) потока, формирующего набор ведомых потоков и распределяя задачи между ними. Таким образом, параллельные вычисления реализуются на программном уровне, а не за счет математического аппарата, что в значительной степени ограничивает их возможности. Одной из причин сложившейся ситуации является отсутствие полной поддержки параллельных вычислений на уровне геометрического ядра системы, ограниченного возможностями математического аппарата. Подобная же

картина наблюдается в географических информационных системах и системах дистанционного зондирования Земли [16–19]. Исходя из вышеизложенного разработка математического аппарата геометрического моделирования, способного обеспечить параллельные вычисления на уровне геометрического ядра является актуальной и востребованной научной задачей.

Вообще, вычислительные алгоритмы компьютерной графики, основанные на методах конструктивного геометрического моделирования, прекрасно поддаются распараллеливанию вычислений. Например, в работе [20] приводится пример структуры алгоритма решения задачи Аполлония, допускающей распараллеливание вычислений и их конвейерную организацию, который относится к распараллеливанию по задачам и предусматривает согласованное и единовременное построение нескольких геометрических объектов. Однако, в процессе проектирования нет острой необходимости единовременного построения разных геометрических объектов. Проектировщик, использующий САПР для построения 3-мерной модели, не будет одновременно строить несколько окружностей, прямых или других геометрических объектов. Он будет строить их по очереди, применяя инструменты САПР. Другое дело, что при построении самих геометрических объектов можно и нужно использовать параллельные вычисления. Это приводит к необходимости разработки новой концепции разработки геометрического ядра САПР, основанного на эффективном использовании всех возможных подходов к реализации параллельных вычислений.

3. МАТЕМАТИЧЕСКИЙ АППАРАТ РЕАЛИЗАЦИИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

В качестве математического аппарата, способного обеспечить эффективную реализацию геометрического ядра САПР нового поколения, предлагается использовать точечное исчисление [21] исходя из следующих соображений. Базовым методом точечного исчисления является метод проецирования геометрического объекта на оси глобальной системы координат, в отличие от начертательной геометрии, в основу которой положен метод проецирования пространственных геометрических объектов на плоскости проекций. Такой подход позволяет переходить от символьных точечных уравнений к системе однотипных параметрических уравнений. Геометрическая интерпретация такого перехода представляет собой параллельное проецирование пространственного геометрического объекта на оси глобальной системы координат. Аналитически этот процесс описывается систематической заменой точек на соответствующие их координаты. Эта операция получила название покоординатного расчета. Вместе с тем покоординатный расчет целесообразно выполнять тогда, когда получено итоговое точечное уравнение моделируемого геометрического объекта, что значительно сокращает время необходимых вычислений. Геометрическое моделирование в точечном исчислении осуществляется на основе геометрической схемы – графического алгоритма построения геометрического объекта с последующим описанием в виде точечных уравнений и вычислительных алгоритмов на их основе. При этом каждой графической операции ставится в соответствие аналитическая операция. Это привело к разработке инструментов геометрического моделирования, инвариантных относительно параллельного проецирования, которые способны обеспечить покоординатный расчет точечных уравнений [22]. Например, определение отрезка прямой в 3-мерном пространстве благодаря инвариантным свойствам параметра точечного исчисления, которым является простое отношение трех точек прямой (рис. 1), выглядит следующим образом:

$$M = (B - A)t + A \Leftrightarrow \begin{cases} x = (x_B - x_A)t + x_A \\ y = (y_B - y_A)t + y_A \\ z = (z_B - z_A)t + z_A \end{cases}.$$

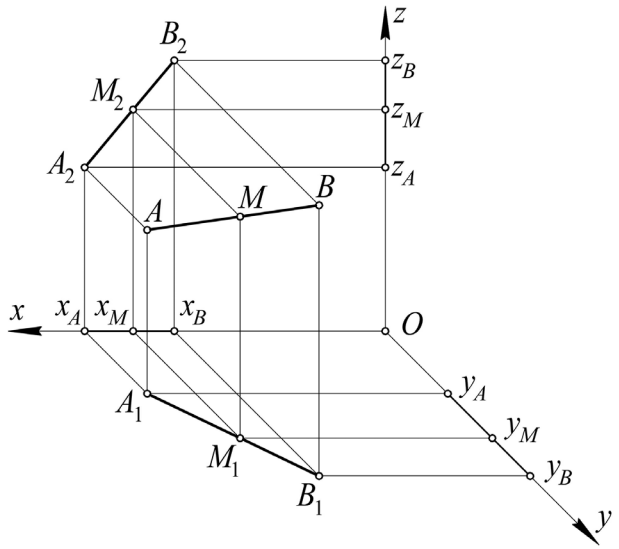


Рис. 1. Геометрическая интерпретация покоординатного расчета для отрезка прямой в 3-мерном пространстве.

Этот же подход справедлив и для многомерного пространства. Фактически размерность пространства определяет необходимое количество осей проекций и соответственно количество параметрических уравнений системы. Теоретически таких осей может быть бесконечное множество, как и однотипных параметрических уравнений, которые являются аналитическим представлением геометрической операции покоординатного расчета.

Также справедливым остается проецирование не только на оси проекций, но и на любую прямую, что позволяет работать с подпространствами (локальными симплексами), а результат получать в глобальной системе координат (глобальном симплексе). Причем переход от локального симплекса к глобальному осуществляется автоматически путем замены точек на их точечные уравнения и не требует дополнительных вычислений.

В общем виде точечное уравнение любого геометрического объекта можно представить в виде суммы произведений точек симплекса на функции от текущих параметров:

$$M = \sum_{i=1}^{n} A_i p_i \Leftrightarrow \begin{cases} x_M = \sum_{i=1}^{n} x_{A_i} p_i \\ y_M = \sum_{i=1}^{n} y_{A_i} p_i \\ z_M = \sum_{i=1}^{n} z_{A_i} p_i \\ \dots \end{cases}, \quad (1)$$

где M – текущая точка, которая своим движением заполняет пространство, формируя геометрический объект; A_i – точки, определяющие исходный симплекс многомерного пространства; p_i – функции от текущих параметров (u, v, w,...), которые обеспечивают движение текущей точки M; n – размерность пространства исходного симплекса.

Условием принадлежности текущей точки исходному симплексу является суммарное значение функций от текущих параметров равное 1:

$$\sum_{i=1}^{n} p_i = 1.$$

Как видно из (1), все параметрические уравнения системы, полученные на основе точечного уравнения, являются полностью однотипными. Меняются только координаты точек симплекса, которые относятся к исходным или промежуточным данным. В этом случае все точки в уравнении (1) по сути являются координатными векторами. Из этих соображений точечное исчисление можно считать особым видом векторного исчисления, способным индуцировать системы однотипных параметрических уравнений, обеспечивая в результате геометрического моделирования, так называемый, скрытый параллелизм. Тогда уравнение (1) будет представлено следующим образом:

$$M = \sum_{i=1}^{n} \begin{pmatrix} x_{A_i} \\ y_{A_i} \\ z_{A_i} \\ \dots \end{pmatrix} p_i.$$

Используя это свойство точечных уравнений, для определения геометрических объектов многомерного пространства будем выполнять параллельные вычисления для каждой отдельной оси проекций. Таким образом, получим декомпозицию задачи геометрического моделирования или распараллеливание вычислений по данным. И чем больше будет размерность пространства, в котором определяется геометрический объект, тем больше вычислительных потоков многоядерного процессора могут быть одновременно задействованы. Такому же распараллеливанию вычислений могут быть подвергнуты все вычислительные алгоритмы определения многомерных геометрических объектов, представленные в виде последовательности точечных уравнений. В результате получим:

$$M = \sum_{i=1}^{n} A_i p_i \Leftrightarrow \begin{cases} x_M = \sum_{i=1}^{n} x_{A_i} p_i & \text{1-й вычислительный поток} \\ y_M = \sum_{i=1}^{n} y_{A_i} p_i & \text{2-й вычислительный поток} \\ z_M = \sum_{i=1}^{n} z_{A_i} p_i & \text{3-й вычислительный поток} \\ \dots & n\text{-й вычислительный поток} \end{cases} \quad (2)$$

Как видно из уравнения (2) параметрические уравнения системы являются полностью независимыми друг от друга и обеспечивают полную синхронизацию расчета геометрического объекта в целом, поскольку они одновременно начинаются и одновременно заканчиваются.

Следует отметить, что при определении метрических характеристик геометрических объектов с помощью метрического оператора может понадобиться ряд промежуточных вычислений, предусматривающих особый вид покоординатного расчета, который усложняет распараллеливание вычислений по данным. Эта особенность не является следствием использования точечного исчисления и относится ко всем современным САПР. Вместе с тем, доля таких вычислений несравненно мала и такие вычисления легко поддаются распараллеливанию по задачам.

В исследованиях [24] отмечалось, что выигрыш по времени вычислений не пропорционален числу ядер процессора. Это связано с невозможностью равномерного распределения работы по всем ядрам. Возможна и другая проблема, если загрузка ядер процессора достаточно равномерная, но скорость обмена данными низкая, то основная часть времени будет тратиться впустую на ожидание информации, необходимой для дальнейшей работы данного ядра процессора. В предложенном подходе к распараллеливанию вычислений путем покоординатного расчета все вычисления не просто проходят параллельно, чего можно было бы достичь с использованием обычных систем параметрических уравнений. Результатом покоординатного расчета точечного исчисления является система однотипных параметрических уравнений, в которых отличаются лишь соответствующие координаты точек, а все остальные функции от параметров остаются неизменными, что следует из уравнения (1). Это приводит к тому, что количество математических операций, необходимых для вычисления координат точек, является одинаковым, обеспечивает согласованное выполнение вычислений и минимизирует простой ядер. Таким образом, предложенный подход повышает общую производительность многоядерной системы за счет согласованного выполнения всех вычислительных операций.

4. МОДЕЛИРОВАНИЕ ГЕОМЕТРИЧЕСКИХ ОБЪЕКТОВ С УЧЕТОМ РЕАЛИЗАЦИИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Рассмотрим несколько примеров моделирования геометрических объектов в точечном исчислении сквозь призму реализации параллельных вычислений. Основным формообразующим инструментом геометрического моделирования является кривая. Следует отметить, что современные САПР ограничены в выборе инструментов геометрического моделирования кривых линий, из которых в основном используются окружности и их дуги, эллипсы, цилиндрические спирали и сплайны, как некий универсальный инструмент моделирования кривых, позволяющий с определенной точность аппроксимировать требуемую геометрическую форму. Некоторые из САПР, такие как Компас 3D, содержат инструменты геометрического моделирования конических сечений, а также проекционных линий и линий пересечения двух поверхностей (в частном случае поверхности с плоскостью). При этом линия, полученная в результате пересечения поверхностей, определяется с помощью сплайна, т. е., по сути, аппроксимирует ее.

Исходя из этого рассмотрим пример моделирования кривых 2-го порядка в точечном исчислении. Вообще, за десятилетия развития точечного исчисления было получено множество параметризаций кривых линий и поверхностей на их основе, некоторые из которых приведены в работе [21]. Геометрическая схема конструирования кривой 2-го порядка, которая определяется с помощью инженерного дискриминанта $k = \dfrac{KC}{T_1C}$, представлена на рис. 2. В данном случае кривая

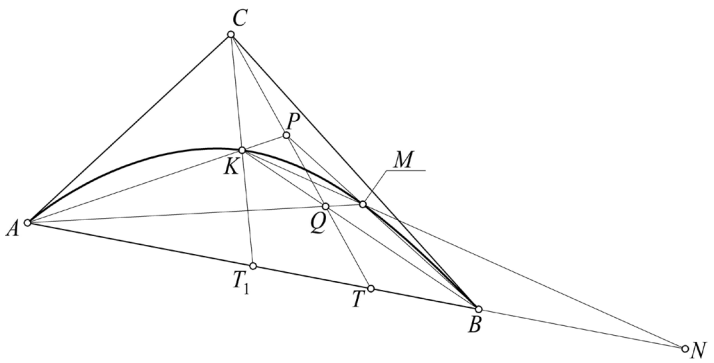


Рис. 2. Геометрическая схема конструирования кривой 2-го порядка.

является дугой обвода (т. е. имеет касательные в начальной и конечной точках) и проходит через 3 точки: A, K и B.

Точечное уравнение такой кривой и соответствующая ему система точечных уравнений представлена ниже:

$$M = (A - C)\frac{k\bar{u}^2}{k(1-2u)^2 + 2\bar{u}u} + (B - C)\frac{ku^2}{k(1-2u)^2 + 2\bar{u}u} + C.$$

$$\Updownarrow$$

$$\begin{cases} x_M = (x_A - x_C)\dfrac{k\bar{u}^2}{k(1-2u)^2 + 2\bar{u}u} + (x_B - x_C)\dfrac{ku^2}{k(1-2u)^2 + 2\bar{u}u} + x_C \\ y_M = (y_A - y_C)\dfrac{k\bar{u}^2}{k(1-2u)^2 + 2\bar{u}u} + (y_B - y_C)\dfrac{ku^2}{k(1-2u)^2 + 2\bar{u}u} + y_C \\ z_M = (z_A - z_C)\dfrac{k\bar{u}^2}{k(1-2u)^2 + 2\bar{u}u} + (z_B - z_C)\dfrac{ku^2}{k(1-2u)^2 + 2\bar{u}u} + z_C \end{cases} \quad (3)$$

где $0 \le u \le 1$ – текущий параметр; $\bar{u} = 1 - u$ – дополнение соответствующего параметра до единицы.

В соответствии с [21], при $k = 0.5$ получим дугу параболы, при $0 < k < 0.5$ – дугу гиперболы, а при $0.5 < k < 1$ – дугу эллипса.

Рассмотрим другой пример моделирования дуги обвода 3-го порядка на основе конфигурации Дезарга [22] (рис. 3).

$$M = \frac{A\bar{u}^3 + Bu^2\bar{u} + C\bar{u}^2u + Du^3}{2u^2 - 2u + 1}, \quad (4)$$

где $0 \le u \le 1$ – текущий параметр.

С учетом покоординатного расчета точечного уравнения (4) для трехмерного пространства получим систему однотипных параметрических уравнений для каждого из трех вычислительных потоков:

$$\begin{cases} x_M = \dfrac{x_A\bar{u}^3 + x_Bu^2\bar{u} + x_C\bar{u}^2u + x_Du^3}{2u^2 - 2u + 1} \\ y_M = \dfrac{y_A\bar{u}^3 + y_Bu^2\bar{u} + y_C\bar{u}^2u + y_Du^3}{2u^2 - 2u + 1} \\ z_M = \dfrac{z_A\bar{u}^3 + z_Bu^2\bar{u} + z_C\bar{u}^2u + z_Du^3}{2u^2 - 2u + 1} \end{cases}.$$

Следует отметить, что уравнения (3) и (4) получены на основе классических проективных алгоритмов [23] и в процессе экспериментов показали высочайший уровень устойчивости при изменении исходных данных.

Проверим масштабируемость точечного исчисления на примере моделирования поверхностей (рис. 4) и тел (рис. 5) в трехмерном пространстве.

Моделирование участка интерполяционной поверхности, проходящей через 9-точек [24], удобно представить в виде вычислительного алгоритма последовательности точечных уравнений, каждое из которых позволяет реализовать параллельные вычисления за счет покоординатного расчета:

$$\begin{cases} M_A = A_1\bar{u}(1-2u) + 4A_2\bar{u}u + A_3u(2u-1) \\ M_B = B_1\bar{u}(1-2u) + 4B_2\bar{u}u + B_3u(2u-1) \\ M_C = C_1\bar{u}(1-2u) + 4C_2\bar{u}u + C_3u(2u-1) \\ M = M_A\bar{v}(1-2v) + 4M_B\bar{v}v + M_Cv(2v-1) \end{cases},$$

где M – текущая точка отсека параболической поверхности; M_A, M_B, M_C – текущие точки направляющих параболических дуг; A_i, B_i, C_i – исходные точки, координаты которых соответствуют исходной экспериментально-статистической информации; u и v – текущие параметры отсека параболической поверхности; $\bar{u} = 1 - u$ и $\bar{v} = 1 - v$ – дополнение текущих параметров до 1.

В данном случае конструктивные алгоритмы геометрического моделирования позволяют реализовать параллельные вычисления не только по данным за счет покоординатного расчета, но и по задачам, предусматривающим определения массива направляющих линий с последующим расчетом образующих.

Следует отметить, что масштабируемость точечных уравнений и вычислительных алгоритмов на их основе также подтверждается геометрической теорией многомерной интерполяции [25] с возможностью обобщения геометрических объектов на многомерное пространство.

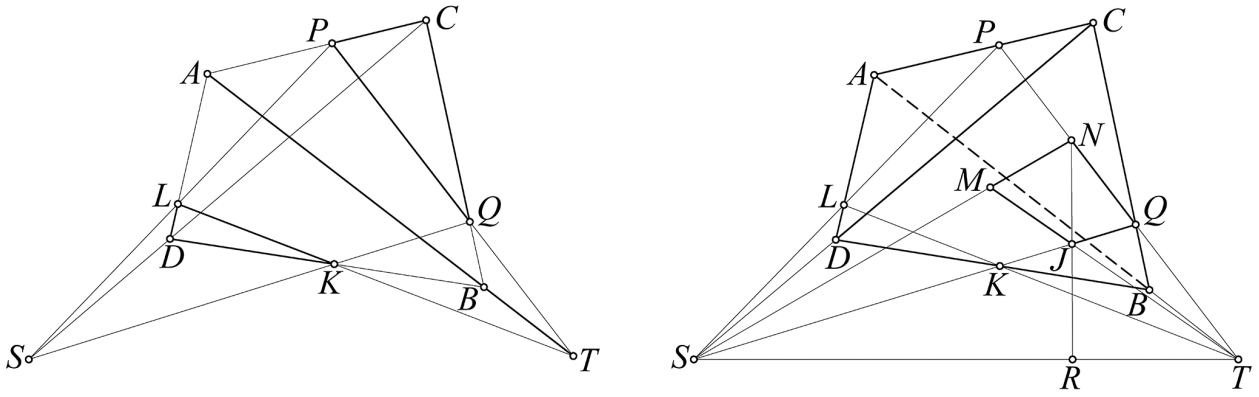


Рис. 3. Геометрическая схема моделирования дуги обвода на основе конфигурации Дезарга.

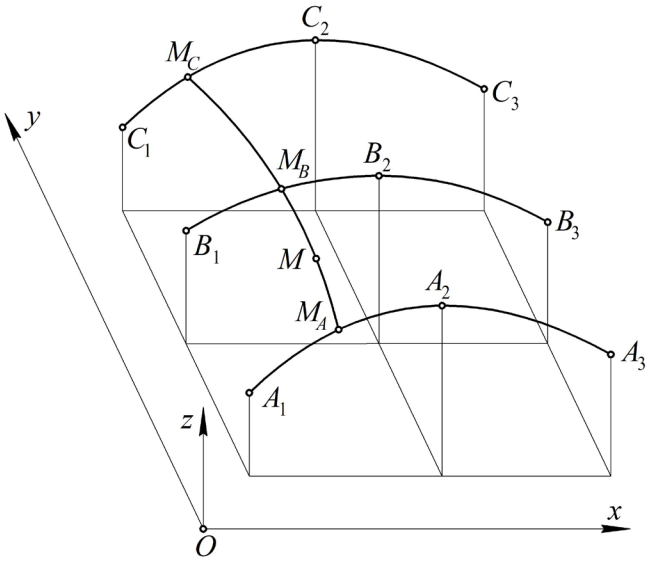


Рис. 4. Геометрическая схема моделирования 9-точечного отсека поверхности.

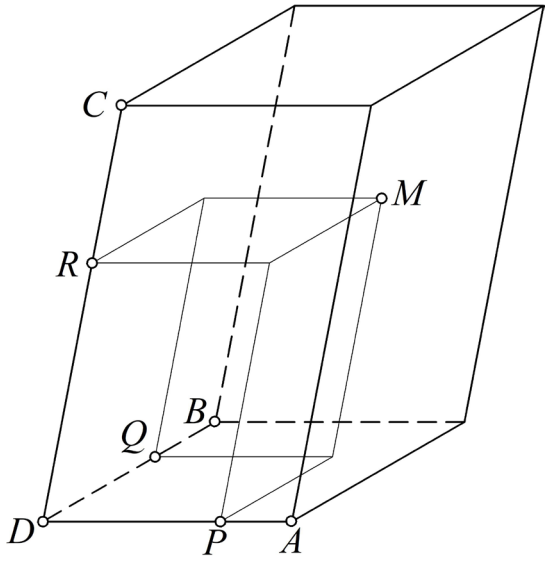


Рис. 5. Геометрическая схема определения твердотельной модели параллелепипеда.

В качестве последнего примера рассмотрим твердотельную геометрическую модель параллелепипеда с вершинами A, B, C, D (рис. 5), которая в линейном пространстве определяется простым точечным уравнением и соответствующей ему системой параметрических уравнений:

$$M = Au + Bv + Cw + D(1 - u - v - w).$$

$$\Updownarrow$$

$$\begin{cases} x_M = x_A u + x_B v + x_C w + x_D (1 - u - v - w) \\ y_M = y_A u + y_B v + y_C w + y_D (1 - u - v - w), \\ z_M = z_A u + z_B v + z_C w + z_D (1 - u - v - w) \end{cases}$$

где A, B, C, D – точки, которые не только сами по себе формируют локальный симплекс трехмерного пространства, но и однозначно определяют положение и размеры параллелепипеда в глобальной системе координат; M – текущая точка, которая своим движением заполняет трехмерное пространство формируя твердотельную модель параллелепипеда; $u = \dfrac{DP}{DA}$, $v = \dfrac{DQ}{DB}$, $w = \dfrac{DR}{DC}$ – текущие параметры, которые изменяются от 0 до 1.

5. ВЫВОДЫ

Исходя из вышеизложенного можно сделать следующие выводы о перспективах использования точечного исчисления в качестве математического аппарата для реализации параллельных вычислений в САПР [26]:

Математический аппарат “Точечное исчисление” позволяет реализовать декомпозицию задач геометрического моделирования на однотипные независимые подзадачи за счет покоординатного расчета и инвариантных свойств точечных уравнений относительно параллельного проецирования.

Все параметрические уравнения системы, полученные на основе точечных уравнений, явля-

ются полностью независимыми друг от друга, каждая из которых составляет отдельную проекцию оригинала на оси глобальной системы координат.

Реализация покоординатного расчета точечных уравнений обеспечивает полную синхронизацию параллельных вычислений, при которой в уравнении меняются исключительно координаты точек. В остальном математическое выражение точечного уравнения остается полностью неизменным, что позволяет минимизировать простой процессора и тем самым оптимизировать общую производительность системы.

Все точечные уравнения и вычислительные алгоритмы на их основе основаны на конструктивных алгоритмах параметризации геометрических моделей с учетом графических алгоритмов аффинной и проективной геометрии, что обеспечивает их отказоустойчивость для любых частных случаев решения задачи.

На приведенных примерах моделирования отсека поверхности видно, что предложенный математический аппарат геометрического моделирования является полностью масштабируемым за счет использования специального метода подвижного симплекса и инвариантных свойств точечного исчисления, способного обеспечить обобщение решаемой задачи на многомерное пространство. Причем, чем выше размерность пространства, тем в большей степени удается достигнуть реализации параллельных вычислений, что позволяет предложенному математическому аппарату быть эффективным инструментом многомерной интерполяции и аппроксимации [25].

Дополнительной реализации параллельных вычислений можно достичь за счет дискретизации текущих параметров геометрических объектов на заданные интервалы значений, что является очень актуальным для рендеринга трехмерных изображений и относится к перспективе дальнейших исследований.

ИСТОЧНИК ФИНАНСИРОВАНИЯ

Работа выполнена в рамках государственного задания Министерства науки и высшего образования Российской Федерации (№ 124012000104-5 “Математический аппарат реализации параллельных вычислений в системах геометрического моделирования и автоматизированного проектирования”).

СПИСОК ЛИТЕРАТУРЫ

1. Разработка параллельного программного кода для расчетов задачи радиационной магнитной газодинамики и исследования динамики плазмы в канале КСПУ / В.А. Бахтин, Д.А. Захаров, А.Н. Козлов, В.С. Коновалов // Научный сервис в сети Интернет. 2019. № 21. С. 105–118. https://doi.org/10.20948/abrau-2019-80
2. *Пекунов В.В.* Предицирующие каналы в параллельном программировании: возможное применение в математическом моделировании процессов в сплошных средах // Программные системы и вычислительные методы. 2019. № 3. С. 37–48. https://doi.org/10.7256/2454-0714.2019.3.30393
3. *Воробьев В.Е., Мурынин А.Б., Хачатрян К.С.* Высокопроизводительная регистрация пространственных спектров морского волнения при оперативном космическом мониторинге обширных акваторий // Исследование Земли из космоса. 2020. № 2. С. 56–68. https://doi.org/10.31857/S0205961420020062
4. *Goncharsky A.V., Romanov S.Y., Seryozhnikov S.Y.* Implementation and performance of wave tomography algorithms on SIMD CPU and GPU computing platforms // Numerical Methods and Programming. 2021. V. 22. No 4. P. 322–332. https://doi.org/10.26089/NumMet.v22r421
5. *Шмаков И.А., Иордан В.И., Соколова И.Е.* Компьютерное моделирование св-синтеза алюминида никеля методом молекулярной динамики в пакете LAMMPS с использованием параллельных вычислений // Высокопроизводительные вычислительные системы и технологии. 2018. Т. 2. № 1. С. 48–54.
6. *Федотов В.Л.* Использование архитектуры параллельных вычислений в подходе к построению самолетных комплексов систем управления // Навигация и управление летательными аппаратами. 2019. № 1(24). С. 12–20.
7. *Пекунов В.В.* Улучшенная балансировка загрузки процессоров при численном решении задач механики сплошной среды, осложненных химической кинетикой // Кибернетика и программирование. 2021. № 1. С. 13–19. https://doi.org/10.25136/2644-5522.2021.1.35101
8. Параллельный алгоритм трассировки лучей для анализа поля излучения и построения обскурограмм излучающего газа / О.Г. Ольховская, В.А. Гасилов, А.М. Котельников, М.В. Якобовский // Препринты ИПМ им. М.В. Келдыша. 2018. № 143. С. 1–16. https://doi.org/10.20948/prepr-2018–143
9. Development of parallel algorithms for intelligent transportation systems / B.N. Chetverushkin, A.A. Chechina, N.G. Churbanova, M.A. Trapeznikova // Mathematics. 2022. V. 10. No. 4. https://doi.org/10.3390/math10040643
10. *Кучеров Д.П., Моргун К.О., Аникеенко Л.С.* Средства управления параллельными вычислениями в задачах компьютерной графики // Наукоємні технології. 2018. Т. 38. № 2. С. 178–186. https://doi.org/10.18372/2310-5461.38.12833

11. *Nizovskikh A.S., Koporushkin P.A., Tarasenko R.R.* Problems of parametric approach in some modern CAD // Современные проблемы теории машин. 2016. No. 4–1. P. 83–85.
12. *Абрамов О.В.* Вычислительная среда для решения задач автоматизации проектирования на многопроцессорных системах // Математические методы в технике и технологиях – ММТТ. 2018. Т. 5. С. 28–30.
13. A large-scale parallel hybrid grid generation technique for realistic complex geometry / Z. Zhao [et al.] // International Journal for Numerical Methods in Fluids. 2020. V. 92. No. 10. P. 1235–1255. https://doi.org/10.1002/fld.4825
14. *Kukreja A., Dhanda M., Pande S.S.* Voxel-based adaptive toolpath planning using graphics processing unit for freeform surface machining. Journal of Manufacturing Science and Engineering. Transactions of the ASME. 2022. 144(1). https://doi.org/10.1115/1.4051535
15. *de Matos Menezes M., Viana Gomes de Magalhães S., Aguilar de Oliveira M., Randolph Franklin W., de Oliveira Bauer Chichorro R.E.* Fast parallel evaluation of exact geometric predicates on GPUs. CAD Computer Aided Design. 2022. 150. https://doi.org/10.1016/j.cad.2022.103285
16. *Ощепков А.Ю., Попов С.Е.* Разработка информационно-вычислительной системы на базе Apache Hadoop для обработки гипер-и мультиспектральных данных дистанционного зондирования земли. Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. 2016. № 3. С. 95–105.
17. *Zieg J., Zawada D.G.* Improving esri arcgis performance of coastal and seafloor analyses with the python multiprocessing module. Journal of Coastal Research. 2021. 37(6). P. 1288–1293. https://doi.org/10.2112/JCOASTRES-D-21-00026.1
18. *Hariri S., Weill S., Gustedt J., Charpentier I.* Pairing GIS and distributed hydrological models using MATLAB. 2022. https://doi.org/10.1007/978-3-030-72543-3_103
19. *Wang Y., Ai B., Qin C., Zhu A.* A load-balancing strategy for data domain decomposition in parallel programming libraries of raster-based geocomputation. International Journal of Geographical Information Science. 2022. 36(5). P. 968–991. https://doi.org/10.1080/13658816.2021.2004603.
20. *Волошинов Д.В., Соломонов К.Н.* Программно-аппаратная реализация конструктивных геометрических моделей // Труды Международной конференции по компьютерной графики и зрению "Графикон". 2020. № 30. С. 83–98. https://doi.org/10.51130/graphicon-2020-1-83-98
21. *Балюба И.Г.* Точечное исчисление: Учебно-методическое пособие / И.Г. Балюба, Е.В. Конопацкий, А.И. Бумага. – Макеевка: Донбасская национальная академия строительства и архитектуры, 2020. 244 с.
22. *Конопацкий Е.В.* Моделирование дуги обвода на основе конфигурации Дезарга / Е.В. Конопацкий, И.Г. Балюба // Омский научный вестник. 2022. № 3(183). С. 5–9. https://doi.org/10.25206/1813-8225-2022-183-5-9
23. *Глаголев Н.А.* Проективная геометрия / Н.А. Глаголев // М.: Высшая школа, 1963. 244 с.
24. *Konopatskiy E.V.* Geometric modeling of multifactor processes and phenomena by the multidimensional parabolic interpolation method / E.V. Konopatskiy, A.A. Bezditnyi // Journal of Physics: Conference Series: XIII International Scientific and Technical Conference "Applied Mechanics and Systems Dynamics", Omsk, 05–07 ноября 2019 года. V. 1441. Omsk: Institute of Physics Publishing, 2020. P. 012063. https://doi.org/10.1088/1742–6596/1441/1/012063.
25. *Конопацкий Е.В.* Геометрическое моделирование многофакторных процессов на основе точечного исчисления: специальность 05.01.00 "Инженерная геометрия и компьютерная графика»: диссертация на соискание ученой степени доктора технических наук / Конопацкий Евгений Викторович. – Нижний Новгород, 2020. 307 с.
26. *Конопацкий Е.В.* Геометрические основы параллельных вычислений в системах компьютерного моделирования и автоматизированного проектирования / Е.В. Конопацкий // Труды Международной конференции по компьютерной графике и зрению "Графикон". 2022. № 32. С. 816–825. https://doi.org/10.20948/graphicon-2022-816-825

THEORETICAL BASES OF MATHEMATICAL APPARATUS OF PARALLEL COMPUTING REALIZATION IN COMPUTER-AIDED DESIGN SYSTEMS

© 2024 E. Konopatskiy[a]

[a] *Nizhny Novgorod State University of Architecture and Civil Engineering*
65 Iljinskaya st., Nizhny Novgorod, 603950, Russia

The purpose of the work is the development of mathematical apparatus and computational algorithms for the implementation of parallel computing in geometric modeling and computer-aided design systems. The analysis of existing approaches to the implementation of parallel computing in CAD systems has been carried out. As a result, it was found that for most information modeling and computer-aided design systems, there is no support for parallel computing at the level of the geometric kernel. A concept for the development of a CAD geometric kernel based on the invariants of parallel projection of geometric objects onto the axes of the global coordinate system is proposed, which combines the potential of constructive methods of geometric modeling capable of providing parallelization of geometric constructions by tasks (message passing) and the mathematical apparatus of "Point Calculus", capable of implementing parallelization by data through coordinate-by-coordinate calculation (data parallel). The use of coordinate-by-coordinate calculation of point equations not only allows you to parallelize calculations along coordinate axes, but also ensures the consistency of computational operations along streams, which significantly reduces the idle time of calculations and optimizes the processor's work to achieve the maximum effect from the use of parallel computing.

Keywords: CAD, mathematical apparatus, parallel computing, point calculus, coordinate-by-coordinate calculation, coordinate vector, invariants of parallel projection, hidden parallelism.

REFERENCES

1. *Bakhtin, V.A., Zakharov, D.A., Kozlov, A.N., and Konovalov, V.S.* Development of parallel program code for calculating radiation magnetic gas dynamics and studying plasma dynamics in the QSPA channel, *Nauchn. Servis Seti Internet,* 2019. No. 21, P. 105–118. https://doi.org/10.20948/abrau-2019-80
2. *Pekunov, V.V.* Predicting channels in parallel programming: Possible applications in mathematical modeling of processes in continuous media, *Program. Sist. Vychisl. Metody,* 2019. No. 3, P. 37–48. https://doi.org/10.7256/2454-0714.2019.3.30393
3. *Vorob'ev, V.E., Murynin, A.B., and Khachatryan, K.S.* High-performance recording of spatial spectra of sea waves during operational space monitoring of vast water areas, *Issled. Zemli Kosmosa,* 2020. No. 2, P. 56–68. https://doi.org/10.31857/S0205961420020062
4. *Goncharsky, A.V., Romanov, S.Y., and Seryozhnikov, S.Y.* Implementation and performance of wave tomography algorithms on SIMD CPU and GPU computing platforms, *Numer. Methods Program.,* 2021. V. 22. No. 4, P. 322–332. https://doi.org/10.26089/NumMet.v22r421
5. *Shmakov, I.A., Iordan, V.I., and Sokolova, I.E.* Computer simulation of SH-synthesis of nickel aluminide by the molecular dynamics method in the LAMMPS package using parallel computing, *Vysokoproizvod. Vychisl. Sist. Tekhnol.,* 2018. V. 2. No. 1, P. 48–54.
6. *Fedotov, V.L.* Using a parallel computing architecture in the approach to constructing airborne complexes of control systems, *Navig. Upr. Letatel'nymi Appar.,* 2019. V. 24. No. 1, P. 12–20.
7. *Pekunov, V.V.* Improved balancing of CPU workload when numerically solving continuum mechanics problems complicated by chemical kinetics, *Kibern. Program.,* 2021. No. 1, P. 13–19. https://doi.org/10.25136/2644-5522.2021.1.35101
8. *Ol'khovskaya, O.G., Gasilov, V.A., Kotel'nikov, A.M., and Yakobovskii, M.V.* Parallel ray tracing algorithm for radiation field analysis and construction of obscurograms of radiative gas, *Preprint of Inst. Prikl. Mat. Keldysha,* Moscow, 2018. No. 143, P. 1–16. https://doi.org/10.20948/prepr-2018-143
9. *Chetverushkin, B.N., Chechina, A.A., Churbanova, N.G., and Trapeznikova, M.A.* Development of parallel algorithms for intelligent transportation systems, *Mathematics,* 2022. V. 10. No. 4. https://doi.org/10.3390/math10040643
10. *Kucherov, D.P., Morgun, K.O., and Anikeenko, L.S.* Parallel computing control in computer graphics problems, *Naukoємni Tekhnol.,* 2018. V. 38. No. 2, P. 178–186. https://doi.org/10.18372/2310-5461.38.12833
11. *Nizovskikh, A.S., Koporushkin, P.A., and Tarasenko, R.R.* Problems of parametric approach in some modern CAD, *Sovrem. Probl. Teor. Mash.,* 2016. No. 4–1, P. 83–85.
12. *Abramov, O.V.* Computing environment for solving CAD problems on multiprocessor systems, Mat. Metody Tekhn. Tekhnol., 2018. V. 5, P. 28–30.
13. *Zhao, Z., et al.* A large-scale parallel hybrid grid generation technique for realistic complex geometry, Int. J. Numer. Methods Fluids, 2020. V. 92. No. 10, P. 1235–1255. https://doi.org/10.1002/fld.4825
14. *Kukreja, A., Dhanda, M., and Pande, S.S.* Voxelbased adaptive toolpath planning using graphics processing

unit for freeform surface machining, J. Manuf. Sci. Eng. Trans. ASME, 2022. V. 144. No. 1. https://doi.org/10.1115/1.4051535

15. *de Matos Menezes, M., Viana Gomes de Magalhães, S., Aguilar de Oliveira, M., Randolph Franklin, W., and de Oliveira Bauer Chichorro, R.E.* Fast parallel evaluation of exact geometric predicates on GPUs, Comput. Aided Des., 2022. https://doi.org/10.1016/j.cad.2022.103285
16. *Oshchepkov, A. Yu. and Popov, S.E.* Development of an information and computing system based on Apache Hadoop for processing hyper- and multispectral Earth remote sensing data, Vestn. Voronezh. Gos. Univ. Ser.: Sist. Anal. Inf. Tekhnol., 2016. No. 3, P. 95–105.
17. *Zieg, J. and Zawada, D.G.* Improving ESRI ArcGIS performance of coastal and seafloor analyses with the Python multiprocessing module, J. Coastal Res., 2021. V. 37. No. 6, P. 1288–1293. https://doi.org/10.2112/JCOASTRESD-21-00026.1
18. *Hariri, S., Weill, S., Gustedt, J., and Charpentier, I.* Pairing GIS and distributed hydrological models using MATLAB, 2022. https://doi.org/10.1007/978-3-030-72543-3_103
19. *Wang, Y., Ai, B., Qin, C., and Zhu, A.* A load-balancing strategy for data domain decomposition in parallel programming libraries of raster-based geocomputation, Int. J. Geogr. Inf. Sci., 2022. V. 36. No. 5, P. 968–991. https://doi.org/10.1080/13658816.2021.2004603
20. *Voloshinov, D.V. and Solomonov, K.N.* Software and hardware implementation of constructive geometric models, Trudy Mezhdunarodnoi konferentsii po komp'yuternoi grafiki i zreniyu "Grafikon" (Proc. Int. Conf. Computer Graphics and Vision "Graphicon"), 2020. No. 30, P. 83–98. https://doi.org/10.51130/graphicon-2020-1-83-98
21. *Balyuba, I.G., Konopatskiy, E.V., and Bumaga, A.I.* Tochechnoe ischislenie: Uchebno-metodicheskoe posobie (Point Calculus: Study Guide), Makeevka: Donbasskaya Nats. Akad. Stroit. Arkhit., 2020.
22. *Konopatskiy, E.V. and Balyuba, I.G.* Contour arc modeling based on the Desargues configuration, Omsk. Nauchn. Vestn., 2022. V. 183. No. 3, P. 5–9. https://doi.org/10.25206/1813-8225-2022-183-5-9
23. *Glagolev, N.A.* Proektivnaya geometriya (Projective Geometry), Moscow: Vysshaya shkola, 1963.
24. *Konopatskiy, E.V. and Bezditnyi, A.A.* Geometric modeling of multifactor processes and phenomena by the multidimensional parabolic interpolation method, Proc. XIII Int. Sci. Techn. Conf. Applied Mechanics and Systems Dynamics, Omsk, 2019. https://doi.org/10.1088/1742-6596/1441/1/012063
25. *Konopatskiy, E.V.* Geometric modeling of multifactor processes based on point calculus, Doctoral Dissertation, Nizhny Novgorod, 2020.
26. *Konopatskiy, E.V.* Geometric foundations of parallel computing in computer-aided modeling and design systems, Trudy Mezhdunarodnoi konferentsii po komp'yuternoi grafike i zreniyu "Grafikon" (Proc. Int. Conf. Computer Graphics and Vision "Graphicon"), 2022. No. 32, P. 816–825. https://doi.org/10.20948/graphicon-2022-816-825

ПРОГРАММИРОВАНИЕ, 2024, № 5, с. 14–30

ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ

УДК 004.031.6

СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ СОПРОГРАММ И СОБЫТИЙ ДЛЯ ПРОГРАММИРОВАНИЯ ВСТРОЕННЫХ СИСТЕМ

© 2024 г. П. В. Минин[a],*

[a]ООО “КБ “ДОРС”

111399 Москва, Федеративный пр., д. 5, к. 2, Россия,

**E-mail: p.minin@dors.ru*

Поступила в редакцию: 24.03.2023 г.
После доработки: 04.11.2023 г.
Принята к публикации: 12.05.2024 г.

Предложена новая методология построения программ для встроенных систем реального времени. Программа, написанная на языке C/C++, исполняется без помощи операционной системы и позволяет реализовать как событийно-ориентированный подход, так и параллельное выполнение с помощью сопрограмм. Обработка событий и выполнение сопрограмм производится в ядре мягкого реального времени на уровне приоритета программных прерываний. Сопрограмма рассматривается как частный случай события, где возобновляемая функция сопрограммы выполняется в качестве обработчика события. Для возобновления сопрограммы после приостановки ее событие повторно отправляется на обработку, до тех пор пока возобновляемая функция не будет полностью выполнена. Таким образом, в ядре происходят как обработка простых событий, так и планирование выполнения сопрограмм. Событиям могут назначаться различные приоритеты обработки. Ядро мягкого реального времени может быть расширено для работы в симметричной многопроцессорной системе. Сочетание сопрограмм и простых событий позволяет реализовать технологию fork/join, а также средства параллельного программирования, сходные с инструментами языков Go и occam. Новая методология была воплощена на языке C++ в виде библиотеки DORSECC, адаптированной для различных процессоров ARM и Blackfin. С ее помощью были созданы встроенные вычислительные системы реального времени, применяемые в серийно выпускаемых машинах для сортировки банкнот. Эти системы используются как для распознавания образов методом каскада классификаторов, так и для управления датчиками и приводами механизма с применением автоматного программирования. Их общее количество в эксплуатации превысило 20000. Ядро мягкого реального времени в указанных системах обеспечивает среднюю длительность обработки события на уровне десятков микросекунд при субмикросекундных накладных затратах.

Ключевые слова: режим реального времени, встроенный, “голое железо”, параллелизм, управляемый событиями, сопрограмма, C++, программное прерывание, синхронизация, многопоточный код

DOI: 10.31857/S0132347424050027, **EDN:** OMADOW

1. ВВЕДЕНИЕ

Чаще всего, при создании встроенных систем используется самодостаточная (bare metal) программа, написанная с использованием языков C, C++ или их комбинации. Самодостаточная программа содержит в себе все необходимые компоненты и не требует операционной системы (ОС). Значительно реже используются программы, работающие под управлением ОС реального времени (FreeRTOS, ThreadX, WxWorks, LynxOS, PikeOS и другие) либо даже ОС общего назначения Linux. Отказ от применения ОС в пользу самодостаточной программы обычно объясняется ресурсными ограничениями аппаратной платформы или же затруднениями при управлении сложной аппаратурой в реальном времени. Самодостаточная программа позволяет избежать накладных затрат, связанных с операционной системой. В конечном счете, она обеспечивает решение, наиболее оптимальное с точки зрения быстродействия и требуемых аппаратных ресурсов. Однако, платой за эту оптимизацию оказывается сложность программирования. Наибольшие трудности, как показывает опыт, связаны с реализацией параллельного выполнения кода, а также асинхронной обработки.

ОС реального времени предоставляют возможность использовать концепцию потоков в качестве основного механизма параллельной обработки. Для самодостаточной программы с использованием C/ C++, до последнего

времени, самим языком программирования не предусматривались средства параллельного выполнения. Только недавно в C++ был реализован механизм сопрограмм (coroutine, начиная со спецификации C++ 20). Однако, сопрограммы C++ во встроенных системах пока еще не нашли широкого применения, отчасти из-за сложности самой концепции и ее непривычной терминологии.

1.1 Сопрограммы

Сопрограммы [1] не являются потоками в современном понимании, поскольку реализуют параллельное исполнение нескольких последовательностей операторов с явной передачей управления между ними, без возможности вытеснения. Достаточно давно известны библиотечные реализации сопрограмм на языках C [2, 3] и C++ [4, 5, 6], которые делятся на стековые и бесстековые. Концепция стековой сопрограммы, по своим выразительным возможностям, максимально приближена к традиционному потоку. Обратной стороной высоких выразительных возможностей оказываются большие накладные расходы на управление сопрограммой. Для бесстековых сопрограмм не обеспечивается сохранение стековых переменных одной сопрограммы на период передачи управления от нее другой сопрограмме. Это накладывает существенные ограничения на стиль программирования. С другой стороны, бесстековые сопрограммы обеспечивают минимальные требования по памяти и затраты времени, поскольку при переключении от одной сопрограммы к другой не требуется сохранения и восстановления стека и связанных с ним элементов контекста. Сравнение стековых и бесстековых сопрограмм вызвало интенсивное обсуждение, которое частично отражено, например, в [7]. Обзор по применимости сопрограмм во встроенных системах приведен в [8].

В некоторых реализациях сопрограммам назначается величина приоритета. Приоритетный запуск сопрограммы на исполнение достигается двумя независимыми способами: с помощью диспетчера сопрограмм и с использованием приоритета потока исполнения. После того, как очередная сопрограмма прерывает свое исполнение, диспетчер передает управление наиболее приоритетной сопрограмме, готовой к выполнению. Если же ОС представляет программе потоки исполнения с разными приоритетами, то для выполнения более приоритетных сопрограмм может выделяться более приоритетный поток. Примерами можно назвать язык Kotlin [9] и ОС реального времени FreeRTOS [10]. Отметим, что в самодостаточной программе имеется единственный поток исполнения, поэтому для обеспечения приоритетов сопрограмм невозможно использовать потоки с разными приоритетами.

На сегодняшний день, существуют десятки реализаций сопрограмм для языков C/C++, ни одна из которых не смогла пока стать стандартом де-факто. Обращает на себя внимание тот факт, что в подавляющем большинстве случаев в них не задействован такой мощный асинхронный инструмент реального времени, как система прерываний процессора. Более того, обслуживание прерываний обычно полностью изолируется от выполнения детерминированной последовательности процессорных команд, создаваемой компилятором для кода сопрограмм.

1.2 Обработка событий

Значительная часть задач управления физическим устройством сводится к описанию в виде конечного автомата. Для конечного автомата естественным представлением является система, управляемая событиями [11]. Использование потоков (threads) для решения многих задач приводит к утяжеленным, неоптимальным решениям, в то время как эти же задачи наиболее просто решаются в концепции управления событиями [12, 13].

В простых встроенных системах реального времени обработка событий, возникающих в аппаратуре, традиционно реализуется с помощью взаимосвязанных обработчиков прерываний. Подобный подход позволяет получить максимальное быстродействие, но основан на приемах программирования, часто напоминающих трюки, порождающих скрытые ошибки и сложных для верификации. Для функционально сложного управления встроенной системой по событиям используются специализированные библиотеки программных инструментов (frameworks), например, Quantum Leaps [14, 15]. Кроме того, в практическом программировании часто применяется шаблон проектирования, известный как “Наблюдатель” (Observer) [16] и используемый в качестве основы менеджера событий. Для каждого получаемого события такой менеджер обеспечивает вызов соответствующего ему обработчика.

В операционных системах общего назначения используется механизм отложенной обработки данных, формируемых в аппаратных прерываниях, на менее приоритетном уровне. По своей сути, он также относится к обработке событий. В качестве примеров можно привести отложенный

вызов процедуры (DPC) в MS Windows [17], или отложенные задачи в Linux [18], которые выполняются в потоке ядра (Workqueue) или в контексте программного прерывания (Tasklet).

Для организации последовательной обработки асинхронно возникающих событий обычно применяется входная очередь системы обработки событий.

2. МОТИВАЦИЯ К РАЗРАБОТКЕ

Бесстековые сопрограммы представляют существенный интерес как средство обеспечения многозадачности во встроенных системах, где не используется ОС. Главное преимущество бесстековой сопрограммы перед потоком ОС состоит в значительно меньших накладных затратах на переключение выполнения задач. Оно связано с размером переключаемого контекста, который в случае бесстековой сопрограммы может быть уменьшен до одного-двух слов процессора.

Однако, классическая сопрограмма плохо подходит для работы в условиях реального времени, поскольку не может обеспечить гарантированное время реакции (латентность, latency) на асинхронный стимул, такой, как аппаратное прерывание. Это можно увидеть в сравнении с приоритетными вытесняющими потоками (задачами, процессами), применяемыми в ОС реального времени. Латентность запуска потока, ранее остановленного в ожидании прерывания, при гарантии вытеснения составляет

$$T_{TL} = T_{IL} + T_{TSW} + T_{OVH} \quad (1).$$

Здесь T_{IL} обозначает латентность обработки прерывания, а T_{TSW} – время переключения контекста с одного потока на другой. При помощи T_{OVH} обозначена общая продолжительность вспомогательных операций, включающих в себя выполнение драйвера прерывания до момента отправки сигнала потоку, а также накладные расходы на синхронизацию и диспетчеризацию. Видно, что T_{TL} складывается как сумма продолжительностей системных операций ОС, которые невелики и хорошо детерминированы.

Классические сопрограммы в самодостаточной программе выполняются в единственном имеющемся потоке исполнения и обеспечивают кооперативный тип многозадачности. Чтобы отреагировать на стимул запуском соответствующей сопрограммы, в потоке исполнения должен работать диспетчер, управляющий выполнением набора сопрограмм. Диспетчер может отправить сопрограмму на исполнение, только дождавшись, чтобы предшествующая сопрограмма самостоятельно уступила процессор. Латентность запуска бесстековой сопрограммы в ответ на стимул составляет

$$T_{CL} = T_{IL} + T_{CY} + T_{OVH} \quad (2),$$

где T_{CY} есть время выполнения текущей сопрограммы от момента обработки стимула в прерывании до момента освобождения процессора. Текущая сопрограмма уступает процессору, когда ее последовательность исполнения доходит до специальной операции YIELD или TRANSFER, без какой-либо синхронизации с появлением стимула. Поэтому $max(T_{CY}) = max(T_{CQ})$, где T_{CQ} есть квант времени, в течение которого сопрограмма владеет процессором. Этот квант определяется только расстановкой YIELD или TRANSFER в коде исполняемых сопрограмм. Иначе говоря, T_{CY} ограничивается сверху наибольшим значением кванта T_{CQ} для всех выполняемых сопрограмм, но, за счет асинхронного появления стимула, может случайным образом принимать любые меньшие значения. Заметим, что квант T_{CQ} должен существенно превышать время переключения сопрограмм, поскольку при слишком частом переключении недопустимо растет доля накладных расходов процессорного времени. За счет всех указанных факторов, латентность T_{CL} оказывается плохо предсказуемой и достаточно большой.

Латентность реакции на стимул считается основным параметром, по которому можно судить о возможности работы программы в реальном времени. По этой характеристике классические сопрограммы значительно уступают вытесняющим потокам. Достижение стабильно малой латентности запуска сопрограммы представляет собой важную проблему, пока еще не нашедшую адекватного разрешения. Вытесняющие сопрограммы ввода-вывода в языке Modula-2 [19] могли бы показаться подходящим решением. Однако, при внимательном рассмотрении видно, что они обладают всеми свойствами потока в современном понимании, и потому их неправомерно называть сопрограммами.

С другой стороны, обработка событий позволяет достичь малой и предсказуемой латентности реакции на внешний стимул. Для наиболее быстрого запуска и исполнения функции-обработчика может использоваться либо программное прерывание, либо вытесняющий поток реального времени. В качестве примеров можно привести упомянутые ранее механизмы Tasklet и Workqueue [18].

В отсутствии других событий в очереди, при использовании программного прерывания, латентность можно приблизительно представить в виде

$$T_{EL} = 2T_{IL} + T_{OVH} \quad (3),$$

где $2T_{IL}$ соответствует сумме латентностей аппаратного и программного прерывания. Когда используется обработка в вытесняющем потоке, то для оценки латентности может использоваться формула (1). В обоих случаях, затраты времени на диспетчеризацию с использованием очереди событий включены в T_{OVH}. Вход в прерывание выполняется быстрее переключения контекста ($T_{IL} \ll T_{TSW}$), благодаря чему программное прерывание обеспечивает существенно более низкую латентность, чем вытесняющий поток. Для самодостаточной программы, работающей без ОС и потоков, обработка в программном прерывании остается наилучшим возможным выбором.

В то же время, использование программного прерывания ограничивает круг задач, которые могут быть решены с помощью обработки событий. Как и для любого обработчика прерывания, в функции обработки события не допускается ожидание заданного условия или готовности примитива синхронизации. Кроме того, обработка должна быть ограничена по длительности, чтобы не нарушить своевременную обработку других событий.

Естественным решением перечисленных проблем было бы объединение высокой скорости реакции на асинхронный стимул, характерной для обработки событий, с возможностями сопрограмм для параллельной обработки и выполнения синхронизации. Насколько нам известно, подобный подход ранее не рассматривался. При его реализации важно сохранить такое фундаментальное преимущество бесстековых сопрограмм, как очень малые накладные затраты времени на переключение.

Мы постарались создать простую методологию для совместного применения сопрограмм и управления событиями в рамках самодостаточной встроенной программы, целиком написанной на C/C++. На ее основе нами была создано переносимое ядро обработки событий и сопрограмм в виде библиотеки DORSECC (DORS Event and Coroutine Core).

Важными требованиями к методологии были ее независимость от архитектуры процессора и отказ от использования ассемблерных вставок, за исключением стандартного кода для вызова функции обработчика прерывания, а также для возбуждения программного прерывания. Возможность дальнейшего масштабирования для использования в симметричных мультипроцессорных системах рассматривалась как обязательное требование.

3. ПРЕДЛАГАЕМОЕ РЕШЕНИЕ

Предполагается, что самодостаточная программа имеет три основных уровня исполнения кода. На наименее приоритетном уровне исполняется код главной функции *main*(). Во встроенных системах эту функцию часто называют суперциклом. Она обеспечивает первоначальную инициализацию и последующее циклическое выполнение действий, не критичных по времени, например таких, которые обеспечивают работу пользовательского интерфейса или встроенного веб-сервера. В случае мультипроцессорной системы, на этом уровне для каждого из ядер исполняется код главной функции, назначенной заданному ядру.

На наиболее приоритетном уровне выполняются обработчики прерываний от внешних устройств системы. Этот уровень распадается на несколько отдельных подуровней, каждому из которых соответствуют аппаратные прерывания, инициируемые определенными внешними устройствами. Организация обработчиков прерываний, которые выполняют действия, наиболее критичные по времени, здесь не отличается от общепринятой. Время нахождения в обработчике прерывания ограничено, и в нем допустимо выполнять только те действия, которые следует выполнить немедленно.

Отличия от общеизвестных встроенных систем возникают на промежуточном уровне приоритета, лежащем между уровнем аппаратных прерываний и уровнем *main*(). Промежуточный уровень соответствует программным прерываниям и предназначен для работы системы событий. Для тех архитектур, где отдельных программных прерываний не существует, может применяться низкоприоритетное аппаратное прерывание, выделенное для отсутствующего устройства и возбуждаемое через установку бита запроса в контроллере прерываний.

Во встроенных системах нет возможности задействовать все выразительные средства языка C++ из-за ограничений по использованию динамического выделения памяти, а также по скорости выполнения. При разработке DORSECC мы ограничились использованием механизма классов с небольшими дополнениями.

3.1. Система событий

Система событий реализована внутри базового класса события. Для каждого вида событий создается отдельный класс, который наследует базовому классу. Экземпляр отдельного класса события соответствует событию этого вида, произошедшему в конкретный момент и с конкретными параметрами. Например, события периодического таймера реализуются отдельным классом, причем каждый экземпляр этого класса соответствует определенному импульсу таймера и содержит номер импульса в качестве параметра.

Обращения к системе событий выполнены как методы экземпляра класса события. Источник события может находиться на любом уровне приоритета. Он создает экземпляр события *Event* с помощью *new*, при необходимости наполняет его параметрами, и передает на обработку вызовом функции *pEvent->Post*(). Эта функция помещает событие *Event* в очередь системы событий (см. рис. 1).

В простейшем случае очередь представляет собой обычную структуру FIFO. Вызов *Post*() может происходить из кода, исполняемого на любом уровне приоритета. Обработка события *Event* состоит в извлечении его из очереди и вызове обработчика *pEvent->Exec*(), индивидуально определенного для каждого отдельного класса события. Когда очередь событий не пуста, и никакое другое событие не обрабатывается, то диспетчер событий запускает на обработку первое по порядку событие в очереди. Весь код обработки события выполняется на промежуточном уровне.

Более конкретно, событие помещается в очередь с помощью *pEvent->Post*(). Если очередь до этого момента была пустой, то также возбуждается программное прерывание. В этом программном прерывании запускается диспетчер событий, который извлекает событие из очереди и вызывает его обработчик *pEvent->Exec*(). Если, после завершения обработчика, в очереди еще остаются другие события, то они обрабатываются одно за другим. Как только очередь оказывается пустой, диспетчер событий прекращает работу и программное прерывание завершается.

Функция обработчика возвращает статус события, который определяет его дальнейшую судьбу. Обычно, обработанное событие уничтожается (статус *done* на рис. 1). Однако, для специальных целей, как будет описано далее, событие может быть сохранено и отправлено на обработку повторно.

Для обеспечения требований реального времени, при создании и уничтожении события не

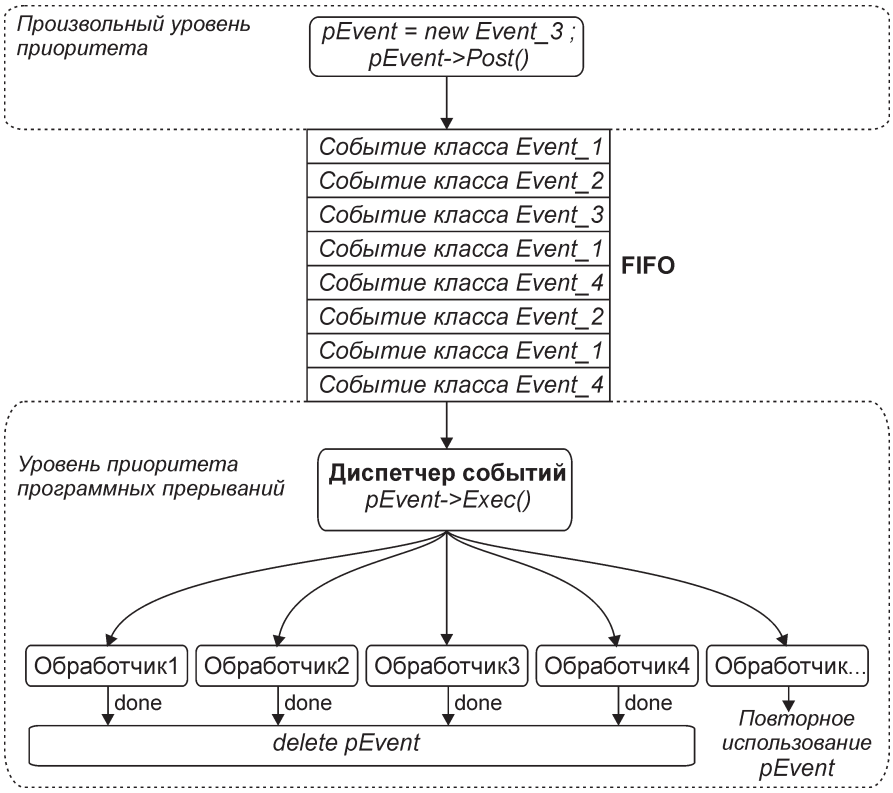


Рис. 1. Система событий. Показана отправка события класса *Event_3* в очередь FIFO (без заполнения события параметрами).

происходит обращения к менеджеру памяти. Вместо этого, заранее создается пул событий, которые не уничтожаются, а только помечаются как неиспользуемые. Затем, они повторно используются при создании события. Для такого решения в C++ имеются средства в виде контейнеров, а также возможности переопределения операторов *new* и *delete*. Если требования реального времени не очень жесткие и менеджер памяти безопасно работает в прерываниях, то можно обойтись без заранее созданного пула. Событие может либо выделяться из числа ранее удаленных, либо создаваться заново в памяти, когда ранее удаленное событие не доступно. Такой вариант обеспечивает автоматическую подстройку использования памяти к реальным потребностям.

Обработка событий диспетчером прерывает выполнение кода уровня *main*(), поскольку приоритет программного прерывания более высокий. Обработка события должна быть ограниченной во времени, чтобы не блокировать обработку следующих событий в очереди. В то же время, она не может заблокировать реакцию на аппаратные прерывания, так как имеет более низкий приоритет. Таким образом, в отличие от аппаратных прерываний, обеспечивающих режим жесткого реального времени, система событий работает в мягком режиме реального времени.

Одна из основных функций системы событий относится к вычислительно сложной обработке аппаратного события. Обработчик аппаратного прерывания быстро реагирует на запрос от периферийного устройства и производит немедленную обработку, которую нельзя отложить. Однако, если требуются дополнительные сложные действия, которые недопустимо выполнять в обработчике аппаратного прерывания по соображениям большой продолжительности, то такие действия выполняет система событий. Для этого, обработчик аппаратного прерывания создает событие по *new* и помещает его в очередь с помощью *pEvent->Post*(). Диспетчер событий далее запускает обработку события *pEvent->Exec*() на промежуточном уровне приоритета, не блокируя аппаратные прерывания.

Система событий позволяет реализовать переходы конечного автомата по событиям. При обработке события выполняется перевод автомата в следующее состояние, соответствующее графу переходов. Источником такого события может быть как низкоприоритетный код уровня *main*() (например, в результате действий в интерфейсе пользователя), так и высокоприоритетное аппаратное прерывание. Кроме того, событие может быть отправлено с промежуточного уровня, то есть, из самой системы событий.

3.2. Сопрограммы

Для реализации бесстековых сопрограмм нами была выбрана концепция протопотоков (protothreads) Дункельса [2], основанная на методе Даффа (Duff's device) и работе Тэтема [20]. Она относится к наиболее часто применяемому способу реализации сопрограмм, в котором используются так называемые возобновляемые функции (resumable functions, подробную дискуссию о них в контексте C++ см. в [7]). Сопрограмма представляет собой последовательность операторов, показанную на рис. 2. С формальной точки зрения, она есть тело функции, в которой необходимые действия сопрограммы реализованы последовательностью операторов Op.1 – Op.12. Служебные участки кода BEGIN, YIELD и END реализованы с помощью макроопределений. Вся сопрограмма целиком выполняется за несколько последовательных вызовов этой функции. Благодаря такому поведению функция получила название возобновляемой. При каждом вызове выполняется только часть операторов, после чего происходит выход из возобновляемой функции.

Начальный участок кода BEGIN представляет собой локальный диспетчер, который направляет выполнение кода на продолжение сопрограммы с того места, где оно завершилось при предшествующем вызове возобновляемой функции. Служебный участок YIELD приостанавливает

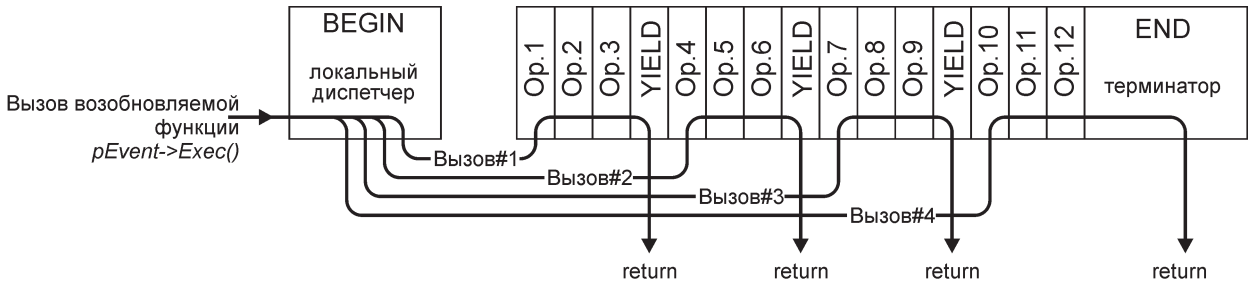


Рис. 2. Последовательность операторов возобновляемой функции сопрограммы.

выполнение сопрограммы, завершая текущий вызов возобновляемой функции. При первом вызове выполняются операторы Op.1 – Op.3, при следующем Op.4 – Op.6, и так далее. Выполнение сопрограммы оканчивается терминатором END, в котором состояние сопрограммы помечается как завершенное. При необходимости повторного исполнения в терминаторе может быть реализован возврат к началу сопрограммы.

Служебные участки кода используют скрытые переменные состояния, которые нужны для сохранения точки возврата в сопрограмму при следующем вызове возобновляемой функции, а также текущего статуса выполнения сопрограммы. Эти участки построены на основе оператора *switch*, где все точки возобновления выполнения сопрограммы отмечены с помощью *case*. Однако конструкция *switch – case* скрыта внутри макроопределений и не видна программисту.

Макрос YIELD, как это изначально было определено Дункельсом, дает возможность выполниться другим сопрограммам. Помимо YIELD, существуют дополнительные макроопределения служебных участков кода, которые могут быть использованы для проверки определенного условия, и остановки выполнения сопрограммы в случае, когда условие не выполнено. Например, макрос WAIT_UNTIL(*condition*) проверяет условие *condition* и в случае его невыполнения приостанавливает выполнение сопрограммы. При этом точка возобновления устанавливается на начало макроса. После выполнения условия *condition* при очередном вызове возобновляемой функции, выполнение сопрограммы продвигается дальше по коду.

Из сопрограммы при помощи макроса SPAWN может быть запущена дочерняя сопрограмма. Такой запуск является синхронным и останавливает выполнение основной сопрограммы до завершения дочерней.

На протяжении всего выполнения сопрограммы, только статически определенные переменные и объекты, выделенные в куче (heap), могут сохранить свои значения. Стековые переменные допустимо использовать исключительно между двумя служебными участками кода, поскольку их значение будет утеряно при следующем вызове функции сопрограммы. Результат выполнения сопрограммы должен, в том или ином виде, сохраняться в куче. Благодаря невозможности полноценно применять стековые переменные, возобновляемая функция не реентерабельна. Эти ограничения можно считать естественным для всех видов бесстековых сопрограмм.

Подобная реализация сопрограмм привносит очень малые накладные расходы процессорного времени и памяти. Это было убедительно показано при разработке ОС реального времени Contiki [21] и сетевого стека uIP [22, 23], применяемых во встроенных системах с весьма ограниченными ресурсами. В частности, высокое быстродействие локального диспетчера достигается за счет табличной реализацией переходов, в которую в ходе компиляции переводится конструкция *switch – case*.

3.3. Сопрограммы как события

Исходная реализация Дункельса [2] имеет ряд существенных недостатков, которые касаются способа вызова функции сопрограммы. В этой реализации, вызов возобновляемых функций всех объявленных сопрограмм последовательно делался из карусели (round robin) суперцикла, работающего на низкоприоритетном уровне *main*(). Суперцикл реальных встроенных систем обычно включает в себя множество действий, выполняемых без привлечения сопрограмм, не относящихся к категории реального времени и снижающих общую частоту повторения. Из-за этого, для сопрограмм было достаточно сложно обеспечить высокую скорость реакции. Кроме того, ожидание условия по типу WAIT_UNTIL(*condition*), хотя и не блокирует работу других сопрограмм, но является активным и потому непродуктивно расходует время процессора на постоянные проверки условия *condition*. Благодаря этому, синхронизации сопрограмм сопряжена со значительными потерями времени процессора.

Нами был предложен иной подход к вызову возобновляемой функции сопрограммы, который полностью свободен от указанных недостатков. Он основан на нескольких принципах, перечисленных ниже.

Класс сопрограммы наследует от класса события, так что каждая сопрограмма представляет собой частный случай события. Возобновляемая функция сопрограммы оформляется как обработчик события *pEvent->Exec*(). Поскольку возобновляемая функция сопрограммы не реентерабельна и не имеет смысла вне соответствующего ей события, то предпочтительно создавать ее, вместе с экземпляром события, как анонимную функцию (лямбда-выражение в C++ 11).

Запуск сопрограммы производится ее отправкой в очередь системы событий с помощью *pEvent->Post*(). После извлечения сопрограммы из очереди, ее возобновляемая функция *pEvent->Exec*() вызывается диспетчером системы событий.

После выхода из *pEvent->Exec*() по выполнению макроса YIELD, событие сопрограммы не уничтожается, но заново отправляется в очередь системы событий. Таким образом, после каждого YIELD возобновляемая функция повторно вызывается системой событий, пока исполнение не дойдет до конца сопрограммы, отмеченного END.

При выполнении YIELD проверяется состояние очереди системы событий. Если она оказывается пустой, то выход из возобновляемой функции не выполняется и исполнение *pEvent->Exec*() продолжается до следующего служебного макроса. Благодаря такой проверке, приостановка сопрограммы производится только тогда, когда это реально требуется, а накладные затраты системы событий снижаются.

Когда сопрограмму необходимо остановить в ожидании какого-либо условия, а затем продолжить ее выполнение по наступлению этого условия, то используется механизм возобновления. Как только внешний код обнаруживает наступление ожидаемого условия, он возобновляет работу сопрограммы, инициируя отправку ее события в очередь.

При выполнении этих принципов, система событий становится одновременно планировщиком и диспетчером сопрограмм. В то же время, полностью сохраняется возможность обработки простых событий, обработчик которых является обычной непрерывной функцией. Простые события и события сопрограммы стоят в общей очереди и единообразно обрабатываются диспетчером событий. Когда работают несколько сопрограмм, то в очереди присутствует смесь событий этих сопрограмм и простых событий, а их последовательная обработка обеспечивает параллельное выполнение сопрограмм.

Грань между простым событием и событием сопрограммы оказывается достаточно размытой. Давно отмечено [24], что подпрограмма представляет собой частный случай сопрограммы. А именно, подпрограмма – это сопрограмма, которая не приостанавливается в ходе выполнения. Аналогичным образом, обработчик простого события можно рассматривать как частный случай сопрограммы, которая не выполняет YIELD.

Если определенное простое событие возникает достаточно часто, то в очереди системы событий одновременно может находиться несколько экземпляров простых событий одного класса, возможно, отличающихся параметрами. Вызов возобновляемой функции сопрограммы также является повторяющимся, но в очереди системы событий в каждый момент времени может быть не более одного события, соответствующего данной сопрограмме. Разница тут в том, что появление экземпляров одного и того же простого события определяется внешними асинхронными действиями, а сопрограмма самостоятельно и синхронным образом планирует исполнение самой себя (см. рис. 3).

При развитии функциональности программного кода может выясниться, что обработка некоторого простого события стала выполняться недопустимо долго и блокирует обработку других событий на слишком большое время. Тогда, это простое событие можно превратить в сопрограмму, сделав функцию обработчика возобновляемой и вставив в ее код один или несколько YIELD. За счет такой вставки общее время обработки события несколько увеличится, но время блокировки им обработки других событий кратно уменьшится.

Для работы на самом низком уровне приоритета, была сохранена возможность вызова возобновляемых функций в суперцикле *main*(), в соответствии с первоначальным подходом Дункельса. Они не обрабатываются в системе событий и не работают в реальном времени, но позволяют структурировать параллельное выполнение медленных задач суперцикла.

3.4. Синхронизация параллельного выполнения

Представленный нами механизм остановки и возобновления сопрограммы реализует пассивное ожидание, не расходующее процессорное время. Он полностью совместим со всеми классическими способами синхронизации, включая семафоры, мониторы, каналы передачи сообщений, мьютексы, а также ожидание готовности объекта. Указатель на событие сопрограммы при остановке и возобновлении используется совершенно аналогично идентификатору (handle) процесса или потока в традиционных операционных системах.

Событие, обработка которого завершилась, считается готовым. Это относится, в том числе, и к завершению сопрограммы. Статус завершения сопрограммы можно получить опросом *pEvent->GetState*(). Такой способ наиболее пригоден для проверки состояния в суперцикле *main*(), в том числе, при активном ожидании. Когда используется управление встроенной системой с помощью конечного автомата, то естественным результатом обработки события и подтверждением ее завершения является изменение состояния автомата самим кодом обработчика.

Кроме того, предусмотрена подача объекту синхронизации сигнала о наступлении готовности события. Для этого, соответствующий объ-

Рис. 3. Обработка простых событий и сопрограмм в системе событий. *Event_XX* может быть классом как простого события, так и сопрограммы.

ект синхронизации должен быть предварительно подписан на сигнал от события *Event*, то есть указатель на него должен быть занесен в поле события *pEvent->SignalOnReady*. Использование функционального обратного вызова (callback) в запускающий код могло бы показаться более удобным способом извещения о готовности события, но оно не рекомендуется нами по соображениям безопасности. Дело в том, что распространение передачи управления при обратном вызове плохо контролируется средствами верификации кода. Обратный вызов исполняется на промежуточном уровне приоритета и может продлиться слишком долго, блокируя другие события.

В реализации Дункельса имеется только синхронный блокирующий запуск дочерней сопрограммы SPAWN, при котором возобновляемая функция родительской сопрограммы вызывает возобновляемую функцию дочерней. Предлагаемый нами подход позволяет асинхронно запускать дочернее событие при помощи макроса FORK, который просто отправляет дочернее событие в очередь на обработку, не останавливая родительской сопрограммы.

В связи с малыми затратами памяти и времени на порождение даже сравнительно большого количества событий, появляется возможность простой реализации методологии fork/join [25]. Для выполнения операции join, то есть ожидания завершения ранее запущенных событий, нами используется специальный объект синхронизации *Joint* с методом *Signal*(). В объекте хранятся две переменные, задаваемые извне: счетчик ожидаемых событий *waited* и указатель на событие продолжения *pContinuation*. Первоначально, в счетчик заносится известное количество ожидаемых событий. Объект *Joint* подписывается на сигналы от тех событий, завершения обработки которых ему необходимо дождаться. По каждому вызову *Signal*() происходит декремент *waited*. Как только *waited* достигнет нуля, что обозначает завершение обработки всех ожидаемых событий, *Joint* запускает событие продолжения по указателю *pContinuation*. В описанной схеме fork/join

простые события и сопрограммы могут сочетаться произвольным образом.

Для родительской сопрограммы возможно организовать пассивное ожидание завершения join. Она должна записать саму себя в качестве *pContinuation* и остановиться после запуска дочерних событий. Когда обработка всех запущенных событий завершится, сопрограмма будет возобновлена.

В класс *cJoint* был добавлен метод *Fork*(*pEvent*), который подписывает объект *Joint* на дочернее событие *Event,* инкрементирует счетчик *waited* и отправляет дочернее событие в очередь при помощи *pEvent->Post*(). С помощью *Fork* полностью автоматизируется запуск дочерних событий и подготавливается ожидание их завершения.

Нужно отметить, что описанные средства управления параллельным исполнением сходны с конструкциями языков Go и occam [26, 27], основанных на теории взаимодействующих последовательных процессов (CSP) Хоара [28]. Так, FORK действует аналогично оператору *go* языка Go, а базовый вариант *Joint* близок к стандартному примитиву синхронизации *sync.WaitGroup* этого языка. Использование *Fork*(*pEvent*) позволяет получить семантику, аналогичную оператору параллельного исполнения *PAR* в языке occam.

Как можно видеть, *Joint* через поле *pContinuation* связан с сопрограммой, в коде которой он используется для ожидания. Данный объект синхронизации естественно включить в объект сопрограммы, что и было выполнено в DORSECC. В результате, сама сопрограмма становится объектом синхронизации. Когда сопрограмма *Event* остановилась в ожидании сигнала и *waited*=1, то обращение *pEvent->Signal*() имеет тот же результат, что и *pEvent->Post*(). Если же сопрограмма не остановлена в ожидании, то поданный ей сигнал просто теряется. Таким образом, сопрограмма может ожидать готовности не только других событий, но также и любых объектов, которые могут отправить ей сигнал при входе в ожидаемое состояние. Подобный способ синхронизации можно считать безопасной заменой обратного вызова функции, который очень широко, но без должной дисциплины используется в программировании встроенных систем.

Рассмотрим пример, когда макрос WAIT_UNTIL(*condition*) в возобновляемой функции обнаруживает невыполнение условия и останавливает сопрограмму в ожидании сигнала. После прихода сигнала, возобновление происходит с первого оператора в макросе WAIT_UNTIL(*condition*), так что сразу же выполняется повторная проверка условия *condition*. Если условие не выполнено, то сопрограмма вновь останавливается. Благодаря этой проверке, сигнал может подаваться как знак не только гарантированного, но и даже лишь вероятного выполнения *condition* (например, при освобождении ресурса, за который конкурируют несколько сопрограмм). Кроме того, дополнительная проверка позволяет учесть изменение условия, если оно произошло уже после подачи сигнала. Этот подход известен как монитор типа Mesa [29].

Макрос JOINT, используемый для ожидания завершения всех запущенных дочерних событий, эквивалентен WAIT_UNTIL(*this->waited* <= 0). Он позволяет корректно обработать ситуацию быстрого завершения дочерних событий, когда родительскую сопрограмму останавливать уже не нужно.

Использование сопрограммы в качестве объекта синхронизации не отменяет, но дополняет ранее перечисленные классические способы, в которых примитив синхронизации существует независимо и хранит в себе указатель на ожидающую сопрограмму для ее прямого возобновления по *pEvent->Post*().

3.5. Приоритеты и вытеснение

При необходимости, в DORSECC может быть разрешена поддержка приоритета события. Для обработки событий с учетом приоритета, система событий клонируется на несколько промежуточных уровней приоритета прерываний, так чтобы каждому уровню приоритета соответствовала своя очередь FIFO и свой диспетчер событий. В этом случае, приоритет обработки события определяется уровнем приоритета программного прерывания, на котором он выполняется. В типовом варианте, установлены два уровня приоритетов: нормальный и высокий.

Отправка на обработку высокоприоритетного события инициирует программное прерывание с высоким приоритетом, и таким образом прерывает обработку менее приоритетного события. Благодаря этому, становится возможным немедленное вытеснение одной сопрограммы более приоритетной. Такое поведение невозможно в классической схеме приоритетов сопрограмм [9, 10], если только для ее реализации не используются приоритетные потоки ОС.

Латентность начала обработки уединенной сопрограммы, как и всякого события, описывается формулой (3). Наличие других событий в очереди увеличивает латентность в соответствии со временем обработки этих событий. Однако, когда в DORSECC используются приоритеты событий, то для наиболее приоритетных событий, всегда

обеспечивается минимальная латентность согласно (3), если только в очереди нет других событий с тем же приоритетом.

Указанный эффект аналогичен вытеснению исполняемого потока ОС более приоритетным, но реализуется без использования ОС и потоков. Поскольку вход в прерывание всегда выполняется быстрее переключения контекста ($T_{IL} \ll T_{TSW}$), то DORSECC обеспечивает меньшую латентность не только в сравнении с классическими сопрограммами согласно (2), но также и с вытесняющими потоками ОС в соответствии с (1).

Сопрограмма с высоким приоритетом является мощным инструментом реального времени, но при этом несет с собой опасность блокировки системы событий. Пока такая сопрограмма выполняется, все события с более низким приоритетом остаются заблокированными. Поэтому, сопрограмма с высоким приоритетом должна быстро завершаться либо останавливаться, что требует дополнительного внимания от программиста. Для устранения риска блокировки нами был реализован так называемый приоритет пробуждения (wakeup priority), который используется в дополнение к обычному приоритету события. При запуске или при возобновлении сопрограммы после остановки, ее событие помещается в очередь диспетчера, соответствующего приоритету пробуждения. Когда в сопрограмме выполняется YIELD, то ее событие перемещается в очередь диспетчера обычного уровня приоритета. Таким образом, сопрограмма начинает или возобновляет работу на уровне приоритета пробуждения, но после первого же вызова YIELD переходит на обычный уровень.

В качестве примера рассмотрим сопрограмму, для которой важна малая латентность запуска в ответ на внешний асинхронный стимул, но которая не должна блокировать выполнение сопрограмм с нормальным уровнем приоритета. Для нее устанавливается нормальный уровень обычного приоритета и высокий уровень приоритета пробуждения. При возникновении внешнего стимула, событие сопрограммы помещается в очередь диспетчера высокого приоритета, и вытесняет выполняемый в этот момент обработчик простого события или сопрограмму с нормальным приоритетом. Однако, как только выполнение сопрограммы на высоком уровне приоритета дойдет до первого YIELD, она перемещается в очередь диспетчера нормального уровня приоритета, и продолжает выполняться на этом уровне до завершения или остановки. Таким образом, достигается малая латентность реакции сопрограммы согласно формуле (3), но блокировка обработки на нормальном уровне приоритета не возникает. Конечно же, сохраняется потенциальный риск блокировки, если программист вовсе не будет использовать YIELD, но такой риск присущ всем системам с кооперативным параллельным исполнением.

3.6. Мультипроцессорная обработка

Предлагаемая методология может быть распространена на случай симметричной мультипроцессорной системы. В самом простом варианте, если очередь событий не пуста, и хотя бы на одном ядре процессора никакое событие не обрабатывается, то диспетчер событий запускает на обработку этим ядром первое по порядку событие в очереди. Это обеспечивает равномерную загрузку промежуточного уровня приоритета для всех ядер процессора.

В мультипроцессорной системе должны быть предусмотрены средства для обеспечения аффинности между ядром процессора и определенным событием. В отношении сопрограмм, аффинность к ядру аналогична хорошо изученному понятию аффинности потока и ядра в операционных системах, например, см. [30, 31]. За счет обеспечения аффинности между ядром процессора и сопрограммой снижаются затраты времени и энергии на поддержание когерентности кэша ядер. Когда речь идет об оптимизации работы кэша, то аффинность носит для диспетчера мягкий, рекомендательный характер, поскольку ее нарушение не приводит к нарушению логики работы программы.

С другой стороны, соблюдение аффинности между ядром процессора и определенным классом простого события может быть строго необходимым для поддержания последовательности обработки событий в том порядке, как они поступали в систему событий. Если несколько экземпляров события одного класса непосредственно следуют в очереди друг за другом, то, без соблюдения аффинности, их обработка может почти одновременно начаться на нескольких ядрах. В некоторых случаях, такое нарушение очередности (race condition) может приводить к разрушению логики обработки событий определенного класса. Жестко заданная афинность событий этого класса к конкретному ядру позволяет сохранить неизменной последовательность обработки.

4. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ

Библиотека DORSECC была использована при создании самодостаточных программ встроенных систем для трех различных платформ: ARM9, Blackfin и ARM Cortex V7-A. С помощью

этих систем, решалась техническая задача высокоскоростной сортировки банкнот. Были разработаны два вида взаимодействующих встроенных систем: контроллер механизма сортировки и модуль проверки банкнот.

Первая из этих систем в реальном времени осуществляла управление датчиками и приводными устройствами механизма, а также поддерживала графический интерфейс пользователя и работу в сети TCP/IP. Вначале она была реализована на процессоре ARM926 с частотой ядра 400 МГц и объемом оперативной памяти 128 МБ (применялся компилятор IAR с максимальной оптимизацией по скорости). Задача управления в реальном времени решалась с помощью двух взаимосвязанных конечных автоматов, графы переходов которых суммарно имели более 90 состояний и были реализованы через систему событий. Наибольший поток событий составил около 12000 в секунду. Сопрограммы применялись для формирования отчетов о работе системы, автоматического регулирования в механизме, а также обеспечения печати и сетевой коммуникации. Во время работы механизма в наиболее нагруженном режиме процессор находился на промежуточном уровне приоритета примерно в течение 36% всего времени, из которых 9% приходилось на выполнение сопрограмм. Собственные накладные расходы времени системы событий (без учета выполнения самого обработчика) для первого события в очереди составляли до 1,5 мкс. Для второго и последующих событий, находящихся очереди, затраты на снижались до 0,5–1,0 мкс, поскольку для них не требовался выход из программного прерывания и повторный вход в него. Обработка подавляющего большинства событий занимала от 20 до 90 мкс, в среднем составляя около 30 мкс. В очереди обычно находилось от 0 до 2 событий, и в редких случаях это число увеличивалось до 5. Латентность обработки события, в среднем, составляла 10–12 мкс, но иногда достигала 220 мкс за счет накопления в очереди часто следующих событий.

Позднее, описанная система была перенесена на процессор ARM Cortex A7 с частотой ядра 1 ГГц (использовался компилятор GCC и уровень оптимизации –o3). Для этой платформы, все накладные расходы системы событий снизились до субмикросекундных величин.

Вторая система использовалась для мультиспектрального сканирования банкноты линейными датчиками изображения и последующего анализа полученных изображений. При распознавании образов применялся метод каскада классификаторов, реализованный с помощью специализированной виртуальной машины. Каскад классификаторов записывался в виде байт-кода, где каждому классификатору соответствовала отдельная инструкция виртуальной машины. Интерпретация байт-кода была реализована с помощью сопрограммы, где после выполнения каждой инструкции вызывался YIELD. В ходе анализа образа банкноты, независимо друг от друга и с перекрытием по времени работали до 3 виртуальных машин. Кроме того, с помощью простых событий, в ходе сканирования выполнялся поворот изображения для компенсации его перекоса. В целом, обработка простых событий занимала не более одной пятой части времени нахождения в программных прерываниях.

Система сканирования и анализа изображений была первоначально выполнена на двухъядерном процессоре Analog Devices Blackfin ADSP-BF607, работающем на частоте ядра 500 МГц (использовался компилятор CrossCore Studio с максимальной оптимизацией по скорости). Позднее, она была перенесена на ранее упомянутую платформу ARM Cortex A7 с частотой ядра 1 ГГц. В обоих вариантах, обработка событий выполнялась на одном из двух ядер процессора. Время исполнения инструкций байт-кода виртуальной машиной на платформе Cortex A7 лежало в пределах от единиц до десятков микросекунд, и только в редких случаях достигало нескольких сотен микросекунд. При пиковой нагрузке во время распознавания, аппаратные и программные прерывания занимали все время процессора. Такая нагрузка, в виде отдельных интервалов продолжительностью от 2 до 10 миллисекунд, в разных режимах отбирала 25–40% общего времени работы. Приведенные значения дают представление о гранулярности распределения процессорного времени, а также о весьма малой доле накладных расходов системы событий.

Мы выявили важный эффект, который можно назвать автоматической балансировкой нагрузки между сопрограммами и простыми событиями. Суть его сводится к следующему. Сопрограмма, выполняющая распознавание образа во второй системе, работает без остановки для ожидания и постоянно перепланирует продолжение своего выполнения при помощи YIELD. Таким образом, она занимает все доступное ей время процессора на промежуточном уровне приоритета прерываний. За счет этого и возникает пиковая нагрузка, при которой для уровня *main*() время не выделяется. Было обнаружено, что при пиковой нагрузке часто повторяющиеся простые события автоматически получают некоторый приоритет перед вычислительными сопрограммами.

Это можно объяснить следующим образом. Простые события обычно синхронизированы со временем или же запускаются извне встроенной системы (например, от срабатывания датчиков). Их частота, таким образом, есть данность, на которую количество событий в очереди не оказывает влияния. Напротив, частота возобновления работы сопрограммы после выполнения YIELD зависит от продолжительности обработки всех событий, находящихся в очереди, и падает при их большом количестве. Таким образом, когда времени для обработки простых событий не хватает, они накапливаются в очереди и снижают долю времени, выделяемую сопрограмме. В результате, сопрограмма получает процессорное время по остаточному принципу, после того, как оно расходуется для обслуживания потока поступающих простых событий.

Из проведенного рассмотрения можно сделать общие выводы о структуре накладных расходов времени на обработку события. Эти расходы складываются, главным образом, из суммарного времени T_{IP} входа в прерывание и завершения прерывания, а также суммарного времени T_{QUEUE} постановки в очередь и получения из очереди. При начале обработки уединенного простого события либо сопрограммы обязательно происходит программное прерывание, что соответствует накладным расходам времени на его обработку $T_{IP} + T_{QUEUE}$. Для простого события к этому добавляется время выполнения *new* и *delete*.

Однако, запуск на параллельное выполнение еще одной сопрограммы не требует вызова программного прерывания, поскольку это прерывание уже обрабатывается. То же относится к возобновлению любой сопрограммы после YIELD. Не требуется прерывание и для простых событий, которые либо возникают с интервалами, меньшими времени их обработки, либо обрабатываются в смеси с сопрограммами. Во всех перечисленных случаях накладные расходы определяются только T_{QUEUE}.

Наименьшие накладные расходы возникают при выполнении YIELD уединенно работающей сопрограммы. В этом случае, в системе событий лишь проверяется отсутствие событий в очереди, а расходы времени T_{IP} и T_{QUEUE} не возникают.

Важное следствие состоит в том, что накладные расходы на переключение между сопрограммами в предложенной методологии близки к расходам на переключение классических сопрограмм. Для классических сопрограмм переключение обычно осуществляется по принципу карусели (round robin), для реализации которой чаще всего применяется связанный кольцевой список сопрограмм. Эта структура данных похожа на структуру данных очереди, и вносит сопоставимые накладные расходы времени. Поскольку программные прерывания для переключения не требуются, T_{IP} в расходы на переключение между сопрограммами не включается.

По мере роста загрузки системы событий и роста заполнения очереди удельные накладные расходы на обработку одного события снижаются. Это происходит потому, что частота программных прерываний F_I и связанная с ней часть общих накладных расходов $F_I T_{IP}$ растут более медленно, чем полная частота событий F_E. Соответственно, уменьшается компонент $F_I T_{IP}/F_E$ удельных накладных расходов.

Обе описанные здесь встроенные системы были созданы для применения в счетно-сортировальных машинах DORS, выпускаемых крупносерийно. На момент написания статьи, их общее количество в эксплуатации превысило 20000 и продолжает увеличиваться.

Небольшие различия кода для реализации системы событий и сопрограмм на используемых платформах связаны с особенностями применяемых компиляторов, а также способом возбуждения программного прерывания. На платформе ARM9 это прерывание возбуждалось эмуляцией аппаратного прерывания от отсутствующего устройства с помощью специального бита запроса прерывания в контроллере прерывания AIC. Процессор Blackfin имеет отдельную инструкцию RAISE для возбуждения программного прерывания. На платформе ARM Cortex V7-A имеется развитая система команд контроллера прерываний GIC-400, которая позволяет гибко управлять возбуждением программных прерываний (SGI).

5. ОБСУЖДЕНИЕ И ВЫВОДЫ

Предложенная методология позволяет построить компактное ядро для обработки событий и выполнения бесстековых сопрограмм в самодостаточной программе встроенной системы реального времени. Обеспечивается малая латентность реакции на внешний стимул не только в сравнении с классическими сопрограммами, но также и с вытесняющими потоками ОС. Накладные расходы на переключение остаются низкими, как и у классических бесстековых сопрограмм.

Мы намеренно использовали слово “методология”, чтобы указать на возможность различных воплощений одного и того же замысла. Для реализации библиотеки DORSECC был выбран язык C++, хотя, при необходимости, можно было бы

ограничиться классическим ANSI C. Применение C++ скрывает детали внутренних механизмов системы событий и выполнения сопрограммы. Использованные нами и намеренно ограниченные выразительные средства языка C++ позволяют не только получить быстрый и оптимизированный ассемблерный код, но и достичь весьма высокого уровня абстракции.

Вместо протопотоков Дункельса, могут быть использованы практически любые виды сопрограмм, реализуемые в виде возобновляемой функции. Перечислим основные отличия нашего подхода от известных реализаций сопрограмм:

• представление сопрограммы как события, а ее возобновляемой функции в качестве обработчика этого события;

• обработку событий, в том числе сопрограмм, в контексте низкоприоритетного программного прерывания;

• обеспечение нескольких уровней приоритета событий и сопрограмм с помощью различных приоритетов программных прерываний для их обработки;

• выполнение YIELD путем повторной отправки события сопрограммы на обработку;

• возможность немедленного вытеснения одной сопрограммы более приоритетной сопрограммой или событием при асинхронном возникновении внешнего стимула, с малым риском блокировки системы событий.

Самодостаточная встроенная программа разделяется на три уровня приоритета исполнения кода. На самом высокоприоритетном уровне жесткого реального времени производится обработка аппаратных прерываний, с субмикросекундными задержками и длительностью на уровне единиц или десятков микросекунд. На промежуточном уровне мягкого реального времени, с характерными временами в десятки или сотни микросекунд, происходит обработка простых событий, а также выполняются сопрограммы. И, наконец, на самом низком уровне приоритета, соответствующего основной функции *main*(), выполняются постоянные действия с миллисекундными разрывами в исполнении, не требующие реального масштаба времени. Загрузка промежуточного уровня автоматически балансируется между простыми событиями и сопрограммами таким образом, что при необходимости обработки интенсивного потока простых событий, процессорное время для этого отбирается у сопрограмм.

Накладные расходы времени для обработки события, в наихудшем случае, заметно превосходят накладные расходы обработки прерывания. Однако, для большинства случаев обработки, эти расходы оказываются существенно меньшими. Как правило, они не превышают единиц процентов от длительности самих обработчиков событий.

Вызов возобновляемой функции в контексте программного прерывания решает вопрос о способе вызова, который был предметом активного обсуждения при выработке стандарта C++ 20 [7]. Когда сопрограмма представляет собой просто конструкцию языка программирования, то ее возобновляемую функцию поневоле приходится синхронно вызывать из контекста потока исполнения программы. В предлагаемом подходе, сопрограмма работает изолированно и асинхронно в контексте программного прерывания. Она запускается путем отправки в систему событий, но ее возобновляемая функция обычно вызывается вне запускающего контекста, то есть асинхронным образом по отношению к запускающему коду. С точки зрения аппаратуры, вызов возобновляемой функции инициируется контроллером прерывания, а не ядром процессора. Он не связан цепочкой вызовов с единственным потоком исполнения самодостаточной программы.

Дополнительное преимущество использования C++ проявилось в возможности динамического создания большого количества простых событий и сопрограмм при малых накладных затратах. Это важно для декомпозиции сложных алгоритмов по принципу fork/join, который пока еще редко используется во встроенных системах реального времени. Для управления параллельным исполнением могут применяться как классические способы синхронизации, так и средства, характерные для языков Go и occam. Поскольку средства и механизмы синхронизации следуют теории взаимодействующих последовательных процессов Хоара [28], то для верификации программ могут применяться формальные методы, основанные на этой теории.

Проведенная реализация сложных самодостаточных программ реального времени для массово выпускаемых устройств с использованием библиотеки DORSECC показала надежность предлагаемого подхода. Протопоток Дункельса, выбранный за основу сопрограммы, обладает неоспоримыми преимуществами: независимостью от типа компилятора и версии языка, высокой скоростью исполнения и малыми затратами памяти. Однако он накладывает ограничения на синтаксис возобновляемой функции и не предполагает автоматических средств проверки соблюдения этих ограничений. В будущем имеет смысл опробовать предлагаемую методологию с другими воплощениями сопрограмм, которые были бы более безопасны и удобны для программиста.

Предложенный нами механизм событий имеет явное сходство с механизмом *sender – executor – receiver* (отправитель – исполнитель – получатель), который планируется к введению в новый стандарт языка C++ [32]. *Executor* есть инструмент доступа к определенному контексту исполнения, который понимается очень широко: от потока программы до сопроцессора SIMD. Представляется важным опробовать абстракцию *executor* в применении к контексту исполнения в программном прерывании.

При внимательном рассмотрении можно заметить, что буфер FIFO системы событий в каждый момент времени содержит динамически генерируемую и исполняемую последовательность символов, известную как шитый код (threaded code). Вообще, символы шитого кода представляют собой единообразно оформленные ссылки на подпрограммы. В данном случае это указатели на объекты события, каждый со своей подпрограммой *Exec*(). Шитый код близок к понятию байт-кода и известен высокой скоростью исполнения, приближающейся к скорости исполнения нативного кода процессора. Понятие шитого кода было введено Беллом [33]. Затем, оно было развито Муром в языке FORTH [34], а также Верноком и Гешке в языке PostScript [35] для хранения тела процедуры в откомпилированной форме. Описанное нами порождение событий, а также планирование и диспетчеризацию их обработки можно рассматривать как особый случай машинной генерации программы в шитом коде для ее немедленного исполнения на выделенном уровне приоритета прерывания.

СПИСОК ЛИТЕРАТУРЫ

1. *Knuth D.* The Art of Computer Programming: Fundamental Algorithms. Addison-Wesley, Reading, Massachusetts, the USA. 1997. P. 193–200.
2. *Dunkels A., Schmidt O., Voigt T., Ali M.* Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems. Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006), Boulder, Colorado, the USA, November 2006.
3. Libtask: a Coroutine Library for C and Unix. https://swtch.com/libtask/. Accessed 21.03.2023.
4. The Boost C++ Libraries. Chapter 51: Boost.Coroutine. https://theboostcpplibraries.com/boost.coroutine. Accessed 21.03.2023.
5. The Boost C++ Libraries. Chapter 32: Boost.Asio. https://theboostcpplibraries.com/boost.asio. Accessed 21.03.2023.
6. CO2 Coroutine. https://github.com/jamboree/co2. Accessed 21.03.2023.
7. *Goodspeed N.* Stackful Coroutines and Stackless Resumable Functions. The C++ Standards Committee – ISOCPP Document N4232. December 13, 2014. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4232.pdf.
8. *Belson B., Holdsworth J., Xiang W., Philippa B.* A Survey of Asynchronous Programming Using Coroutines in the Internet of Things and Embedded Systems. ACM Transactions on Embedded Computer Systems. April 2019. V. 18. № 3. Article 21. https://doi.org/10.1145/3319618.
9. *Moskała M.* Kotlin Coroutines: Deep Dive. Kt. Academy, Warsaw, Poland. 2022.
10. FreeRTOS Kernel. Co-routines. https://www.freertos.org/croutine.html. Accessed 01.11.2023.
11. *Шалыто А.А.* Парадигма автоматного программирования. Научно-Технический вестник ИТМО. 2008. Выпуск 53: Автоматное программирование.
12. *Ousterhout J.* Why Threads are a Bad Idea (for most purposes). January 1996. USENIX Winter Technical Conference, San Diego, California, the USA.
13. *Lee E.* The Problem with Threads. Computer. 2006. V. 39. P. 33–42.
14. *Samek M.* Who Moved My State? Dr. Dobb's (online). April 1, 2003. https://drdobbs.com/who-moved-my-state/184401643
15. *Samek M.* Practical Statecharts in C/C++: Quantum Programming for Embedded Systems. CMP Books, San-Francisco, the USA. 2002.
16. *Gamma E., Helm R., Johnson R., Vlissides J.* Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Massachusetts, the USA. 1995. P. 293.
17. Kernel-Mode Driver Architecture Design Guide: Introduction to DPC Objects. https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/introduction-to-dpc-objects. Accessed 21.03.2023.
18. Deferred Work – The Linux Kernel Documentation. https://linux-kernel-labs.github.io/refs/heads/master/labs/deferred_work.html. Accessed 21.03.2023.
19. *Nicholl R.A.* A Specification of Modula-2 Process (Coroutine) Management. Journal of Pascal, Ada & Modula-2. 1988. V. 7. № 5. P. 16–22.
20. *Tatham S.* Coroutines in C (online). https://www.chiark.greenend.org.uk/~sgtatham/coroutines.html. 2000.
21. *Dunkels A., Grönvall B., Voigt T.* Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors. Proceedings of the Conference on Local Computer Networks. 2004. P. 455–462. https://doi.org/10.1109/LCN.2004.38
22. *Barnett D., Massa A.* Inside the uIP Stack. Dr Dobbs Journal (online). February 1, 2005. https://www.drdobbs.com/inside-the-uip-stack/184405971
23. *Dunkels A.* Programming Memory-Constrained Networked Embedded Systems. Swedish Institute of Computer Science Doctoral Thesis (online). SICS Dissertation Series 47. February 2007.

http://www.diva-potal.org/smash/get/diva2:1041306/FULLTEXT01.pdf

24. *Perlis A.* Epigrams on programming. ACM SIGPLAN Notices. September 1982. V. 17. № 9. P. 7–13. https://doi.org/10.1145/947955.1083808
25. *McCool M., Reinders J., Robison A.* Structured Parallel Programming: Patterns for Efficient Computation. Morgan Kaufmann, Burlington, Massachusetts, the USA. 2012. P. 209–252.
26. *Cox-Buday K.* Concurrency in Go. O'Reily Media, Sebastopol, California, the USA. 2017.
27. *Roscoe A., Hoare C.A.R.* The Laws of Occam Programming. Theoretical Computer Science. 1988. V. 60. P. 177–229. https://doi.org/10.1016/0304-3975(88)90049-7
28. *Hoare C.A.R.* Communicating sequential processes. Prentice-Hall, Englewood Cliffs, New Jersey, the USA. 1985.
29. *Lampson B.W., Redell D.D.* Experience with processes and monitors in Mesa. Comm. of the ACM. 1980. V. 23. № 2. P. 105–117.
30. *Love R.* CPU Affinity. Linux Journal (online). July 1, 2003. https://www.linuxjournal.com/article/6799
31. *Gujarati A., Cerqueira F., Brandenburg B.* Multiprocessor real-time scheduling with arbitrary processor affinities: from practice to theory. Real-Time Systems. 2015. V. 51. P. 440–483. https://doi.org/10.1007/s11241-014-9205-9
32. *Hoberock J., Garland M., Kohloff C. et al.* A Unified Executors Proposal for C++. The C++ Standards Committee – ISOCPP Document P0443R14. September 15, 2020. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p0443r14.html
33. *Bell J.* Threaded code. Communications of the ACM. 1973. V. 16. № 6. P. 370–372. https://doi.org/10.1145/362248.362270.
34. *Rather E., Colburn D., Moore C.* The evolution of Forth. In: History of programming languages – II. ACM Other Books. 1 January 1996. P. 625–670. https://doi.org/10.1145/234286.1057832 ISBN 9780201895025
35. *Reid G.* Thinking in PostScript. Addison-Wesley, Reading, Massachusets, the USA. 1990. P. 105–118.

UNIFIED PROCESSING OF EVENTS AND COROUTINES IN EMBEDDED PROGRAM

© 2024 P. V. Minin[a]

[a] *DORS R&D LLC*

7–2, Federativny prospect, Moscow, 111399, Russia

A new architectural approach to real-time embedded programming is described. A bare-metal program is written in C/C++ and combines event-driven technique with concurrency based on co-routines. Events are processed by soft real-time core at the priority level of software generated interrupts. Event is first posted to input queue of the core and then processed by invocation of its event handler. A special case of event is co-routine, its resumable function being a co-routine event handler. The co-routine that either yielded or needs to be resumed is queued for event processing once again. As a result, it is processed multiple times until execution of resumable function comes to the end of its operator sequence. Different levels of processing priority may be assigned to an event. Soft real-time core could be further expanded to run on symmetrical multiprocessor hardware. A combination of co-routines and basic events could easily be used in fork/join model. Concurrency constructs resemble those of Go and occam languages. Virtually all classic types of synchronization primitives could be implemented. The new approach was implemented for various ARM and Blackfin processors in C++ language as portable DORSECC library. This library was further used to program real-time embedded systems for mass-produced banknote sorting machines. One type of systems was used to recognize and validate banknote images by the method of cascade of one-class classifiers. The other system worked as a motion controller and used finite automata to control sensors and actuators. The total number of systems in operation is currently over 20000. The event and co-routine core in these systems provides average event processing time in the range of dozens of microseconds with sub-microsecond overhead time per each event.

Keywords: real-time, embedded, bare metal, concurrency, event-driven, co-routine, C++, software interrupt, synchronization, threaded code

REFERENCES

1. *Knuth D.* The Art of Computer Programming: Fundamental Algorithms. Addison-Wesley, Reading, Massachusetts, the USA. 1997. P. 193–200.
2. *Dunkels A., Schmidt O., Voigt T., Ali M.* Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems. Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006), Boulder, Colorado, the USA, November 2006.
3. Libtask: a Coroutine Library for C and Unix. https://swtch.com/libtask/. Accessed 21.03.2023.

4. The Boost C++ Libraries. Chapter 51: Boost.Coroutine. https://theboostcpplibraries.com/boost.coroutine. Accessed 21.03.2023.
5. The Boost C++ Libraries. Chapter 32: Boost.Asio. https://theboostcpplibraries.com/boost.asio. Accessed 21.03.2023.
6. CO2 Coroutine. https://github.com/jamboree/co2. Accessed 21.03.2023.
7. *Goodspeed N.* Stackful Coroutines and Stackless Resumable Functions. The C++ Standards Committee – ISOCPP Document N4232. December 13, 2014. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4232.pdf.
8. *Belson B., Holdsworth J., Xiang W., Philippa B.* A Survey of Asynchronous Programming Using Coroutines in the Internet of Things and Embedded Systems. ACM Transactions on Embedded Computer Systems. April 2019. V. 18. № 3. Article 21. https://doi.org/10.1145/3319618.
9. *Moskała M.* Kotlin Coroutines: Deep Dive. Kt. Academy, Warsaw, Poland. 2022.
10. FreeRTOS Kernel. Co-routines. https://www.freertos.org/croutine.html. Accessed 01.11.2023.
11. *Shalyto, A.A.* Paradigm for automata based programming, in Nauchno-tekhnicheskii vestnik ITMO (ScientificTechnical Herald of ITMO Univ.). Issue 53: Avtomatnoe programmirovanie (Automata Based Programming), St. Petersburg, 2008.
12. *Ousterhout J.* Why Threads are a Bad Idea (for most purposes). January 1996. USENIX Winter Technical Conference, San Diego, California, the USA.
13. *Lee E.* The Problem with Threads. Computer. 2006. V. 39. P. 33–42.
14. *Samek M.* Who Moved My State? Dr. Dobb's (online). April 1, 2003. https://drdobbs.com/who-moved-my-state/184401643
15. *Samek M.* Practical Statecharts in C/C++: Quantum Programming for Embedded Systems. CMP Books, San-Francisco, the USA. 2002.
16. *Gamma E., Helm R., Johnson R., Vlissides J.* Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Massachusetts, the USA. 1995. P. 293.
17. Kernel-Mode Driver Architecture Design Guide: Introduction to DPC Objects. https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/introduction-to-dpc-objects. Accessed 21.03.2023.
18. Deferred Work – The Linux Kernel Documentation. https://linux-kernel-labs.github.io/refs/heads/master/labs/deferred_work.html. Accessed 21.03.2023.
19. *Nicholl R.A.* A Specification of Modula-2 Process (Coroutine) Management. Journal of Pascal, Ada & Modula-2. 1988. V. 7. № 5. P. 16–22.
20. *Tatham S.* Coroutines in C (online). https://www.chiark.greenend.org.uk/~sgtatham/coroutines.html. 2000.
21. *Dunkels A., Grönvall B., Voigt T.* Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors. Proceedings of the Conference on Local Computer Networks. 2004. P. 455–462. https://doi.org/10.1109/LCN.2004.38
22. *Barnett D., Massa A.* Inside the uIP Stack. Dr Dobbs Journal (online). February 1, 2005. https://www.drdobbs.com/inside-the-uip-stack/184405971
23. *Dunkels A.* Programming Memory-Constrained Networked Embedded Systems. Swedish Institute of Computer Science Doctoral Thesis (online). SICS Dissertation Series 47. February 2007. http://www.diva-potal.org/smash/get/diva2:1041306/FULLTEXT01.pdf
24. *Perlis A.* Epigrams on programming. ACM SIGPLAN Notices. September 1982. V. 17. № 9. P. 7–13. https://doi.org/10.1145/947955.1083808
25. *McCool M., Reinders J., Robison A.* Structured Parallel Programming: Patterns for Efficient Computation. Morgan Kaufmann, Burlington, Massachusetts, the USA. 2012. P. 209–252.
26. *Cox-Buday K.* Concurrency in Go. O'Reily Media, Sebastopol, California, the USA. 2017.
27. *Roscoe A., Hoare C.A.R.* The Laws of Occam Programming. Theoretical Computer Science. 1988. V. 60. P. 177–229. https://doi.org/10.1016/0304-3975(88)90049-7
28. *Hoare C.A.R.* Communicating sequential processes. Prentice-Hall, Englewood Cliffs, New Jersey, the USA. 1985.
29. *Lampson B.W., Redell D.D.* Experience with processes and monitors in Mesa. Comm. of the ACM. 1980. V. 23. № 2. P. 105–117.
30. *Love R.* CPU Affinity. Linux Journal (online). July 1, 2003. https://www.linuxjournal.com/article/6799
31. *Gujarati A., Cerqueira F., Brandenburg B.* Multiprocessor real-time scheduling with arbitrary processor affinities: from practice to theory. Real-Time Systems. 2015. V. 51. P. 440–483. https://doi.org/10.1007/s11241-014-9205-9
32. *Hoberock J., Garland M., Kohloff C. et al.* A Unified Executors Proposal for C++. The C++ Standards Committee – ISOCPP Document P0443R14. September 15, 2020. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p0443r14.html
33. *Bell J.* Threaded code. Communications of the ACM. 1973. V. 16. № 6. P. 370–372. https://doi.org/10.1145/362248.362270.
34. *Rather E., Colburn D., Moore C.* The evolution of Forth. In: History of programming languages – II. ACM Other Books. 1 January 1996. P. 625–670. https://doi.org/10.1145/234286.1057832 ISBN 9780201895025
35. *Reid G.* Thinking in PostScript. Addison-Wesley, Reading, Massachusets, the USA. 1990. P. 105–118.

ПРОГРАММИРОВАНИЕ, 2024, № 5, с. 31–41

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.92

ОПРЕДЕЛЕНИЕ СТЕПЕНИ СЛОЖНОСТИ ОБЪЕКТОВ НА ИЗОБРАЖЕНИЯХ

© 2024 г. В. Б. Бокшанский[a], В. А. Кулин[a], Г. С. Финякин[a, b], А. С. Харламов[c], А. А. Шацкий[a, *]

[a] *Московский государственный технический университет имени Н.Э. Баумана*
105005 Москва, ул. Бауманская 2-я, д. 5, стр. 1, Россия
[b] *Национальный исследовательский университет "МЭИ"*
111250, Москва, вн. тер. г. муниципальный округ Лефортово, ул. Красноказарменная, д. 14, стр. 1, Россия
[c] *Московский государственный технический университет гражданской авиации*
125993, Москва, Кронштадтский бульвар, д. 20, Россия
**E-mail: shatskiyalex@gmail.com*

Поступила в редакцию: 23.09.2023 г.
После доработки 19.02.2024 г.
Принята к публикации 13.05.2024 г.

Предложен новый метод оценки сложности геометрических фигур (пятен), учитывающий внутреннюю структуру пятен, а не только их внешний контур. Задача по вычислению степени сложности объектов разделена на составляющие: сегментация пятен и оценка сложности изолированных пятен. Новый метод обладает относительно низкой вычислительной сложностью по сравнению с рассмотренными в работе альтернативными методами. С помощью нового метода был создан алгоритм на основе параллельных вычислений языка CUDA для графических ускорителей (видеокарт), что дополнительно повышает быстродействие нашего метода. Проведен качественный и количественный анализ существующих (альтернативных) методов, выявлены их преимущества и недостатки по сравнению с нашим методом и друг с другом. Реализованный на основе нового метода алгоритм апробирован как на искусственных, так и на реальных изображениях.

Ключевые слова: инварианты Hu, сложность изображения, сжатие изображения, контур изображения, сегментация изображения, выделение контуров, классификация изображений, статистические моменты изображений, вычислительная сложность

DOI: 10.31857/S0132347424050036, **EDN:** OLZNYI

1. ВВЕДЕНИЕ

В экспериментальных исследованиях часто встаёт проблема поиска различных аномалий и новых особенностей среди большого количества экспериментальных данных.

В качестве примера можно привести автоматизированный поиск микроорганизмов (амёб, инфузорий и т. п.) в изображениях, полученных с оптических микроскопов. Для решения подобных задач существуют нейросетевые алгоритмы, но обучить нейросети распознавать все возможные типы (известных и неизвестных) микроорганизмов не представляется возможным, поэтому актуальной становится проблема поиска наиболее сложных аномалий в изображениях с микроскопов, что может являться альтернативным методом обнаружения.

Еще одним примером для поиска аномалий может быть фиксация пролета воздушных судов над аэропортами. Как показывают эксперименты, летящий самолет на фоне неба является относительно сложным объектом (аномалией) – в смысле, о котором пойдет речь далее. И на фоне неба самолет можно однозначно выделить как аномалию по сравнению с другими небесными объектами (Луна, Солнце, звёзды, спутники, метеоры и т. п.). Как показывают эксперименты, иногда самолет является даже более сложной аномалией, чем разнообразные наземные строения на фоне горизонта.

Также в качестве примера можно привести поиск аномалий в картах космологического (реликтового) фона. Эксперименты WMAP CMB [1] и Planck CMB [2] получили огромный массив данных таких наблюдений – аномалий

реликтового фона. Астрофизики и астрономы всего мира используют эти данные для самых разнообразных научных целей – от изучения ранней горячей Вселенной от стадии рекомбинации, до поиска топологических аномалий (космологических струн и червоточин). Этих данных настолько много, что их ручная сортировка попросту невозможна, и необходим автоматизированный алгоритм по поиску и отбору аномалий различного типа, по структуризации этих аномалий и разбиению их по типам и классам.

Наша работа предлагает новый метод для автоматизированного поиска и классификации аномалий в любых данных двумерного типа (на двумерных картах-изображениях). Мы разбиваем найденные аномалии по степени их сложности, вводя при этом математическое определение сложности двумерного изображения. При этом наш математический метод определяет сложность текущего двумерного объекта (на изображении) инвариантно по отношению к трем группам геометрических преобразований: к трансляциям, к масштабированию и к вращению.

Подобные исследования не являются принципиально новыми. Например в общепринятом подходе поиска аномалий в космологическом фоне принято разлагать космологический фон по мультиполям и строить спектр (гистограмму) такого разложения. Данный подход (разложения по мультиполям) имеет существенный недостаток – инвариантность мультиполей возможна только если полный монопольный заряд равен нулю. В разложении микроволнового излучения это достигается специальным выбором нулевого уровня яркости изображения, ниже которого яркость считается эффективно отрицательной, так что суммарная эффективная яркость изображения оказывается равной нулю. Такой выбор эффективной яркости возможен только благодаря плотному расположению пятен флуктуаций микроволнового фона друг относительно друга. Если площадь всех пятен на изображении много меньше площади всего изображения, то выбор нулевого уровня яркости приводит к обрезанию краёв пятен. Иногда и сами пятна полностью уходят под уровень фона.

В данной работе мы предлагаем новый метод определения аномалий на изображениях, а главное – вычисление уровня сложности таких аномалий (пятен). Помимо того, что наш метод является инвариантным к сдвигам, к масштабированию и к поворотам изображения, он еще и не зависит от выбора уровня фона на изображении (от выбора уровня нулевой яркости).

Потенциальные применения таких исследований:

• Поиск аномалий в изображениях, полученных с оптических микроскопов – для выявления на снимках микроорганизмов.

• Анализ медицинских снимков для выявления потенциальной онкологии и других заболеваний.

• Применение рентгеновской микротомографии для количественного и структурного анализа фармацевтических многокомпонентных систем.

• Космология (поиск аномалий реликтового фона).

2. ИНВАРИАНТЫ HU

Моменты изображения, или статистические моменты хорошо известны из математической статистистики. Для дискретного (пиксельного) изображения в оттенках серого они определяются согласно формулам:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y) \qquad (1)$$

Здесь x и y – пиксельные координаты на изображении, $I(x,y)$ – яркость пикселя. Моменты M_{ij} не инвариантны ни к каким преобразованиям (за исключением моментов нулевого порядка M_{00}). Мы будем рассматривать моменты M_{ij} с общим порядком степеней координат не выше 3-й степени ($i+j \le 3$). Таких моментов (далее обозначим их как M_{pq}) существует 10 штук.

С помощью моментов M_{pq} можно получить центр яркости изображения (математические ожидания по x и по y), его координаты определяются формулами:

$$\hat{x} := \frac{M_{10}}{M_{00}}, \quad \hat{y} := \frac{M_{01}}{M_{00}}. \qquad (2)$$

С помощью (1) и (2) можно ввести центральные моменты:

$$\mu_{pq} = \sum_x \sum_y \left(x - \hat{x}\right)^p \left(y - \hat{y}\right)^q I(x,y) \qquad (3)$$

Центральные моменты инвариантны относительно сдвига (трансляций). Ненулевых величин μ_{pq} 8 штук.

Можно также определить масштабные инварианты:

$$\eta_{pq} := \frac{\mu_{pq}}{\mu_{00}^{\left[1+0.5(p+q)\right]}}. \qquad (4)$$

Величины η_{pq} инвариантны и к трансляциям, и к изменению масштаба – их тоже 8 штук.

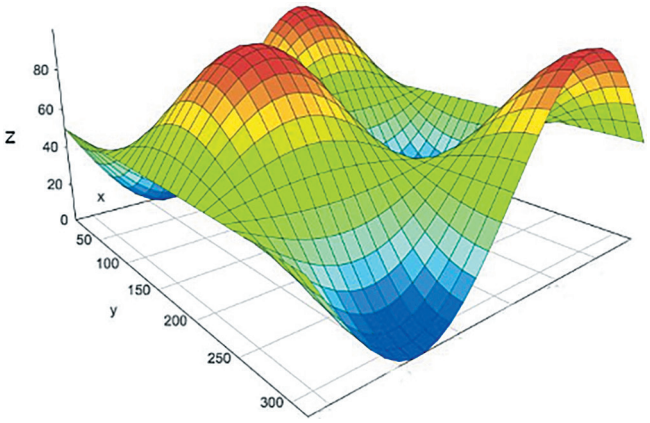
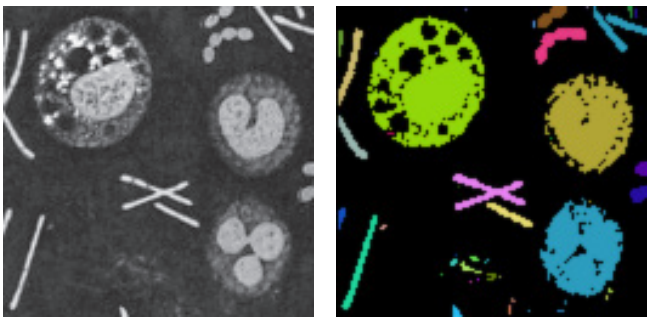
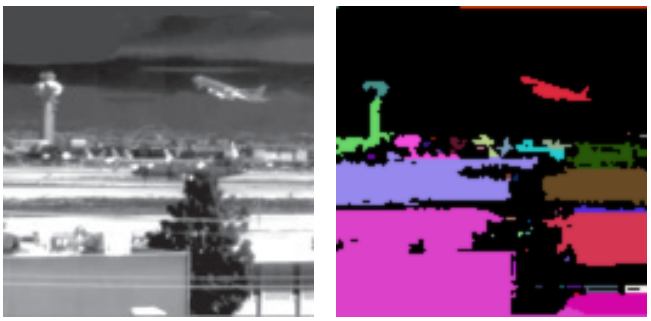


Рис. 1. Интенсивность яркости изображения.

(a) Микроорганизмы:



(b) Взлетающий самолет:



(c) Аномалии космологического фона реликтового излучения:

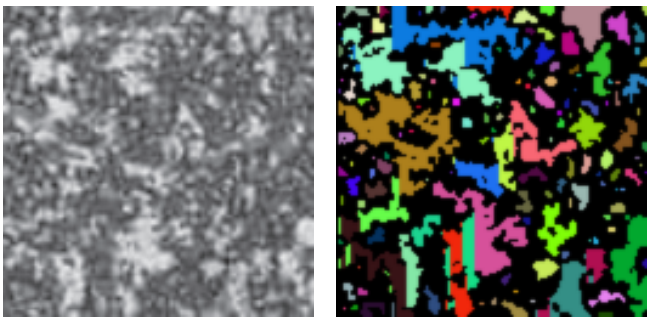


Рис. 2. Изображения в оттенках серого (слева), и сегментация этих же изображений по отдельным сегментам (объединяемых в пятна), выделенных разными цветами (справа).

С помощью выражений (1–4) можно вычислить еще 7 величин, которые будут дополнительно инвариантны еще и к поворотам. Они называются инвариантами *Hu* [3], [4]. Можно показать, что традиционный набор инвариантов *Hu* не является ни независимым, ни полным. Третий инвариант Hu_3 не очень полезен, так как он зависит от других инвариантов. В оригинальном наборе инвариантов *Hu* отсутствует инвариант независимого момента третьего порядка, а вместо него вводится 8-й инвариант Hu_8.

Таким образом, всего можно определить 8 инвариантов *Hu*:

$$Hu_1 = (\eta_{20} + \eta_{02}). \tag{5}$$

$$Hu_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2. \tag{6}$$

$$Hu_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2. \tag{7}$$

$$Hu_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2. \tag{8}$$

$$\begin{aligned} Hu_5 = {} & (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - \right. \\ & \left. - 3(\eta_{21} + \eta_{03})^2\right] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03}) \times \\ & \times \left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]. \end{aligned} \tag{9}$$

$$\begin{aligned} Hu_6 = {} & (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + \\ & + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}). \end{aligned} \tag{10}$$

$$\begin{aligned} Hu_7 = {} & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - \right. \\ & \left. - 3(\eta_{21} + \eta_{03})^2\right] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03}) \times \\ & \times \left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]. \end{aligned} \tag{11}$$

$$\begin{aligned} Hu_8 = {} & \eta_{11}\left[(\eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2\right] - \\ & -(\eta_{20} - \eta_{02})(\eta_{30} + \eta_{12})(\eta_{03} + \eta_{21}). \end{aligned} \tag{12}$$

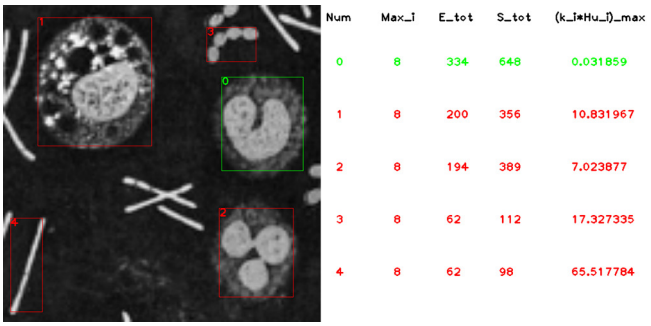
3. НАХОЖДЕНИЕ ПЯТЕН НА ИЗОБРАЖЕНИИ

Перед тем как исследовать конкретное пятно изображения на сложность сначала необходимо определить границы этого пятна. Для этого мы используем алгоритм **WaterShed** [5]. Суть алгоритма **WaterShed** – разбиение изображения на множество покрывающих его областей. За основу этого алгоритма был выбран алгоритм сегментации по водоразделам. Если значение яркости изображения откладывать по оси *OZ*, то в местах

(a) Искусственно созданные пятна:

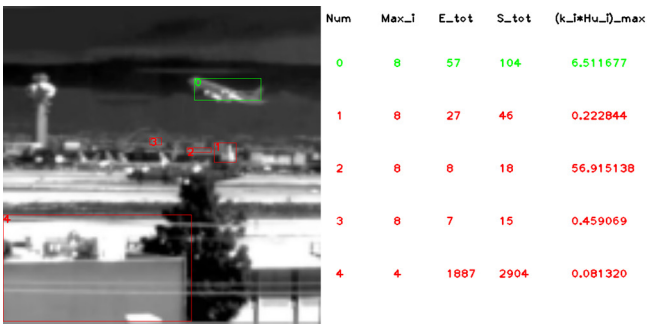
Num	Max_i	E_tot	S_tot	(k_i*Hu_i)_max
0	8	965	2216	0.216645
1	8	22	22	0.565753
2	4	2679	2690	0.022558
3	2	104	104	0.024970
4	2	80	159	0.449212

(b) Микроорганизмы:



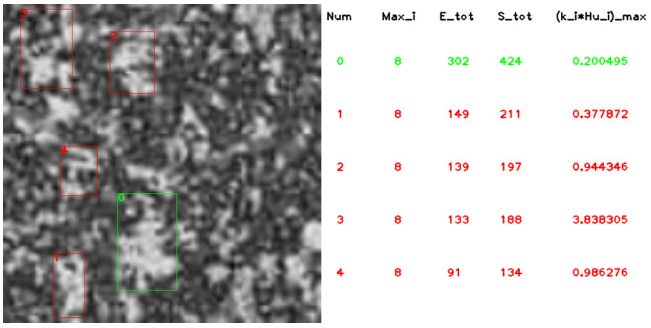
Num	Max_i	E_tot	S_tot	(k_i*Hu_i)_max
0	8	334	648	0.031859
1	8	200	356	10.831967
2	8	194	389	7.023877
3	8	62	112	17.327335
4	8	62	98	65.517784

(c)Взлетающий самолет:



Num	Max_i	E_tot	S_tot	(k_i*Hu_i)_max
0	8	57	104	6.511677
1	8	27	46	0.222844
2	8	8	18	56.915138
3	8	7	15	0.459069
4	4	1887	2904	0.081320

(d) Аномалии космологического фона реликтового излучения:



Num	Max_i	E_tot	S_tot	(k_i*Hu_i)_max
0	8	302	424	0.200495
1	8	149	211	0.377872
2	8	139	197	0.944346
3	8	133	188	3.838305
4	8	91	134	0.986276

Рис. 3. Сортировка пятен изображения по убыванию уровня сложности и выделение главного пятна зелёной рамкой. Показан вывод top-5 пятен изображения, ранжированных сначала по величине номера *Max_i* у величины $|k_i Hu_i|_{max}$, а потом по яркости пятна. Справа приведена таблица характеристик пятен: номер пятна, номер *Max_i* у максимальной величины $|k_i Hu_i|_{max}$, яркость пятна E_{tot}, площадь пятна S_{tot} и значение $|k_i Hu_i|_{max}$.

наибольшей интенсивности образуются “хребты”, наименьшей – “впадины”, а в однородных регионах – равнины (см. рис. 3).

Объектом будет считаться некоторая область, состоящая в общем случае из пикселей разной интенсивности (отдельное пятно изображения). Обязательное условие для данной области, что интенсивность пикселей, находящихся на краю пятна выше некоторого значения фона и меньше или равно интенсивности пикселей объекта граничащих с ними. Иными словами, выделяемый объект по сути это пятно, выделяющееся на некотором фоне. Причем изменением значения фона можно регулировать уровень чувствительности алгоритма **WaterShed**. Яркость пикселей ниже уровня фона алгоритмом считается нулевой яркостью.

Изображения на левых частях рис. 3 представляют из себя оригинальные кадры, содержащие объекты исследования. На правых частях этого же рисунка разными цветами обозначены сегменты пятен, найденные алгоритмом **WaterShed**.

Для определения сложности объектов следует сегментировать изображение, а затем найти на нем кандидаты на объекты (отдельные пятна) для последующей обработки. Сегменты с разной яркостью, касающиеся друг друга хотя бы в одном пикселе, должны быть объединены в одно пятно. Сегментация отдельных пятен может быть затруднена из за высокой зашумленности изображения, поэтому первой задачей является определение уровня фона, а уже потом поиск сегментов и отдельных пятен.

4. УРОВЕНЬ СЛОЖНОСТИ ПЯТНА, ОПРЕДЕЛЕННЫЙ ЧЕРЕЗ ИНВАРИАНТЫ *Hu*

Дискретный уровень сложности пятна, выраженный через инварианты Hu_3 можно определить как номер максимального элемента вектора:

$$Compl = \left\{ |k_1 Hu_1|, |k_2 Hu_2|, |k_3 Hu_3|, |k_4 Hu_4|, |k_5 Hu_5|, |k_6 Hu_6|, |k_7 Hu_7|, |k_8 Hu_8| \right\} \quad (13)$$

Здесь k_i – специально подобранные нормировочные коэффициенты.

Коэффициенты k_i подбираются эмпирическим путем, главным критерием для такого подбора является вид гистограммы величин $|k_i Hu_i|$, состоящей из восьми бинов. При выборе коэффициентов k_i мы исходили из того, что наиболее простые геометрические фигуры (круг, или окружность) должны иметь максимум гистограммы, смещенный максимально влево (к первому

бину), а наиболее сложные геометрические фигуры – наоборот: максимум гистограммы должен соответствовать восьмому бину.

Стоит отметить, что пятна с центрально-симметричным распределением яркости имеют только первый ненулевой инвариант Hu_1. Пятна с осевой симметрией имеют ненулевые инварианты *Hu* с номерами не выше четвертого. Максимально асимметричные пятна содержат все ненулевые инварианты *Hu*.

Один из возможных, подобранных по этому принципу, вариантов набора коэффициентов k_i может быть следующим:

$$k_i = \{1, 3.5, 12.25, 43, 150, 525, 1840, 6430\}.$$

Если несколько пятен будут иметь одинаковую степень сложности, то эти пятна между собой можно будет ранжировать по яркости пятна, либо по площади пятна, либо по номеру *Max_i* бина гистограммы, соответствующего её максимуму: $|k_i Hu_i|_{max}$, либо по комбинации этих величин.

5. РЕЗУЛЬТАТ РАБОТЫ АЛГОРИТМА

В результате работы алгоритма **WaterShed** мы получаем маску с сегментированным изображением (правая часть рис. 3), где пиксели одного сегмента помечены одинаковой меткой и образуют связную область. Основным недостатком данного алгоритма является использование процедуры предварительной обработки для картинок с большим количеством локальных минимумов (изображения со сложной текстурой и с обилием различных цветов). Для ускорения работы, изображение обрабатывается на графическом ускорителе (видеокарте). При этом, каждый пиксель обрабатывается своим собственным потоком с использованием архитектуры CUDA, а информация о пикселях сохраняется в память видеокарты. В итоге после обработки изображения мы получаем массив структур с данными обо всех объектах (пятнах) на изображении. После обработки этого массива структур с информацией о сегментированном объекте далее могут быть вычислены центральные статистические моменты всех пятен изображения, необходимые для вычисления инвариантов Hu.

На рис. 4 показаны результаты работы алгоритма по вычислению сложности на искусственно-созданных пятнах, а также на изображениях реальных объектов: микроорганизмов, взлетающего самолета и аномалии космологического фона реликтового излучения.

Исходя из этого, для выделения наиболее приоритетного объекта, следует в первую очередь выбрать объекты с максимальным значением номера *Max_i* у величины $|k_i Hu_i|_{max}$, а затем отдать предпочтение объекту с наибольшим значением одного из параметров[1]: E_{tot}, S_{tot}, $|k_i Hu_i|_{max}$. Какой из них выбрать, можно решить экспериментальным путем, в зависимости от поставленной задачи. Также можно использовать все три параметра E_{tot}, S_{tot}, $|k_i Hu_i|_{max}$ путем голосования по большинству.

6. АЛЬТЕРНАТИВНЫЕ МЕТОДЫ ОПРЕДЕЛЕНИЯ СЛОЖНОСТИ

Про метод максимального мультиполя было написано во введении. Здесь мы хотим привести еще несколько методов определения сложности пятен и сравнить их:

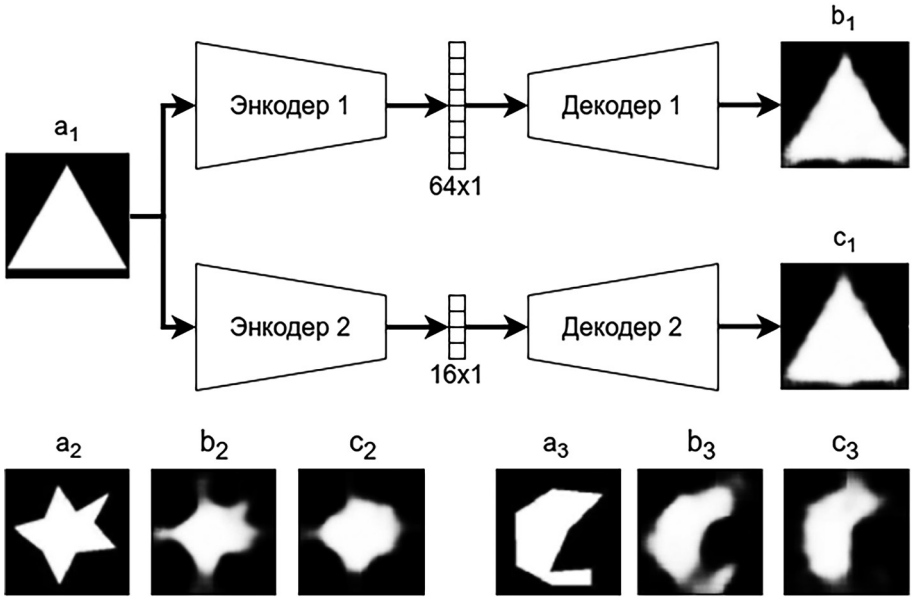
1) Метод Шайдука-Останина [6];
2) Метод быстрого преобразования Фурье (БПФ) в координатной сетке Декарта [7];
3) Метод вычисления коэффициента сжатия изображения [7], [8];
4) Нейросетевой метод с использованием вариационного автоэнкодера (ВАЭ) [7].

6.1. Метод Шайдука-Останина

Суть метода Шайдука-Останина заключается в вычислении энтропии спектральной плотности мощности (СПМ) центрированной сигнатуры радиуса, проведённого из центра тяжести контура изображения к каждой из точек этого контура. По сравнению с нашим методом (на основе инвариантов *Hu*) метод Шайдука-Останина обладает как преимуществами, так и недостатками. Оба метода дают оценку изображения (или участка изображения), инвариантную к смещению, к вращению и к масштабированию.

Оценивая сложность пятна по нашему методу, мы получаем оценку сложности целым числом от 1 до 8 (по числу инвариантов *Hu*), и поэтому метод оценки сложности, предложенный нами, требует дополнительного сравнения для пятен. В свою очередь, по методу Шайдука-Останина, оценка сложности является числом, лежащим в диапазоне от 0 до бесконечности, то есть для любого из двух пятен мы можем однозначно сказать, какое из них сложнее (тогда как по нашему методу мы можем распределить в одну из 8 категорий). Более “подробного” ранжирования сложности нашим методом можно добиться, используя значение $|k_i Hu_i|_{max}$ пятна, либо яркость пятна, либо его

[1] Величина E_{tot} является суммой яркостей всех пикселей пятна изображения, а величина S_{tot} – его полной площадью (в пикселях).



Здесь:

1. a_1, a_2, a_3 – оригинальные изображения (далее обозначенные как $S\_\{i\}$);
2. b_1, b_2, b_3 – результат трассировки через слой скрытого представления размером 64×1 (далее обозначенные как $R64\_\{i\}$);
3. c_1, c_2, c_3 – результат трассировки через слой скрытого представления размером 16×1 (далее обозначенные как $R16\_\{i\}$).
4. $C\_\{i\} = \Sigma|R64\_\{i\} - R16\_\{i\}| \;/\; \Sigma|S\_\{i\}|$ – численно рассчитанная сложность для исходного изображения $S\_\{i\}$:

$C\_\{1\} = 137.7/1661 = 0.083;$

$C\_\{2\} = 204.0/1064 = 0.192;$

$C\_\{3\} = 433.5/1293 = 0.335.$

Рис. 4. Схема архитектуры нейронной сети – вариационного автоэнкодера, пример трассировки тестового бинарного изображения и оценки его степени сложности.

площадь (что, конечно, является недостатком нашего метода).

К недостаткам метода Шайдука-Останина можно отнести оценку сложности пятна только по его внешней форме (только по контуру вокруг пятна). Таким образом, сложность, полученная методом Шайдука-Останина, никак не зависит от внутренней структуры пятна и распределения яркости в пятне – как в нашем методе.

Также к недостаткам метода Шайдука-Останина можно отнести вычислительную сложность. Для получения результата необходимо выполнить следующие действия:

1) Выделить пятна на общем фоне.
2) Найти контуры вокруг выделенных пятен.
3) Вычислить усреднённую сигнатуру радиуса.
4) Найти СПМ сигнатуры радиуса.
5) Вычислить модифицированную энтропию Шеннона [9].

Видим, что помимо более сложной предобработки (необходимо выделить контуры рассматриваемых объектов), метод Шайдука-Останина требует вычисления корней при нахождении радиусов, преобразования Фурье для нахождения СПМ и взятия логарифмов при вычислении энтропии. В свою очередь наш метод требует только более быстрых с точки зрения вычислений на ЭВМ операций сложения, вычитания, деления, умножения и сравнения. Таким образом, несмотря на меньший динамический диапазон оценки сложности изображения, предложенный нами метод обладает более низкой вычислительной сложностью, а потому может быть использован в системах реального времени. Более того, наш метод хорошо поддаётся распараллеливанию вычислений (на физические потоки), чего нельзя сказать о методе Шайдука-Останина, в котором необходимо вычислять контуры, что требует последовательных вычислений.

6.2. Метод БПФ в координатной сетке Декарта

Метод БПФ в декартовых координатах заключается в предположении, что присутствие в спектре интенсивных высокочастотных гармоник является показателем сложности формы, что в то

же время может указывать на присутствие шума на изображении. При отсутствии методов нахождения отличий воздействия шума от сложности формы в качестве меры может быть использовано некоторое среднее или интегральное значение мощности спектра амплитуд изображения. Присутствие на изображении относительно сложных элементов выражается в увеличении относительной интенсивности высокочастотных гармоник.

6.3. Метод вычисления коэффициента сжатия изображения

Метод сжатия заключается в расчете отношения байтового объема занимаемого пространства на информационном накопителе сжатого и несжатого (исходного) изображений при использовании алгоритма сжатия без потерь. В настоящей работе использовался python-модуль zlib [8] для сжатия изображений в представлении массива байтов.

6.4. Нейросетевой метод с использованием вариационного автоэнкодера (ВАЭ)

Нейронной сетью выступает предварительно обученный вариационный автоэнкодер (ВАЭ), предложенный в работе [7]. Нейронная сеть решает задачу воспроизведения на выходном слое изображения, поступившего на входной. Входное изображение рассматривается с позиции тензорного представления и трассируется через нейронную сеть, включающую в себя энкодер, слой скрытого представления и декодер. Тензор в энкодере последовательно сжимается по ширине и высоте слоями свёртки (Conv2d), активации (ReLU) и прореживания (MaxPooling2d), при этом увеличивается количество слоев (глубина) тензора. Слой скрытого представления (latent layer) является “бутылочным горлышком” системы, в нем сосредоточена сжатая информация о входном изображении. Декодер последовательно выполняет развертку данных. В настоящем (частном) случае выходное изображение, поступающее на дальнейшую обработку из декодера, согласовано по высоте и ширине со входным изображением. Размер скрытого слоя влияет на способность нейронной сети воспроизводить (реконструировать) поступающие на входной слой изображения. В экспериментах работы [7] размер скрытых слоев ограничен 16 и 64 нейронами для двух ВАЭ. Для оценки сложности используется сумма абсолютной попиксельной разницы между продуктами трассировки входного изображения через два ВАЭ с разным размером слоя скрытого представления. Сложность вычисляется как частное суммы абсолютных отклонений и интегральной яркости исходного изображения. Нужно отметить, что авторами работы [7] не рассматриваются альтернативные методы оценки схожести продуктов трассировки ветвей модели. На рис. 4 представлена схема архитектуры нейронной сети, состоящей из двух независимых ветвей ВАЭ, а также представлен пример трассировки некоторого бинарного изображения и расчета его сложности. Схема взята из оригинальной публикации [7].

6.5. Сравнение методов оценки сложности

Метод Шайдука-Останина базируется на инвариантной к вращению математической модели. Вращение влияет только на фазовый спектр сигнатуры контура. Спектр мощности не зависит от поворота контура относительно центра тяжести. Метод не анализирует внутреннюю структуру пятен. Следовательно, метод Шайдука-Останина не может рассматриваться для достоверного анализа пятен на предмет двоякой сложности: сложность окаймляющего контура и сложность внутреннего распределения интенсивности (структура пятна).

В настоящей работе были поставлены эксперименты, в которых сравнивались методы: БПФ, нейросетевой, сжатие и предлагаемый нами (на основе моментов Ху). Анализировалось изменение оценки сложности модели при добавлении преобразований сдвига, масштабирования и вращения, а также при добавлении синтетического шума к тестовым изображениям. Тестовые изображения приведены на рис. 5. Устойчивость модели к преобразованиям типа сдвиг, масштабирование и вращение при оценке сложности важна, так как одна и та же геометрическая форма может быть зарегистрирована на изображении с произвольным положением в плоскости кадра и с различным масштабом. Шум является трудно устранимым явлением, в особенности, при недостаточном значении светосилы оптической системы, низкой чувствительности матричного приемника излучения и при недостаточной

Рис. 5. Тестовые изображения, используемые для сравнения методов оценки сложности на предмет инвариантности к некоторым из аффинных преобразований и к аддитивному шуму: яблоко (слева), птица (в центре) и муха (справа).

ширине диапазона варьирования времени экспозиции.

Из группы преобразований, используемых при сравнении методов, наибольший интерес представляет именно устойчивость метода к вращению. Устойчивость к сдвигу обеспечивается выделением области интереса на изображении (ROI), которая получена с использованием алгоритма сегментации. Устойчивость к масштабу в нашем методе обеспечивается автоматически (инварианты Hu масштабно-инвариантны). В нейросетевом методе все изображения (пятна с объектами) масштабируются перед применением свёрточных слоёв, поэтому нейросетевой метод также гарантирует инвариантность к масштабу. Что касается метода БПФ и метода сжатия, то эти методы не содержат в явном виде инвариантности к масштабу, но её в них можно добавить – также масштабируя пятно с объектом к заданному размеру перед применением этих методов.

В первом эксперименте анализировалось поведение оценки сложности при добавлении шума на изображения. Сложность оценивалась и усреднялась для трех предварительно отобранных исходно бинарных (черно-белых) тестовых изображений при наложении синтетического шума. Выбранная модель шума – шум Гаусса. Среднеквадратичное отклонение (СКО) шума рассматривается из дискретного набора значений $[0,0.5,1,2,4,8,16,32,64]$(у.е.). Нулевое значение СКО было добавлено в ряд с целью последующих графических построений усредненной вычисленной сложности в зависимости от СКО шума из “нуля”. Глубина кодирования интенсивности (яркости) пикселя изображения равняется 8 битам, возможные значения находятся в диапазоне $BrigthnessRange = [0...255]$(у.е.). При добавлении реализации шума на изображение получаемое значение оценки в общем случае изменяется. Положительным качеством метода является способность стабильно удерживать числовую оценку.

Анализ результатов полученной оценки проводится не на одном изображении, а на статистической выборке. Объем статистической выборки $N_{\text{МК}} = 1000$. По этой выборке могут быть рассчитаны статистические моменты (например, среднее или СКО). Это используется для достижения репрезентативности результатов анализа. В рамках одной статистической выборки СКО шума неизменно. Для просмотра динамики оцениваемой сложности рассматриваются значения СКО шума из дискретного набора (показан выше). Выборка для конкретного значения СКО шума составляется как массив изображений, на которых добавлена реализация шума с текущим значением СКО. Далее в рамках полученной выборки каждым методом по порядку производится оценка сложности всех изображений. Полученные оценки усредняются. Результатом является значение средней сложности, полученной каждым методом для каждого значения СКО шума из набора.

Результаты математически представляются так, что средняя сложность являются функцией двух переменных: метода оценки и СКО шума. Каждый метод представлен отдельной кривой на графике, по оси абсцисс которого откладывается СКО шума, а по оси ординат – нормированная средняя сложность. Нормировка средней сложности позволяет уравнять порядки получаемых величин разными методами и оценивать относительную динамику изменения.

По определению считаем, что стабильность метода при воздействии шума SN(*StabilitytoNoise*) является описанной выше нормированной средней сложностью. Стабильность метода при воздействии шума зависит от СКО шума (аргумент функции), параметром выступает конкретный метод. Ниже представлена формула, по которой рассчитывается стабильность метода при воздействии шума.

$$SN_{id}(\sigma_{IN}) = \frac{M\left[C_{id}(\sigma_{IN})\right]}{C_{id}^{ref}}, \qquad (14)$$

где: SN – стабильность метода при воздействии шума; id – идентификатор метода; σ_{IN} – СКО шума; M – оператор вычисления среднего значения по статистической выборке случайной величины; $C_{id}(\sigma_{IN})$ – сложность, оцененная методом id при добавлении шума с СКО σ_{IN}; C_{id}^{ref} – сложность, оцененная методом id на исходном (оригинальном) изображении;

На рис. 6 представлено семейство графиков, показывающих зависимость стабильности метода от СКО шума для различных методов. На рис. 7 представлен пример одного из тестовых изображений (муха) с реализацией аддитивного шума при $CKO = 64$ (*у.е.*).

Во втором эксперименте отслеживалось изменение оценки сложности разными методами при вращении тестововых изображения (рис. 5). Вращение производилось без “подрезания” содержимого изображения. Под “относительной сложностью” $C^N(\alpha)$ при повороте оцениваемого объекта на некоторый угол α понимается отношение значения оцененной сложности при повороте на определенный угол $C^{rot}(\alpha)$ и значения

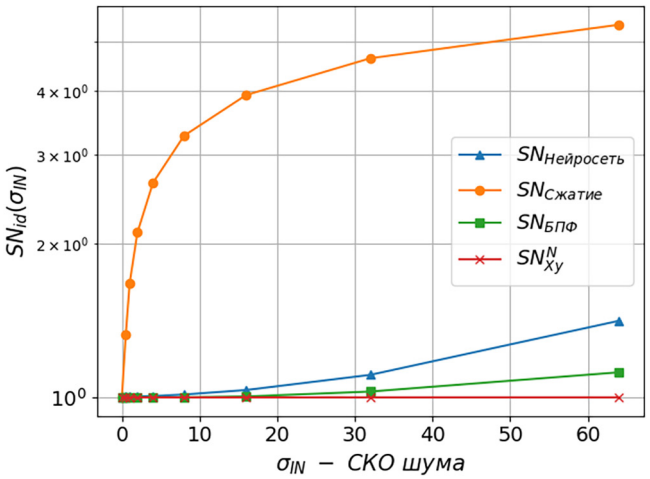


Рис. 6. Стабильность при воздействии шума, полученная разными методами.

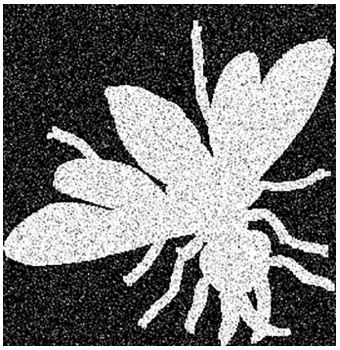


Рис. 7. Пример тестового изображения мухи с реализацией шума при СКО = 64(у. е.). Для бинарных изображений, у которых значения яркости равны либо 0, либо 255 (при 8-битной глубине кодирования) добавление шума с СКО = 64 не является фактором, в следствие которого исходные геометрические формы зрительно не могут быть распознаны.

сложности исходного изображения C^{ref}. Параметрами выступают идентификаторы метода *id* и тест-изображения I_i из рисунка 5.

$$C^{N}_{id,I_i}(\alpha) = \frac{C^{rot}_{id,I_i}(\alpha)}{C^{ref}_{id,I_i}} \qquad (15)$$

Далее объект выделялся по содержимому. Шаг поворота составлял $\delta\alpha = 1°$, при визуализации сглаживались высокочастотные колебания показаний одномерным усредняющим фильтром с размером ядра равным $\Delta\alpha = 5°$. Использовались три обозначенных ранее тестовых изображения рис. 5. На рис. 8 представлены три блока графиков, каждый блок соответствует своему тест-изображению из рис. 5. Графики показывают изменение относительной сложности для разных методов при вращении тест-объектов.

6.6. Выводы по сравнению методов

По результатам поставленного эксперимента метод сжатия показал наименьшую “стабильность” при воздействии шума, в то же время наш метод наиболее стабилен.

Изменчивость нормированной оценки сложности по нашему методу на каждом тестовом изображении минимальна в диапазоне вращения изображения от нуля до 360°.

Нейросетевой метод наиболее изменчив при вращении всех рассмотренных тест-изображений рис. 5.

Метод сжатия является наиболее сложным с вычислительной точки зрения и в исходном виде не имеет параметров для для настройки.

Из недостатков метода БПФ можно выделить отсутствие параметров для калибровки метода и потенциальная чувствительность к геометрическому шуму (коррелированному), в том числе к периодическому.

Методы сжатия, БПФ и наш метод обладают таким достоинством как “интерпретируемость”. В настоящем контексте под интерпретируемостью понимается возможность прямого обоснования полученных результатов, данное свойство

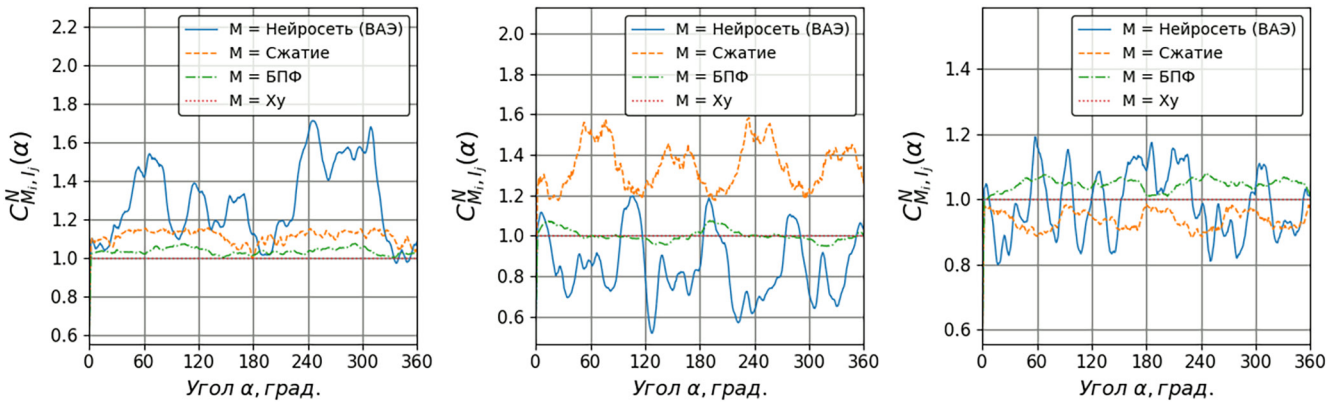


Рис. 8. Изменение относительной сложности для разных методов при вращении тест-объектов: яблоко (слева), птица (в центре) и муха (справа).

модели является особенно ценным в процессе отладки и калибровки под конкретную предметную область. Нейросетевой метод включает в себя блоки последовательной обработки (в том числе нелинейные операции), в которых, к сожалению, обоснование получаемых результатов только косвенное.

Нейросетевой (ВАЭ) метод показал среднюю стабильность оценки при воздействии шума, а также наибольшую изменчивость оценки к вращению на тестовом наборе данных. Кроме этого результаты работы нейросетевого метода зависят от тренировочной выборки, настроек обучения, параметров нейронной сети и способа сравнения изображений.

Среди потенциальных преимуществ нейросетевого метода может быть отмечена возможность получения высокой производительности при вычислении на графических ускорителях (видеокартах). Еще одним преимуществом нейросетевого метода можно считать возможность тренировки сети на базовую фильтрацию шумов. Вероятно, при более оптимальным подборе разрешения слоя скрытого представления или тренировочного набора данных, а также при использовании методов аугментации нейросетевым методом возможно удастся получить лучшие результаты.

Резюмируя, можно отметить, что рассмотренные методы сжатия, БПФ, нейросетевой метод и наш метод при наличии определенных преимуществ также обладают и недостатками.

7. ВЫВОДЫ

• Нами предложен новый метод оценки сложности геометрических фигур (пятен), учитывающий внутреннюю структуру пятен, а не только их внешний контур.

• Задача по вычислению степени сложности объектов разделена на составляющие: сегментация пятен + оценка сложности изолированных пятен.

• Предлагаемый метод обладает относительно низкой вычислительной сложностью по сравнению с вышеуказанными альтернативными методами.

• Нами проведен качественный и количественный анализ существующих (альтернативных) методов, выявлены их преимущества и недостатки по сравнению с нашим методом и друг с другом.

• Реализованный на основе нашего метода алгоритм апробирован как на искусственных, так и на реальных изображениях.

СПИСОК ЛИТЕРАТУРЫ

1. https://ru.wikipedia.org/wiki/WMAP
2. https://ru.wikipedia.org/wiki/Планк_(космическая_обсерватория)
3. *Hu M.K.* Visual pattern recognition by moment invariants. IRE Trans. Info. Theory, IT-8:179–187, 1962.
4. Alastair Florence Frederik Doerr. A micro-xrt image analysis and machine learning methodology for the characterisation of multi-particulate capsule formulations. International Journal of Pharmaceutics, 2, 2020.
5. *Kornilov A.S., Safonov I.V.* An overview of watershed algorithm implementations in open source libraries. Journal of Imaging, 4(10):123, 2018.
6. *Шайдук А.М., Останин С.А.* Количественная оценка сложности контура медицинских изображений // Журнал радиоэлектроники. 2013. № 2.
7. *Ritter H.,* Rothg¨anger M., Melnik A. Shape complexity estimation using vae. arXiv preprint arXiv:2304.02766., 2023.
8. Gilchrist J. Parallel data compression with bzip2. Proceedings of the 16th IASTED international conference on parallel and distributed computing and systems, 16:559–564, 2004.
9. Shannon C.E. A mathematical theory of communication. The Bell system technical journal, 27 (3):379–423, 1948.

ESTIMATING THE COMPLEXITY OF OBJECTS IN IMAGES

© 2024 V. B. Bokshanskiy[a], V. A. Kulin[a], G. S. Finiakin[a, b], A. S. Kharlamov[c], A. A. Shatskiy[a]

[a] *Bauman Moscow State Technical University, Moscow, Russia*
[b] *National Research University "Moscow Power Engineering Institute", Moscow, Russia*
[c] *Moscow State Technical University of Civil Aviation, Moscow, Russia*

A new method for estimating the complexity of geometric shapes (spots) is proposed, taking into account the internal structure of the spots, and not only their external contour. The task of calculating the degree of complexity of objects is divided into components: segmentation of spots and estimation of the complexity of isolated spots. The new method has a relatively low computational complexity compared to the alternative methods considered in the work. Using the new method, an algorithm based on parallel computing of the CUDA programming language for graphics accelerators (video cards) was created, which further increases the performance of our method. A qualitative and quantitative analysis of existing (alternative) methods has been carried out, their advantages and disadvantages in comparison with our method and with each other have been revealed. The algorithm implemented on the basis of the new method has been tested on both artificial and real digital images.

Keywords: Hu invariants, image complexity, image compression, image contour, image segmentation, contour selection, image classification, statistical moments of images, computational complexity

REFERENCES

1. https://ru.wikipedia.org/wiki/WMAP/
2. https://en.wikipedia.org/wiki/Planck_(spacecraft)
3. *Hu M.K.* Visual pattern recognition by moment invariants, IRE Trans. Info. Theory, 1962. V. IT-8, P. 179–187 (1962).
4. *Doerr F.J.S., Florence, A.J.* A micro-xrt image analysis and machine learning methodology for the characterisation of multi-particulate capsule formulations, Int. J. Pharm., 2020. V. 2.
5. *Kornilov A.S., Safonov I.V.* An overview of watershed algorithm implementations in open source libraries, J. Imaging, 2018. V. 4. No. 10. P. 123.
6. *Shaiduk A.M., Ostanin S.A.* Quantitative estimation of shape comlexity in medical images // Zh. Radioelektron. 2013. V. 2.
7. *Rothganger M., Melnik A., Ritter H.* Shape complexity estimation using VAE. ArXiv:2304.02766, 2023.
8. *Gilchrist J.* Parallel data compression with bzip2, Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems, 2004. V. 16. P. 559–564.
9. *Shannon C.E.* A mathematical theory of communication. Bell Syst. Techn. J., 1948. V 27, No. 3. P. 379–423.

ПРОГРАММИРОВАНИЕ, 2024, № 5, с. 42–53

КОМПЬЮТЕРНАЯ ГРАФИКА И ВИЗУАЛИЗАЦИЯ

УДК 004.93'1

УНИВЕРСАЛЬНЫЙ АЛГОРИТМ ДИСКРЕТИЗАЦИИ БИХРОМАТИЧЕСКИХ ДВУМЕРНЫХ ГРАФИЧЕСКИХ КОДОВ

© 2024 г. А. А. Трубицын[a, b, *], М. В. Шадрин[b, **], С. И. Холкин[c]

[a] *Рязанский государственный радиотехнический университет имени В.Ф. Уткина*
390005 Рязань, ул. Гагарина 59/1, Россия
[b] *ООО "Квантрон Групп"*
390010 Рязань, ул. Энгельса, д. 28а, помещ. Н2, Россия
[c] *ООО "Оператор-ЦРПТ"*
123376, Москва, ул Рочдельская, д. 15 стр. 16А, помещ. I, Россия,
**E-mail: assur@bk.ru*
***E-mail: m.shadrin@kvantron.com*

Поступила в редакцию: 26.01.2024 г.
После доработки: 16.04.2024 г.
Принята к публикации: 17.04.2024 г.

Представляются математические основы и алгоритмы распознавания бихроматических двумерных графических кодов вне зависимости от их вида (QR-коды, DataMatrix, GridMatrix и др.). Этапы достижения результата включают в себя обнаружение кода, локализацию его произвольным четырехугольником, трансформацию четырехугольника в канонический квадрат, построение сетки элементов (модулей) квадратного кода и заполнение ее последовательностью битов. Показано, что формулы преобразования перспективы позволяют трансформировать локализованные четырехугольные области в канонические квадраты с допустимым для дальнейшей обработки уровнем погрешности. Плоская сетка элементов квадратного кода формируется на основе поиска экстремумов производных от распределения интенсивности пикселей изображения квадрата по осям 0x и 0y. В алгоритме заполнения ячеек сетки (модулей кода) последовательностью 0 и 1 используется информация о средней интенсивности каждой такой ячейки. В завершение статьи проводится тестирование алгоритмов на множестве реальных изображений двумерных кодов, исследуются ограничения предложенных алгоритмов.

Ключевые слова: двумерный графический код, QR-код, DataMatrix, распознавание образов, преобразование Хафа, преобразование перспективы, численное дифференцирование, бинаризация

DOI: 10.31857/S0132347424050044, **EDN:** OLSFZO

1. ВВЕДЕНИЕ

По мере компьютеризации и цифровизации систем управления обществом возникла необходимость в новой форме представления информации – в форме компактных графических кодов.

Наибольшей информационной емкостью и устойчивыми перспективами применения обладают двумерные коды – DataMatrix и QR-коды.

Эти коды стали повсеместными и используются для внутреннего учета – на складах, в цехах и компаниях. Благодаря кодам возможно оптимизировать формирование заказов, контроль остатков и запасов и отслеживание всей цепочки поставок или производства. На производстве маркировка кодами позволяет повысить конкурентоспособность продукции за счет постоянного контроля качества продукции и организации обратной связи с клиентами. Система маркировки позволяет успешно бороться с контрафактной продукцией.

DataMatrix считаются кодами промышленного применения. Ключевой областью применения DataMatrix в России является маркировка товарных групп, подлежащих обязательному отслеживанию через систему "Честный ЗНАК". Сегодня к таким группам относятся лекарство, табачные изделия, молочные продукты, парфюм, фототехника, одежда, обувь и текстиль, автомобильные шины. Данный список постоянно расширяется за счет добавления новых групп.

Двумерные коды другой разновидности, так называемые QR-коды, относятся к кодам общего назначения и широко применяются в логистике, для хранения данных (ссылки на сайты, номера

телефонов или тексты), в сфере проведения платежей, в качестве ссылок на ресурс компьютерной сети и т. д.

Стандартные DataMatrix и QR-коды, считающиеся весьма удобными в использовании, на самом деле представлены несколькими версиями, обеспечивающими запись либо большей, либо сокращенной по объему информации с целью оптимизации расходов на запись/считывание данных в каждом конкретном случае.

Список кодов не ограничивается лишь DataMatrix и QR-кодами, в частности при предоставлении транспортных услуг выбор компаний сделан в пользу Aztec Code, который недавно рекомендован международной ассоциацией воздушного транспорта для трансфера электронных билетов (стандарт BCBP IATA). Другой пример – двумерный графический код GridMatrix специально разработан для кодирования китайских иероглифов. Список существующих на настоящий момент кодов можно продолжить.

Следует также учесть, что вполне вероятно появление новых типов двумерных кодов, обеспечивающих оптимальные соотношения надежности считывания и объема информации, более высокий уровень ее защищенности и др.

В условиях множественности двумерных кодов и вероятности появления новых возникает проблема разработки универсального сканера. Очевидно, что острые углы этой проблемы могут быть сглажены построением алгоритма считывания закодированной информации, мало зависящего от типа кода и его размера. Причем алгоритм должен работать в жестких условиях съемки, т. е. допускать различные углы и масштаб записи графической информации в условиях естественной освещенности и реальных искажений. Техническая часть проблемы в таком случае не является острой и разрешается разработкой сканеров, являющихся аналогами уже существующих устройств – 2D код-ридеров. В идеале результатом работы предлагаемого к рассмотрению универсального алгоритма должна стать последовательность битовых 0 и 1, соответствующих светлым и темным модулям двумерного кода, по которой уже на заключительном этапе операции обработки легко определяется тип кода, его версия и, наконец, вся зашифрованная информация.

Целью настоящих исследований является разработка универсального алгоритма, превращающего в последовательность 0–1 записанные в реальных условиях двумерные коды, версии которых имеют квадратную форму.

В статье рассматриваются черно-белые двумерные коды, зарегистрированные в оттенках серого.

Прежде чем перейти к изложению основного материала подчеркнем, что в литературе представлено множество методов и алгоритмов обработки (дискретизации и подготовки к декодированию) двумерных кодов, зарегистрированных в реальных условиях съемки. Подавляющее большинство описанных алгоритмов укладываются в одну схему: 1 – определение вида кода, 2 – дискретизация на модули, 3 – бинаризация модулей и декодирование. В нашей работе продвигается альтернативная схема, создающая предпосылки к разработке универсального алгоритма: 1 – превращение любого двумерного кода в канонический квадрат, 2 – высоконадежная дискретизация его на модули, 3 – бинаризация модулей, 4 – установление типа и версии кода по битовой последовательности с последующим декодированием.

Логика всего цикла распознавания требует, чтобы на начальной стадии объект был обнаружен и изолирован от фона. В таком случае, полный цикл обработки двумерного кода будет состоять из последовательности следующих процедур: обнаружение кода, его локализация, дискретизация на модули и декодирование. Первые две процедуры – обнаружение и локализация кодов – подробно исследованы в авторских работах [1, 2], поэтому нет большого смысла на них здесь останавливаться, можно лишь отметить, что предложенная в [1] функция осцилляций яркости служит надежному обнаружению кодов на изображении, а преобразование Хафа оказалось эффективным инструментом решения задачи локализации четырехугольных кодов.

Важная особенность предлагаемых в рамках данной работы методов достижения поставленной цели заключается в использовании строгого математического аппарата без применения эвристических алгоритмов, страдающих сильной неустойчивостью по отношению к вариациям входных данных и особенно к изменению класса этих данных.

2. ДИСКРЕТИЗАЦИЯ КОДА

Здесь мы приступаем к описанию предлагаемого метода дискретизации двумерного графического кода произвольной четырехугольной формы, предварительно локализованного на плоскости изображения.

Задача дискретизация кода на черные и белые ячейки (структурные единицы, модули) не

является до конца решенной, поскольку представленные в разное время в литературе алгоритмы имеют серьезные недостатки и, главное, не могут рассматриваться в качестве универсальных.

В работе [3] предложен алгоритм считывания информации кодов Data Matrix, в котором после локализации кода, в общем случае повернутого на произвольный угол по отношению к горизонтальному направлению, по сканированию шаблона синхронизации (Timing Pattern), расположенного на границе кода, определяется количество строк и столбцов кода и их ширина. Пересечения этих строк и столбцов образуют темные и светлые квадратные ячейки (модули) кода, в общем случае повернутые относительно канонического горизонтального направления. Темные и светлые модули затем трансформируются в бинарную матрицу. Недостаток данного способа дискретизации кода на модули состоит в необходимости строго фронтальной съемки кода, сложность и невысокая точность алгоритма при установлении наклонных границ переходов светлых и темных модулей шаблона синхронизации (Timing Pattern), чувствительность его к шуму, а также применимость лишь для Data Matrix.

В запатентованном способе [4] определение размеров ячеек (модулей) двумерного кода производится на основе анализа распределения длин белых и черных последовательностей пикселей при сканировании изображения кода вверх, вниз, влево и вправо, начиная от некоторой точки внутри области двумерного кода. Минимальная длина последовательности, встречающаяся с частотой не ниже предопределенного уровня, выбирается в качестве ширины и высоты квадратной ячейки (модуля), по которой восстанавливается вся сетка модулей локализованного кода. Недостаток данного способа заключается в сложности и низкой надежности алгоритма определения размеров ячейки (модуля) кода, вследствие необходимости оценки уровня частоты в распределении длин черных и белых последовательностей, ниже которого результаты не должны приниматься в расчет; ввиду обязательного исключения черных и белых последовательностей существенно малой длины, частота появления которых будет всегда высока при наличии шума, а также при расфокусировке и смазывании изображения. Следует отметить, что в данном способе предполагается канонический квадратный вид двумерных графических кодов как объектов обработки, который в общем случае невозможно обеспечить в реальных условиях съемки.

В работе [5] предлагается алгоритм, в котором ячейки двумерного кода извлекаются за четыре шага. На первом шаге из анализа распределения градиента интенсивности изображения локализованного, произвольно повернутого кода строится гистограмма ориентации ребер, и выбираются две доминирующие ориентации, расположенные примерно на величину 90° друг от друга, и поэтому соответствующие границам резких переходов интенсивности между строками (одна группа ребер) и столбцами (перпендикулярная им группа ребер) двумерного кода. На втором шаге для пикселов, образующих обнаруженные ребра, выполняется преобразование Хафа. Нахождение максимумов преобразования Хафа позволяет математически описать прямые, соответствующие границам строк и столбцов двумерного кода, на третьем шаге алгоритма. На четвертом шаге определяются координаты центрального пикселя каждой ячейки (модуля) кода как координаты минимума по осям x и y между двумя последовательными максимумами по осям x и y соответственно. Недостатком способа является математическая сложность четырех-шагового алгоритма, лежащего в основе способа, и, как следствие, низкая надежность получения результатов в общем случае. Главное ограничение алгоритма заключается в искажении направлений ребер при больших углах съемки и, как следствие, невозможность определения границ строк и столбцов по преобразованию Хафа.

Авторами статьи [6] представляется способ распознавания QR-кода, в котором на основе горизонтального и вертикального сканирования бинаризованного изображения обнаруженного и изолированного QR-кода определяются три опознавательные метки (шаблоны Finder Patterns). После определения ориентации QR-кода по анализу взаимного расположения опознавательных меток, находится ширина ячейки (модуля) простым делением ширины одного из трех шаблонов на 7 (7×7 – число модулей в опознавательной метке) и уточняется версия кода. Затем по формулам преобразования перспективы [7] код приводится к каноническому квадратному виду и производится дискретизация его на квадратные ячейки (модули) в соответствии с найденным значением их ширины. Недостатками предложенного способа являются его узкая специализация для расшифровки только QR-кодов, допустимые лишь небольшие отклонения съемки по углу от фронтальной, низкая точность определения ширины ячейки, и как следствие малая надежность результатов дискретизации кода на модули.

В предлагаемом авторами данной статьи алгоритме, претендующем на роль универсального для двумерных кодов произвольного вида, выделяются две процедуры – трансформация кода в канонический квадрат и дискретизация кода на модули по анализу максимумов средних производных от интенсивности по декартовым осям.

2.1. Трансформация кода в канонический квадрат

При решении задачи трансформации локализованного четырехугольника (1′2′3′4′) в квадрат (1234) (см. рис. 1) должны быть определены функции преобразования f и g, в общем записываемые как

$$\begin{aligned} x' &= f(x, y, x', y'), \\ y' &= g(x, y, x', y'). \end{aligned} \qquad (1)$$

В рамках настоящих исследований были протестированы два способа трансформации (1): 1) с использованием простых формул, предложенных (без всякого, кстати, обоснования) в работе [8], 2) с использованием известных формул преобразования перспективы [7].

Заметим, что иллюстрирующий графический материал (рис. 1) подходит для описания обоих способов. Пусть функции f и g (1) допускают свое определение по известным координатам вершин четырехугольника и квадрата. С целью минимизации яркостных искажений и исключения пустых пикселей на изображениях трансформация одной фигуры в другую производится в форме обратного преобразования, т. е. квадрата (1234) в четырехугольник (1′2′3′4′) (рисунок 1а). В таком случае, после определения конкретного вида функций f и g (1) при построении изображения квадрата целочисленные координаты его пикселей пробегают весь выбранный диапазон по x и y (рисунок 1б), а полученные по формулам преобразований (1) (нецелочисленные) координаты x' и y' позволяют вычислить интенсивность в точке (x', y') по формулам двумерной интерполяции, например, билинейной, с использованием значений интенсивности ближайших четырех пикселей (i, j), $(i+1, j)$, $(i, j+1)$ и $(i+1, j+1)$ исходного изображения. Вычисленная таким образом интенсивность точки (x', y') транслируется в интенсивность пиксела (x, y) на изображении квадрата (см. рисунок 1б).

Способ 1.

Формулы преобразования (1), предложенные в [8], имеют вид

$$\begin{aligned} x' &= c_1 x + c_2 y + c_3 xy + c_4, \\ y' &= c_5 x + c_6 y + c_7 xy + c_8, \end{aligned} \qquad (2)$$

где $c_1 - c_6$ – неизвестные коэффициенты, которые необходимо найти для определения функций преобразования f и g (1).

Формулы преобразования (2) каждой из четырех вершин записываются как

$$\begin{aligned} x'_k &= c_1 x_k + c_2 y_k + c_3 x_k y_k + c_4, \\ y'_k &= c_5 x_k + c_6 y_k + c_7 x_k y_k + c_8, \end{aligned}$$

где $k = 1, 2, 3, 4$.

Поскольку полное число уравнений для четырех преобразуемых вершин равно восьми, то получаем замкнутую систему уравнений относительно восьми неизвестных коэффициентов c_i, $i = 1, 2 \ldots 8$. В этом способе трансформации решение системы удается получить в виде последовательности достаточно простых выражений

$$c_1 = \frac{c_{15}^{+} + c_{15}^{-}}{2}, \quad c_5 = \frac{c_{15}^{+} - c_{15}^{-}}{2},$$

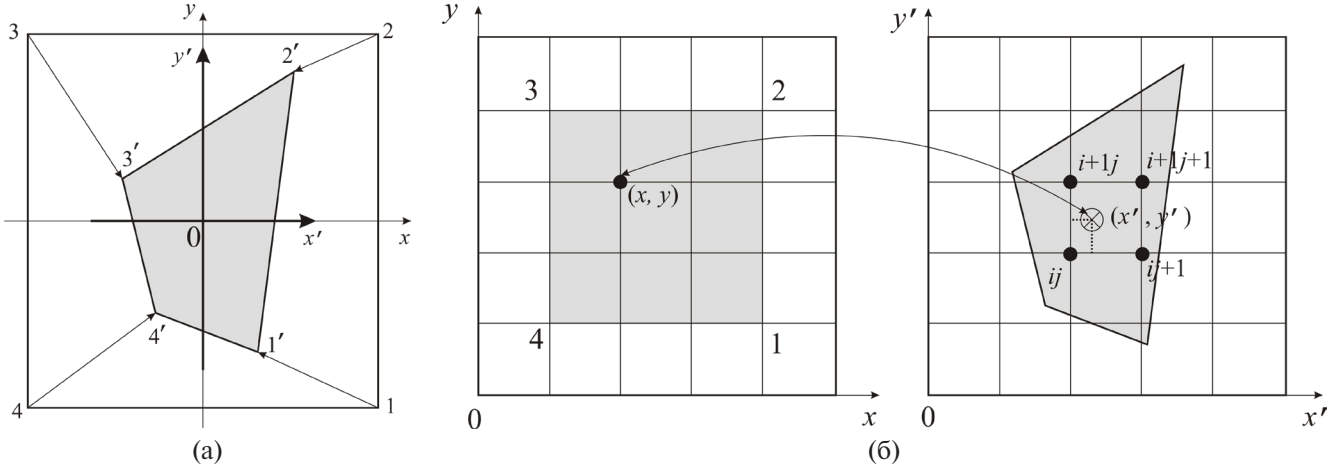


Рис. 1. Преобразование четырехугольника (1′2′3″) в квадрат (1234): (а) схема обратного преобразования, (б) схема вычисления интенсивности пикселей.

$$c_2 = \frac{c_{26}^+ + c_{26}^-}{2}, c_6 = \frac{c_{26}^+ - c_{26}^-}{2},$$

$$c_3 = \frac{c_{37}^+ + c_{37}^-}{2}, c_7 = \frac{c_{37}^+ - c_{37}^-}{2},$$

$$c_4 = \frac{c_{48}^+ + c_{48}^-}{2}, c_8 = \frac{c_{48}^+ - c_{48}^-}{2},$$

$$c_{15}^\pm = \frac{b_1^\pm a_{11} - b_2^\pm a_{01}}{a_{00}a_{11} - a_{10}a_{01}}, c_{26}^\pm = \frac{b_1^\pm - a_{10}c_{15}^\pm}{a_{01}},$$

$$c_{37}^\pm = \frac{s_1^\pm - s_2^\pm - (x_1 - 2)c_{15}^\pm - (y_1 - y_2)c_{26}^\pm}{g_{01}},$$

$$c_{48}^\pm = s_1^\pm - c_{15}^\pm x_1 - c_{26}^\pm y_1 - c_{37}^\pm x_1 y_1,$$

$$b_k^\pm = \left(s_k^\pm - s_{k+1}^\pm\right) g_{k+1,k+2} - \left(s_{k+1}^\pm - s_{k+2}^\pm\right) g_{k,k+1},$$

$$k = 1,2,$$

$$a_{00} = (x_1 - x_2)g_{23} - (x_2 - x_3)g_{12},$$

$$a_{01} = (y_1 - y_2)g_{23} - (y_2 - y_3)g_{12},$$

$$a_{10} = (x_2 - x_3)g_{34} - (x_3 - x_4)g_{23},$$

$$a_{11} = (y_2 - y_3)g_{34} - (y_3 - y_4)g_{23},$$

$$g_{i,\,i+1} = x_i y_i - x_{i+1} y_{i+1},\ i = 1,2,3,$$

$$s_k^\pm = x_k' \pm y_k',$$

$$k = 1,2,3,4.$$

Восстановление тестовых квадратных изображений (в статье не приводятся) по формулам преобразования (2), полученных при съемке под разными углами, показало практическое отсутствие искажений в диапазоне углов съемки от 0° до 10° по отношению к нормали. Еще большее увеличение угла съемки приводит к недопустимым искажениям. Т.е. предложенное в [8] преобразование (2) годится только для случаев строго фронтальной съемки.

На рис. 2 представлен пример трансформации локализованного QR-кода (рис. 2а), расположенного на верхней грани куба, угол съемки которого составляет около 45°, в канонический квадрат (рис. 2б). Обращает на себя внимание искажение размеров и вертикально-горизонтальных пропорций всех трех меток обнаружения. Отношение линейных размеров самой большой метки (левый нижний угол) к размерам самой маленькой (правый верхний угол) составляет примерно 7:5 и является количественной характеристикой уровня искажений при трансформации (2).

Способ 2.

Строго обоснованные математические формулы преобразования перспективы [7] имеют вид

$$\begin{aligned} x' &= a_{11}x + a_{21}y + a_{31} - a_{13}xx' - a_{23}yx', \\ y' &= a_{12}x + a_{22}y + a_{32} - a_{13}xy' - a_{23}yy'. \end{aligned} \quad (3)$$

Записывая эту пару уравнений для четырех соответствующих вершин произвольного четырехугольника и квадрата, получим систему восьми линейных алгебраических уравнений относительно неизвестных коэффициентов a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, a_{32}, a_{13} и a_{31}, которую удобно представить в матрично-векторной форме:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{13} \\ a_{23} \end{bmatrix} = \begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \\ y_1' \\ y_2' \\ y_3' \\ y_4' \end{bmatrix}.$$

(а) (б) (в)

Рис. 2. Трансформация QR-кода (а), предварительно локализованного четырехугольником (1234) на верхней грани куба по формулам: (б) – (2), (в) – (3).

Решение данной системы уравнений методом Гаусса с частичным выбором главного элемента и последующее восстановление по формулам преобразования перспективы (3) тестовых квадратных изображений (в статье не приводятся), продемонстрировало практическое отсутствие яркостных и геометрических искажений в диапазоне углов съемки вплоть до 60°.

На рис. 2в представлен пример трансформации локализованного QR-кода, расположенного на верхней грани куба (рис. 2а). Сравнение качества способов трансформаций (2) и (3) делается в пользу второго, поскольку оно сохраняет исходное соотношение линейных размеров структурных элементов кода.

Таким образом, второй способ трансформации четырехугольных областей искажает их при обработке наименьшим образом и поэтому является предпочтительным при решении поставленной проблемы.

2.2. Построение сетки

После трансформации двумерного кода строки и столбцы его модулей будут ориентированы почти горизонтально, а столбцы – почти вертикально. Некоторые отклонения в ориентации от строгих вертикального и горизонтального направлений могут быть связаны с погрешностью локализации кода. Канонический квадратный вид двумерных кодов позволяет легко найти границы раздела между его структурными единицами – чередующимися темными и светлыми модулями, поскольку на этих границах будут наблюдаться скачки производных интенсивности в направлениях обеих декартовых осей. В обнаружении этих скачков и заключается суть предлагаемого алгоритма, позволяющая применять его для обработки двумерных кодов произвольных типов. Для анализа величины и положения скачков удобнее использовать абсолютные значения частных производных. Поскольку не каждая пара соседних модулей характеризуется изменением интенсивности, то для поиска границ модулей следует анализировать средние абсолютные значения производных по строкам и столбцам. Усреднение производных производится их суммированием, которое, ко всему прочему, снижает уровень случайных гауссовских шумов на исходных изображениях. Вполне очевидно, что найденные основные максимумы средних производных будут соответствовать границам раздела модулей кода.

На рис. 3 представлен пример трансформации локализованного четырехугольного кода (рис. 3а) в канонический квадрат (рис. 3б) с последующей дискретизацией его на структурные единицы.

На рис. 3в показан средний модуль частной производной по y; средний модуль частной производной по x имеет похожий вид и поэтому не демонстрируется. Практика применения алгоритма позволила повысить надежность процедуры поиска максимумов функции вида рис. 3в за счет предварительного вычитания ее кусочно-линейного фона, проходящего через минимумы. После вычитания фона удается однозначно определить положительный уровень функции, выше которого располагаются все искомые максимумы (обозначенные маленькими кружками) (рис. 3г). В соответствии с координатами максимумов на плоскости рисунка квадратного кода располагаются горизонтальные и вертикальные линии сетки (номера максимумов совпадают с номерами линий), т. е. производится его дискретизация на $n \times n$ модулей (рис. 3д).

Стоит заметить, что частные производные $I_y(x, y) = \dfrac{\partial I}{\partial y}$ от интенсивности $I(x, y)$ каждого столбца с номером x цифрового изображения размером $L \times L$ могут быть найдены во всех L пикселах столбца по формулам численного дифференцирования [9]:

$$I_y(x, y) = I(x, y-2) - 8I(x, y-1) + 8I(x, y+1) - I(x, y+2), y = 3,4 \ldots L-2,$$

причем для производных в крайних пикселях слева $y = 1$, $y = 2$ и справа $y = L-1$, $y = L$ можно использовать формулы с эксклюзивными весовыми коэффициентами [9] или приравнять их значениям производных в узлах $x = 3$ и $x = L-2$ соответственно.

Среднее значение модуля производной определяется с помощью выражения

$$p(y) = \overline{\left|I_y(x, y = \text{const})\right|} = \frac{1}{L}\sum_{x=1}^{L}\left|I_y(x, y)\right|, \quad y = 1,2 \ldots L.$$

Способ нахождения средних частных производных по x аналогичен выше описанному.

3. БИНАРИЗАЦИЯ МОДУЛЕЙ

Следующий этап обработки двумерного кода заключается в бинаризации его $n \times n$ модулей. Во многих случаях здесь могут использованы стандартные методы пороговой бинаризации изображений [10], причем в нашем случае за интенсивность пиксела изображения размером

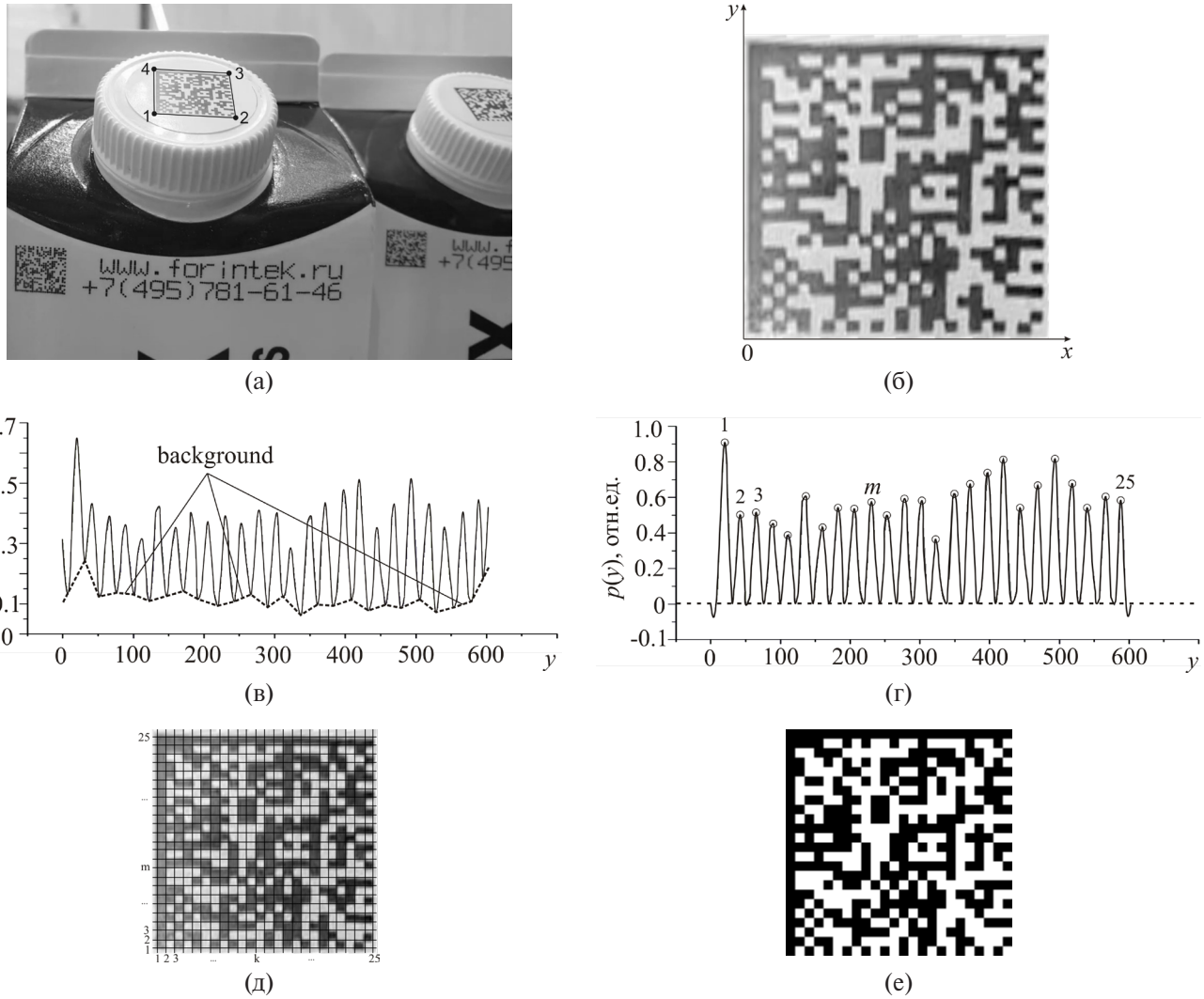


Рис. 3. Трансформация и дискретизация DataMatrix: (а) – исходный для обработки локализованный код, (б) – результат трансформации четырехугольника (1234) в канонический квадрат, (в) – средний модуль производной по y интенсивности квадратного кода и кусочно-линейный фон (background), (г) – средний модуль производной после вычитания кусочно-линейного фона, (д) – наложение сетки с $n \times n = 24 \times 24$ ячейками, (е) – код с бинарными модулями.

$n \times n$ принимается среднее значение интенсивности $\bar{I}_{ij}$ ячейки сетки (модуля). Еще лучшие результаты показывают методы локальной бинаризации [11]. Вполне возможна в перспективе разработка эффективных методов бинаризации модулей кода при использовании дополнительной информации о распределении интенсивности внутри каждого из них, а не только о ее средней величине.

На рис. 3е показан результат бинаризации ячеек сетки, изображенной на рис. 3д. Таким образом, исходный код DataMatrix на рис. 3а получает представление в виде матрицы

$$d_{ij} = \begin{cases} 0, \bar{I}_{ij} \geq I_t, \\ 1, \bar{I}_{ij} < I_t, \end{cases}$$

где $i, j = 1, 2, \ldots n$; n – размер кода; I_t – пороговое значение интенсивности. С целью визуального контроля качества распознавания исходного кода данная матрица закрашивается белыми ($d_{ij} = 0$) и черными ($d_{ij} = 1$) небольшими квадратами (рис. 3е). Однако еще раз подчеркнем важный момент – на выходе предложенных алгоритмов дискретизации графических кодов получаем двумерную битовую последовательность 0 и 1.

Рис. 4 демонстрирует универсальность предложенных алгоритмов. Тестирование описанной методики на множестве различных кодов показала ее эффективность при восстановлении исходной битовой последовательности 0 и 1 без предопределения типа кода.

Отсутствие необходимости определения (предопределения) типа кода на рассмотренных выше

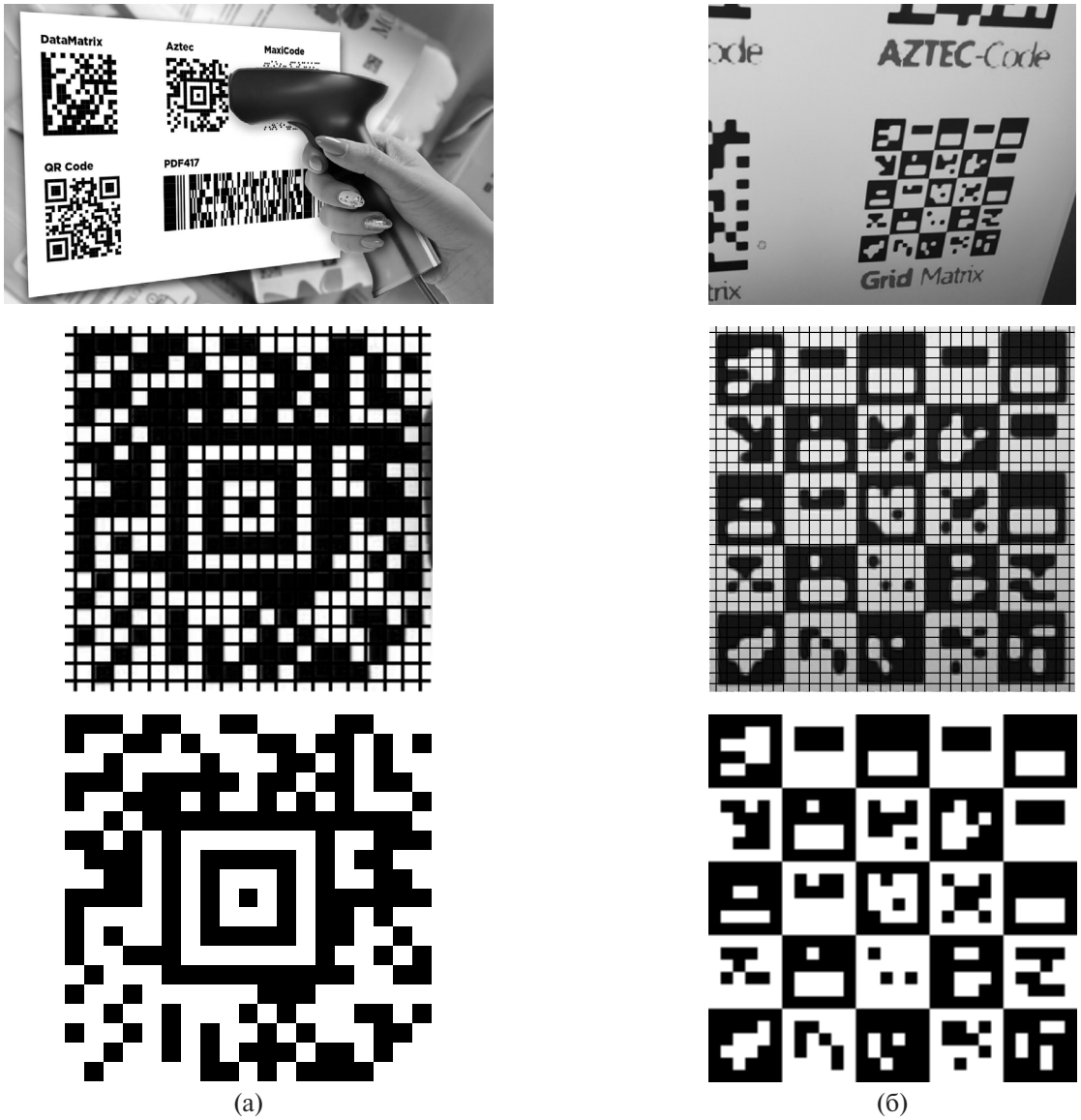


Рис. 4. Обработка двумерных кодов: (а) – Aztec, (б) – GridMatrix. Верхний ряд 1 – исходные изображения, средний ряд 2 – результат обнаружения, локализации, трансформации в канонический квадрат и дискретизации на модули двумерного кода, нижний ряд 3 – результат бинаризации модулей кода (двумерная последовательность 0 и 1).

стадиях обработки реально облегчает весь процесс распознавания, поскольку установление типа по графической информации не может иметь общего, стандартного решения вследствие разнообразия кодов и произвольности условий съемки. В то же время тип кода и его версия могут быть элементарно и быстро идентифицированы по двумерной последовательности 0 и 1 с учетом специфики меток, сопровождающих каждый код.

Например, DataMatrix сопровождается L-образной опознавательной меткой, поэтому сумма элементов первой или последней строки, а также первого или последнего столбца будет равна размеру кода n, т. е.

$$\text{или } \sum_{i=1}^{n} d_{i1} = n \text{ или } \sum_{i=1}^{n} d_{in} = n,$$

$$\text{или } \sum_{j=1}^{n} d_{1j} = n, \text{ или } \sum_{j=1}^{n} d_{nj} = n.$$

причем элементы противоположной строки и столбца в этом случае есть чередующиеся 0 и 1 (метки синхронизации). Более того, тестирование алгоритмов показало, что строгие равенства в этих условиях могут быть заменены на $\geq (n - k)$, где k – небольшое число, а также могут допускаться нарушения строгого чередования 0 и 1. В таком случае при определенных потерях

информации (при обработке некачественных изображений) тип кода все равно определяется однозначно.

QR-код отличается тем, что в трех его углах располагаются черно-белые квадратные опознавательные метки размером 7×7, первая и седьмая строки которой, а также первый и седьмой столбцы есть последовательность семи битовых 1. Имеются другие очевидные особенности этих меток, которые могут быть использованы при идентификации. Практика тестирования предложенных алгоритмов также показала допустимость небольших отклонений содержания меток обрабатываемого QR-кода от установленного идеального образца.

Вывод, который может быть здесь сделан, заключается в том, что предлагаемый метод распознавания допускает не строгое соответствие между метками стандартного образца двумерного кода и объекта обработки. Такой вывод оказывается вполне логичным, но в некоторой степени неожиданным.

Версия кода определяется аналогичным образом по специфическим меткам.

После определения типа двумерного кода сменой строк и столбцов битовой матрицы d_{ij} может быть обеспечена его стандартная каноническая ориентация.

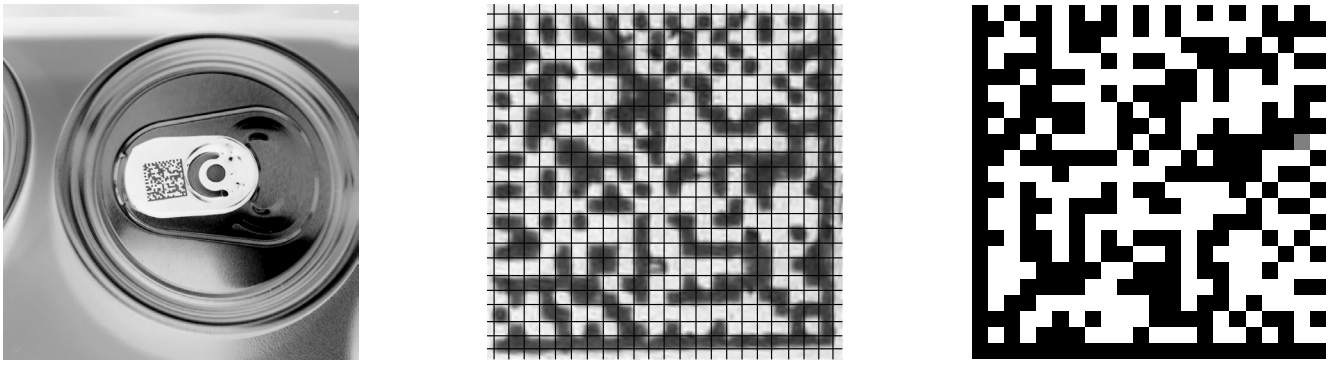
После определения типа и версии кода, смены его ориентации на каноническую можно запускать процедуру его дешифрования, которая не является сложной и регламентирована стандартами.

Для отладки предложенных алгоритмов и целей практического распознавания двумерных графических кодов разработано (C++ Qt) программное приложение Quantron CodeReader pro. В приложение включены дополнительные опции адаптивной медианной фильтрации [12], фильтрации гауссовских шумов и морфологической обработки изображений. В качестве финальной демонстрации достоинств предложенных алгоритмов на рис. 5 представляются результаты полного цикла распознавания изображения кода DataMatrix плохого качества, которое не удается дешифровать сканером смартфона Redmi Note 12.

На рис. 5а показано исходное изображение кода, на рис. 5б – код после обнаружения и локализации на основе оригинальных авторских методов [1], трансформации в канонический квадрат, дискретизации на модули, и на рис. 5в – код после бинаризации модулей и переориентации. В предпоследней колонке модулей на рис. 5в обнаруживается модуль серого цвета как индикатор того, что используемый в программе алгоритм бинаризации не смог однозначно идентифицировать 0 или 1 в соответствующей ячейке сетки (второй ряд сверху) на рис. 5б. Однако такой сбой не может оказать влияние на окончательный результат дешифрования, поскольку двумерные коды содержат в себе дополнительную информацию – коды Рида-Соломона коррекции ошибок, позволяющие восстанавливать до 30% потерянных данных. Результатом распознавания двумерной двоичной последовательности на рис. 5в является строка символов 010460372577000321 5ABIEM: Y7DR? Y{GS}93QoBp.

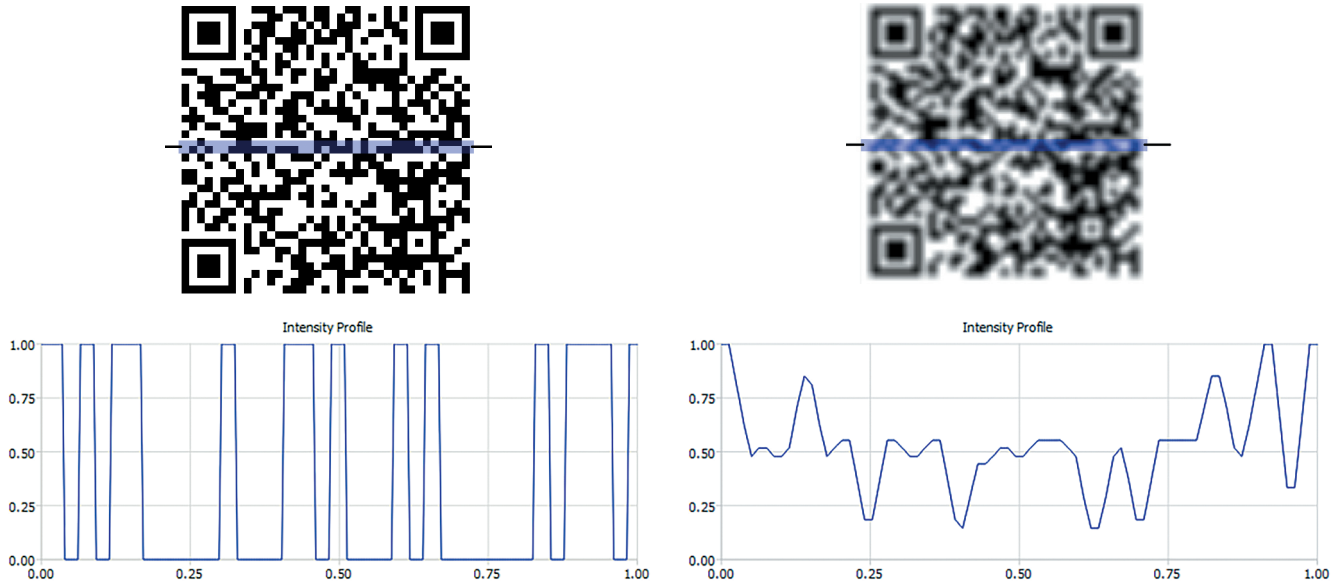
4. ТЕСТИРОВАНИЕ МЕТОДА

Вследствие отсутствия специальных тестовых примеров для оценки эффективности методов распознавания двумерных кодов в нашем эксперименте использовались изображения Data Matrix и QR-кодов из открытого доступа в интернете, размер которых варьировался от 0.3 Мп



(а) (б) (в)

Рис. 5. Результаты полного цикла распознавания DataMatrix низкого качества: (а) – исходное изображение кода на жестяной банке, (б) – код после обнаружения и локализации, трансформации и дискретизации, (в) – бинаризованный канонически ориентированный код.



(а) (б)

Рис. 6. Эффект расфокусировки QR-кода: (а) – идеальный код и профиль интенсивности выделенной строки ($\sigma = 0.49$), (б) – расфокусированный код и профиль интенсивности выделенной строки ($\sigma = 0.20$).

до 5 Мп, а минимальный размер модуля превышал 4 пикселя. Распознавание кода принималось успешным, если однозначно определялся его тип и отсутствовали визуально контролируемые значительные искажения структуры.

По результатам тестирования сделаны следующие выводы относительно эффективности предлагаемого алгоритма. На результативность распознавания оказывают три фактора: точность локализации кода, степень расфокусировки (размазывания) изображения кода и корректность бинаризации модулей после построения сетки.

Погрешность локализации кода влияет на точность построения сетки дискретизации и определяется погрешностью определения каждой из вершин четырехугольника, которая не может превышать половины размера модуля кода.

Расфокусировка и размазывание изображения кода снижает максимумы производных и поэтому затрудняет определение положений линий сетки дискретизации. Падение качества двумерного кода при расфокусировке продемонстрировано на рис. 6. Среднеквадратическое отклонение σ интенсивности каждой строки (столбца) идеального черно-белого Data Matrix или QR-кода заключено в пределах от 0.4 до 0.5 в формате записи яркости пикселей байтовыми числами от 0 до 1 (рис. 6а). При расфокусировке изображения σ уменьшается (рис. 6б). Исследования показали, что граница надежного построения сетки модулей находится выше значения среднеквадратического отклонения, приблизительно равного 0.2–0.3.

И наконец, корректность бинаризации модулей кода определяется конкретным используемым алгоритмом, поэтому здесь можно лишь заметить то, что его функциональные возможности могут существенным образом снизить требования к точности локализации кода и степени расфокусировки исходного изображения.

4. ЗАКЛЮЧЕНИЕ

Предложенная в работе последовательность процедур трансформации локализованного двумерного кода в канонический квадрат по формулам преобразования перспективы и дискретизации его на модули на основе анализа максимумов производных интенсивности строк и столбцов изображения является универсальным алгоритмом дешифрования всех известных версий двумерных бихроматических кодов квадратной формы.

Ограничения предложенного алгоритма определяются погрешностью локализации кода, степенью расфокусировки (размазывания) изображения кода и корректностью бинаризации модулей после построения сетки дискретизации.

СПИСОК ЛИТЕРАТУРЫ

1. *Trubitsyn A.A., Shadrin M.V., Serezhin A.A.* Localization of image fragments with high frequency intensity oscillation. Journal of Autonomous Intelligence. 2023. V. 6. № 2. P. 1–16.

2. *Трубицын А.А., Шадрин М.В.* Обнаружение, локализация и трансформация двумерного графического кода. ГрафиКон 2023: 33-я Международная конференция по компьютерной графике и машинному зрению, 19–21 сентября 2023 г., Институт проблем управления им. В.А. Трапезникова Российской академии наук, г. Москва, Россия. 2023. С. 509–516.
3. *Karrach L., Pivarciova E.* Options to use data matrix codes in production engineering. Management Systems in Production Engineering. 2018. V. 26. № 4. P. 231–236.
4. *Yamaguchi et al.* Code type determining method and code boundary detecting method. US Patent 2005/O121520 A1. 09.06.2005.
5. *Szentandrási I., Herout A., Dubská M.* Fast detection and recognition of QR codes in high-resolution images. SCCG '12: Proceedings of the 28th Spring Conference on Computer Graphics March. 2013. P. 129–136.
6. *Lin J.A., Fuh C.S.* 2D barcode image decoding. Mathematical Problems in Engineering. 2013. 2013: 848276.
7. *Heckbert P.S. F*undamentals of texture mapping and image warping [M.S. thesis]. Department of Electrical Engineering, University of California, Berkeley, Calif, USA. 1989.
8. *Gonzalez R, Rafael R.* Digital image processing. NY: Pearson. 2018. 1168 p.
9. *Abramowitz M., Stegun I.A.* Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables. NY: Dover Publications Inc. 1965. 1046 p.
10. *Otsu N.A.* Threshold Selection Method from Gray-Level Histograms. IEEE Trans. Sys. Man. Cyber. 1979. V. 9. № 1. P. 62–66.
11. *Koleda P, Hrčková M.* Global and Local Thresholding Techniques for Sawdust Analysis. Acta Facultatis Technicae. 2018. Vol XXIII. No 1. P. 33–42.
12. *Trubitsyn A., Grachev E.* Switching median filter for suppressing multi-pixel impulse noise. Computer Optics. 2021. V. 45. № 4. P. 580–588.

A UNIVERSAL ALGORITHM FOR DISCRETIZING BICHROMATIC TWO-DIMENSIONAL GRAPHIC CODES

© 2024 A. A. Trubitsyn[a, b], M. V. Shadrin[b], S. I. Holkin[c]

[a] *Ryazan State Radio Engineering University named after V.F. Utkina*
59/1 st. Gagarina, Ryazan, 390005 Russia
[b] *Kvantron Group LLC*
28a st. Engelsa, room. H2, Ryazan, 390010 Russia
[c] *LLC "Operator-CRPT"*
15 Rochdelskaya st., building 16A, premises I, Moscow, 123376 Russia

Mathematical foundations and algorithms for recognizing bichromatic two-dimensional graphic codes, regardless of their type (QR codes, DataMatrix, GridMatrix, etc.) are presented. The stages of achieving the result include detecting the code, localizing it within an arbitrary quadrilateral, transforming the quadrilateral to a canonical square, constructing a grid of elements (modules) of the square code, and filling it with a sequence of bits. It is shown that perspective transformation formulas make it possible to transform localized quadrangular regions to canonical squares with an acceptable error level for further processing. A flat grid of square code elements is formed based on the search for extrema of the derivatives of the pixel intensity distribution of the square image along the axes x and y. The algorithm for filling grid cells (code modules) with a sequence of zeros and ones uses information about the average intensity of each such cell. At the end of the paper, the algorithms are tested on a variety of real images of two-dimensional codes, and the limitations of the proposed algorithms are examined.

Keywords: two-dimensional graphic code, QR code, DataMatrix, pattern recognition, Hough transform, perspective transformation, numerical differentiation, thresholding

REFERENCES

1. *Trubitsyn A.A., Shadrin M.V., Serezhin A.A.* Localization of image fragments with high frequency intensity oscillation. Journal of Autonomous Intelligence. 2023. V. 6. № 2. P. 1–16.
2. *Trubitsyn A.A., Shadrin M.V.* Obnaruzheniye, lokalizatsiya i transformatsiya dvumernogo graficheskogo koda (Detection, localization and transformation of two-dimensional graphics code). GraphiCon 2023: 33rd International Conference on Computer Graphics and Computer Vision, September 19–21, 2023, Institute of Control Problems. V.A. Trapeznikov Russian Academy of Sciences, Moscow, Russia. 2023. P. 509–516.
3. *Karrach L., Pivarciova E.* Options to use data matrix codes in production engineering. Management Systems in Production Engineering. 2018. V. 26. № 4. P. 231–236.

4. *Yamaguchi et al.* Code type determining method and code boundary detecting method. US Patent 2005/ O121520 A1. 09.06.2005.
5. *Szentandrási I., Herout A., Dubská M.* Fast detection and recognition of QR codes in high-resolution images. SCCG '12: Proceedings of the 28th Spring Conference on Computer Graphics March. 2013. P. 129–136.
6. *Lin J.A., Fuh C.S.* 2D barcode image decoding. Mathematical Problems in Engineering. 2013. 2013: 848276.
7. *Heckbert P.S. F*undamentals of texture mapping and image warping [M.S. thesis]. Department of Electrical Engineering, University of California, Berkeley, Calif, USA. 1989.
8. *Gonzalez R, Rafael R.* Digital image processing. NY: Pearson. 2018. 1168 p.
9. *Abramowitz M., Stegun I.A.* Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables. NY: Dover Publications Inc. 1965. 1046 p.
10. *Otsu N.A.* Threshold Selection Method from Gray-Level Histograms. IEEE Trans. Sys. Man. Cyber. 1979. V. 9. № 1. P. 62–66.
11. *Koleda P, Hrčková M.* Global and Local Thresholding Techniques for Sawdust Analysis. Acta Facultatis Technicae. 2018. Vol XXIII. No 1. P. 33–42.
12. *Trubitsyn A., Grachev E.* Switching median filter for suppressing multi-pixel impulse noise. Computer Optics. 2021. V. 45. № 4. P. 580–588.

ПРОГРАММИРОВАНИЕ, 2024, № 5, с. 54–74

ТЕОРИЯ ПРОГРАММИРОВАНИЯ: ФОРМАЛЬНЫЕ МОДЕЛИ И СЕМАНТИКА

УДК 004.436

ФОРМАЛЬНАЯ СПЕЦИФИКАЦИЯ И ВЕРИФИКАЦИЯ ТРЕБОВАНИЙ В АРХИТЕКТУРЕ И СТРОИТЕЛЬСТВЕ НА ОСНОВЕ ЯЗЫКА МОДЕЛИРОВАНИЯ EXPRESS

© 2024 г. В. А. Семенов[a],*, С. В. Морозов[a],, С. В. Аришин[a],***, О. Н. Кузина[b],*****, В. И. Римшин[b],*****, Е. В. Макиша[b],********

[a] *Институт системного программирования им. В.П. Иванникова РАН*
109004 Россия, Москва, ул. А. Солженицына, д. 25
[b] *Национальный исследовательский Московский государственный строительный университет*
129337 Россия, Москва, Ярославское шоссе, д. 26
**e-mail: sem@ispras.ru*
***e-mail: serg@ispras.ru*
****e-mail: arishin@ispras.ru*
*****e-mail: kuzinaon@mgsu.ru*
******e-mail: RimshinVI@mgsu.ru*
*******e-mail: makishaev@mgsu.ru*

Поступила в редакцию: 23.04.2024 г.
После доработки: 03.05.2024 г.
Принята к публикации: 03.05.2024 г.

В настоящее время цифровые технологии моделирования зданий, сооружений и инфраструктуры успешно применяются в международной и российской практике реализации сложных строительных проектов и масштабных программ. Вместе с тем, реализуемый во многих странах переход к машиночитаемым стандартам с целью повышения качества проектной документации и автоматизации ее проверки сталкивается с серьезными методологическими и инструментальными проблемами. Прежде всего они связаны со сложностью самих цифровых моделей, а также с разнообразием требований, формулируемых на естественных языках и предъявляемых к моделям на государственном, региональном, ведомственном и корпоративном уровне. Предпринимаемые попытки создания реестров требований и программных инструментов для их ведения и использования, как правило, носят специализированный характер и не обеспечивают необходимую полноту, нормализацию, согласованность, взаимосвязанность, однозначность, прослеживаемость и проверяемость описаний требований. Конструктивным, в связи с этим, представляется применение формальных методов спецификации и верификации требований, зарекомендовавших себя в системной и программной инженерии. В работе проводится сравнительный анализ программных инструментов для автоматизированной проверки нормативных требований в области строительства. Отмечается возросшая популярность инструментов, ориентированных на международные стандарты IFC (Industry Foundation Classes), IDS (Information Delivery Specification) и обеспечивающих контроль полноты объектного и атрибутного состава моделей, а также уточнение допустимых областей значений. Вместе с тем, стандарт IDS не формализован и не предусматривает задание требований, выражаемых произвольными алгебраическими условиями. Перспективным для формальной спецификации и верификации требований к цифровым моделям в строительстве представляется применение языка моделирования объектно-ориентированных данных EXPRESS, на котором специфицирована и информационная схема IFC. В качестве обоснования показывается представимость IDS спецификаций логическими выражениями и функциями EXPRESS, а также возможность задания произвольных алгебраических условий в виде декларативных правил языка EXPRESS. В качестве иллюстраций предлагаемого подхода приводятся примеры формализации строительных нормативов и сводов правил РФ, предъявляемых к безопасности зданий, сооружений и процессов. Также обсуждаются возможности гармонизации предлагаемого формального подхода со стандартом IDS в результате определения новых паттернов для представления локальных, уникальных и глобальных правил языка EXPRESS.

Keywords: BIM, IFC, EXPRESS, IDS, спецификация и верификация требований, инженерия требований

DOI: 10.31857/S0132347424050052, **EDN:** OLMAMS

1. ВВЕДЕНИЕ

В настоящее время цифровые технологии моделирования зданий, сооружений и инфраструктуры успешно применяются в международной и российской практике реализации сложных строительных проектов и масштабных программ. Модели разрабатываются в соответствующих САПР, а при обмене между участниками проектной деятельности, как правило, преобразуются в файловые форматы в соответствии с международными и национальными стандартами по функциональной совместимости (интероперабельности) программного обеспечения. Ключевыми среди них являются открытые стандарты и сервисы openBIM, разработанные международным альянсом buildingSMART [1]. Прежде всего, это IFC (Industry Foundation Classes) [2], IDM (Information Delivery Manual) [3], BCF (BIM Collaboration Format) [4], bSDD (buildingSMART Data Dictionary) [5], которые в настоящее время активно используются ведущими индустриальными странами. Детальная информация о стандартах, сервисах и технологиях buildingSMART содержится в монографии [6].

Вместе с тем, реализуемый во многих странах переход к машиночитаемым стандартам с целью повышения качества проектной документации, автоматизации и интеллектуализации ее проверки сталкивается с серьезными методологическими проблемами [7]. Одной из главных причин является отсутствие единого понимания, каким образом должны быть организованы реестры требований, чтобы обеспечить необходимую полноту (complete), нормализацию (normalized), детализируемость (granular), уникальность (unique set), согласованность (consistent), взаимосвязанность (linked set), модифицируемость (modifiable) описаний требований, а также каким образом должны быть представлены сами требования, чтобы обеспечить необходимый уровень абстрактности (abstract) при однозначной интерпретируемости (unambiguous), прослеживаемости (traceable) и проверяемости (validatable). Данные принципы закреплены в международном стандарте по системной инженерии [8].

Методами управления требованиями в соответствии с перечисленными принципами занимается дисциплина “инженерия требований” [9], в которой для обеспечения математически строгих гарантий корректной спецификации требований и их однозначной интерпретации применяются формальные методы [10]. Например, для контроля целостности данных и согласованности состояний в приложениях системной и программной инженерии, разработанных на основе универсального языка моделирования UML (Unified Modeling Language) [11], часто применяется метод контрактных спецификаций с инвариантами, пред- и постусловиями на языке объектных ограничений OCL (Object Constraint Language) [12].

Однако в архитектурно-строительной отрасли формальные методы ограниченно применяются из-за сложности в освоении пользователями, не являющимися IT-специалистами. Хотя стандартная информационная схема IFC, определяющая типы данных и правила целостности цифровых моделей, специфицирована на декларативном языке моделирования объектно-ориентированных данных EXPRESS [13], конечные пользователи, как правило, не способны формализовать и описать на этом языке собственные правила, выражающие те или иные требования к моделям. Другими факторами, затрудняющими переход к машиночитаемым стандартам, являются разнообразие норм и сводов правил, предъявляемых к моделям на государственном, региональном, ведомственном, корпоративном уровне; неоднозначность и нестрогость формулировок на естественных языках; необходимость в определении алгебраически полного языка, обеспечивающего формальную и конструктивную спецификацию разнообразных требований.

В качестве примера реализации формального подхода к спецификации и верификации требований следует указать систему EDM Express Data Checker [14], в которой требования оформляются в виде декларативных правил EXPRESS. Поскольку информационные схемы цифровых моделей, подлежащих верификации, также специфицируются на EXPRESS, решается проблема несоответствия метамоделей (impedance mismatch) и обеспечивается необходимая математическая строгость в представлении и интерпретации требований. Вместе с тем, данный инструмент не предусматривает функциональных расширений, которые могли бы существенно упростить формализацию и спецификацию норм и правил, например, с учетом особенностей стандарта IFC.

Важное место в практике верификации цифровых моделей на основе стандарта IFC занимают инструменты, имеющие в своем арсенале предопределенные, готовые к использованию шаблоны проверок. Самым популярным, безусловно, является система Solibri Model Checker [15]. К этому семейству следует также отнести 7D Modeler

[16], BIMcollab Zoom [17], BlenderBIM [18], Open IFC Viewer [19], usBIM.checker [20], которые позволяют выполнить типовые проверки объектной организации модели, ее атрибутного состава и пространственного размещения элементов. Принципиальным недостатком подобных инструментов и императивного подхода, в целом, является функциональная ограниченность. Наличие прикладных программных интерфейсов не решает проблему, поскольку отсутствуют спецификации исходных требований, нейтральные по отношению к платформам, языкам и системам реализации, а их неформальная интерпретация может приводить к разным вердиктам.

Альтернативу рассмотренным подходам составляют инструменты, сочетающие в себе декларативные средства для описания требований, как правило, в терминах специализированных словарей или классификационных систем, и императивные средства для проверки выполнимости требований применительно к заданным цифровым моделям. Требования обычно задаются с помощью специальных редакторов и представляются на диалектах языков правил, подобных RuleML, LRML, SWRL, LDML [21]. Несмотря на кажущуюся гибкость эклектичных инструментов, они неизбежно порождают проблемы несоответствия метамоделей и недостоверности результатов из-за необходимости интерпретации правил в терминах целевых моделей и реализации промежуточного слоя между декларативными и императивными средствами.

В качестве особого семейства следует указать инструменты, ориентированные на перспективный открытый стандарт IDS (Information Delivery Specification) [22], официальный релиз 1.0 которого планируется к принятию альянсом buildingSMART во второй половине 2024 года. Стандарт предусматривает спецификацию требований в терминах информационной схемы IFC, что решает упомянутую выше проблему несоответствия. Примером полнофункциональной реализации стандарта IDS являются сервисы и редактор машиночитаемых требований ИСП РАН [23]. Сервисы обеспечивают контроль соответствия цифровой модели схеме IFC, а также выполняют проверки полноты ее объектного и атрибутного состава, корректности областей значений, заданных в виде регулярных выражений, перечислимых множеств, числовых интервалов, а также отношений композиции и агрегации. Вместе с тем, на сегодняшний день стандарт IDS не формализован, а также не предусматривает средств для спецификации сложных требований, выражаемых параметрическими, пространственными, топологическими, метрическими, темпоральными и прочими алгебраическими условиями. Известны подходы к расширению IDS с использованием императивных средств и отмеченными выше проблемами [24, 25].

Настоящая статья адресована проблемам и возможностям применения языка моделирования EXPRESS для формальной спецификации и верификации требований в архитектуре и строительстве. В разделе 2 обсуждаются открытые международные стандарты IFC, IDS и BCF, применяемые для автоматизированной проверки цифровых моделей зданий, сооружений и инфраструктуры. Раздел 3 посвящен особенностям декларативного языка EXPRESS и его основным конструкциям для определения объектных типов данных и задания правил целостности на них. В разделе 4 проводится формализация стандарта IDS в результате представления стандартных паттернов логическими выражениями и функциями EXPRESS. Примеры формализации строительных нормативов и сводов правил РФ, предъявляемых к безопасности зданий, сооружений и процессов, приводятся в разделе 5 в качестве обоснования универсальности и гибкости языка EXPRESS для спецификации требований, выражаемых сложными алгебраическими условиями. Наконец, в разделе 6 описываются результаты апробации предлагаемого формального подхода, возможности его гармонизации со стандартом IDS и расширения последнего путем определения новых паттернов для представления локальных, уникальных и глобальных правил языка EXPRESS.

2. ОТКРЫТЫЕ СТАНДАРТЫ ИНФОРМАЦИОННОГО МОДЕЛИРОВАНИЯ В АРХИТЕКТУРЕ И СТРОИТЕЛЬСТВЕ

Industry Foundation Classes (IFC) – это открытая информационная схема и стандартизованный формат данных, независимые от поставщиков программного обеспечения (ПО) и разработанные международным альянсом buildingSMART для обеспечения интероперабельности ПО в области архитектуры и строительства (AEC). Будучи одобренным ISO в качестве международного стандарта [2], а также принятым в РФ в качестве соответствующего ГОСТ [26], IFC (версия 4.0.2.1) широко используется различными программными приложениями и инструментами [27] для обмена междисциплинарными данными (включая архитектуру, описание механических,

электрических систем, систем отопления, вентиляции и кондиционирования, графики работ, сметы, модели для структурного и энергетического анализа и т. д.) и их совместного использования различными участниками архитектурно-строительных проектов. В марте 2024 года ISO приняла версию 4.3.2.0 [28], ранее одобренную buildingSMART и расширяющую информационную схему IFC понятиями, используемыми в области строительства инфраструктурных объектов (дороги, мосты, станции, тоннели, водные пути, каналы, пристани, порты и т. п.).

Базируясь на методологии STEP (STandard for the Exchange of Product model data, ISO 10303) [29], модель данных IFC специфицируется на языке объектно-ориентированного моделирования EXPRESS [13] и поддерживает хранение и транспортировку данных в форматах, предоставляемых STEP, включая:

• IFC STEP Physical File (SPF) – формат кодирования STEP-данных открытым текстом в соответствии со стандартом ISO 10303–21:2016 [30] (наиболее широко используемый формат в настоящее время);

• IFC XML – формат XML для представления STEP-данных, определенный стандартом ISO 10303–28:2007 [31] (не получил широкого распространения из-за раздутого размера XML-файлов по сравнению с SPF);

• IFC ZIP – файл IFC SPF или IFC XML, сжатый в формате ZIP;

• IFC HDF – бинарный Hierarchical Data Format, определенный технической спецификацией ISO/TS10303–26:2011 [32] для представления STEP-данных (имеющий экспериментальный статус для IFC).

Заметим, что модель данных IFC сочетает в себе как объектно-ориентированный, так и онтологический [33] подходы. К последнему относятся следующие возможности:

• отдельные определения для типов объектов (наследники типа *IfcTypeObject*) и экземпляров объектов или индивидов (наследники типа *IfcObject*) для расширения семантики модели путем создания новых экземпляров типов и их ассоциации с соответствующими индивидами;

• объектификация взаимоотношений (наследники типа *IfcRelationship*), позволяющая расширить семантику языка EXPRESS путем определения иерархии различных типов взаимоотношений (включая ассоциации, композиции, назначения, определения, декларации, связи) и хранить характеристики взаимоотношений отдельно от типов взаимосвязанных объектов;

• определение дополнительных статических свойств объектов и их типов наряду со стандартными атрибутами в виде динамически расширяемых наборов качественных (*IfcPropertySet*) и количественных (*IfcQuantitySet*) характеристик;

• дополнительная категоризация объектов и их типов с использованием предопределенных таксономических схем или систем классификации (*IfcClassification*).

Детальную информацию об IFC, его использовании и развитии можно найти в разделе 3.2 монографии [6], а также в следующих источниках [34–37].

В настоящее время все большую популярность приобретает открытый стандарт IDS (Information Delivery Specification) [22], разрабатываемый сообществом buildingSMART и предназначенный для перевода требований к цифровой информационной модели (ЦИМ) объекта строительства, представленной в соответствии со стандартом IFC, в машиночитаемый вид и их дальнейшей верификации с использованием инструментов, поддерживающих данный стандарт [27]. Текущая версия IDS0.9.6 позволяет формализовать типовые проверки ЦИМ, в частности, полноту ее атрибутивного состава, наличия требуемых дополнительных свойств, соответствия установленным правилам классификации, именования, соответствия значения указанному образцу, его принадлежность заданному множеству или интервалу.

С технической точки зрения IDS документ представляет собой XML файл с приписанной схемой XSD, которая должна применяться для контроля соответствия данных, содержащихся в этом файле, стандарту IDS. В частности, подобный XML файл должен содержать общий заголовок документа и набор формализованных правил. Общий заголовок включает в себя название документа, краткое назначение, развернутое описание, а также версию документа, владельца авторских прав, e-mail авторов, дату создания и этапа жизненного цикла объекта капитального строительства, на котором предполагается использование данных правил.

Каждое формализованное правило состоит из следующих компонентов:

• общая информация (general), в которой указывается название спецификации, идентификатор, назначение, инструкции, версии стандарта IFC, которым соответствует данное правило, а также статус, устанавливающий обязательный или рекомендуемый характер соответствующего требования;

• контекст или применимость (applicability), задаваемый декларативным образом, используя объектный запрос или, менее формально, фильтр, результатом которого являются выбранные объекты исходной модели в IFC, подлежащие проверке выполнимости соответствующего требования;

• требование (requirement), которое задается с использованием тех же декларативных конструкций, что позволяет интерпретировать правило как формальное суждение о принадлежности каждого объекта, удовлетворяющего условиям контекста, множеству объектов, удовлетворяющих условиям требования.

Для задания контекста и требования в IDS предусмотрено шесть предопределенных паттернов (facets), каждый из которых позволяет определить условия соответствующего вида:

• *Entity*: принадлежность объектов указанным объектным типам данных IFC с возможностью их уточнения предопределенными типами при их наличии для заданных типов объектов в спецификации модели IFC (например, в наборе данных должны существовать плиты *IfcSlab* с предопределенным типом “перекрытие”

IfcSlabTypeEnum.FLOOR:
Name="IFCSLAB",
PredefinedType="FLOOR");

• *PartOf*: задание отношений композиции между объектами заданных типов с возможностью уточнения вида композиции: агрегация, группировка, принадлежность пространственной структуре, вложение, высечение, заполнение (например, некоторый объект должен быть частью навесной стены типа *IfcCurtainWall*: *Relation="IFCRELAGGREGATES", Entity="IFCCURTAINWALL"*);

• *Attribute*: наличие значений у атрибутов, предопределенных схемой IFC, и принадлежность их заданным областям определения допустимых значений, при этом отсутствие области определения значений соответствует наличию определенного (*NOT NULL*) значения (например, имя объекта *Name* должно быть не длиннее 50 символов: *Name="Name", Value="MaxLength=50"*);

• *Property*: наличие установленных дополнительных свойств объектов (property или quantity), а также, опционально, указание соответствия их значений заданному типу данных и области определения допустимых значений (например, в наборе основных количественных параметров плиты должен быть указан ее чистый объем, значение которого должно быть в диапазоне от 20 до 100 кубических метров: *Property Set="Qto_SlabBaseQuantities", Name="NetVolume", DataType="IFCVOLUMEMEASURE", Value="20<= Value<=100"*);

• *Classification*: наличие у объектов назначенных классификаторов и их соответствие заданным системам классификации UniFormat [38], OmniClass [39], КСИ [40] и т. п., при этом указание только системы классификации означает возможность использования произвольных классификаторов из данной системы, а задание паттерна без параметров соответствует наличию произвольных классификаторов у данного объекта (например, объекты должны быть классифицированы с использованием системы OmniClass, классификатор должен начинаться с
'23-': System="OmniClass", Value="23-"*);

• Material: наличие у строительных элементов IFC назначенных материалов, сопоставляемых по имени или категории, при этом пустое значение в данном паттерне соответствует наличию произвольного материала (например, элементы должны быть стальные либо бетонные: *Value=["steel", "concrete"]*).

Для более детального знакомства со стандартом IDS можно обратиться к статье [41], разделу 3.7 монографии [6] и технической документации на сайте проекта [42].

В процессе верификации требований необходимо сформировать детальный отчет, включающий, в том числе, и информацию об обнаруженных нарушениях. С этой целью наиболее удобно использовать журнал замечаний в формате BCF (BIM Collaboration Format) [4], открытая спецификация которого также разрабатывается и поддерживается альянсом buildingSMART. Текущая версия BCF – 3.0. BCF представляет собой файловый архив в формате ZIP, включающий несколько XML файлов со стандартными схемами для описания используемой версии формата и ее расширения, самого журнала замечаний, включающего традиционные для отслеживания замечаний понятия (замечание, комментарий, точка обзора, снимок, документ), каталог используемых документов со ссылками на них, а также сопутствующие документы (изображения, нормативы, и т. п.) для детального разъяснения обнаруженных замечаний. Замечания BCF ссылаются на соответствующие объекты в IFC файле, используя их GUID. buildingSMART предоставляет API для работы с BCF в режимах обмена файлами между отдельными приложениями и REST API для использования BCF в веб-сервисах. Детальную информацию о BCF и его использовании можно найти в разделе 3.4 монографии [6], техническую

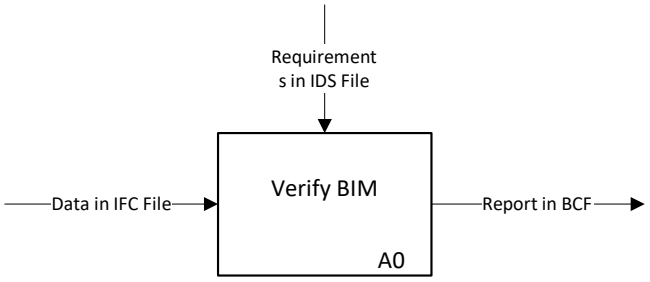


Рис. 1. Процесс верификации требований в строительстве на основе открытых стандартов.

документацию по BCF API – на сайте [43]. Список поддерживающего данный формат программного обеспечения находится в общей базе совместимого ПО buildingSMART [27].

Таким образом, международный альянс buildingSMART предоставляет развитое семейство открытых стандартов для представления ЦИМ объектов строительства, описания требований к ним и формирования журналов замечаний к ЦИМ в процессе их формирования и применения. Организация процесса верификации с использованием данных стандартов изображена на рис. 1. Приложение верификации принимает данные в формате IFC и требования к ним в IDS файле и выдает отчет в виде журнала замечаний в формате BCF.

3. ЯЗЫК МОДЕЛИРОВАНИЯ ДАННЫХ EXPRESS

EXPRESS [13] является декларативным языком информационного моделирования, предназначенным для независимого от особенностей реализации формального описания объектов реального мира, определения их статических свойств (структур данных) и ограничений целостности. Являясь частью методологии стандартов семейства STEP [29], он широко используется для описания моделей данных в различных отраслях промышленности (машиностроение, строительство и архитектура, нефте- и газодобыча, судостроение и т. д.).

EXPRESS – объектно-ориентированный язык моделирования данных и основной структурой данных в нем является объект типа *ENTITY*. Остальные типы данных используются при определении атрибутов объектных типов. К ним относятся простые типы данных с общепринятой семантикой: числовые (*NUMBER*, *INTEGER*, *REAL*), логические (*BOOLEAN*, *LOGICAL*), символьные и битовые строки (*STRING*, *BINARY*). К сложным типам, кроме объектного (*ENTITY*), относят также перечисляемые типы (*ENUMERATION*), выборки (*SELECT*), различные коллекции: упорядоченные (*ARRAY*, *LIST*) и неупорядоченные (*BAG*, *SET*). Типы данных, включая простые, возможно переопределять с помощью конструкции *TYPE* с указанием имени нового типа и, опционально, ограничения определения области допустимых значений. В EXPRESS предусматриваются и обобщенные типы данных, которые могут использоваться при определении формальных параметров функций и процедур. Тип *GENERIC* является обобщением всех допустимых в языке типов данных. Тип данных *AGGREGATE* является обобщением для типов коллекций *ARRAY*, *LIST*, *BAG*, *SET* и их специализаций. Более подробно о типах данных языка EXPRESS можно узнать из стандарта [13], а также из следующих статей [36, 44–46].

Для настоящей статьи наиболее интересно рассмотреть возможности спецификации правил, определяющих ограничения целостности данных. В языке EXPRESS допустимы определения правил трех типов:

• локальные правила (*WHERE*) в объектных или переопределенных типах, определяющие ограничения области допустимых значений отдельного атрибута или типа данных, или же некоторые зависимости между значениями атрибутов в объектном типе;

• правила уникальности (*UNIQUE*) в объектных типах, устанавливающие условия уникальности значений одного атрибута или их комбинации на экстенте соответствующего объектного типа данных;

• глобальные правила (*RULE*), задающие ограничения целостности для экстентов одного или нескольких объектных типов.

Несмотря на то, что язык EXPRESS является декларативным, в определениях локальных и глобальных правил применяются императивные конструкции: выражения, а также вызовы предопределенных и определяемых пользователем функций и процедур, в теле которых допускается использование операторов присваивания, условного перехода, выбора, циклов, возврата, как и в традиционных языках программирования. Выражения языка EXPRESS отличаются богатством синтаксических конструкций. В них можно использовать следующие операции и предопределенные функции:

• арифметические (включая целочисленное и вещественное деление), сравнения, нахождение значения в интервале, тригонометрические, возведение в степень, вычисление экспоненты, логарифмов, квадратного корня, модуля, опре-

деление четности, преобразование числа в строку и строки в число для числовых типов данных;

• логические (and, or, xor, not), сравнения для логических типов данных;

• конкатенация, определение длины, выделение подстроки, символа/бита по индексу, сравнения для символьных и битовых строк, сравнение с образцом только для символьных строк;

• сравнения для перечисляемых типов данных;

• глубокое сравнение на равенство или неравенство, определение идентичности или неидентичности экземпляров, определение множества имен ролей, исполняемых экземпляром в связях с другими объектами, определение мультимножества экземпляров;

• использующих заданный объект в указанной роли, конструирование экземпляра (с возможностью конкатенации конструкторов для создания экземпляра сложного типа), обращение к атрибуту, доступ к подтипу (взятие всех атрибутов, определенных в этом подтипе) для объектных типов данных;

• теоретико-множественные, сравнение на равенство или неравенство, определение идентичности или неидентичности элементов, подмножества и надмножества, принадлежности элемента (по равенству или идентичности), наличия уникальных по значению элементов, количества элементов, нижней и верхней границы (индекса для массивов), запрос (итерирование с вычислением заданного логического выражения для каждого элемента и включением элемента в результирующую коллекцию в случае истинности этого выражения), вставка/удаление элемента (только для списков), взятие одного или нескольких элементов для типов коллекций;

• проверка существования значения, задание альтернативного значения для неопределенного состояния, определение множества имен типов данных, которым принадлежит заданное значение для любого типа данных.

Продемонстрируем пример записи локального правила для трехмерного объекта, получаемого путем экструзии: перемещения сечения в заданном направлении на фиксированное расстояние (см. рис. 2). Правило *ValidExtrusionDirection* устанавливает, что заданное направление экструзии не должно быть перпендикулярно оси Z в локальной системе координат объекта. В нем используется вызов функции *IfcDotProduct* для вычисления скалярного произведения двух векторов.

Таким образом, на языке EXPRESS можно сформулировать достаточно сложные локальные и глобальные правила, используя богатый репертуар его синтаксических конструкций.

4. ФОРМАЛИЗАЦИЯ СТАНДАРТА IDS

Рассмотрим, как можно представить паттерны IDS с помощью конструкций языка EXPRESS. Подобные конструкции (логические выражения

```
ENTITY IfcExtrudedAreaSolid
 SUPERTYPE OF
(IfcExtrudedAreaSolidTapered)
 SUBTYPE OF (IfcSweptAreaSolid);
  ExtrudedDirection : IfcDirection;
  Depth : IfcPositiveLengthMeasure;
 WHERE
  ValidExtrusionDirection :
IfcDotProduct(
  IfcRepresentationItem() ||
  IfcGeometricRepresentationItem() ||
  IfcDirection([0.0,0.0,1.0]),
  SELF.ExtrudedDirection) <> 0.0;
END_ENTITY;
```

```
FUNCTION IfcDotProduct
  (Arg1, Arg2 : IfcDirection) : REAL;
LOCAL
  Scalar : REAL;
  Vec1, Vec2 : IfcDirection;
  Ndim   : INTEGER;
END_LOCAL;

  IF NOT EXISTS (Arg1) OR NOT EXISTS (Arg2) THEN
    Scalar := ?;
  ELSE
    IF (Arg1.Dim <> Arg2.Dim) THEN
      Scalar := ?;
    ELSE
      BEGIN
        Vec1 := IfcNormalise(Arg1);
        Vec2 := IfcNormalise(Arg2);
        Ndim := Arg1.Dim;
        Scalar := 0.0;
        REPEAT i := 1 TO Ndim;
          Scalar := Scalar +
Vec1.DirectionRatios[i]*Vec2.DirectionRatios[i];
        END_REPEAT;
      END;
    END_IF;
  END_IF;
  RETURN (Scalar);
END_FUNCTION;
```

Рис. 2. Пример спецификации объекта с локальным правилом на языке EXPRESS.

и функции) используются для указания контекста применимости требований (выделения множества объектов для проверки их выполнимости) и формулирования самих требований, выражаемых простой конъюктивной формой в IDS. Тем самым, представимость паттернов IDS конструкциями EXPRESS служит обоснованием формализации всего стандарта IDS. Заметим, что при отображении в EXPRESS значений следует учитывать их реальные типы данных в соответствии со схемой IFC, поскольку типы данных в XML ограничиваются только строками, числами и булевыми значениями.

4.1. Представление паттерна Entity

Паттерн *Entity* можно представить в виде логического выражения, в состав которого входит функция *TYPEOF*. Так как согласно стандарта EXPRESS данная функция возвращает множество строк, содержащих названия всех допустимых типов данных для значения, передаваемого в нее в качестве параметра, при этом названия типов квалифицируются именем схемы, содержащей определение этого типа, то строку с именем типа, полученную из паттерна IDS, следует квалифицировать именем схемы стандарта IFC соответствующей версии. В IDS имена типов в данном паттерне могут задаваться как в виде простого значения (одиночной строки), так и в виде перечисления (множества строк). Поэтому для сравнения имен в первом случае следует использовать операцию *IN* для проверки вхождения названия типа во множество, возвращаемое функцией *TYPEOF*. Во втором случае следует выполнить операцию пересечения двух множеств и сравнить размер результата с нулем.

Если в паттерне используется необязательный параметр *PredefinedType*, то условие проверки принадлежности типу данных должно быть связано конъюнкцией с условием равенства значения атрибута *PredefinedType* в соответствующем типе объекта IFC заданному в IDS для случая простого значения или с условием вхождения этого значения во множество допустимых (операция *IN*) для случая перечисления значений в IDS. При этом строковые значения IDS должны быть преобразованы к соответствующим им перечисляемым литералам EXPRESS, так как в объектных типах IFC атрибут *PredefinedType*, как правило, определяется типами перечислений. Так, для приведенного выше примера паттерна *Entity* (*Name="IFCSLAB", PredefinedType="FLOOR"*) используется следующее логическое выражение EXPRESS:

```
'IFC4.IFCSLAB' IN TYPEOF(Obj) AND
Obj.PredefinedType = IfcSlabTypeEnum.FLOOR
```

Будем считать, что существует одноместный объектный предикат *IsClassOf*, первый параметр *Obj* которого является объектом корневого типа *IfcRoot*, чей конкретный тип необходимо проверить, второй параметр *Name* является множеством строк названий допустимых типов, а третий параметр *PredefinedType* задает множество (возможно, пустое) перечисляемых констант для уточнения предопределенного типа объекта. Типом данных для элементов множества *PredefinedType* должен быть селективный тип, объединяющий все перечислимые типы схемы данных IFC, используемые для определения предопределенных типов объектов.

```
FUNCTION IsAggregatedIn(
  PartObj : IfcObjectDefinition;
  ParentTypes : SET [1:?] OF STRING) : LOGICAL;
LOCAL
  ParentObj : IfcObjectDefinition;
END_LOCAL;
IF SIZEOF(PartObj.Decomposes) = 0 THEN
  RETURN (FALSE);
END_IF;
ParentObj :=
PartObj.Decomposes[1].RelatingObject;
IF (SIZEOF(TYPEOF(ParentObj) * ParentTypes) >
0) THEN
  RETURN (TRUE);
ELSE
  RETURN (IsAggregatedIn(ParentObj,
ParentTypes));
END_IF;
END_FUNCTION;
```

Рис. 3. Функция на языке EXPRESS для определения наличия отношений агрегации.

4.2. Представление паттерна PartOf

Рассмотрим представление паттерна *PartOf* для отношений агрегации. Так как поиск родительского объекта, представляющего целое в данном отношении, должен осуществляться рекурсивно согласно стандарту IDS, для этих целей разработана рекурсивная функция *IsAggregatedIn* (см. рис. 3). Допустимые типы родительского объекта передаются как множество строк и при сопоставлении используется операция пересечения данного множества с результатом функции *TYPEOF*, примененной к родительскому объекту.

Для отношений другого вида можно написать аналогичные функции на языке EXPRESS, которые в статье не приводятся для краткости изложения. Также существует функция *PartOf*, которая в зависимости от типа передаваемого объекта и списка типов отношений вызывает упомянутые вспомогательные функции для проверки его вхождения в те отношения, которые задаются входным параметром и являются допустимыми для его типа согласно спецификации IFC.

4.3. Представление паттерна Attribute

Паттерн *Attribute* можно представить в виде логических выражений согласно таблице 1 в зависимости от способа задания области определения допустимых значений в соответствующем паттерне IDS. В качестве примеров в таблице используются следующие требования к представлению данных о работах по проекту (тип данных *IfcTask*): для работ следует задавать описание; их длительность должна определяться как рабочее время (по календарю); статус работ указывается только тремя величинами: *NOTSTARTED*, *STARTED*, *COMPLETED*; значение идентификатора работ должно соответствовать следующему шаблону: первые три символа *'DT-'*, за которыми следуют два произвольных символа, далее через тире может быть записана произвольная строка; приоритет работы должен являться целым числом от 1 до 100; длина имени работы не должна превышать 50 символов.

Заметим, что синтаксис регулярных выражений в XML не соответствует применяемому в операции *LIKE* языка EXPRESS. Детально проблема отображения регулярных выражений не исследовалась, но, несомненно, простые шаблоны в образцах IDS могут быть представлены в EXPRESS с применением небольших и несложных преобразований. Если атрибут является необязательным согласно схеме IFC, то логическое выражение EXPRESS является конъюнкцией вызова функции *EXISTS* для проверки наличия определенного значения и соответствующей конструкции из таблицы 1 для проверки соблюдения области допустимых значений.

4.4. Представление паттерна Property

Рассмотрим возможное представление паттерна *Property* на языке EXPRESS. При этом для краткости ограничимся вариантом, где указываются только имена набора свойств и свойства для проверки их наличия в объекте без указания допустимого типа значений и области их определения. Соответствующая универсальная функция *HasProperty* применяется к объектам-наследникам типа *IfcObjectDefinition* (см. рис. 4). Поскольку в модели данных IFC наборы свойств назначаются по-разному различным потомкам типа *IfcObjectDefinition*: контекстам *IfcContext*, объектам *IfcObject* и их типам *IfcTypeObject* (в первых двух случаях – через отношение *IfcRelDefinesByProperties*, в последнем – напрямую), в ней сначала формируется вспомогательная коллекция назначенных наборов свойств *PropSets*, а затем осуществляется поиск нужного набора и свойства в нем по заданным именам. Функция для поиска свойства в наборе по имени *FindPropertyInSet* является вспомогательной.

Заметим, что в более сложном случае при указании допустимого значения и его типа для проверки области значений дополнительно вызывается функция *GetPropertyValue* (ее спецификация не приводится для краткости изложения), которая возвращает значение найденного свойства, а затем осуществляется проверка типа данных этого значения путем вызова функции *TYPEOF*, а также проверка области определения с помощью соответствующей конструкции согласно табл. 1 в зависимости от вида ограничения области определения значений в паттерне IDS.

4.5. Представление паттерна Classification

Паттерн *Classification* выражается на языке EXPRESS с помощью следующих логических выражений. При отсутствии в нем названий классификационной системы и классификатора данное выражение имеет вид:

```
SIZEOF(QUERY(temp <*
SELF\IfcObjectDefinition.HasAssociations |
'IFC4.IFCRELASSOCIATESCLASSIFICATION' IN
TYPEOF(temp))) > 0;
```

где *SELF* соответствует объекту одного из типов, унаследованных от *IfcObjectDefinition*, куда классификаторы могут быть назначены через отношение *IfcRelAssociatesClassification* согласно модели данных IFC.

```
FUNCTION FindPropertyInSet
  (PropSet : IfcPropertySetDefinition;
   Name : STRING) : LOGICAL;
LOCAL
  Properties : SET [1:?] OF IfcProperty;
  Quantities : SET [1:?] OF
IfcPhysicalQuantity;
END_LOCAL;
IF 'IFC4.IFCPROPERTYSET' IN TYPEOF(PropSet)
THEN
  Properties :=
PropSet\IfcPropertySet.HasProperties;
  REPEAT i:=1 TO HIINDEX(Properties);
    IF Properties[i].Name = Name THEN
      RETURN (TRUE);
    END_IF;
  END_REPEAT;
ELSE
  IF 'IFC4.IFCELEMENTQUANTITY' IN
TYPEOF(PropSet) THEN
    Quantities :=
PropSet\IfcElementQuantity.Quantities;
    REPEAT i:=1 TO HIINDEX(Quantities);
      IF Quantities[i].Name = Name THEN
        RETURN (TRUE);
      END_IF;
    END_REPEAT;
  END_IF;
END_IF;
RETURN (FALSE);
END_FUNCTION;


FUNCTION HasProperty
  (Obj : IfcObjectDefinition;
   PropSetName : STRING;
   PropName : STRING) : LOGICAL;
LOCAL
  PropSets : SET [1:?] OF
IfcPropertySetDefinition := [];
  Relations : SET [0:?] OF
IfcRelDefinesByProperties := [];
  Definition : IfcPropertySetDefinitionSelect;
  DefinitionSet : IfcPropertySetDefinitionSet;
END_LOCAL;
IF 'IFC4.IFCTYPEOBJECT' IN TYPEOF(Obj) THEN
  IF EXISTS(Obj\IfcTypeObject.HasPropertySets)
THEN
    PropSets :=
Obj\IfcTypeObject.HasPropertySets;
  ELSE
    RETURN(FALSE);
  END_IF;
ELSE
  IF 'IFC4.IFCOBJECT' IN TYPEOF(Obj) THEN
    Relations := Obj\IfcObject.IsDefinedBy;
  ELSE
    Relations := Obj\IfcContext.IsDefinedBy;
  END_IF;
  IF SIZEOF(Relations) = 0 THEN
    RETURN(FALSE);
  END_IF;
  REPEAT i:=1 TO HIINDEX(Relations);
    Definition :=
Relations[i].RelatingPropertyDefinition;
    IF 'IFC4.IFCPROPERTYSETDEFINITION' IN
TYPEOF(Definition) THEN
      PropSets := PropSets + Definition;
    ELSE
      IF 'IFC4.IFCPROPERTYSETDEFINITIONSET' IN
TYPEOF(Definition) THEN
        DefinitionSet := Definition;
        REPEAT j:= 1 TO
HIINDEX(DefinitionSet);
          PropSets := PropSets +
DefinitionSet[j];
        END_REPEAT;
      END_IF;
    END_IF;
  END_REPEAT;
END_IF;
REPEAT i:=1 TO HIINDEX(PropSets);
  IF PropSets[i]\IfcRoot.Name = PropSetName
THEN
    RETURN (FindPropertyInSet(PropSets[i],
PropName));
  END_IF;
END_REPEAT;
RETURN (FALSE);
END_FUNCTION;
```

Рис. 4. Функции на языке EXPRESS для поиска назначенных свойств.

Таблица 1. Отображение паттерна *Attribute* в язык EXPRESS

Параметр *Value* паттерна *Attribute* в IDS	**Конструкция языка EXPRESS**
Отсутствие *Name="Description"*	Вызов функции *EXISTS* *EXISTS(Description)*
Простое значение *Name="DurationType", Value="WORKTIME"*	Операция сравнения = *TaskTime.DurationType = IfcTaskDurationEnum.WORKTIME*
Перечисление *Name=»Status", Value=["NOTSTARTED", "STARTED", "COMPLETED"]*	Операция принадлежности множеству *IN* *Status IN ['NOTSTARTED', 'STARTED', 'COMPLETED']*
Образец *Name="Identification", Value="DT-..-*"*	Операция подобия *LIKE* *Identification LIKE 'DT-??-*'*
Интервал *Name="Priority", Value="0<Value<=100"*	Операции сравнения *0 < Priority <= 100*
Длина *Name="Name", Value="MaxLength=50"*	Вызов функции *LENGTH* и операции сравнения *LENGTH(Name) <= 50*

Если же в паттерне IDS указываются требуемые классификационные системы, например, OmniClass, то выражение имеет следующий вид:

```
SIZEOF(QUERY(temp <*
SELF\IfcObjectDefinition.HasAssociations |
('IFC4.IFCRELASSOCIATESCLASSIFICATION' IN
TYPEOF(temp)) AND
HasClassificationSystems(temp\IfcRelAssocat
esClassification.RelatingClassification,
['OmniClass'], FALSE)))) > 0;
```

Функция *HasClassificationSystems* (см. рис. 5) находит классификационную систему по назначенному классификатору с помощью вспомогательной рекурсивной функции *GetClassificationSystem* и осуществляет сопоставление ее имени с параметром, который может быть задан в паттерне в виде простой строки, перечисления или образца. Перечисление передается в виде множества строк *Systems*. В случае простой строки или образца данное множество состоит из одного элемента. В случае образца параметр *IsPattern* равен *TRUE*.

Если в паттерне указывается допустимое множество классификаторов, которое также может быть задано в виде простой строки, перечисления или образца, то в вышеприведенном логическом выражении после вызова функции *HasClassificationSystems* через конъюнкцию добавляется вызов функции *HasClassifiers*, которая формирует полное имя назначенного классификатора с учетом иерархии в классификационной системе и осуществляет его сопоставление с соответствующим параметром паттерна. Для краткости данная функция в статье не приводится.

4.6. Представление паттерна Material

Наличие произвольных материалов, назначенных на строительный элемент (если в паттерне IDS не указываются названия материалов), можно проверить с помощью следующего логического выражения на языке EXPRESS:

```
SIZEOF(QUERY(temp <*
SELF\IfcObjectDefinition.HasAssociations
|   'IFC4.IFCRELASSOCIATESMATERIAL'   IN
TYPEOF(temp))) > 0;
```

где *SELF* соответствует объекту одного из типов строительных элементов.

Если в паттерне IDS задаются имена материалов (в виде строки, перечисления или образца), то логическое выражение будет иметь следующий вид:

```
FUNCTION GetClassificationSystem(Ref : IfcClassificationReference) :
IfcClassification;
LOCAL
  Result : IfcClassification;
END_LOCAL;
IF 'IFC4.IFCCLASSIFICATION' IN TYPEOF(Ref.ReferencedSource) THEN
  Result := Ref.ReferencedSource;
ELSE
  Result := GetClassificationSystem(Ref.ReferencedSource);
END_IF;
RETURN(Result);
END_FUNCTION;


FUNCTION HasClassificationSystems(Classifier : IfcClassificationSelect;
  Systems : SET [1:?] OF STRING; IsPattern : BOOLEAN) : LOGICAL;
LOCAL
  Classification : IfcClassification;
END_LOCAL;
IF 'IFC4.IFCCLASSIFICATIONREFERENCE' IN TYPEOF(Classifier) THEN
  Classification := GetClassificationSystem(Classifier);
ELSE
  Classification := Classifier;
END_IF;
IF IsPattern THEN
  IF Classification.Name LIKE Systems[1] THEN
    RETURN (TRUE);
  END_IF;
ELSE
  IF Classification.Name IN Systems THEN
    RETURN (TRUE);
  END_IF;
END_IF;
RETURN (FALSE);
END_FUNCTION;
```

Рис. 5. Функция на языке EXPRESS для сопоставления используемых систем классификации.

```
SIZEOF(QUERY(temp <*
SELF\IfcObjectDefinition.HasAssociations
|   ('IFC4.IFCRELASSOCIATESMATERIAL'   IN
TYPEOF(temp)) AND
(HasMaterials(temp\IfcRelAssociatesMateri
al.RelatingMaterial, ['steel', 'concrete'],
FALSE)))) > 0;
```

Здесь используется приведенный выше пример наличия стальных или бетонных элементов. Функция *HasMaterials* сопоставляет назначенные материалы по имени или категории согласно требованиям стандарта IDS. На рис. 6 приведен небольшой ее фрагмент для двух типов материалов в IFC: простого материала и списка материалов. Остальные 8 типов сложных материалов IFC обрабатываются аналогично: сначала сопоставление идет по собственному имени или категории сложного материала, а затем сопоставляются простые материалы, входящие в состав сложного. Вспомогательная функция *MatchSimpleMaterial* работает только с типом простого материала *IfcMaterial*. Имена для сопоставления передаются как множество строк, что соответствует определению перечисления в IDS. В случае простого строкового значения это множество состоит из одного элемента. В случае образца во множество включается также один элемент, при этом параметр *IsPattern* равен *TRUE*. Во всех остальных случаях он должен быть равен *FALSE*.

Таким образом, все паттерны, определенные в текущей версии стандарта IDS, могут быть представлены с помощью конструкций языка EXPRESS и определены формально как одноместные объектные предикаты.

5. ФОРМАЛЬНАЯ СПЕЦИФИКАЦИЯ НОРМАТИВОВ И СВОДОВ ПРАВИЛ

В ходе формализации стандарта IDS была разработана библиотека функций на языке EXPRESS, которые проверяют факт выполнения условий, устанавливаемых паттернами, для объектов, типы которых определяются схемой IFC. С математической точки зрения данные функции представляют собой одноместные объектные предикаты, где первым аргументом является некоторый IFC объект, а остальные аргументы являются параметрами соответствующего паттерна. Эти функции можно рассматривать

```
FUNCTION MatchSimpleMaterial(Material : IfcMaterial;
  Names : SET [1:?] OF STRING; IsPattern : BOOLEAN) : LOGICAL;
IF IsPattern THEN
  IF Material.Name LIKE Names[1] THEN
    RETURN (TRUE);
  ELSE
    IF (EXISTS(Material.Category)) AND (Material.Category LIKE Names[1]) THEN
      RETURN (TRUE);
    END_IF;
  END_IF;
ELSE
  IF Material.Name IN Names THEN
    RETURN (TRUE);
  ELSE
    IF (EXISTS(Material.Category)) AND (Material.Category IN Names) THEN
      RETURN (TRUE);
    END_IF;
  END_IF;
END_IF;
RETURN (FALSE);
END_FUNCTION;


FUNCTION HasMaterials(Material : IfcMaterialSelect;
  Names : SET [1:?] OF STRING; IsPattern : BOOLEAN) : LOGICAL;
IF 'IFC4.IFCMATERIAL' IN TYPEOF(Material) THEN
  RETURN (MatchSimpleMaterial(Material, Names, IsPattern));
ELSE
  IF 'IFC4.IFCMATERIALLIST' IN TYPEOF(Material) THEN
    RETURN (SIZEOF(QUERY(temp <* Material\IfcMaterialList.Materials |
MatchSimpleMaterial(temp, Names, IsPattern))) > 0);
  ELSE
  ...
END_IF;
RETURN (FALSE);
END_FUNCTION;
```

Рис. 6. Функция на языке EXPRESS для сопоставления материалов по имени или категории.

как расширения стандартной схемы IFC, также специфицированной на языке EXPRESS. Тогда требования могут быть формализованы как правила языка EXPRESS (локальные, глобальные, уникальности), формулируемые с использованием определенного набора предикатов. В таком случае для верификации требований необходим транслятор языка EXPRESS в исполняемый код на одном из императивных языков для генерации части кода приложения верификации либо данное приложение должно включать интерпретатор EXPRESS для непосредственного вычисления соответствующих выражений в процессе работы.

Заметим, что для выражения пространственных условий (допустимого расстояния между объектами или их взаимного расположения) одноместных предикатов недостаточно. Будем предполагать, что для типов объектов, имеющих в IFC геометрическое представление (например, строительные элементы), могут быть реализованы двуместные предикаты, аргументами которых являются два объекта соответствующих типов для проверки выполнимости наложенных пространственных ограничений.

Проиллюстрируем предложенный подход на нескольких примерах из строительных нормативов и сводов правил РФ, предъявляемых к безопасности зданий, сооружений и процессов.

Согласно СП 60.13330.2020 [47] соотношение сторон для воздуховодов прямоугольного сечения не должно превышать 1 к 4. Поскольку данное требование определяет условие, которое должно выполняться для любого воздуховода прямоугольного сечения, то его можно сформулировать виде локального правила *CorrectRectangularDuct* для объектного типа *IfcDuctSegment*, определяющего элемент воздуховода. В модели данных IFC стандартные параметры воздуховодов (форма сечения *Shape*, размеры *NominalDiameterOrWidth*, *NominalHeight*) задаются как свойства внутри стандартного набора *PSet_DuctSegmentTypeCommon*, назначаемого объектам типа *IfcDuctSegment*. Тогда в логическом выражении данного правила можно задействовать определенные ранее функции *HasProperty* и *GetProperyValue*, как показано на рис. 7.

Согласно СП 1.13130.2020 [48] число эвакуационных выходов из здания должно быть не менее числа эвакуационных выходов с любого этажа здания. Чтобы проверить выполнение данного требования, необходимо считать количество объектов в модели (выходов из здания типа *IfcDoor*), удовлетворяющих соответствующему условию. Поэтому в данном случае требование формализуется как глобальное правило *EvacuationRule*, определенное на экстенте объектного типа *IfcDoor* (см. рис. 8).

При проверке условия используются три свойства у объектов-дверей, значения которых извлекаются с помощью определенной ранее функции *GetPropertyValue*:

1) идентификатор (как правило, номер) этажа, на котором находится дверь (свойство "*Этаж*" в наборе "*Местоположение*");
2) функциональное назначение двери в соответствии с пунктом 4.2 ГОСТ 475-2016 [49] (свойство "*Назначение*" в наборе "*Идентификация*", для наружных выходов из здания равно "*Н*");

```
ENTITY IfcDuctSegment SUBTYPE OF (IfcFlowSegment);
  PredefinedType : OPTIONAL IfcDuctSegmentTypeEnum;
 WHERE
  CorrectPredefinedType : NOT(EXISTS(PredefinedType)) OR
    (PredefinedType <> IfcDuctSegmentTypeEnum.USERDEFINED) OR
    ((PredefinedType = IfcDuctSegmentTypeEnum.USERDEFINED) AND
      EXISTS(SELF\IfcObject.ObjectType));
  CorrectTypeAssigned : (SIZEOF(IsTypedBy) = 0) OR
    ('IFCHVACDOMAIN.IfcDuctSegmentType' IN
      TYPEOF(SELF\IfcObject.IsTypedBy[1].RelatingType));
  CorrectRectangularDuct : HasProperty(SELF, 'Pset_DuctSegmentTypeCommon',
'Shape') AND
    HasProperty(SELF, 'Pset_DuctSegmentTypeCommon', 'NominalDiameterOrWidth')
AND
    HasProperty(SELF, 'Pset_DuctSegmentTypeCommon', 'NominalHeight') AND
    ((GetPropertyValue(SELF, 'Pset_DuctSegmentTypeCommon', 'Shape') <>
PEnum_DuctSegmentShape. RECTANGULAR) OR
    (0.25 <= (GetPropertyValue(SELF, 'Pset_DuctSegmentTypeCommon',
'NominalDiameterOrWidth') / GetPropertyValue(SELF,
'Pset_DuctSegmentTypeCommon', 'NominalHeight') <= 4.0));
END_ENTITY;
```

Рис. 7. Формализация требования для допустимого соотношения сторон воздуховодов в виде локального правила на языке EXPRESS.

```
RULE EvacuationRule FOR IfcDoor;
LOCAL
  Storeys : SET OF IfcText := [];
  BuildingExit : INTEGER := 0;
  Result  : LOGICAL := TRUE;
  Purpose : IfcText := '';
  Emergency : BOOLEAN := FALSE;
  Storey : IfcText := '';
END_LOCAL;
  REPEAT i := 1 TO HIINDEX(IfcDoor);
    Purpose := GetPropertyValue(IfcDoor[i], 'Идентификация', 'Назначение');
    Emergency := GetPropertyValue(IfcDoor[i], 'Пожарные параметры',
'Эвакуационный');
    IF ((Purpose = 'H') AND (Emergency = TRUE)) THEN
      BuildingExit := BuildingExit + 1;
    END_IF;
    Storey := GetPropertyValue(IfcDoor[i], 'Местоположение', 'Этаж');
    IF EXISTS(Storey) THEN
      Storeys := Storeys + Storey;
    END_IF;
  END_REPEAT;
  REPEAT i := 1 TO HIINDEX(Storeys);
      IF (SIZEOF(QUERY(temp <* IfcDoor |
         ((Storeys[i] = GetPropertyValue(temp, 'Местоположение', 'Этаж')) AND
           (GetPropertyValue(temp, 'Пожарные параметры', 'Эвакуационный') =
TRUE)))) > BuildingExit) THEN
      Result := FALSE;
      ESCAPE;
    END_IF;
  END_REPEAT;
  WHERE
    WR1 : Result = TRUE;
END_RULE;
```

Рис. 8. Формализация требования для количества эвакуационных выходов из здания в виде глобального правила на языке EXPRESS.

3) булевское значение, указывающее, что данная дверь является (или не является) эвакуационным выходом (свойство "*Эвакуационный*" в наборе "*Пожарные параметры*").

В первом цикле подсчитывается количество эвакуационных выходов из здания, а также формируется множество идентификаторов этажей, на которых располагаются двери. Во втором цикле по этажам здания с помощью операции запроса, применяемой к экстенту типа *IfcDoor*, формируется множество эвакуационных выходов на каждом этаже, подсчитывается число элементов в нем и сравнивается с подсчитанным ранее количеством эвакуационных выходов из здания.

Согласно требованиям ЦГЭ.ЦИМ-3.0 [50] помещения (тип *IfcSpace*) в модели должны иметь уникальный номер, задаваемый свойством "*Номер*" в наборе "*Идентификация*". Его можно представить в виде правила уникальности в объектном типе *IfcSpace*, как показано на рис. 9, переопределив данное свойство как производный атрибут *SpaceNumber* в данном объектном типе, поскольку в EXPRESS уникальность может устанавливаться только для атрибутов. Для извлечения значения свойства используется определенная ранее функция *GetPropertyValue*.

Рассмотрим, как можно представить требования об отсутствии геометрических коллизий между строительными элементами, например, приведенные в ЦГЭ.ЦИМ-3.0 для стен. Пусть реализована функция для проверки пересечения двух объектов, имеющих геометрическое представление (в IFC это объекты, чей тип является производным от *IfcProduct*) со следующей сигнатурой:

```
FUNCTION LieCloserThan(Obj1: IfcProduct; Obj2
: IfcProduct; Distance: REAL): LOGICAL;
```

Тогда данное требование формулируется как глобальное правило на экстенте объектного типа *IfcWall*, что продемонстрировано на рис. 10.

Таким образом, использование языка EXPRESS в сочетании со вспомогательными одноместными и двуместными предикатами позволяет формализовать сложные требования, что было невозможно в рамках стандарта IDS.

6. АПРОБАЦИЯ ПРЕДЛОЖЕННОГО ПОДХОДА

Благодаря проведенной формализации стандарта IDS перспективной видится идея его гармонизации с предложенным формальным

```
ENTITY IfcSpace SUBTYPE OF (IfcSpatialStructureElement);
  PredefinedType : OPTIONAL IfcSpaceTypeEnum;
  ElevationWithFlooring : OPTIONAL IfcLengthMeasure;
 INVERSE
  HasCoverings : SET [0:?] OF IfcRelCoversSpaces FOR RelatingSpace;
  BoundedBy : SET [0:?] OF IfcRelSpaceBoundary FOR RelatingSpace;
 DERIVE
  SpaceNumber : IfcText := GetPropertyValue(SELF, 'Идентификация', 'Номер');
 UNIQUE
  UR1 : SpaceNumber;
 WHERE
  CorrectPredefinedType : NOT(EXISTS(PredefinedType)) OR
    (PredefinedType <> IfcSpaceTypeEnum.USERDEFINED) OR
    ((PredefinedType = IfcSpaceTypeEnum.USERDEFINED) AND
      EXISTS(SELF\IfcObject.ObjectType));
  CorrectTypeAssigned : (SIZEOF(IsTypedBy) = 0) OR
    ('IFCPRODUCTEXTENSION.IfcSpaceType' IN
      TYPEOF(SELF\IfcObject.IsTypedBy[1].RelatingType));
END ENTITY;
```

Рис. 9. Формализация требования уникальности номеров помещений в виде правила уникальности на языке EXPRESS.

```
RULE CollisionMissingRule FOR IfcWall;
LOCAL
  Result  : LOGICAL := TRUE;
END_LOCAL;
  REPEAT i := 1 TO HIINDEX(IfcWall);
    REPEAT j := i + 1 TO HIINDEX(IfcWall);
      IF LieCloserThan(IfcWall[i], IfcWall[j],
0.02) THEN
        Result := FALSE;
        ESCAPE;
      END_IF;
    END_REPEAT;
    IF NOT(Result) THEN
      ESCAPE;
    END_IF;
  END_REPEAT;
  WHERE
    WR1 : Result = TRUE;
END_RULE;
```

Рис. 10. Формализация требования об отсутствии геометрических коллизий между стенами на языке EXPRESS.

подходом. Подобная гармонизация способна обеспечить простоту задания типовых требований, универсальность и гибкость алгебраической спецификации сложных требований, а также строгость и достоверность верификации цифровых моделей, тем самым, сочетая в себе достоинства декларативных и императивных средств.

Предлагаемый способ гармонизации состоит в развитии стандарта IDS и его расширении следующими возможностями:

• поддержка нескольких контекстов применимости требований, что необходимо, например, для определения пространственных и топологических условий;

• расширение набора паттернов IDS, включая, как минимум, паттерны, соответствующие трем видам декларативных правил языка EXPRESS (локальные, глобальные, уникальности);

• разработка библиотеки функций на языке EXPRESS, являющихся расширением стандартной схемы IFC и реализующих предикаты для типовых пространственных, топологических, метрических и прочих условий.

Предложенные идеи были успешно апробированы в ходе реализации редактора машиночитаемых требований ИСП РАН, который обеспечивает просмотр, модификацию спецификаций требований в соответствии со стандартом IDS, а также их документирование в виде файлов HTML на русском и английском языках. На рис. 11 показан снимок главного окна перспективного редактора. В верхней части снимка располагается общий заголовок документа, включающий параметры в соответствии со стандартом IDS, как описано в разделе 2. В качестве дополнительной опции указывается количество контекстов, необходимое для задания спецификации.

В таблице спецификаций, располагаемой ниже слева, выводятся название спецификации и ее статус, указывающий на соответствие стандарту и возможность ее корректного сохранения в виде XML файла. Для создания, редактирования, перемещения и удаления спецификаций предусмотрены меню команд и панель инструментов. Справа от таблицы выводится заголовок выделенной спецификации и ее текстовая нотация, дающая полное и непротиворечивое описание соответствующего требования.

Для просмотра и редактирования конкретной спецификации предназначено модальное окно с вкладками доступа к ее заголовку, контекстам и требованию (см. рис. 12). Задание условий на категории объектов, отношения, классификаторы, атрибуты, свойства и материалы осуществляется

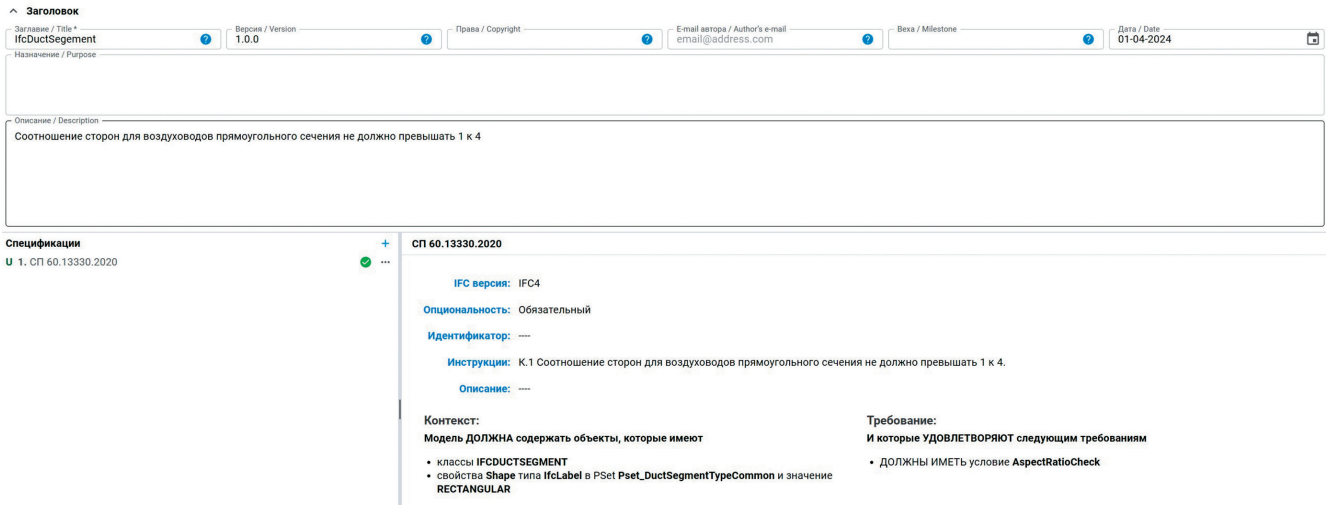


Рис. 11. Экранный снимок главного окна перспективного редактора требований.

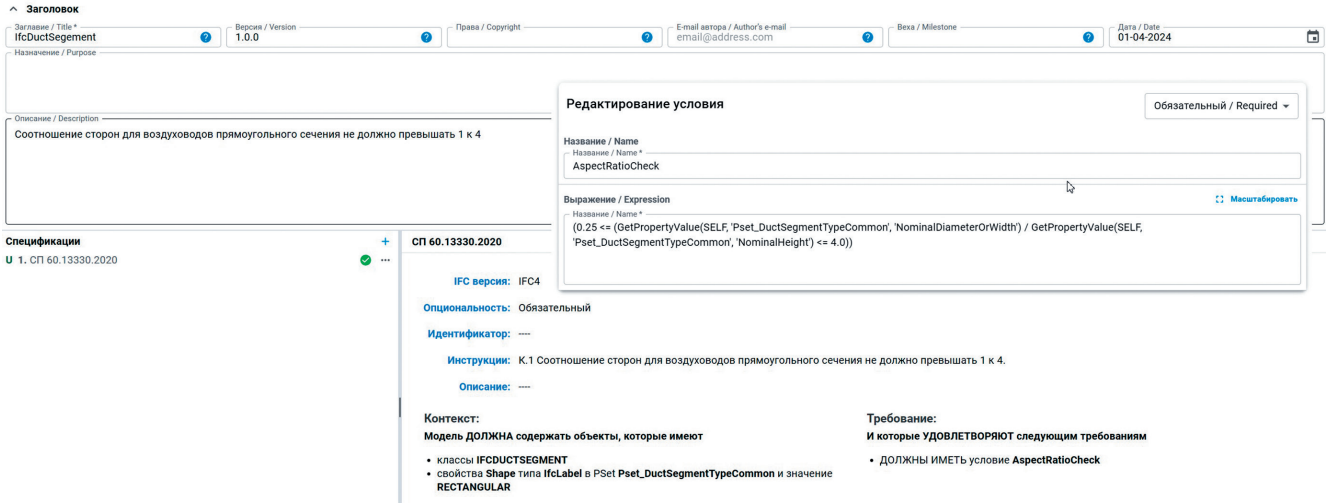


Рис. 12. Экранный снимок окна редактора спецификаций требований.

с помощью стандартных IDS паттернов, для поддержки которых предусмотрены меню в таблице условий и соответствующие элементы редактирования снизу. Для быстрого и верного ввода названий простых и объектных типов данных, а также многочисленных атрибутов, определяемых IFC схемой, в элементах редактирования реализована контекстная помощь, позволяющая выбирать допустимые названия из предлагаемого списка или иерархии.

Для реализации предлагаемого формального подхода в поддерживаемый набор стандартных IDS паттернов добавлено три новых паттерна, соответствующих декларативным правилам языка EXPRESS. При инстанциировании соответствующего правила в нижней части диалога отображаются элементы для ввода его названия и тела спецификации на языке EXPRESS. Парсер позволяет оперативно выявить допущенные пользователем синтаксические ошибки, а семантический контроль осуществляется на этапе верификации цифровой модели и интерпретации спецификации требования. На приведенном скриншоте показано представление требования соотношения сторон для воздуховодов прямоугольного сечения, ранее рассмотренного в разделе 5.

7. ЗАКЛЮЧЕНИЕ

Таким образом, в статье проведен сравнительный анализ программных инструментов для автоматизированной проверки нормативных требований в области архитектуры и строительства. Отмечена возросшая популярность инструментов, ориентированных на международные открытые стандарты IFC, IDS и BCF, а также выделены их ограничения для верификации сложных требований. Исследованы возможности применения

языка моделирования EXPRESS для формальной спецификации и верификации требований в архитектуре и строительстве, а также определены перспективы и способы гармонизации предложенного формального подхода с актуальным стандартом IDS. Планируемые дальнейшие исследования будут связаны с полнофункциональной реализацией редактора машиночитаемых требований с использованием декларативных правил языка EXPRESS, а также соответствующих сервисов верификации цифровых моделей, представленных в соответствии со стандартом IFC.

СПИСОК ЛИТЕРАТУРЫ

1. About buildingSMART International. https://www.buildingsmart.org/about, 22.04.2024.
2. ISO 16739-1:2018. Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries – Part 1: Data schema. Edition 1. 1474 p.
3. ISO 29481–1:2016. Building information models – Information delivery manual – Part 1: Methodology and format. Edition 2. 29 p.
4. BIM Collaboration Format (BCF) – buildingSMART Technical. https://technical.buildingsmart.org/standards/bcf, 22.04.2024.
5. buildingSMART Data Dictionary (bSDD) – buildingSMART Technical. https://technical.buildingsmart.org/services/bsdd, 22.04.2024.
6. *Eichler C.C., Schranz Ch., Krischmann T., Urban H., Hopferwieser M., Fischer S.* BIMcert Handbook – Basic knowledge openBIM. Edition 2024. Mironde-Verlag, Niederfrohna. 2024. 224 p. https://doi.org/10.34726/5383.
7. *Nisbet N.* Using uncertainty to link compliance and creativity. ECPPM 2021 – eWork and eBusiness in Architecture, Engineering and Construction: Proceedings of the 13th European Conference on Product & Process Modelling. 2021. CRC Press, London. P. 29–34. https://doi.org/10.1201/9781003191476-4.
8. ISO/IEC/IEEE29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering. Edition 2. 92 p.
9. *Hull E., Jackson K., Dick J.* Requirements Engineering, Third Edition. Springer-Verlag, London. 2011. 207 p.
10. *Kuliamin V.V.* Integration of verification methods for program systems. Programming and Computer Software. 2009. V. 35. № 4. P. 212–222. https://doi.org/10.1134/S0361768809040057.
11. ISO/IEC19505-1:2012. Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure. Edition 1. 220 p.
12. ISO/IEC19507:2012. Information technology – Object Management Group Object Constraint Language (OMG OCL). Edition 1. 233 p.
13. ISO 10303-11:2004. Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual. Edition 2. 255 p.
14. EXPRESS Data Manager™ – Jotne Connect. https://jotneit.com/products/express-data-manager, 22.04.2024.
15. *Soliman-Junior J., Tzortzopoulos P., Baldauf J.P., Pedo B., Kagioglou M., Formoso C.T., Humphreys J.* Automated compliance checking in healthcare building design. Automation in Construction. 2021. V. 129. https://doi.org/10.1016/j.autcon.2021.103822.
16. 7D Modeler | Open BIM Systems. https://www.openbimsystems.ru/en/7d-modeler, 22.04.2024.
17. BIMcollab Zoom: model validation that gets issues solved. https://www.bimcollab.com/en/products/bimcollab-zoom, 22.04.2024.
18. BlenderBIM Add-on – beautiful, detailed, and data-rich OpenBIM. https://blenderbim.org, 22.04.2024.
19. Open IFC Viewer. https://openifcviewer.com, 22.04.2024.
20. IFC Checker and BIM Validation | usBIM.checker | ACCA software. https://www.accasoftware.com/en/ifc-checker, 22.04.2024.
21. *Li B., Schultz C., Dimyadi J., Amor R.* Defeasible reasoning for automated building code compliance checking. ECPPM 2021 – eWork and eBusiness in Architecture, Engineering and Construction: Proceedings of the 13th European Conference on Product & Process Modelling. 2021. CRC Press, London. P. 229–236. https://doi.org/10.1201/9781003191476-32.
22. Information Delivery Specification IDS – buildingSMART Technical. https://technical.buildingsmart.org/projects/information-delivery-specification-ids, 22.04.2024.
23. Национальная ТИМ Платформа | Сервис валидации IFC моделей. URL: https://bim.ispras.ru/validate-ifc, 22.04.2024.
24. *van Berlo L., Willems P., Pauwels P.* Creating Information Delivery Specifications using linked data. Proceedings of the 36th International CIB W78 Conference. 2019. P. 647–660.
25. *Kremer N., Beetz J.* Extending Information Delivery Specification for linking distributed model checking services. Proceedings of the 40th International CIB W78 Conference. 2023. https://doi.org/10.35490/EC3.2023.266.
26. ГОСТ Р 10.0.02-2019/ИСО 16739-1:2018. Система стандартов информационного моделирования зданий и сооружений. Отраслевые базовые классы (IFC) для обмена и управления данными об объектах строительства. Часть 1. Схема данных. 28 с.
27. Software Implementations – buildingSMART Technical. https://technical.buildingsmart.org/resources/software-implementations, 22.04.2024.

28. ISO 16739-1:2024. Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries – Part 1: Data schema. Edition 2. 496 p.
29. ISO 10303–1:2024. Industrial automation systems and integration – Product data representation and exchange – Part 1: Overview and fundamental principles. Edition 3. 23 p.
30. ISO 10303–21:2016. Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure. Edition 3. 3 p.
31. ISO 10303–28:2007. Industrial automation systems and integration – Product data representation and exchange – Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas. Edition 1. 309 p.
32. ISO/TS10303–26:2011. Industrial automation systems – Product data representation and exchange – Part 26: Implementation methods: Binary representation of EXPRESS-driven data. Edition 1. 5 p.
33. *Staab S., Studer R. (eds.)* Handbook on Ontologies, Second Edition. Springer-Verlag, Berlin. 2009. 811 p.
34. IFC Implementation Guidance – buildingSMART Technical. https://technical.buildingsmart.org/resources/ifcimplementationguidance, 22.04.2024.
35. *van Berlo L.A.H.M., Beetz J., Bos P., Hendriks H., van Tongeren R.C.J.* Collaborative engineering with IFC: New insights and technology. ECPPM 2012 – eWork and eBusiness in Architecture, Engineering and Construction: Proceedings of the 9th European Conference on Product & Process Modelling. 2012. CRC Press, London. P. 811–818.
36. *Borrmann A., Beetz J., Koch C., Liebich T., Muhic S.* Industry Foundation Classes – A standardized data model for the vendor-neutral exchange of digital building models. Building Information Modeling – Technology Foundations and Industry Practice. Springer, Cham. 2018. P. 81–126. https://doi.org/10.1007/978-3-319-92862-3_5.
37. *van Berlo L., Krijnen T.F., Tauscher H., Liebich T., van Kranenburg A., Paasiala P.* Future of the Industry Foundation Classes: towards IFC5. Proceedings of the 38th International CIB W78 Conference. 2021. P. 123–137.
38. UniFormat® – Construction Specifications Institute. https://www.csiresources.org/standards/uniformat, 22.04.2024.
39. OmniClass® – Construction Specifications Institute. https://www.csiresources.org/standards/omniclass, 22.04.2024.
40. Классификатор Строительной Информации. http://ksi.faufcc.ru, 22.04.2024.
41. *Tomczak A., Benghi C., van Berlo L., Hjelseth E.* Requiring Circularity Data in BIM with Information Delivery Specification. Journal of Circular Economy. 2024. V. 1. N. 2. P. 1–13. https://doi.org/10.55845/REJY5239.
42. IDS – Machine readable Information Delivery Specification. URL: https://github.com/buildingSMART/IDS/blob/master/Documentation, 22.04.2024.
43. BIM Collaboration Format v3.0 Technical Documentation. https://github.com/buildingSMART/BCF-XML/tree/release_3_0/Documentation, 22.04.2024.
44. *Semenov V., Ilyin D., Morozov S., Tarlapan O.* Effective consistency management for large-scale product data // Journal of Industrial Information Integration. 2019. V. 13. P. 13–21. https://doi.org/10.1016/j.jii.2018.11.006.
45. *Морозов С.В., Ильин Д.В., Семенов В.А., Тарлапан О.А.* Библиотека ограничений для спецификации индустриальных моделей данных. Труды Института системного программирования РАН. 2015. Т. 27. № 4. С. 69–110. https://doi.org/10.15514/ISPRAS-2015–27(4)-5.
46. *Semenov V.A., Arishin S.V., Semenov G.V.* Formal rules to produce object notation for EXPRESS schema-driven data. Programming and Computer Software. 2022. V. 48. N. 7. P. 455–468. https://doi.org/10.1134/S0361768822070076.
47. Свод правил СП 60.13330.2020. Отопление, вентиляция и кондиционирование воздуха. 149 с.
48. Свод правил СП 1.13130.2020. Системы противопожарной защиты. Эвакуационные пути и выходы. 46 с.
49. ГОСТ 475–2016. Блоки дверные деревянные и комбинированные. Общие технические условия. 35 с.
50. Требования к цифровым информационным моделям объектов капитального строительства, представляемым для проведения экспертизы. Редакция 3.0 (ЦГЭ.ЦИМ-3.0). СПб ГАУ “Центр Государственной Экспертизы”. 2022. 203 с.

FORMAL SPECIFICATION AND VERIFICATION OF REQUIREMENTS IN ARCHITECTURE AND CONSTRUCTION USING THE EXPRESS MODELING LANGUAGE

© 2024 V. A. Semenov[a,*], S. V. Morozov[a,], S. V. Arishin[a,***], O. N. Kuzina[b,****], V. I. Rimshin[b,*****], E. V. Makisha[b,******]**

[a] *Ivannikov Institute for System Programming, RAS*
25, Alexander Solzhenitsyn str., Moscow, 109004 Russia
[b] *National Research University Moscow State University of Civil Engineering*
26, Yaroslavskoye shosse, Moscow, 129337 Russia

Currently, digital technologies for modeling buildings and infrastructure are successfully used in international and national practice in the implementation of complex construction projects and large-scale programmes. At the same time, the transition to machine-readable standards being implemented in many countries in order to improve the quality of design documentation and to automate its verification faces serious methodological and technical problems. First of all, they are associated with the complexity of the digital models, as well as with the variety of requirements formulated in natural languages and imposed on models at the state, regional, departmental and corporate levels. Attempts to create catalogues of requirements and software tools for their maintenance and use, usually have specialized nature and do not provide the necessary completeness, normalization, consistency, linked set, unambiguity, traceability and validatability of requirement descriptions. In this regard, it seems constructive to use formal methods for specification and verification of requirements that have proven themselves in system and software engineering. The paper provides a comparative analysis of software tools for automated verification of regulatory requirements in the construction domain. There is an increased popularity of tools focused on the international standards IFC (Industry Foundation Classes), IDS (Information Delivery Specification) and providing control of the completeness of the object and attribute composition of models, as well as clarification of acceptable ranges of values. At the same time, the IDS standard is not formalized and does not provide for specifying the requirements expressed by arbitrary algebraic conditions. The use of the object-oriented data modeling language EXPRESS, in which the IFC information schema is also specified, looks promising for the formal specification and verification of requirements for digital models in construction. As a justification, it is shown that IDS specifications can be represented by logical expressions and EXPRESS functions, as well as arbitrary algebraic conditions can be specified as EXPRESS declarative rules. Examples of the formalization of some requirements from national construction standards and set of rules on the safety of buildings, structures and processes are provided in the paper as illustrations of the proposed approach. The possibilities of harmonizing the proposed formal approach with the IDS standard as a result of defining new facets for representing EXPRESS local, unique and global rules are also discussed.

Keywords: BIM, IFC, EXPRESS, IDS, specification and verification, requirements engineering

REFERENCES

1. About buildingSMART International. https://www.buildingsmart.org/about, 22.04.2024.
2. ISO 16739-1:2018. Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries – Part 1: Data schema. Edition 1. 1474 p.
3. ISO 29481–1:2016. Building information models – Information delivery manual – Part 1: Methodology and format. Edition 2. 29 p.
4. BIM Collaboration Format (BCF) – buildingSMART Technical. https://technical.buildingsmart.org/standards/bcf, 22.04.2024.
5. buildingSMART Data Dictionary (bSDD) – buildingSMART Technical. https://technical.buildingsmart.org/services/bsdd, 22.04.2024.
6. *Eichler C.C., Schranz Ch., Krischmann T., Urban H., Hopferwieser M., Fischer S.* BIMcert Handbook – Basic knowledge openBIM. Edition 2024. Mironde-Verlag, Niederfrohna. 2024. 224 p. https://doi.org/10.34726/5383.
7. *Nisbet N.* Using uncertainty to link compliance and creativity. ECPPM 2021 – eWork and eBusiness in Architecture, Engineering and Construction: Proceedings of the 13th European Conference on Product & Process Modelling. 2021. CRC Press, London. P. 29–34. https://doi.org/10.1201/9781003191476-4.
8. ISO/IEC/IEEE29148:2018. Systems and software engineering – Life cycle processes – Requirements engineering. Edition 2. 92 p.
9. *Hull E., Jackson K., Dick J.* Requirements Engineering, Third Edition. Springer-Verlag, London. 2011. 207 p.
10. *Kuliamin V.V.* Integration of verification methods for program systems-. Programming and Computer Software. 2009. V. 35. № 4. P. 212–222. https://doi.org/10.1134/S0361768809040057.

11. ISO/IEC19505-1:2012. Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure. Edition 1. 220 p.
12. ISO/IEC19507:2012. Information technology – Object Management Group Object Constraint Language (OMG OCL). Edition 1. 233 p.
13. ISO 10303-11:2004. Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual. Edition 2. 255 p.
14. EXPRESS Data Manager™ – Jotne Connect. https://jotneit.com/products/express-data-manager, 22.04.2024.
15. *Soliman-Junior J., Tzortzopoulos P., Baldauf J.P., Pedo B., Kagioglou M., Formoso C.T., Humphreys J.* Automated compliance checking in healthcare building design. Automation in Construction. 2021. V. 129. https://doi.org/10.1016/j.autcon.2021.103822.
16. 7D Modeler | Open BIM Systems. https://www.openbimsystems.ru/en/7d-modeler, 22.04.2024.
17. BIMcollab Zoom: model validation that gets issues solved. https://www.bimcollab.com/en/products/bimcollab-zoom, 22.04.2024.
18. BlenderBIM Add-on – beautiful, detailed, and data-rich OpenBIM. https://blenderbim.org, 22.04.2024.
19. Open IFC Viewer. https://openifcviewer.com, 22.04.2024.
20. IFC Checker and BIM Validation | usBIM.checker | ACCA software. https://www.accasoftware.com/en/ifc-checker, 22.04.2024.
21. *Li B., Schultz C., Dimyadi J., Amor R.* Defeasible reasoning for automated building code compliance checking. ECPPM 2021 – eWork and eBusiness in Architecture, Engineering and Construction: Proceedings of the 13th European Conference on Product & Process Modelling. 2021. CRC Press, London. P. 229–236. https://doi.org/10.1201/9781003191476-32.
22. Information Delivery Specification IDS – buildingSMART Technical. https://technical.buildingsmart.org/projects/information-delivery-specification-ids, 22.04.2024.
23. Natsional'naya TIM Platforma | Servis validatsii IFC modelej [National BIM Platform | IFC model validation service] (in Russian). URL: https://bim.ispras.ru/validate-ifc, accessed 22.04.2024.
24. *van Berlo L., Willems P., Pauwels P.* Creating Information Delivery Specifications using linked data. Proceedings of the 36th International CIB W78 Conference. 2019. P. 647–660.
25. *Kremer N., Beetz J.* Extending Information Delivery Specification for linking distributed model checking services. Proceedings of the 40th International CIB W78 Conference. 2023. https://doi.org/10.35490/EC3.2023.266.
26. GOST R10.0.02-2019/ISO 16739-1:2018 Sistema standartov informatsionnogo modelirovaniya zdanij i sooruzhenij. Otraslevye bazovye klassy (IFC) dlya obmena i upravleniya dannymi ob ob'ektah stroitel'stva. Chast' 1. Shema dannyh [GOST R 10.0.02-2019/ISO 16739-1:2018. System of standards for building information modeling. Industry Foundation Classes (IFC) for exchanging and managing construction site data. Part 1. Data schema]. 28 p. (in Russian).
27. Software Implementations – buildingSMART Technical. https://technical.buildingsmart.org/resources/software-implementations, accessed 22.04.2024.
28. ISO 16739-1:2024. Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries – Part 1: Data schema. Edition 2. 496 p.
29. ISO 10303-1:2024. Industrial automation systems and integration – Product data representation and exchange – Part 1: Overview and fundamental principles. Edition 3. 23 p.
30. ISO 10303-21:2016. Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure. Edition 3. 3 p.
31. ISO 10303-28:2007. Industrial automation systems and integration – Product data representation and exchange – Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas. Edition 1. 309 p.
32. ISO/TS10303-26:2011. Industrial automation systems – Product data representation and exchange – Part 26: Implementation methods: Binary representation of EXPRESS-driven data. Edition 1. 5 p.
33. *Staab S., Studer R. (eds.)* Handbook on Ontologies, Second Edition. Springer-Verlag, Berlin. 2009. 811 p.
34. IFC Implementation Guidance – buildingSMART Technical. https://technical.buildingsmart.org/resources/ifcimplementationguidance, accessed 22.04.2024.
35. *van Berlo L.A.H.M., Beetz J., Bos P., Hendriks H., van Tongeren R.C.J.* Collaborative engineering with IFC: New insights and technology. ECPPM 2012 – eWork and eBusiness in Architecture, Engineering and Construction: Proceedings of the 9th European Conference on Product & Process Modelling. 2012. CRC Press, London. P. 811–818.
36. *Borrmann A., Beetz J., Koch C., Liebich T., Muhic S.* Industry Foundation Classes – A standardized data model for the vendor-neutral exchange of digital building models. Building Information Modeling – Technology Foundations and Industry Practice. Springer, Cham. 2018. P. 81–126. https://doi.org/10.1007/978-3-319-92862-3_5
37. *van Berlo L., Krijnen T.F., Tauscher H., Liebich T., van Kranenburg A., Paasiala P.* Future of the Industry Foundation Classes: towards IFC5. Proceedings of the 38th International CIB W78 Conference. 2021. P. 123–137.
38. UniFormat® – Construction Specifications Institute. https://www.csiresources.org/standards/uniformat, accessed 22.04.2024.

39. OmniClass® – Construction Specifications Institute. https://www.csiresources.org/standards/omniclass, accessed 22.04.2024.
40. Klassifikator Stroitel'noj Informatsii [Construction Information Classifier] (in Russian). http://ksi.faufcc.ru, accessed 22.04.2024.
41. *Tomczak A., Benghi C., van Berlo L., Hjelseth E.* Requiring Circularity Data in BIM with Information Delivery Specification. Journal of Circular Economy. 2024. V. 1. № 2. P. 1–13. https://doi.org/10.55845/REJY5239
42. IDS – Machine readable Information Delivery Specification. URL: https://github.com/buildingSMART/IDS/blob/master/Documentation, accessed 22.04.2024.
43. BIM Collaboration Format v3.0 Technical Documentation. https://github.com/buildingSMART/BCF-XML/tree/release_3_0/Documentation, accessed 22.04.2024.
44. *Semenov V., Ilyin D., Morozov S., Tarlapan O.* Effective consistency management for large-scale product data // Journal of Industrial Information Integration. 2019. V. 13. P. 13–21. https://doi.org/10.1016/j.jii.2018.11.006.
45. *Morozov S.V., Ilyin D.V., Semenov V.A., Tarlapan O.A.* Biblioteka ogranichenij dlya spetsifikatsii industrial'nyh modelej dannyh [A constraint library for specification of industrial data models]. 2015. V. 27. N. 4. P. 69–110 (in Russian). https://doi.org/10.15514/ISPRAS-2015–27(4)-5.
46. *Semenov V.A., Arishin S.V., Semenov G.V.* Formal rules to produce object notation for EXPRESS schema-driven data. Programming and Computer Software. 2022. V. 48. № 7. P. 455–468. https://doi.org/10.1134/S0361768822070076.
47. Svod pravil SP 60.13330.2020. Otoplenie, ventilyatsiya i konditsionirovanie vozduha [Set of rules SP 60.13330.2020. Heating, ventilation and air conditioning]. 149 p. (in Russian).
48. Svod pravil SP 1.13130.2020. Sistemy protivopozharnoj zashchity. Evakuatsionnye puti i vyhody. [Set of rules SP 1.13130.2020. Fire protection systems. Evacuation routes and exits]. 46 p. (in Russian).
49. GOST 475-2016. Bloki dvernye derevyannye i kombinirovannye. Obshchie tehnicheskie usloviya [GOST 475–2016. Wooden and combined door blocks. General technical conditions]. 35 p (in Russian).
50. Trebovaniya k tsifrovym informatsionnym modelyam ob'ektov kapital'nogo stroitel'stva, predstavlyaemym dlya provedeniya ekspertizy. Redaktsiya 3.0 (TGE. TIM-3.0) [Requirements for digital information models of capital construction projects submitted for expertise. Edition 3.0 (TGE.TIM-3.0)]. SPb GAU "Tsentr Gosudarstvennoj Ekspertizy" [SPb SAI "Center for State Expertise"]. 2022. 203 p (in Russian).